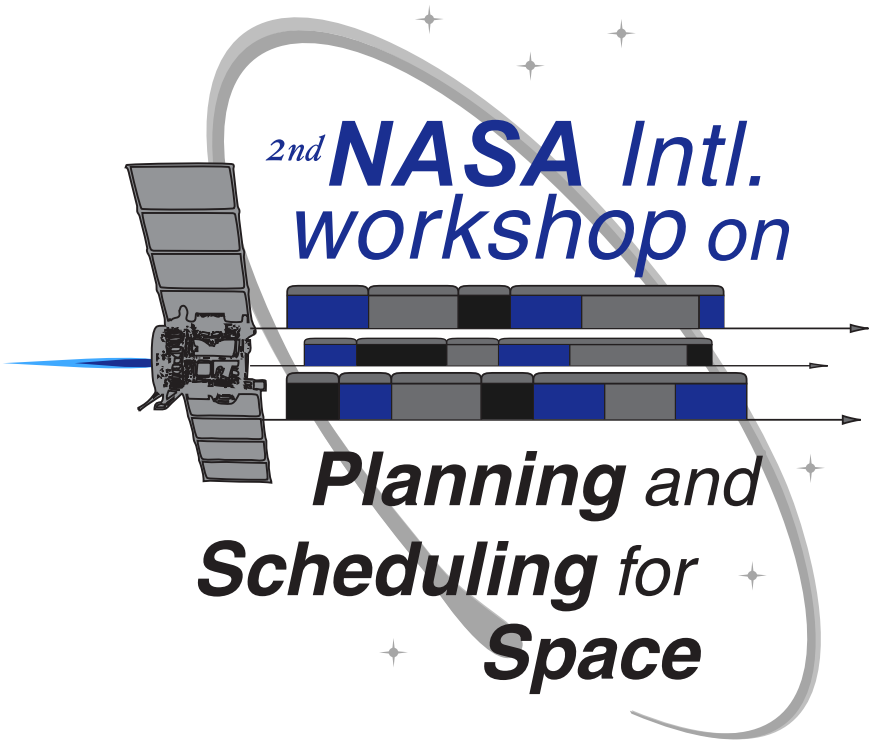
NASA Conference Proceedings NASA/CP– 2000-209590

Proceedings of the



Nicola Muscettola, NASA Ames (Chair)
Steve Chien, JPL (Co-Chair)

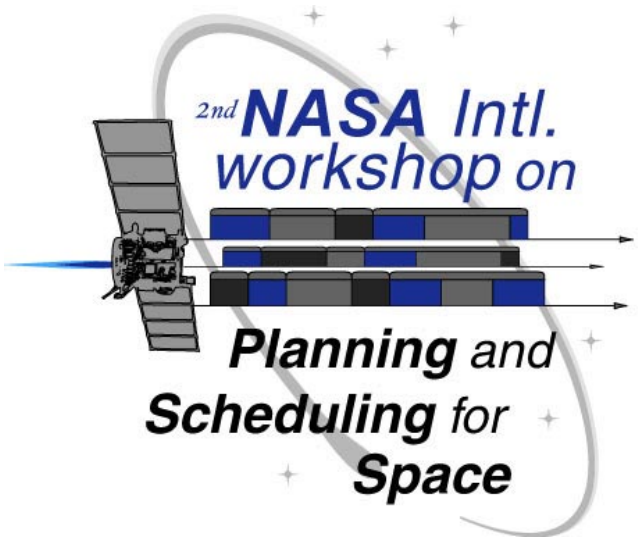
March 16-18 2000
San Francisco
California, USA

Sponsored by:

Computational Sciences Division, NASA Ames Research Center, Moffett Feild, CA

and

Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Feild, CA



March 16th to 18th, 2000
San Francisco, California

Organizing Committee:

Jeremy Frank	NASA Ames Research Center	(*frank@ptolemy.arc.nasa.gov*)
Keith Golden	NASA Ames Research Center	(*kgolden@ptolemy.arc.nasa.gov*)
Rich Washington	NASA Ames Research Center	(*richw@ptolemy.arc.nasa.gov*)

Program Committee:

Matthew Barry	United Space Alliance	(*mrb@alumni.rice.edu*)
Steve Chien	Jet Propulsion Laboratory (*co-chair*)	(*steve.chien@jpl.nasa.gov*)
Richard Creasey	European Space Agency	(*rcreasey@estec.esa.nl*)
Tara Estlin	Jet Propulsion Laboratory	(*tara.estlin@jpl.nasa.gov*)
Paul Hempel	NASA Goddard Space Flight Center	(*phempel@csc.com*)
David Kortenkamp	NASA Johnson Space Center	(*kortenkamp@jsc.nasa.gov*)
Nicola Muscettola	NASA Ames Research Center (*chair*)	(*mus@ptolemy.arc.nasa.gov*)
Karen Myers	SRI International	(*myers@ai.sri.com*)
Martha Pollack	University of Pittsburgh	(*pollack@cs.pitt.edu*)
Kanna Rajan	NASA Ames Research Center	(*kanna@ptolemy.arc.nasa.gov*)
Steve Smith	Carnegie Mellon University	(*sfs@cs.cmu.edu*)

This workshop proceedings will be available for purposes of citation as NASA Conference Proceedings number **NASA/CP—2000-209590** issued by the NASA Ames Research Center, Moffett Field, California. It will also be available electronically with the above number via the world wide web at **http://atrs.arc.nasa.gov.**

Contents

Preface / vii
Schedule of workshop events / ix
Biographies of workshop commentators / xv

Plenary Session

Long Range Planning for a New Class of Observatories / 2
Laurence A. Kramer
Commentary by David E. Smith / 8

Using Generic Preferences to Incrementally Improve Plan Quality / 11
Gregg Rabideau, Barbara Engelhardt and Steve Chien
Commentary by Alice F. Bowman / 17

Mission Planning Systems for Earth Observation Missions / 19
Marc Niézette
Commentary by Pete Bonasso / 25

Acting on Information: A plan language for manipulating data / 28
Keith Golden
Commentary by Patricia Johnson / 34

Automatic Planning for Autonomous Spacecrafts Constellation / 36
Eric Bornschlegl, Christophe Guettier and Jean-Clair Poncet
Commentary by Mark Boddy / 42

Automated Planning for Spacecraft and Mission Design / 45
Ben Smith and George Stebbins
Commentary by Peter Norvig / 50

Forecasting Telecommunications Support Boundaries for Satellite Constellations Using Deterministic Scheduling / 52
David Joesting and Gus Larsson
Commentary by Ken Galal / 58

Approximate Planning in the Probabilistic-Planning-as-Stochastic-Satisfiability Paradigm / 60
Stephen M. Majercik and Michael L. Littman
Commentary by Robert N. Brooks, Jr. / 66

Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling / 67
Steve Chien, Russell Knight, Andre Stechert, Rob Sherwood, and Gregg Rabideau
Commentary by Mario Merri / 73

SOFIA's Choice: Automating the Scheduling of Airborne Observations / 76
Jeremy Frank
Commentary by Sanford M. Krasner / 82

Toward Scheduling Over-Constrained Remote-Sensing Satellites / 84
Joseph C. Pemberton
Commentary by Gérard Verfaillie / 92

Merging the Semantics of State Constraints for Collections of Activities / 92
Russell Knight, Gregg Rabideau, and Steve Chien
Commentary by Miriam H. Nadel / 98

The XMM Mission Planning Process and Tools / 100
Mario Merri, Ian M. Palmer, Fabienne Delhaise Nestor Peccia
Commentary by Erann Gat / 106

Dealing with Temporal Uncertainty and Reactivity in a space mission plan / 107
Thierry Vidal
Commentary by Keyur Patel / 113

SWIM: An AI-based System for Organisation Management / 114
Pauline M. Berry and Brian Drabble
Commentary by Robert D. Rasmussen / 120

Designing and Evaluating an On-line On-board Autonomous Earth Observation Satellite Scheduling System / 122
Gérard Verfaillie and Éric Bornschlegl
Commentary by Peter R. Glück / 127

Planning in Interplanetary Space: Theory and Practice / 128
Ari Jónsson, Paul Morris, Nicola Muscettola, Kanna Rajan and Ben Smith
Commentary by Troy P. LeBlanc / 134

Challenges and Methods in Testing the Remote Agent Planner / 136
Ben Smith Martin S. Feather and Nicola Muscettola
Commentary by G. Michael Tong / 142

High-Stake Planning / 144
Sven Koenig
Commentary by Roger Arno / 151

Can Multiple Learning Agents Implement Responsiveness, Conflict Resolution, and Collaboration Between Autonomous Scheduling Systems and Human Operators? / 155
Keiki Takadama, Shinichi Nakasuka and Katsunori Shimohara
Commentary by Reid Simmons / 161

Optimizing Resource Utilization in Planetary Rovers / 163
Shlomo Zilberstein and Abdel-Illah Mouaddib
Commentary by Abdullah Aljabri / 169

From Rovers to Orbiters: Continuous Task-Distribution-Based Coordination / 170
Anthony Barrett
Commentary by Jonathan Gal-Edd / 176

Poster Session

Adapting Control Methods for Autonomous Exploration of Unknown Environments / 178
Barbara Engelhardt, Steve Chien and Darren Mutz

Automated Planning for the Antarctic Mapping Mission / 184
Ben Smith, Barbara Engelhardt and Darren Mutz

A Call for Knowledge-based Planning / 187
David E. Wilkins and Marie desJardins

Can Customers Schedule Their Own Payload Activities? / 193
John Jaap and Elizabeth Davis

Dispatchable Execution Of Schedules involving consumable resources / 199
R. J. Wallace and E. C. Freuder

Features of an Onboard, Adaptive, Observation Plan Executive for NASA's Next Generation Space Telescope / 205
Gary Welter, Leslye Boyce, Lou Hallock, Keith Kalinowski, Ken Rehm, Steve Tomkins, Glenn Cammarata, Jim Legg, Vicki Balzano, Rodger Doxsey, John Isaacs, and Ray Kutina

Heuristic Management of Execution-Time Task Conflicts / 211
Michael Freed

Integrating Planning and Execution for Multiple Rover Operations / 217
Tara Estlin, Gregg Rabideau, Darren Mutz, and Steve Chien

Interactive Autonomy for Space Applications / 223
Amedeo Cesta, Enrico Giunchiglia and Paolo Traverso

A Prototype for Ground-based Automated Rover Command Generation / 229
Rob Sherwood, Tara Estlin, Steve Chien, Gregg Rabideau, Barbara Engelhardt, Andrew Mishkin, Brian Cooper

Using Artificial Intelligence Planning to Generate Antenna Tracking Plans / 235
Forest Fisher, Tara Estlin, Darren Mutz and Steve Chien

Utility Estimation for Contingent Execution: Preliminary Report / 241
Richard Washington and John L. Bresina

Mixed Initiative Scheduling for the Air Force Satellite Control Network / 247
Adele E. Howe, L. Darrell Whitley, Laura Barbulescu and Jean-Paul Watson

Preface

This meeting is the second in a regular series started in October 1997 at Oxnard, California. Since then, the importance of automated planning and scheduling for the space enterprise has become increasingly clear. NASA technologists and computer scientists have also demonstrated the practical feasibility of these technologies in the context of real missions. For example, the Deep Space 1 Remote Agent Experiment in May 1999 for the first time demonstrated the use of a planner/scheduler operating within the high-level closed-loop control of a spacecraft traveling in interplanetary space. However, to make Planning and Scheduling a ubiquitous technology for space missions, many challenges still remain, including issues in design, development and fielding of such systems in mission critical areas of spacecraft operations. For example:

- **Responsiveness**: When operating within a closed-loop control system, issues related to responsiveness and balance between deliberation and reactivity become more and more important. So far we do not have good answers on how to coherently insert planning activities with different reactivity guarantees at the different levels of an optimizing, hierarchical control system.

- **Validation**: Validating the behavior of an automated planner/scheduler in an operational context is a major challenge. An automated planner makes it possible for a system to adapt its actions to changing execution conditions. However, we still do not know how to guarantee that a plan generated in a previously untested situation will indeed operate the system correctly and safely.

- **Mixed-Initiative autonomy**: As planning systems become an integral part of mission operation concepts, it becomes crucial to solve the problem of guaranteeing a seamless collaboration between automated schedulers and human operators. This includes support for variable levels of autonomy, representational formalisms for doing mixed-initiative reasoning, resolving conflicts between operator requests and existing plans or flight rules and providing the operator with explanations or insight into the behavior of the planning system.

- **Mission acceptance**: Gaining acceptance of planning and scheduling technology for real missions requires balancing the promise of advanced technology with the need for safety and reliability. The underlying representation, algorithms, interface with existing tools, and user interface all play important roles in the final usefulness and usability of the technology.

The papers in this workshop address these and other issues in the context of space missions and applications involving both completely automated systems and those with human intervention in the exploration of space. Within this area, planning and scheduling is important in (but not restricted to)

- Spacecraft commanding and payload operations;
- Operations of air, space and ground based scientific observatories;
- Scheduling of critical resources whether on the ground or onboard;
- Science data analysis;
- Design and analysis of spacecraft systems;
- Planning and scheduling of scientific experiments;
- Planning and scheduling for life support systems;
- Operations and payload scheduling for space transportation systems.

One important goal of the workshop was to foster a lively debate between academic researchers, technologists and mission users. For this reason the plenary session papers are accompanied by a commentary. We selected the author of the commentary to be a member of a different community than that

of the paper's author. For example, a paper with dealing primarily with scheduling research issues was commented by a mission operation specialist and vice versa.

I want to express my gratitude to the various organizations and individuals that have made this workshop possible. The program committee and the commentators were instrumental in guaranteeing the success of the paper review process. I am also grateful to USRA-RIACS and to the Computational Sciences Division of NASA Ames for their generous support. Least, but definitely not last, this workshop is happening primarily for the tireless effort of the organizing committee: Kanna Rajan, Jeremy Frank, Rich Washington, and Keith Golden. In addition, Bob Duffy and Nicole Masjedizadeh of the Code IC Outreach group have been of tremendous help with the publicity and publications aspect of the workshop. In particular, Kanna has been the major propulsive force behind the workshop, not just organizationally but also technically. Without him nothing would have happened.

Nicola Muscettola
Mountain View, CA
March 2000

Schedule of Workshop Events

March 15			
6:30-8:00	Registration and Reception		
March 16			
Time	Paper Title	Speaker	Commentator
8:45 - 9:00	Opening Remarks		
9:00 - 10:00	Invited talk Computing in the New Millenium: Uncertainty, Change and Limited Resources	Michael Georgeff Agentis International Inc.	
10:00-10:35	Break		
Session Chair: John Bresina			
10:35-11:10	Long Range Planning for a New Class of Observatories	L. Kramer	D. Smith
11:10-11:45	Generic Preference to Improve Plan Quality	Rabideau et. al.	A. Bowman
11:45-12:20	Mission Planning Systems for Earth Observation Missions	M. Niezette	P. Bonasso
12:20-1:20	Break		
1:20-2:00	Poster Session		
Session Chair: Steve Chien			
2:00-2:35	Acting on Information: A Plan Language for Manipulating Data	K. Golden	P. Johnson
2:35-3:10	Automatic Planning for Autonomous Spacecrafts Constellations	Bornschlegl et. al.	M. Boddy
3:10-3:45	Automated Planning for Spacecraft and Mission Design	Smith et. al.	P. Norvig
3:45-4:20	Break		
4:20-4:55	Forecasting Telecommunications Support Boundaries for Satellite Constellations Using Deterministic Scheduling	Joesting et. al.	K. Galal
4:55-5:30	Approximate Planning in the Probabilistic-Planning-As-Stochastic-Satisfiability Paradigm	Majercik et. al.	B. Brooks
5:30-6:30	Panel Discussion The Future of Planning and Scheduling Technology for Spacecraft Operations	Chair: Robert Morris RIACS	
6:30-7:30	Break		
7:30-10:00	Banquet		

	March 17		
Time	Paper Title	Speaker	Commentator
Session Chair: Amedeo Cesta			
8:45-9:00	Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling	Chien et. al.	M. Merri
9:00-9:35	SOFIA's Choice: Automating the Scheduling of Airborne Observations	J. Frank	S. Krasner
10:10-10:35	Break		
10:35-11:10	Towards Scheduling Over-Constrained Remote-Sensing Satellites	J. Pemberton	G. Verfaillie
11:10-11:45	Merging the Semantics of State Constraints for Collections of Activities	Knight et. al.	M. Nadel
11:45-12:20	The XMM Mission Planning	Merri et. al.	E. Gat
12:20-1:20	Break		
1:20-2:00	Poster Session		
Session Chair: Martha Pollack			
2:00-2:35	Dealing with Temporal Uncertainty and Reactivity in a Space Mission Plan	T. Vidal	K. Patel
2:35-3:10	SWIM: An AI-Based System for Organization Management	Berry et. al.	B. Rasmussen
3:10-3:45	Designing and Evaluating an On-Line On-Board Autonomous Earth Observation Satellite Scheduling System	G. Verfaillie	P. Gluck
3:45-4:20	Break		
4:20-4:55	Planning in Interplanetary Space: Theory and Practice	Jónsson et. al.	T. LeBlanc
4:55-5:30	Challenges and Methods in Testing the Remote Agent Planner	Smith et. al.	M. Tong
5:30-6:30	Panel Discussion Putting AI Planning and Scheduling to Work in Space Applications	Chair: Karen Myers SRI	
6:30-8:30	Break		
8:30-10:30	Breakout Session Mission Data Systems (MDS) Overview	Sanford Krasner JPL	

	March 18		
Time	Paper Title	Speaker	Commentator
9:00 - 10:00	Invited talk Solar System Exploration: Autonomy Challenges and NASA's Technology Programs	David Atkinson JPL	
10:00-10:35	Break		
Session Chair: Pierre Maldague			
10:35-11:10	High-Stake Planning	S. Koenig	R. Arno
11:10-11:45	Can Multiple Learning Agents Implement Responsiveness, Confict Resolution, and Collaboration Between Autonomous Scheduling Systems and Human Operators?	Takadama et. al.	R. Simmons
11:45-12:20	Optimizing Resource Utilization in Planetary Rovers	Zilberstein et. al.	A. Aljabri
12:20-12:45	From Rovers to Orbiters: Continuous Task Distribution Based Coordination	A. Barrett	J. Gal-Edd
12:55-1:20	Closing Remarks		

Biographies of workshop commentators

Abdullah Aljabri, JPL (*Abdullah.S.Aljabri@jpl.nasa.gov*): Abdullah is a group supervisor for the autonomy technology and infusion group at JPL. He has a B. Tech in Aeronautical Engineering and Design from Loghborough University of Technology (England) and MS in Aerospace Engineering from the Pennsylvania State University. He was the Autonomy/Flight software manager for the DS1 project. He has also served as the technical manager for the Guidance and Control Research Technology Objective Program. His current interests include spacecraft autonomy, model-based engineering and software process improvement.

Roger Arno, NASA Ames Research Center (*rarno@mail.arc.nasa.gov*): Roger joined NASA's Mission Analysis Division in 1966 where he began to apply decision analysis methods to advanced mission and technology tradeoffs. Later, working with remote sensing data, he developed training modules for sampling strategies. Subsequently, he managed the Space Station Life Sciences Research Facility Project, and has participated in numerous advanced mission concept developments from Earth-orbiting telescopes to Mars airplanes.

Mark Boddy, Honeywell Technology Center (*Mark.Boddy@honeywell.com*): Mark received a Ph.D. in Computer Science from Brown University in 1990 (title: "Solving Time-Dependent Problems: A Decision-Theoretic Approach to Planning in Dynamic Environments"). Since that time, he has been employed at the Honeywell Technology Center as a research scientist. His research interests include planning and scheduling, constraint-based reasoning, distributed problem solving, resource-bounded reasoning, and temporal reasoning.

Pete Bonasso, Metrica, Inc., (*r.p.bonasso@jsc.nasa.gov*): Pete is a senior staff consultant for AI & Robotics at Metrica, based at Johnson Space Center. He currently supports the Automation, Robotics, and Simulation Division investigations of intelligent monitoring and control using layered architectures. He is the co-developer of the 3T Robot Control Architecture, and has applied that architecture to the control of robotic and life support machines. He received his B.S. in engineering from the U.S. Military Academy at West Point in1968. After serving in the Army in Europe and in Vietnam, he received master's degrees in operation research and computer utilization from Stanford in 1974.

Alice F. Bowman, Johns Hopkins University Applied Physics Laboratory, (*Alice.Bowman@jhuapl.edu*): Alice is currently the manager of the MSX Operations Planning Center at the Applied Physics Laboratory in Laurel, MD. Prior to MSX, she worked on the Defense Support Program and in long wave infrared detector research. She holds a degree in Physics and Chemistry from the University of Virginia.

Robert Brooks, JPL (*Robert.N.Brooks@jpl.nasa.gov*): Bobby is currently with JPL, serving as a Service System Development Engineer for JPL's Telecommunications and Mission Operations Directorate (TMOD). He has been with JPL for 22 years. During that time he has been heavily involved in such projects as Voyager, Mars Observer, Mars Global Surveyor, Mars Surveyor Operations Project (MSOP), Space Infrared Telescope Facility, Space Interferometer Mission, the Genesis mission as well as several other

missions currently in development. He was the Mission Planning and Sequencing Team Chief for the MSOP from its inception in 1994 until 1998 when he took his current position with TMOD. He and his team developed the MSOP Automated Sequence Processor (ASP), which is a fully automated command and sequence generation processor. His current tasks include the development of a multi-mission flight operations capability for a wide variety of space missions planned for launch through 2010. He is the recipient of several NASA and JPL awards, including two agency medals. He has a BA in Astronomy from the University of Southern California.

Ken Galal, NASA/Ames Research Center (*kgalal@mail.arc.nasa.gov*): Ken performs mission design for the Project Operations Branch at NASA's Ames Research Center. His most recent project involved the Lunar Prospector mission, where he contributed to orbit maneuver design and operations. Prior to coming to Ames, Ken worked at Goddard Space Flight Center for close to ten years, where he performed flight dynamics analysis and operations on several missions, including ERBS,COBE, GRO, and GOES-I. Ken received his undergraduate degree in Mechanical Engineering from VA Tech, and a Masters degree in Engineering from Stanford.

Jonathan Gal-Edd, Goddard Space Flight Center (*jgaledd@pop500.gsfc.nasa.gov*): Jonathan has been working at NASA/GSFC since 1997. His current assignment is as lead software engineer for the Next Generation Space Telescope (NGST). NGST is the follow on program to HST. NGST is in Phase 1 (conceptual design) and is planned to fly 2008. Previously at GSFC Jonathan was part of development of the Earth Observing (EOS) information system (DIS). Prior to moving to GSFC from JSC, Jonathan was the Software Development Integration Lab (SDIL) for the International Space Station Program.

Peter Gluck, Jet Propulsion Laboratory (*peter.r.gluck@jpl.nasa.gov*): Peter joined NASA's Jet Propulsion Laboratory in 1986. He has served in the Reliability Engineering Section, Electric Power Systems Section, and the Autonomy and Control Section. Peter has worked on the Magellan, TOPEX/Poseidon, Mars Observer, Cassini, and Deep Space One missions, including three years as a flight software team leader for Deep Space One. In July of 1999 he was one of two applicants selected that year to JPL's Architect Development Program. Peter earned a B.S. in Mathematics from CSU, Northridge, and an M.S. in Aerospace Engineering from USC.

Patricia Johnson, Goddard Space Flight Center (*pjohnson@pop500.gsfc.nasa.gov*):
Patricia is currently monitoring the development, testing, and operations of the Earth Observing Sciences (EOS) Mission Planning and Scheduling System for Terra and Aqua satellites. In 1998 she managed the generation of EOS test science data needed for testing the Level-0, Level-1 and Level-2 science data processes. Before EOS, she worked with the Flight Dynamics Division monitoring software development and testing of orbit determination and orbit related programs, real time launch support system, and ground and space network acquisition planning data systems. From 1977 to 1990 she was a contractor and operated the real time launch support system, real time orbit determination system, and the real time acquisition data generation system. She has a B.A in Mathematics from the Univ. of Tennessee.

Sanford Krasner, Jet Propulsion Laboratory (*skrasner@jpl.nasa.gov*): Sandy specializes in system design and integration of spacecraft flight software. He has worked on Galileo, Cassini, Mars Observer and Deep Space 1 missions. He is currently the Guidance, Navigation & Control lead for the Mission Data System Project.

Troy P. LeBlanc, United Space Alliance (*troy.p.leblanc1@jsc.nasa.gov*): Troy has spent the past 8 years providing direct support to NASA/JSC in the Space Station Operations and Tool Development Group. He is responsible for integrating all aspects of software such as the Consolidated Planning System (CPS) and the Onboard Short Term Plan Viewer (OSTPV) targeted for use in ISS planning. Troy also led an effort called the International Distributed Planning Simulation #2 (IDPS#2) for demonstrating the international planning and scheduling interfaces for ISS. Troy holds a M.S. in Computer Science from the University of Houston. (Thesis topic: "Selection of Test Cases to Cost-Effectively Reduce the Risk of Failure of a Rule-Based Scheduling Engine"). He also holds a B.S. in Engineering Physics and an A.S. in Aeronautical Science, with a FAA Commercial Pilot License.

Mario Merri, European Space Operations Centre (*mmerri@esoc.esa.de*): Mario obtained his "Laurea" degree from the Politecnico di Milano, Italy, and his Masters and Ph.D. degrees from the University of Rochester, Rochester, NY, all in Electrical Engineering. He has been with the European Space Agency since 1989. He was responsible for the development of the CLUSTER Mission Planning System and Data Distribution System and is currently the Technical Officer of the XMM Science Control System. He is also active in space standardization.

Miriam H. Nadel, The Aerospace Corporation (*miriam.nadel@aero.org*): Miriam received her S.B. degree in 1980 from the Massachusetts Institute of Technology (M.I.T.) and her Ph.D. in 1990 from the University of California at Berkeley, both in mechanical engineering. Dr. Nadel is a Senior Project Engineer at the Aerospace Corporation, currently in the High Component Ground Segment group of the Space Based Infrared Surveillance program. Her responsibilities include simulation and performance assessment of algorithms for sensor tasking, tracking and discrimination.

Peter Norvig, NASA Ames Research Center (*PNorvig@arc.nasa.gov*): Peter is chief of the Computational Sciences Division at Ames and is the author of the books "*Artificial Intelligence: A Modern Approach*" and "*Paradigms of Artificial Intelligence: Case Studies in CommonLisp*" as well as many articles on a wide variety of AI topics. Peter obtained his Ph.D. from the University of California, Berkeley.

Keyur Patel, Jet Propulsion Laboratory (*keyur.c.patel@jpl.nasa.gov*): Keyur is currently the Project Engineer for the ST-3 mission and a Principal engineer in the Avionic Systems Engineering Section at the Jet Propulsion Laboratory. He started at JPL in 1985 and has worked on Voyager as the Attitude and Articulation Control Subsystem (AACS) and Flight Software (FSW) cognizant engineer, TOPEX AACS FSW, Mars Observer AACS and testbed development and operations, Mars Global Surveyor avionics team, Cassini avionics integration and test, and multi-mission modeling and simulation.

Robert D. Rasmussen, Jet Propulsion Laboratory (*Robert.D.Rasmussen@jpl.nasa.gov*): Bob is a principal engineer in the Avionic Systems Engineering section of the Jet Propulsion Laboratory, California Institute of Technology, where he is the Mission Data System's chief architect. He holds BS, MS, and PhD degrees in Electrical Engineering from Iowa State University. He has extensive experience in spacecraft attitude control and computer systems, test and flight operations, and automation and autonomy -- particularly in the area of spacecraft fault tolerance. He has worked on several JPL interplanetary missions since 1975, including Voyager and Galileo. Most recently, he was cognizant engineer for the Attitude and Articulation Control Subsystem on the Cassini mission to Saturn. During this time he also initiated and helped define the Remote Agent, an experimental application of artificial intelligence to spacecraft autonomy, which flew successfully on the Deep Space-1 mission.

Reid Simmons, Carnegie Mellon University (*reids@cs.cmu.edu*): Reid's research interests are in reliable autonomous systems. He has helped develop over a dozen autonomous robots, including the NASA-sponsored Ambler, Nomad and the Xavier indoor delivery robot. He has developed architectures for autonomy that integrate reactive and deliberative behaviors, probabilistic planning and reasoning for robot navigation, and was a member of the NASA Remote Agent project on Deep Space One.

David E. Smith, NASA Ames Research Center (*desmith@arc.nasa.gov*): Dave is head of the planning and scheduling group in the Computational Sciences Division where he is involved in research on contingency planning for rover operations, temporal planning, and search control for planning. Prior to joining NASA he was a Senior Scientist at the Rockwell Science Center, where his work led to a commercially fielded system for newspaper imposition planning, and to the Design Sheet conceptual design system in use throughout Rockwell and Boeing. He received his Ph.D. from Stanford University in 1985. Current research interests include temporal planning, planning under uncertainty, constraint satisfaction approaches to planning, and search control and preprocessing techniques for planning.

G. Michael Tong, Goddard Space Flight Center (*Gordon.M.Tong@gsfc.nasa.gov*): Mike is a software engineer in the Mission Applications Branch. He has worked on flight software, control center applications, and planning and scheduling systems.

Gerard Verfaillie, ONERA, Toulouse, France (*verfaillie@cert.fr*): Graduated from Ecole Polytechnique (Paris) and from ENSAE (National French Engineering School in Aerospace, Computer Science Specialization). Basic research on models, methods, and tools for constrained optimization. Numerous studies for CNES (French Space Agency) and ESA (European Space Agency) on Earth observation satellite scheduling problems.

Plenary Session

Generating a Long Range Plan for a New Class of Astronomical Observatories

Laurence A. Kramer

Space Telescope Science Institute (STScI)
3700 San Martin Drive
Baltimore, MD 21218
kramer@stsci.edu

Abstract

We present a long range planning (LRP) system, the Spike Plan Window Scheduler, which has been in use for observations on the Hubble Space Telescope (HST) for the past four years and which is being adapted for the Space Infrared Telescope Facility (SIRTF) and Next Generation Space Telescope (NGST) orbiting astronomical observatories. Due to the relatively underconstrained nature of this domain, generating a long range plan is not handled in the traditional AI planning sense of generating operators to achieve goals. Rather, producing an LRP is treated as a type of scheduling problem where what is being scheduled are not the scientific observations themselves, but "plan windows" for the scientific observations. This paper investigates planning subproblems which arise in this type of domain. In particular, we discuss the SIRTF Long Range Plan which requires planning of "instrument campaigns" in conjunction with observation plan window scheduling.

Introduction

Automated planning systems for planetary missions are faced with a problem involving months of relative inactivity followed by a few hours of intense activity that must be planned with great precision. In contrast, orbiting astronomical observatories have a much more constant rate of activity, but the bulk of the scientific activities are relatively underconstrained as to when they can schedule and how they should schedule with respect to each other. Scientific observations for the new observatories SIRTF and NGST will be even less constrained than for HST in that these observatories will not be placed in a low Earth orbit and will thus have less restricted observing windows. For these observatories, though, there will continue to be a premium on achieving high observatory efficiency while addressing planning and scheduling problems peculiar to each observatory.

In this paper we describe the Spike Plan Window Scheduler [Giuliano, 1997] which is used to generate a Long Range Plan for the Hubble Space Telescope. We have confronted a number of interesting challenges in modifying the Plan Window Scheduler for the SIRTF observatory. Addressing these challenges has shown that while long range planning can be treated as a scheduling problem, there are still planning issues to tackle during the process of scheduling.

The Domain

The advent of orbiting astronomical observatories has generated a need for software systems to automate planning and scheduling of activities on those observatories. For the purposes of this paper we mainly will consider activities that are of interest to a long range planner. These are the actual scientific observations, which can be thought of as the "pictures" that the telescope takes (in fact, these "pictures" could be spectrograph readings or images in non-visible light spectra).

For the most part, these observations are of targets that are fixed with respect to our solar system -- galaxies, stars, even visibly blank areas of the sky that may prove interesting. Now, if an astronomer wants to take a picture of the Eagle Nebula (M16), it doesn't matter too much whether she takes the picture today, tomorrow, or next week. The target's not going anywhere. If the observatory, like HST, is in a relatively close in orbit of the Earth, it is not accurate to say that she can take the picture of the Eagle Nebula at *any* time. Most instruments on the telescope are sensitive to too much light, so there are certain observing times when the Eagle Nebula is too close to the Sun, or the Moon, or the bright Earth itself to take an observation. In addition, there are many other valid scientific reasons why an observation can't be performed during certain times with certain instruments. For fixed targets, though, the conjunction of all these constraints on observing leaves one with a (possibly discontinuous) observation window which is still on the order of days or weeks in duration. We refer to these feasible observation windows as **constraint windows**.

Often, two or more observations have constraints placed upon them linking their relative execution times. For instance we might specify several observations of a Cephid variable star to be taken one week after each other. Even these types of constraints do not generally alter the duration very much of the *initial* constraint windows for the individual observations. We emphasize *initial*, since as observations are fixed to absolute times on a short term schedule, their linked successor observations can see their constraint windows collapse radically through constraint propagation.

This issue is treated in great depth in [Kramer and Giuliano, 1997] and we will investigate some consequences for long range planning later in this paper.

There is another significant, though numerically much smaller, class of observations for HST and other observatories. This is the class of "moving" targets. Generally, these are objects in our solar system like planets or moons and objects that enter the solar system on a periodic basis (e.g. comets). These observations may have a number of small, choppy constraint windows throughout the year, or a singular small window for even rarer events such as a view of Saturn's rings edge-on. In addition, a small percentage of viewing time for orbiting observatories is allocated to events that cannot be predicted in advance. These "targets of opportunity" include such phenomena as supernova.

Several newer observatories such as SIRTF and NGST are being designed to orbit farther away from the Earth. This will have the effect of increasing the duration of baseline constraint windows for observations from the days to weeks range experienced with HST to a range of weeks to months. And while it will certainly be the case that many *short term* activities for these observatories will be much easier to schedule, there will still be a number of the same issues -- moving targets, targets of opportunity, and contingent observations -- which complicates long range planning.

Although the bulk of observations for HST and other orbiting observatories are of the relatively underconstrained variety, there are a number of reasons why constraint windows, while of reasonable duration, do not remain completely stable over the course of a long range planning period (typically, a year to year-and-a-half). That is, the constraint windows will remain fairly stable in size, but may shift in time. There are a number of reasons for this:

- The ephemeris (position over time) of the observatory is not known accurately more than a few months in advance, causing constraint calculations to change over time.
- Performance of components on the satellite may change or degrade over time, requiring new operating scenarios.
- Observers refine their observing programs over time. [Giuliano, 1997]

Success Criteria for a Long Range Planner for Orbiting Astronomical Observatories

Efficiency. Orbiting astronomical observatories are an expensive investment with a limited life span. It is important that the long range planner be designed so as to maximize the amount and quality of scientific observations performed by adequately informing a short term scheduler when and what to schedule. Also, the long range planner should help optimize the use of finite resources such as propellants and coolants so as to prolong the life of the observatory.

Stability. It is usually the case that principal investigators (PIs) responsible for an observing program use a long range plan's "best promise" of when their observations will schedule to plan their future work: hiring of graduate students, data analysis, and planning of coordinated observations. It is important that a long range planner produce a plan that remains as stable as possible over an extended time span.

Mutability. As pointed out, there are a number of causes which might affect where an individual observation or set of observations might feasibly be planned. A long range planner should be able to revise the parts of the plan which need to change without adversely affecting the stability or goodness of the long range plan as a whole.

Long Range Planning for HST

Since launch in 1990 the Spike system [Johnston and Miller, 1994] has been used to generate a long range plan for the Hubble Space Telescope. For the first five years of operations the Spike Long Range Planner produced plans which were not successful in meeting any of three criteria listed above. The plans led to low efficiency schedules, were unstable, and were very resistant to incremental change. Why was this? An important reason was that the original planner planned observations to week long bins for input to a short term scheduler. By design these bins had to be well oversubscribed so that there would be an adequate mix for the scheduler. Unfortunately, it is hard to tell what a good mix will be without access to most of the knowledge of the short term scheduler. Months ahead of time this task is virtually impossible. As a result, many weeks' schedules based on input from the Spike LRP were well undersubscribed and some well oversubscribed. Observations that couldn't be scheduled in their target week would need to be replanned, and possibly put off an entire year if highly constrained.

To address these problems the long range planning system for HST was redesigned. Central to this redesign was a rethinking of the HST planning and scheduling operations concept. Based on this new operations concept and maintaining the very sound underpinnings of the Spike system -- the constraint propagator and astronomical objects model, a new long range planner was substituted for the old one.

The Plan Window Operations Concept

> The long range planner creates approximate 4-8 week **plan windows** for observations. A plan window is a subset of an observation's constraint windows, and represents a best effort commitment to schedule in the window. Plan windows for different observations can be overlapping. In addition the windows for a single observation can be non-contiguous. [Giuliano, 1997]

By design now the long range plan is slightly undersubscribed, but when the short term schedule creates a time line for a given week it has a large pool of observations from which to choose. A good and efficient scheduling mix can be generated. Many observations in the pool produced by the planner will *not* be chosen for a given week's schedule, but typically these observations will be able to be scheduled in a later week of their plan window. Observations which are in the last week of their plan window are given priority for scheduling on the next short term schedule.

The SPIKE Plan Window Scheduler

The new long range planner which implements the plan windows operations concept is the Spike Plan Window Scheduler. It can be described as

> ...a function which maps a set of input observing programs, search criteria, and a previous long range plan into a new long range plan. On execution of a long range plan the observations are partitioned into those which have [plan] windows assigned in the input LRP and those which do not have input windows. In general the scheduler will assign new windows to those observations which are not scheduled in the input plan and will pass through windows for observations which are scheduled in the input plan. The system assigns windows for observations which are not scheduled in the input plan in two steps. First, the system uses user defined criteria to greedily find the best window for each observation. In the second step a stochastic search is used to adjust the resource levels. [Giuliano, 1997]

Basically the Plan Window Scheduler is treating the planning process as a problem of constrained optimization where it tries to find the best subinterval of an observation's constraint windows to assign as a plan window. The objective function which is evaluated considers a number of criteria specific to an observation as well as criteria which will produce a good plan globally. Each individual active criterion will return a score as a real number between 0 and 1 inclusive. The criteria are weighted individually to express their relative importance to the overall scoring function. New planning criteria can be added to the planner as needed.

An example of a criterion that affects a small group of observations would be: "Attempt to schedule plan windows for observations in the same observing program as closely together as possible." Thus, if OB1 and OB2 are in the same observing program, but are planned a year apart, this criterion will return a score close to 0. If the plan windows for the two observations are identical, the score will be 1. An example of a planning criterion that is global in nature is one which balances resources across the extent of the planning period.

Plan Window Scheduler Architecture

The Spike Plan Window Scheduler is implemented as a CLOS program in Allegro Common Lisp, currently running on Sun/Solaris. There are separate modules for the Planning Criteria, the Resource Model, and the Scheduler. The Scheduler is comprised of one or more Scheduling Steps, which can be independent of each other, and thus added to, deleted, and specialized as necessary. For HST we initially implemented two scheduling steps: FIND-BEST and REPAIR. The FIND-BEST step finds the best plan window for each observation mostly independent of other observations. The REPAIR step is implemented as a specialization of FIND-BEST to level resource imbalances such as expected observation duration per day. REPAIR uses the same code as FIND-BEST, but adds two additional criteria to the scoring function: a randomizing criterion which adds a stochastic element to the search and a resource criterion.

The Plan Window Scheduler is run daily to generate a new Long Range Plan for HST. Each new plan is basically the previous plan plus additions for new observations and alterations for observations that have been changed or fixed on a short-term schedule. Small changes in an observation's constraint windows do not cause replanning of that observation. Its assigned plan windows are intersected with the new constraint windows to produce possibly somewhat smaller plan windows. In some cases a change in constraint windows necessitates no change in plan windows.

A recent modification illustrates the modularity of the Plan Window Scheduler design. As noted above plan windows are adjusted for constraint windows that are slightly smaller or that have shifted away from the plan windows somewhat. One scenario that wasn't being automatically handled was generating new plan windows for an observation whose constraint windows have "relaxed" around an existing tightly restricted plan window. In order to increase scheduling flexibility it was desirable to automatically expand existing plan windows as constraint windows grow. This problem was solved without modifying existing code, but by adding a new EXPAND-PWS step to the scheduler. It reuses code from FIND-BEST, but applies different planning criteria which maintain an existing plan window in place (it may shift in time by a small amount) while expanding it.

As well as being a good model for software maintainability and reuse, the Spike Plan Window Scheduler meets the three success criteria for a long range planner listed above. It generates a plan which leads to **efficient** schedules by assigning plan windows which are optimized for individual observations. The long range plan is **stable** by retaining existing plan windows or modifying them only slightly from plan to plan, and by only replanning those whose constraint windows have altered radically. The long range plan is **mutable** as new observations may be planned or existing ones replanned without affecting the bulk of the plan.

The SIRTF Mission

The SIRTF mission has much in common with that of HST. Like the Hubble, it will be a an orbiting "space telescope" that will mainly concentrate on fixed astronomical targets. It too will allocate a smaller percentage of time to viewing moving targets and targets of opportunity. As does HST, SIRTF operations will routinely give their observers an indication of approximately when their observations should execute while allowing them to modify their observation specifications until fairly shortly before they are actually scheduled to execute. For both missions it is assumed that all planned observations *will* execute (as opposed to ground-based observatories where many observations are simply dropped due to bad weather), so all observations are of basically equal priority. One of the primary goals of the SIRTF mission is science efficiency. The combination of these attributes points to SIRTF as a good mission to reuse the Spike Plan Window Scheduler in producing a stable yet flexible LRP.

The major difference between HST and SIRTF also makes it a good candidate for the Plan Window Scheduler. SIRTF will be placed in an "Earth trailing" (orbiting the Sun, lagging behind the Earth) orbit as opposed to Hubble's

"Low Earth" orbit. Thus, typical observing windows for SIRTF targets will be much less affected by Earth light and Earth occultations and will be much longer in duration. Some SIRTF targets will be observable year-round, with most others observable for months or weeks at a time. Of course, there will be additional constraints placed on many observations that will narrow the observing windows somewhat, but in general baseline windows for SIRTF will be significantly larger than for HST. This underconstrained problem argues for fairly large plan windows to guide a short term scheduler.

There is one major difference between HST and SIRTF, though, that seriously complicates the generation of a long range plan using the plan windows concept. HST has several scientific instruments, most of which can be run in parallel, so that for instance a wide-field image of a target can be executed at the same time as a spectrograph of the same target. This is not the case for SIRTF which will have three science instruments -- IRS, IRAC, MIPS -- which must be run sequentially. In addition, all of its instruments are cryocooled and it will cost a reasonable amount of time and some of the fixed supply of cryogen every time there is a change over from one instrument to another. These requirements argue that SIRTF observations be planned to "instrument campaigns" where only observations using a certain instrument be scheduled for days or even weeks at a time. In fact, this is a design requirement for the SIRTF mission. But on the face of it seems to be at odds with the plan windows concept which allows and encourages plan windows for observations to overlap, only optimizing the extent and number that overlap to resource limits.

We have been able to address this concern by introducing the concept of **instrument windows** to implement instrument campaigns. Instrument windows are defined to be discrete, but adjoining, subintervals of the nominal planning interval.The Spike Plan Window Scheduler has been extended to schedule plan windows and instrument windows in concert.

The Spike SIRTF Plan Window/Instrument Window Scheduler

A prototype implementation of the Spike SIRTF Plan Window/Instrument Window Scheduler has been completed and work is in progress to refine the prototype into an operational system before the planned SIRTF launch in December, 2001. A major design goal of this long range planning system is to reuse much of the existing Spike code and class hierarchy. To this date no existing Spike code has been rewritten. All enhancements necessary for SIRTF have been achieved by specializing Spike base classes and methods as well as writing some new code. Objects have been added to model SIRTF's resources and new planning criteria have been added to supplement some which have been reused from HST. New scheduling steps have been added to the Scheduling module, which we will describe in some detail shortly.

The main technical hurdle to be overcome in the design of the SIRTF long range planner was the issue of concurrent scheduling of plan windows and instrument windows. The former are by nature non-exclusive, while the latter must be exclusive. We can schedule multiple, overlapping plan windows "within" an instrument window as long as those plan windows are only for observations which require the same scientific instrument. In fact, we allow plan windows for an observation to schedule in multiple instrument windows as long as those instrument windows are designated for the same instrument.

For example, suppose we have an observation which will use the Infrared Array Camera (IRAC) and which has relatively unrestricted constraint windows. The nominal duration of plan windows for SIRTF will be 40 days to cover the natural viewing cycle for most observations (the majority will be observable for 40 days every six months). Even if there are two-week instrument campaigns for MIPS and IRS intervening between IRAC campaigns, a 40-day plan window for an IRAC observation could easily intersect two IRAC campaigns, while "skipping" the four-week interval between.

We considered two approaches to achieving this scheduling problem:

Approach I, plan windows first.

1. Generate plan windows for all observation first.
2. Assign instrument windows to areas where groupings of plan windows for like instrument windows occur.
3. Repair plan window assignments to segregate assignments by instrument window and to balance resources.

Approach II, instrument windows first.

1. Generate instrument windows to some idealized model based on projected resource usage and other criteria.
2. Assign plan windows to the best instrument window(s) based on scoring of planning criteria.
3. Repair plan window assignments for resource usage.

Given SIRTF observations' generally large constraint windows, we quickly rejected Approach I. There was nothing to drive plan window assignment, and thus just about anything could go anywhere. Segregating by instrument window then becomes a huge problem.

Approach II was chosen, with some refinements. Before discussing these, certain other attributes of SIRTF scheduling should be noted. First of all, it is anticipated that a significant fraction -- maybe 10% -- of observations may be classified as needing **absolute time** scheduling. We consider those observations that must schedule in no more than a one-day window to be an absolute time event. Secondly, there is a preferred ordering of instrument campaigns for SIRTF instruments in order to best utilize resources. The existence of absolute time observations helps enable our refined Approach IIa; instrument campaign ordering makes it more interesting.

Approach IIa, absolute time observations first.

1. Schedule plan windows for all absolute time observations. These plan windows will by definition be small, since the constraint windows are small. As each absolute time

observation is planned, create an instrument window "around it" such that this instrument window is somewhat larger than the plan window, but must not intersect unlike instrument windows. If an instrument window already exists where an observation needs to go, either it will be a window matching the observation's instrument or it will not. If it matches the plan window can be scheduled in that window. If it doesn't match, a conflict is noted to be handled later.

2. Generate instrument windows to fill in gaps between the instrument windows generated in (1) based on projected resource usage, desired instrument window size minima and maxima, and the preferred window ordering criterion.

3. For all non-absolute time observations, sorted by most highly constrained first, assign plan windows to the best instrument window(s) generated in (2) based on scoring of planning criteria. If a like instrument window does not exist where a highly constrained observation needs to be planned, a new instrument window may be generated by shrinking an existing one or an existing instrument window may be extended to accommodate the observation. In any case, instrument windows may not be modified so as to "orphan" an interval that is needed by an absolute time observation. Other, **relative time** observations may be replanned in a different instrument window if necessary. During this process instrument window size and ordering are maintained as much as possible.

4. Repair plan window and instrument window assignments for resource usage and other imbalances not handled in (3).

Approach IIa has been implemented by reusing much code from the HST Plan Window Scheduler and by defining four new Scheduling Steps, corresponding to (1-4) above: ABSOLUTE-TIME, INSTRUMENT-CAMPAIGN, RELATIVE-TIME, and REPAIR. Let's consider the INSTRUMENT-CAMPAIGN in some more detail.

Planning Within a Scheduling Problem

In the INSTRUMENT-CAMPAIGN step we are faced with the following initial conditions: A set of assigned instrument windows with gaps in between them. We would like to achieve a state where all the gaps are filled with new (empty, in the sense that no plan windows are assigned to them yet) instrument windows that are of some optimal size and ordering. Basically we have a problem that is amenable to classical AI planning approaches. The domain is highly constrained and there are clear goals that can be achieved by applying a sequence of operators to a defined world state.

There are a number of approaches that could be considered for implementing the Instrument Campaign Planner, however, most of them are overkill. Consider the gap-filling problem. Do we generate N instrument windows of length M to fill the gap, or X windows of length Y, and so on? For large gaps there are many possible assignments that satisfy the instrument window ordering problem. On the other hand, we are in general trying to generate as large instrument windows as possible within some bounds. This cuts the search space a good deal.

A more important consideration is that at this point in our overall planning and scheduling problem (instrument windows + plan windows for all observations) it isn't important to try to get an optimal but computationally expensive solution, as this solution will likely be undone in the RELATIVE-TIME and REPAIR steps. We need a planner that gives us a quick solution which is *good enough*.

The solution we provide is somewhat similar to that employed in *case-based* planners [Weld, 1994] in that we use preexisting plans to synthesize a new plan. We differ from case-based planning in that our domain is very well defined and constrained so we don't have the problem of matching and possibly modifying plans to use. Instead, we take a brute-force approach of selecting all plans that apply to the preconditions, scoring each one, and choosing the best.

We call these "pre-packaged" plans **window optimization scripts**. Window optimization scripts come in three types: standalone, auxiliary, and compound. A standalone script is one that is atomic. It performs one action that can be scored as a solution to forwarding plan goals. An auxiliary script is atomic as well, but is always combined with other scripts to produce a script which can be scored. A compound script refers to a list of sub-scripts -- standalone, auxiliary, and compound -- which it calls to do its work.

An example of a gap-filling windows optimization script is CREATE-WINDOWS-TO-RIGHT, a compound script whose sub-scripts are CREATE-WINDOW-TO-RIGHT, MAKE-NEXT-WINDOW-CURRENT, CREATE-WINDOWS-TO-RIGHT. During the INSTRUMENT-CAMPAIGN scheduling step, CREATE-WINDOWS-TO-RIGHT is one of a number of scripts which is tested and scored as a plan to fill the gaps between existing instrument windows. At all times one instrument window is designated as the *current* window. CREATE-WINDOWS-TO-RIGHT calls CREATE-WINDOW-TO-RIGHT to generate a new window (to the right of the current). If this action is successful, MAKE-NEXT-WINDOW-CURRENT is called to advance the current window pointer to the newly created window. Next, CREATE-WINDOWS-TO-RIGHT calls itself recursively to continue the process of generating new windows to fill the gap. This plan is scored and if better than the current best is saved as best plan. Its effects are undone and the next applicable plan on the list is tested.

Another example of a windows optimization script is EXTEND-BOTH-TO-MAX-LEFT-FIRST. It fills in a gap by calling EXTEND-WINDOW-TO-MAX-ON-RIGHT, MAKE-NEXT-WINDOW-CURRENT, and EXTEND-WINDOW-TO-MAX-ON-LEFT. This and several other scripts are sufficient to do a good job of filling gaps between instrument windows, as our choices of operators is relatively small. As testing has revealed a few cases that are not solved by existing scripts, several new ones have been added.

Issues and Future Work

While the Plan Window Scheduler approach has proved to be quite successful in generating long range plans for several observatories, some issues keep it from being a fully automated solution. Consider the issue of "Plan Window Conflicts" which is being addressed for the HST LRP.

Inherent in the Plan Window Concept is the notion that plan windows for a number of different observations will overlap a given time period. Typically, plan windows for any given observation will be fairly large (more than one week in duration), and the overlaps tend to be somewhat large as well. It is usually the case, then, that a feasible ordering of observations can be found once a short term schedule is constructed. Many observations can be placed on one week's short term schedule, and those that can't can be placed on a succeeding week's.

When some observations are scheduled, though, occasionally conflicts occur. That is, two or more observations claim the same time slot on a short term schedule and the conflict is irresolvable without manual work including modification of the observation specifications. How do these conflicts occur?

1. Sometimes the conflicts are due to observations which have very tight constraint windows (a few hours) which intersect. While this type of conflict derives from the nature of the observations themselves, it is also the easiest to detect beforehand. After detection, the conflict can be reported so that one or more observations can be altered.

2. Some conflicts occur when observations have plan windows which appear to have a good deal of scheduling flexibility built in, but which in practice do not. The cause of this is that the short term scheduler has knowledge of some constraints which are either not known by the long range planner, or which are only computed by the LRP in a statistical sense. A possible solution to this problem is to provide better communication between the short term scheduler and the long range planner. In theory the LRP could detect those observations that are "prone" to this problem and call the short term scheduler to test schedule them for possible conflicts.

3. The third type of conflict is even more difficult to detect. It occurs when a number of observations have long plan windows, but all of them terminate a short time into a given week. Now suppose each of these observations is not scheduled during the first few weeks of their plan windows. This is a very common occurrence as there will be higher priority observations to schedule whose plan windows do end during a given week and thus *must* be scheduled. If enough of the observations with long plan windows are "ignored" on earlier weeks, they may all end up being "must go" for the present week. But, if they all need to schedule in the first part of the week, there may not be enough time to schedule all of them. One possible solution to this problem is to detect groups of observations whose constraint windows all end at the same time and stagger plan window end times.

The Plan Window Conflict problem is one that affects a small percentage of observations, but it presents some interesting problems which do not seem insoluble within the plan window scheduling paradigm. Given the nature of the observatory domain these problems might be avoided with a solution of just short term scheduling on demand, but the costs of this approach would far outweigh the benefits.

Open questions also remain as to how easy it will be to maintain a stable Instrument Campaign schedule for SIRTF in a operational environment with observations changing over time, new observations being planned, and unforeseen events occurring. More work remains to be done on the REPAIR phase of the planner. We will get a better idea of how the Plan Window/Instrument Window Scheduler will perform as we conduct more realistic simulations, however some modifications will be inevitable during the life of the mission. In any case, the Plan Window concept has been shown to provide a good deal of flexibility -- both for scheduling observations, and for handling different planning challenges.

Summary

We have looked at a domain -- long range planning for orbiting observatories -- where planning may be best handled by methods other than the classical goals/operators paradigm of AI. This is still a planning problem in a more generic sense of creating a template so that activities can later be sequenced efficiently and resources managed effectively over time.

The plan window scheduling methodology we describe will continue to have applicability for observatory long range planning, and should be considered for other domains where similar attributes are found: most activities are loosely constrained and coupled, need to be roughly planned well in advance, but may not be completely well defined until they are ready to schedule.

Acknowledgments. I would like to thank Mark Giuliano as the architect of the Spike Plan Window Scheduler, who designed a system that is truly reusable and extensible.

References

Giuliano, Mark 1997. *Achieving Stable Observing Schedules in an Unstable World*. ADASS '97, Sonthofen, Germany.

Johnston M.; Miller G. 1994. *Spike: Intelligent Scheduling of Hubble Space Telescope Observations*. in Zweben M., and Fox M. eds. *Intelligent Scheduling* (San Francisco: Morgan-Kaufmann), ISBN 1-55860-260-7 (1994), pp 391-422.

Kramer L.; Giuliano M. 1997. *Reasoning About and Scheduling Linked HST Observations with Spike*. In the Workshop Notes of the International Workshop on Planning and Scheduling for Space Exploration and Science. Oxnard, CA.

Weld, Daniel 1994. An Introduction to Least Commitment Planning. *AI Magazine* 15(4):27-61.

A Commentary On: Generating a Long Range Plan for a New Class of Astronomical Observatories

David E. Smith

NASA Ames Research Center
Mail stop 269–2
Moffett Field, CA 94035
de2smith@ptolemy.arc.nasa.gov

Introduction

The paper *Generating a Long Range Plan for a New Class of Astronomical Observatories* by Laurence Kramer, describes how scheduling for the Hubble Space Telescope (HST) is broken up into two phases:

1. *Long Range Planning* (LRP) – where the observations for an extended period (a year or more) are each assigned to plan windows of typically 4 to 8 weeks duration.
2. *Short Term Scheduling* – where a detailed schedule of observations is constructed for a single impending week.

The paper goes on to describe a proposal for adapting this same architecture to the Space Infrared Telescope Facility (SIRTF). For SIRTF, observations are typically much less constrained than for HST. However, resource management is much more critical, because there is a finite supply of cryogen for cooling the instruments, and switching instruments uses both coolant and time. As a result, the author proposes to construct explicit instrument windows during the LRP process and to use those windows to help assign plan windows for observations.

To start, I want to comment briefly on the general nature of observation scheduling problems. Second, I wish to comment on the issue of maintaining both flexibility and stability in a schedule, and offer a different way of thinking about stability. Finally, I wish to comment on the proposed use of instrument windows for SIRTF and suggest an alternative.

Observation Scheduling

There is a great deal of similarity between the scheduling problem for HST and SIRTF, and observation scheduling problems for other facilities such as automated ground-based observatories [5], airborne observatories (KAO and SOFIA) [6], and earth observing satellites (EOS). For all of these observatories, investigators submit specific observing proposals, usually well in advance. Proposals are then reviewed and ranked in some fashion. The result is a large collection of requested observations with constraints on date, time, sky condition, and other operational parameters. The challenge is to maximize scientific return, subject to the constraints. The problem is that for all of these facilities there are significant sources of operational uncertainty – weather conditions for ground-based observatories and EOS, water vapor conditions and air traffic delays for KAO and SOFIA, and uncertainty due to slewing, target acquisition, and observation duration.

Are these planning problems or scheduling problems? Both and neither. On the one hand, there are usually more observations than can be performed in a given period, so the task involves *choosing* some subset of the available observations. Choosing the operations to perform sounds like planning, but the choices are simple ones; the choice of one operation does not lead to other choices (as with preconditions). In addition, these problems involve rich temporal constraints and reasoning about continuous resources. While planning problems can include these characteristics, the solution techniques are not robust or mature (see [11] for discussion of this issue).

In contrast, scheduling problems often involve temporal constraints between tasks, continuous resources, and optimization. As a result, the constraint-based representation commonly used for scheduling problems is particularly relevant. However, there is little relationship between the common job shop scheduling problem and observation scheduling problems. In particular, there is no parallelism in observation scheduling – only one target can be observed at a time. The optimization criteria is also much different.

In sum, observation scheduling is a rather different animal, and neither the scheduling or the planning communities have a well developed set of tools for addressing this class of problems.

Flexibility and Stability

In the paper, Kramer argues that successful scheduling systems for HST and SIRTF need to generate schedules that are both flexible and stable. Flexibility is needed because of uncertainty – observation requests change, orbits are uncertain, and instrument behavior and performance are not entirely predictable. Stability is needed because astronomers have expectations about when their observations will be performed and they must make commitments based on those

expectations. As a result, changes to the observing schedule must be made with care.[1]

The properties of flexibility and stability are fundamentally at odds with one another. Flexibility is obtained by delaying commitment – that is, scheduling decisions are not locked in until the last moment. In contrast, stability is guaranteed by early commitment – locking down decisions about the date and time of each observation. In Spike, this trade-off is accomplished by:

1. Early partial commitment to a 4-8 week range of dates for each observation
2. Delaying commitment on the exact dates and times for individual observations until shortly before execution

From a theoretical point of view, the early partial commitment done in LRP is somewhat different than operations usually performed in constraint-based scheduling. To see this, consider the observation scheduling problem as a constrained optimization problem in which there are variables representing the possible start times for each observation. (For each observation, the value of this variable will be a set of time intervals corresponding to the intervals in which the observation is possible.) By constraint propagation techniques such as arc-consistency and edge finding [4, 9], we can narrow down these intervals by virtue of other constraints on the observation, constraints on telescope operation, and time constraints between observations (see [7] for details). But for HST and SIRTF, this can still leave a wide range of possible start times for each observation. The purpose of the LRP process for HST and SIRTF is to narrow down these ranges so that 1) the remaining short-term scheduling problem is somewhat more constrained and hence easier to solve, and 2) astronomers can be given some idea when their observations are likely to be performed. When viewed as constraint optimization, the LRP is making commitments to subsets of the possible values for observation start time variables, instead of making commitments to individual values for those variables.

Schedule stability is enforced, because on subsequent rescheduling the LRP will not rescind these interval choices (except for observations that have been modified). In effect, the LRP refuses to backtrack on window narrowing choices that were made at an earlier date. This approach certainly guarantees the desired schedule stability, but could be problematic in other more highly constrained scheduling domains. The trouble is that there is no guarantee that a solution will exist to the resulting (more constrained) scheduling problem. In fact, scheduling systems typically need to search through many alternative choices for variable values in order to find acceptable schedules. The LRP usually gets away with this kind of irrevocable commitment, because the HST and SIRTF problems are generally very underconstrained.

For a domain in which this kind of commitment will not always work, how can we preserve a reasonable level of schedule stability? I contend that stability is an optimization criterion in rescheduling – there is some penalty associated with delaying an observation beyond its original predicted window, and the extent of the delay is presumably correlated to the magnitude of the penalty. (There is no penalty for performing an observation early, but doing so might displace another observation, which could result in a penalty.) Using this optimization criterion, a scheduler would prefer schedules in which all observations take place prior to the end of their predicted window, but alternatives could be considered and evaluated. Note that treating schedule stability as an optimization criterion, and allowing window revision does not require that we reschedule from scratch each time. It is still possible, and rational, to reschedule using local search methods, treating the previous schedule as the seed for the local search.

Of course, the LRP could use this approach when rescheduling. It would allow greater flexibility in rescheduling, and could allow more principled recovery from larger scale failures such as equipment problems aboard the telescope or unusual events that preempt existing observations.

Instrument Windows

For SIRTF, observations are typically much less constrained than for HST. This means bigger initial time windows. However, resource management is much more critical, because there is a finite supply of cryogen for cooling the instruments, and switching instruments uses both coolant and time. In the paper, the authors propose to 1) assign time windows for tasks with very small constraint windows, 2) build explicit instrument windows around those time windows, and 3) assign time windows for the remaining observations, splitting or adjusting instrument windows if necessary to accommodate observations.

I am not convinced that the construction of instrument windows is the best approach to this problem, because the instrument windows must be chosen, then continually revised to accommodate additional observations. In the final schedule, instrument windows are merely consequences of the choices of observation windows. In other words, it is the observation constraints that determine the best set of instrument windows, not the other way around. The paper argues that the observations are so under constrained that instrument windows are needed to help constrain the assignment of observation windows. If this were so, there would be no need to revise the initial set of observation windows to accommodate the remaining tasks.

Switching instruments on SIRTF is a classic example of a *setup step* common to many job shop scheduling problems. For example, consider a milling machine on a factory floor. Changing the bit or the clamping arrangement on the table is a time consuming and hence costly operation. If the previous task uses the same bit and the same clamping arrangement, no cost is incurred. Thus, there is considerable incentive to

1. The need for both flexibility and stability is not unique to observation scheduling; flexibility and stability are required in any domain where there is significant uncertainty, but commitments must be made early. This is more common than not in real world scheduling problems.

schedule tasks in such a way as to minimize the number and cost of such setup steps. In general, simple search techniques like A*, Limited Discrepancy Search [8], and HBSS [3] should work quite well. To see how this might work for SIRTF, consider the partial schedule shown in Figure 1.

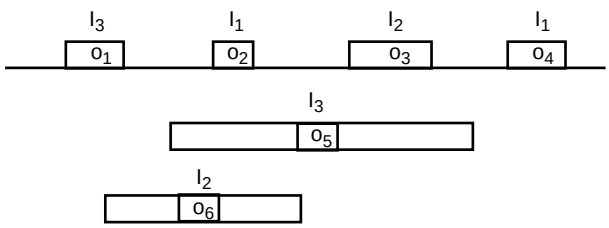


Figure 1: The search for Instrument windows.

Initially there are several observations in the schedule that can only be performed at specific times. For each observation, the required instrument is indicated above the observation. Initially, we have a lower bound on the instrument change cost for this schedule. Somewhere between O_1 and O_2, we will at least need to change from instrument I_3 to I_1. Similar transitions must occur between the other observations. If we let C_{ij} be the cost of changing from instrument i to j, the cost for this partial schedule is $C_{31}+C_{12}+C_{21}$.

Now consider an observation O_5 that has a much larger range of possible start times. Given the existing schedule, there are three places we could put this observation: before O_2, between O_2 and O_3, and after O_3. If O_5 occurs before O_2, there is no additional instrument change cost; only the transition from I_3 to I_1 is required. If O_5 occurs between O_2 and O_3, the cost increases because we must now transition from instrument I_1 to I_3, then back to instrument I_2. The results are similar if O_5 occurs after O_3. For subsequent observations a bit more reasoning is required, because we need to consider the possible ordering relative to flexible observations like O_5. However, the general idea is to search the space of relative observation orderings, in order to minimize instrument change cost. The resulting best ordering will dictate the best set of instrument windows.

In doing this search, one could also make use of probabilistic instrument usage profiles similar to those used by Sadeh [10] and Fox [1, 2]. In Figure 2, I have illustrated a simple profile for instrument I_2 from the example in Figure 1.

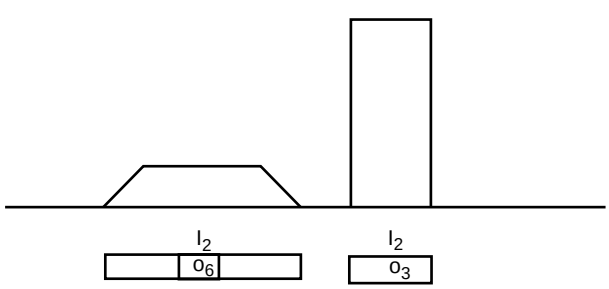


Figure 2: Probabilistic usage profile for instrument I_2.

For O_3, there is no time flexibility so I_2 is used with probability 1 throughout the interval. For O_6 there is flexibility, so the probability mass is distributed in a trapezoid. A composite usage profile for an instrument is constructed by summing up the profiles for all observations using that instrument. These profiles provide heuristic guidance on the best windows to assign for a task – namely those for which there is high demand for the instrument and lower demand for other instruments.

In summary, I believe that for SIRTF, a more systematic search of the space of possible observation windows is practical, and that these observation windows will dictate the resulting instrument windows.

References

1. Beck, J., Davenport, A., Sitarski, E., and Fox, M. 1997. Texture-based heuristics for scheduling revisited. In *Proc. 14th Nat. Conf. on AI*. 241–248.

2. Beck, J. C. and Fox, M. 1998. A generic framework for constraint-directed search and scheduling. In *AI Magazine* 19(4).

3. Bresina, J. 1996. Heuristic-Biased Stochastic Search. In *Proc. 13th National Conf. on AI (AAAI-96).*

4. Carlier, J. and Pinson, E. 1989. An Algorithm for Solving the Job Shop Scheduling Problem. *Management Science, 35(2)*. 164-176.

5. Drummond, M., Bresina, J., Edgington, W., Swanson, K., Henry, G., and Drascher, E. 1995. Flexible Scheduling of Automatic Telescopes over the Internet. In *Robotic Telescopes: Current Capabilities, Present Developments, and Future Prospects for Automated Astronomy.* Edited by G. Henry and J. Eaton. ASP Conf. Series, Vol. 79. Astronomical Society of the Pacific.

6. Frank, J. 2000. SOFIA's choice. In *International Workshop on Planning and Scheduling for Space Exploration and Science* (this volume), San Francisco, CA.

7. Kramer, L. and Giuliano, M. 1997. Reasoning About and Scheduling Linked HST Observations with Spike. In *International Workshop on Planning and Scheduling for Space Exploration and Science*, Oxnard, CA.

8. Harvey, W. and Ginsberg, M. 1993. Limited Discrepancy Search. In *Proc. 14th Int. Joint Conf. on AI (IJCAI-93).*

9. Nuijten, W. 1994. *Time and Resource Constrained Scheduling.* Ph.D. dissertation, U. Eindhoven.

10. Sadeh, N. 1994. Micro-opportunistic scheduling: the micro-boss factory scheduler. Technical report CMU-RI-TR-94-04, Carnegie Mellon Robotics Institute.

11. Smith, D., Frank, J., and Jónsson, A. 2000. Bridging the Gap Between Planning and Scheduling. To appear in *Knowledge Engineering Review* 15:1.

First published by the authors under the same title in AIPS 2000, p. 236-245
Copyright (c) 2000, American Association for Artificial Intelligence. Reprinted with permission from AAAI

Using Generic Preferences to Incrementally Improve Plan Quality

Gregg Rabideau, Barbara Engelhardt, Steve Chien

Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive M/S 126-347, Pasadena, CA 91109
{firstname.lastname}@jpl.nasa.gov

Abstract

We describe a methodology for representing and optimizing user preferences on plans. Our approach differs from previous work on plan optimization in that we employ a generalization of commonly occurring plan quality metrics, providing an expressive preference language. We introduce a domain independent algorithm for incrementally improving the quality of feasible plans with respect to preferences described in this language. Finally, we show that plan quality can be significantly increased with very little modeling effort for the domain.

Introduction

For many planning problems, quality as well as feasibility must be considered while generating a plan. There may be many undesirable, yet feasible solutions that satisfy the goals. In addition, strict feasibility constraints may be too weak for most problems, but necessary for completeness. For example, while it may be physically possible to completely drain a battery, reasons of risk and longevity will make it preferable to maintain a certain level of charge. However, this preferred charge level, if encoded as a hard constraint, would preclude solutions where a full battery drain was necessary. We build on the traditional representation of discrete *hard constraints* and mandatory goals to include continuous *soft constraints* (i.e., preferences) and optional goals. In other words, we extend the notion of what *must* be accomplished (and how) to what *should* be accomplished (and how). In this way, the user can specify which feasible solutions are more desirable, establishing a basis for automatically generating high quality plans.

In many NASA domains, the user can have a complicated definition of plan quality. For example, scientists typically would like to complete as many experiments as possible within given windows of opportunity. Other users, such as engineers, might have a preference for fewer power switches of a spacecraft instrument in order to extend the life of the instrument. Certain system states may be more desirable than other states. For example, extending an arm of a rover might leave the rover unstable, making it preferable to keep the arm stowed when not in use. Some timing constraints may be flexible but also have a preferred time. For example, a calibration may be most useful immediately before an experiment, but still have some utility up to five minutes earlier. We present a general representation of plan quality that is capable of encoding a wide range of preferences including the ones just described.

We implement a representation of plan quality and an optimization algorithm in the ASPEN planning and scheduling system [1, 2]. During *iterative optimization*, low scoring preferences are detected and addressed individually until the maximum score is attained, or a user-defined time limit has been exceeded. A preference is a quality metric for a plan variable, and can be improved by making certain modifications to the plan. The most common plan modifications include moving an activity, adding a new instance of an activity, and deleting an activity. For each preference, a domain-independent improvement expert automatically generates modifications that could potentially improve the preference score. This is a generalization of the iterative repair technique [3, 4] used to resolve violations of hard constraints.

The iterative optimization algorithm has many of the same desirable properties as iterative repair. Both algorithms can be invoked at any time on any plan, making them more amenable to mixed-initiative planning. Repairing or improving existing plans enables fast turn-around times when small changes create conflicts or degrade plan quality. Changes may occur from manual modifications or from automatically detecting unexpected differences during execution. Further discussions with applications to spacecraft commanding can be found in [5].

The ASPEN Planning Model

In ASPEN, we have adopted a planning model with an explicit representation of constraints for time, resources and states [6]. Plan operators, called activities, have a set of local variables including a start time and duration. Activities may have a set of temporal constraint variables, each specifying a minimum and maximum separation between two

activities in the plan. Activities also share a number of global resources or state variables. Local constraint variables may be defined in an activity specifying the required value of a resource or state variable for the activity. The combined effects of the activities define the time-varying profiles (i.e., timelines) for the values of the resources and state variables. Global constraints can be defined for each timeline, restricting its set of legal values. For resources, these are capacity constraints. For state variables, the set of legal states and transitions can be specified. The ASPEN planning model also includes a representation for activity hierarchies. Activities can have a disjunctive set of decompositions, each of which expands the activity into different set of sub-activities. A local variable represents the currently selected decomposition. Arbitrary functional relationships can be expressed between any of the variables in the activities. This allows ASPEN to make external calls to special reasoning modules for calculating plan values, if necessary.

Finally, ASPEN has an explicit representation of mandatory and optional goals. Goals are simply activity specifications that do not immediately appear in the plan. A mandatory goal is a conflict until the activity has been inserted into the plan (i.e., the goal is satisfied). Optional goals are not considered conflicts when not satisfied but instead degrade plan quality.

Representing Plan Quality

We define preferences as quality metrics for variables in complete plans. Preferences provide a mechanism for specifying which plan variables are important to plan quality. Certain values of these variables are preferred over others, without regard for legality. We define a set of preference classes that directly corresponds to the set of plan variable classes.

Preference Variables

To better understand what types of preferences are included in our semantics, we must describe the types of plan variables that can contribute to plan quality. There are five basic types: local activity variable, activity/goal count, resource/state variable, resource/state change count, state duration.

An activity variable preference indicates a ranking for the values of a local variable in an activity instance in the plan. Local activity variables include domain-specific variables as well as internal variables for start time, end time, duration, resource usage, temporal distance from other activities, and selected decomposition. Typically, a preference is made for variables with a particular name defined in a particular type of activity. For example, minimizing tardiness in [7, 8] is a preference on the end times of activities that fulfill factory orders. Minimizing work in process (WIP) is a preference on the distance between the order request and order fulfillment activities. Other preferences can score the plan based on the number of existing activities of specific types (i.e., activity count). Or, one can make a general preference for satisfying more of the optional goals. A preference can also be made for certain values of a global resource or state variable. A resource/state variable preference ranks the set of resource/state values that exist within the planning horizon. For example, a preference can be made for maximizing the minimum value of a battery over time. Other preferences can score the plan based on the number changes occurring on a resource/state variable (i.e., resource/state change count). This type of preference could be used to limit the number of power spikes on the battery. Finally, a preference can be made on the duration of a particular state on a state variable. Pointing a spacecraft antenna towards earth, for example, is preferred when the spacecraft is not constrained to any other state.

Mapping to Quality Metrics

A preference is a mapping from a plan variable to a

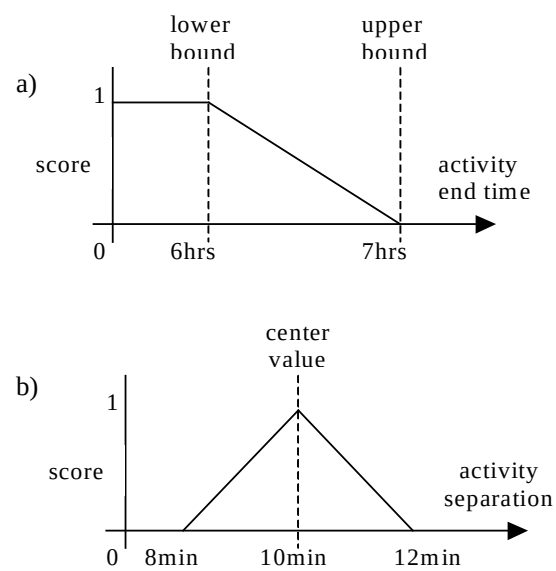


Figure 1: a) Mapping the end time of an activity to a score. This implements a preference for minimizing tardiness of an activity. The deadline is at the sixth hour and the score decreases to zero one hour after the deadline. b) Mapping the distance between two activities to a score centered on a given value. This implements a preference for maintaining a 10 minute separation with a ±2 minute tolerance.

quality metric (i.e., score) in the interval [0,1] (see Figure 1). Specifically, a preference indicates whether the score is monotonically increasing or decreasing with respect to the plan variable within certain bounds. The user can also specify that the score increases as the difference with a given fixed value decreases. In other words, the high score is centered on a value selected from the domain of the variable. From this high-level specification, mapping functions are generated that take preference variables as arguments and return real-valued scores.

Each preference includes an upper and lower bound to indicate the range of the variable for which the score increases or decreases. Any values outside this range produce a score of either zero or one. For example, anything over 90% battery charge may be indistinguishable in terms of quality. Therefore, a preference can be defined as increasing with minimum charge and reaching a maximum score at 90% charge. Each preference also includes a weight for specifying the relative importance of the preference to overall plan quality. The score of a plan is computed as the weighted average of scores for plan variables with preferences.

An aggregate preference is defined for many plan variables, and can either score each variable independently, or score the result of applying a function to the variables. If the preference scores each variable, then the scores are weighted equally and averaged. The built-in functions that can be used in aggregate preferences include average, sum, minimum, and maximum. These functions constitute the set of functions most commonly observed in preferences from various domains. For example, minimizing makespan is a preference on the maximum end time of all activities in the schedule. The specified function is computed for the current set of plan variables, and the result is mapped to a score for the preference.

Improving Plan Quality

Preferences allow us to define quality metrics for evaluating feasible plans and making quantitative distinctions between different plans. The next step is to use these preferences to generate high quality plans.

Local Improvement Experts

In addition to establishing quality metrics, preferences can provide insight into how to improve plan quality. We define a domain-independent *improvement expert* for each class of preference to aid in optimization (see Figure 2). The expert uses the preference specification to find plan modifications that will improve the score for the given preference and current plan. In other words, an expert is a link between changes in the plan and the change in quality. For example, if less resource usage were preferred, expert improvements would include deleting an activity that is currently using the resource. It is a local expert, however, and does not guarantee an increase in overall plan quality. Improvement experts provide a framework for optimization algorithms, defining the search space of possible improvements. We define a separate class of improvement expert for each class of preference.

Local activity variable expert. One class of expert is used for improving preferences on local activity variables. The most obvious modification for improving this preference is to change the value of the local variable. The expert considers variables that contribute to the low score. For example, only the end time of activity a2 in Figure 2 can be changed to improve the score for this preference. If score is a decreasing function of the variable, then making an improvement requires assigning a value less than its current value. Similar improvements exist for increasing functions. In cases where the variable is the start or end time of the activity, assigning a value implies moving the activity to earlier or later times. An expert might create activities with high scoring values or delete activities with low scoring values on a variable.

Activity/goal count expert. A different class of improvement expert is used preferences on the number

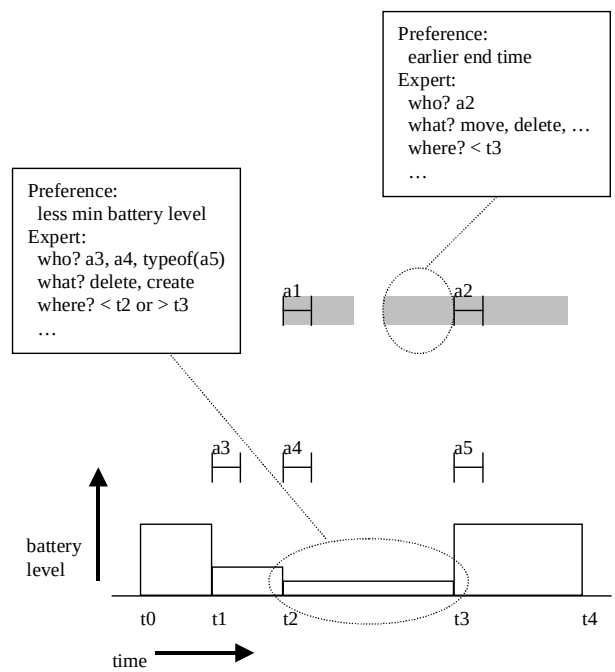


Figure 2: Local improvement experts.

of activities/goals. When the preference is for more occurrences of a goal/activity, creating new activities is the only beneficial modification. When the preference is for fewer occurrences, deleting existing activities is the only improvement.

Resource/state variable expert. Another class of expert improves preference scores for the values of resources or state variables. Only activities that use the resource or state variable are considered. For a high resource preference, the expert selects activities that increase the resource when adding and activities that decrease the resource when deleting. When moving, if the preference is for a higher minimum resource value, activities that decrease the resource during this time can be moved away from the minimum value. In Figure 2, activities a3 and a4 both contribute to the low minimum battery level. Similar cases exist for lower maximum, higher maximum, and lower minimum resource values.

Resource/state change count expert. A simpler class of expert is used for improving scores of preferences on the number of times a resource or state variable changes over time. Adding activities that use the resource or state variable will increase the number of changes. Deleting will decrease the number of changes.

State duration expert. The last class of improvement expert works on state duration preferences. Activities that change the state variable can be created, deleted, or moved in order to change the amount of time spent in a particular state. When the preference is for a longer duration, activities that change to the specified state can be created at times when the variable is in a different state. Conversely, when the preference is for a shorter duration, activities that change to any other state can be created at times when the variable is in the specified state. Similar reasoning is used when deleting or moving activities.

Monotonic Preference Assumption

In order to make improvement calculations tractable, we make a *monotonic preference assumption*, requiring each mapping from plan variable to quality metric to either be consistently increasing or decreasing within a given range of the variable. For preferences centered on a value, the score must increase for values less than the specified center value, and decrease for values greater than the center value. This assumption allows the problem to be restated as simply identifying modifications that increase or decrease the current values of plan variables participating in preferences. For example, if a variable with integer domain [1,10] and current value 4 has a decreasing preference, then only values in the range [1,3] will increase the score for this preference.

Iterative Optimization

The full set of potential plan improvements can be quite large. Once the automated expert has identified this set, we search for more optimal plans by iteratively selecting and making improvements (see Figure 3). We call this technique *iterative optimization* because of its similarity to iterative repair. The iterative optimization algorithm first selects a preference from the list of sub-optimal (i.e., score < 1) preferences. Typical heuristics for this decision include selecting a preference with one of the lowest scores or one with the most potential gain (weight * (1 – score)).

The algorithm must then decide which type of modification to perform for the selected preference, based on the local expert. After making a local improvement, the resulting plan may not be optimal or even improved. The iterative optimization algorithm continues by selecting another preference, and repeating the improvement process. After each improvement, if the current score exceeds the best score and the plan is feasible, the current plan is saved. The algorithm halts when the maximum score is attained, or a specified time limit is reached. If an optimal plan was not found, the saved plan with the best score is returned.

When making modifications during iterative optimization, adhering to plan constraints may be too restrictive, precluding modifications necessary for improving quality. Iterative optimization algorithm may create conflicts while searching for an optimal plan. Because it is unknown how the plan will change

```
Iterative Optimize (T)
    Let P = P_best  = current plan
    Let S = S_best  = current score
    While (S <1 and time < T)
        If conflicts exist, Then repair(T-time)
        Let Q = set of preferences with sub-optimal score
        q = choose(Q)
        M = E_q(P) // get the set of modifications
        m = choose(M)
        P = m(P) // apply the chosen modification
        S = score(P)
        If (S > S_best) // save if best-so-far
            S_best = S
            P_best = P
    Return P_best
```

Figure 3: The ASPEN optimization algorithm. $E_q(P)$ returns the set of modifications for plan P calculated by the expert E for improving preference q.

to achieve feasibility, we do not define quality for infeasible plans. Iterative repair algorithm is invoked to restore feasibility when optimization yields an infeasible schedule.

The iterative optimization algorithm does not perform strict hill-climbing. Decreasing score suggests that a subset of the preferences represent competing objectives. We only attempt to increase the score of a single preference by stochastically choosing one potential improvement rather than optimizing the preference. We would expect competing preferences with a large disparity to reach a compromise rather than thrash between a high score for one and a low score for the other.

Continuous Improvements

During execution we may notice differences between actual and expected values for activities or resources. These differences may violate hard constraints or degrade plan quality. The CASPER system [9] was developed to continuously initiate and monitor the execution of an ASPEN plan, updating the plan when necessary. As the result of a plan update, CASPER uses the iterative algorithms to fix new conflicts and improve preference scores. In this way, CASPER provides continuous planning and optimization during the course of execution.

Case Study

New Millennium Earth Observer 1 (EO-1) is an earth imaging satellite featuring an advanced multi-spectral imaging device. EO-1 mission operations consists of managing spacecraft operability constraints (power, thermal, pointing, buffers, consumables, telecomm, etc.) and science goals (imaging surface targets within specific observation constraints). One interesting constraint involves the Solar Array Drive (SAD) which keeps the solar arrays facing the sun. For a few minutes before and during each data-take, the SAD must be locked to avoid spacecraft jitter, which can corrupt data. The EO-1 model consists of 14 resources, 10 state variables and total of 38 different activity types.

The EO-1 model includes preferences for (see Figure 4): more science goals, more time with the SAD tracking the sun, fewer changes of SAD state, and less deviation from the preferred separation of data-take and SAD locking activities. The last preference has a high score centered on a value because if the settling time is too small there will be too much jitter, but if the separation is too large the solar array power output will suffer.

Optimization begins with no optional goals satisfied and no violated constraints. In order to increase plan quality, a goal might be added to the plan. Because this activity has many requirements, constraint violations are immediately identified and addressed. After repairing all conflicts, optimization continues to make improvements by either adding another goal or working on another preference. Optimization and repair continue until a time limit is reached, after which the best saved plan is reloaded. Random EO-1 problems were run on a Sun Sparc Ultra 60 for five minutes each. Resulting values for each preference variable were averaged over the 100 problems. Approximate “optimal” values were estimated manually considering each preference individually. Average values for all preference variables were within 50% of the “optimal” value or better. This is good considering many of the preferences represent competing objectives.

Studies on other models were performed with similar results, including the New Millennium ST-4 spacecraft, the Data-Chaser shuttle payload, and the JPL Rocky-7 planetary rover [10].

Related Work

Much of the recent work in plan optimization has been looking at ways to integrate linear programming (LP) techniques with symbolic AI and constraint propagation [11, 12, 13, 14]. While LP formulations have the advantage of taking a global view of plan quality, they can be difficult to develop and computationally expensive to solve when including representations for state, resource, and temporal constraints. PYRRHUS [7] is a partial-order planner that must evaluate the utility of partial-plans in order to address optimization. To compute the upper bound on utility of partial plans, they make the restrictive assumption that *overall* quality does not decrease

```
Prefer linearly more goal all total occurrences between 1 and 30 weighted 200
Prefer linearly more sad_sv total duration of tracking value
Prefer linearly less sad_sv total occurrences
Prefer linearly centered activity ali_data_take parameter sad_front_bound each
  value between 250 and 350
```

Figure 4: Preference specifications for the EO-1 model.

when making refinements. Myers and Lee [15] view the optimization problem as providing a set of qualitatively different plans that can be refined by human planners. The CABINS [8] system uses a similar iterative optimization algorithm to improve complete, sub-optimal schedules. Case-based reasoning (CBR) is used to learn preferences from the user's evaluation of the plans. Finally, our approach is a specialization of black-box optimization techniques. The large search space of black-box optimization makes both finding and applying the appropriate technique prohibitively expensive.

Conclusions

We have described an approach to represent and optimize quality metrics using generic preferences for values of arbitrary variables in the plan. In our approach, we efficiently compute the set of local improvements for each preference independent of the domain by restricting the representation to monotonic functions for mapping plan values to quality metrics. We have demonstrated the feasibility in a spacecraft operations domain. More details can be found in [10].

Acknowledgements

This work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

[1] Fukunaga, A.; Rabideau, G.; Chien, S.; and Yan, D.; 1997. Toward an Application Framework for Automated Planning and Scheduling, In *Proc. of iSAIRAS-97*, Tokyo, Japan.

[2] Rabideau, G.; Knight, R.; Chien, S.; Fukunaga, A.; Govindjee, A. 1999. Iterative Repair Planning for Spacecraft Operations Using the ASPEN System, In *Proc. of iSAIRAS-99*.

[3] Minton, S.; and Johnston, M. 1988. Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems, *Artificial Intelligence*, 58:161-205.

[4] Zweben, M.; Daun, B.; Davis, E.; and Deale, M. 1994. Scheduling and Rescheduling with Iterative Repair, In *Intelligent Scheduling*, Morgan Kaufmann, San Francisco, CA. 241-256.

[5] Chien, S.; Smith, B.; Rabideau, G.; Muscettola, N.; and Rajan, K. 1998. Automated Planning and Scheduling for Goal-Based Autonomous Spacecraft, *IEEE Intelligent Systems*, September/October, 50-55.

[6] Smith, B.; Sherwood, R.; Govindjee, A.; Yan, D.; Rabideau, G.; Chien, S.; and Fukunaga, A. 1998. Representing Spacecraft Mission Planning Knowledge in ASPEN, *Artificial Intelligence Planning Systems Workshop on Knowledge Acquisition*, Pittsburgh, PA.

[7]Williamson, M.; and Hanks, S. 1994. Optimal Planning with a Goal-Directed Utility Model, In *Proc. of AIPS-94*, 176-181. AAAI Press.

[8] Miyashita, K.; and Sycara, K. 1995. CABINS: A Framework of Knowledge Acquisition and Iterative Revision for Schedule Improvement and Reactive Repair, *Artificial Intelligence*, 76(1-2):377-426.

[9] Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 1999. Integrated Planning and Execution for Autonomous Spacecraft, In *Proc. of the 1999 IEEE Aerospace Conference.*

[10] Rabideau, G.; Engelhardt, B.; Chien, S. 2000. Using Generic Preferences to Incrementally Improve Plan Quality, In *Proc. of AIPS-2000* (to appear).

[11] Baptiste, P.; Le Pape, C.; and Nuijten, W. 1995. Incorporating Efficient Operations Research Algorithms in Constraint-Based Scheduling, In *Proc. of the First International Joint Workshop on Artificial Intelligence and Operations Research.*

[12] Hooker, J. N.; Ottosson, G.; Thorsteinsson, E. S.; and Kim, H. 1999. On Integrating Constraint Propagation and Linear Programming for Combinatorial Optimization, In *Proc. of AAAI-99*, 136-142.

[13] Kautz, H.; and Walser, J. 1999. State-space Planning by Integer Optimization, In *Proc. of AAAI-99*, 526-533.

[14] Vossen, T.; Ball, M.; Lotem, A.; and Nau, D. 1999. On the Use of Integer Programming Models in AI Planning, In *Proc. of IJCAI-99*.

[15] Myers, K. L.; and Lee, T. J. 1999. Generating Qualitatively Difference Plans through Metatheoretic Biases, In *Proc. of AAAI-99*, 570-576.

Commentary on "Using Generic Preferences to Incrementally Improve Plan Quality" by Gregg Rabideau, Barbara Engelhardt, Steve Chein

Alice F. Bowman

Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Road
Laurel, MD 20723
Alice.Bowman@jhuapl.edu

Summary

In an effort to address some deficiencies of automated planning, the authors of "Using Generic Preferences to Incrementally Improve Plan Quality", develop and test a method that allows the human planner to add preferences and optional goals. As defined by the authors, preferences are quality metrics for variables in completed plans and provide the mechanism for specifying which plan variables are important to plan quality. The authors claim this can dramatically increase plan quality with very little modeling effort. The methodology is developed as an extension to the **A**utomated **S**cheduling and **P**lanning **EN**vironment (ASPEN) platform. This platform develops plans based on discrete hard constraints and mandatory goals and resolves conflicts with an iterative repair algorithm. The resulting plans are feasible but may not be acceptable because they do not include what human planners call soft constraints and optional goals. With the inclusion of these additional considerations, the authors establish a basis for automatically generating high quality plans.

The authors extend the ASPEN platform to increase plan quality by the use of preference variables and iterative optimization. The human planner chooses the time period of the plan to be generated by ASPEN. Hard constraints and mandatory goals are defined as well as optional goals and preferred values of plan variables. Mandatory goals are considered a conflict until the activity that satisfies the goal has become part of the plan. Optional goals that are not satisfied degrade the overall plan score. A plan score is calculated based on the preferred value of specified variables (preference variables). The iterative repair process, a standard part of ASPEN, resolves hard constraints and unmet mandatory goals, and iterative optimization, an extension to ASPEN, strives to increase preference variable scores. Those that are low scoring are identified and addressed until a maximum score is attained. Moving activities in time, reordering activities, etc., can increase scores. Only the highest scoring plan is kept.

The authors describe five different types of plan variable classes that contribute to plan quality. For each of these basic plan variable classes, preferred values are specified (plan variables with preferences or preference variables) and real-valued scores are returned. The plan score is computed as the weighted average of scores for plan variables with preferences. In this way, preferences allow the user to evaluate feasible plans and make quantitative distinctions between different plans. In order to increase plan quality, the plan score must be increased. This is done by iterative optimization, which represents a series of improvements made by the improvement expert algorithms. An improvement expert algorithm is developed for each of the five different types of plan variable classes and is invoked depending on the type of preference variable that is low scoring. The improvement experts iteratively select and make improvements, and thereby search for more optimal plans. This is repeated for all of the low scoring preference variables until the best overall plan score is achieved or the specified time has been reached. Experimental results of this technique are presented for the New Millennium ST-4, New Millennium EO-1, Data Chaser, and Rocky-7 Mars Rover.

Commentary

This paper presents an interesting alternative approach of increasing plan quality by developing an extension to ASPEN, a plan generating platform. According to the authors, plan quality is increased in less time and with less computer resources than without the extensions. It does so by allowing the planner to specify soft constraints (or preferences) and optional goals.

Throughout the development of this extension, the authors make a point of working with mission planners so that their extensions to ASPEN are "user

friendly". This is a very important point and is so often overlooked by software developers. In my experience the easier the program/algorithm is to use, the more it gets used and the more its tools are exploited. While other programs/algorithms may offer better features, it is the program/algorithm that is easiest to use and understand that becomes a mainstay of a mission.

Because the trend is toward smaller mission operations teams that are responsible for more tasks, it follows that the mission planner is expected to have more spacecraft mission expertise. In the past this knowledge may have been spread out among many planners. With the use of the described extensions to ASPEN, much of the required expertise to generate quality plans resides in the algorithms themselves, and thereby alleviates some of the burden of knowledge required by the now smaller planning team. This is beneficial from a quality point of view and a manpower point of view.

Some spacecraft mission concepts call for the operations team to swell and shrink based on whether the mission is in the cruise phase or data taking phase. Using an ASPEN type plan generation platform with the described extensions would be beneficial in that the expertise would be "on line" and those coming back onto the mission after an absence would be able to rely upon it. I expect that once a platform such as this is in use, mission planners will want to extend the capabilities of the algorithms to include more and more automated scheduling based on feedback from the spacecraft itself. According to the authors, these types of extensions could be accomplished.

If the intention of the authors is to have this paper read by persons whose backgrounds do not include much programming or AI experience, the addition of a diagram or flowchart outlining the process/data flow would be beneficial. This would make the relationships of the different variables and processes easier to understand. I find myself getting quickly bogged down in the descriptions of all the different variables and trying to remember how they relate as I continue reading through the paper.

As a mission planner, I would like to have a few lines summarizing what decision processes/compromises and quality checks went into the resulting plan. Currently, the only information that a planner receives on a resulting plan is a plan score. The authors state that in some cases resulting plans are a compromise between competing objectives and/or resource usage. No indication of this is given to the human planner, and this information may be very useful when combined with information outside of the scope of the plan generating algorithms. Quality issues are of major concern to any mission planner and in order to feel confident about using an automatically generated plan, some assurance has to be given that the plan was checked against some standard quality checklist.

From the point of view of a project leader, the authors might consider addressing what type of mission would benefit from this type of plan generation concept and also may want to include a section outlining the steps necessary to use this technique. Included in the discussion would be topics such as 1) how the complexity of the mission affects the development work required to use this technique, 2) what type of personnel background is required to prepare this technique for use, and what type of background is needed to use this technique on the mission, and 3) what savings are gained by using this concept (i.e., operations teams size, time required to generate plans automatically versus by standard techniques, etc.). In order to help project costs and time requirements, project leaders would also benefit from a general discussion of the work needed to ready this extended ASPEN platform for use on a mission. Since some missions may already be planning to use the ASPEN platform, the authors may want to point out that much of the modeling work required would already be done. On the other hand, missions not previously considering the ASPEN platform may have more costs associated with preparing the required modeling. The discussion may include computer platform requirements, time estimates to learn to use the extended ASPEN platform, computer models for all the subsystems, etc. This information can then be used to project costs and allow the project leader to weigh them against projected life cycle costs.

On a final note, I would find it very interesting if the authors could extend their presentation of experimental results to include not only results obtained by using the presented extensions to ASPEN but also the results obtained when normal planning tools for the specific NASA missions are used. This would give great insight to the merits of using their technique by allowing the reader to do a direct compare.

I find that this is a very interesting paper presenting many ideas that are very useful in mission operations. Although not a programmer, I can appreciate the difficult task that the authors have undertaken and understand how complicated it is to code for all the different thought processes and decisions that go into developing mission plans. I can also appreciate how hard it is to write a paper that can explain their work in terms that a person unfamiliar with programming and AI can easily understand. They are to be commended for the great work that they have done.

Mission Planning Systems for Earth Observation Missions

Marc Niézette

Anite Systems GmbH
Robert Bosch Straße 7
Darmstadt, Germany
Marc.Niezette@AniteSystems.de

Abstract

This paper describes two different approaches to Mission Planning for earth observation missions: the approach followed in European Space Agency's ATOS-4 mission planning system prototype, and the approach followed for the operational Flight Operations Segment Mission Planning System of the Envisat earth observation mission. Both systems are introduced in their respective context, together with the underlying goals and constraints that have driven their development. Finally the re-usability of the ATOS-4 approach in the scope of an operational mission is discussed.

Introduction

The Advanced Technology Operations System (ATOS) of the European Space Agency is a programme of studies into the integration of advanced applications (including knowledge based systems (KBS)) with ground systems for the support of spacecraft mission operations. The automated functions of present-day Mission Control Systems (MCS) do not yet support all aspects of mission operations. Continuous advances in the engineering technology of the space segment make possible, and bring demands for, ever-increasing complexity of mission goals and products, and hence of the mission operations tasks. At the same time there is continual pressure to reduce the costs of mission operations. The ATOS studies were conceived to tackle the problems of integrating advanced applications into a MCS while promoting re-use of existing applications, and to demonstrate the benefits to mission operations of such an advanced MCS.

The Mission Planning problem has been addressed in the scope of several of the ATOS studies, with the ambition of leading to a better understanding of the use that could be made of advanced applications in operational systems.

The sections below describe the approach developed in the ATOS-4 project. They are followed by the description of the operational Mission Planning System developed as part of the Flight Operations Segment of the Envisat-1 mission, which illustrates the differences between prototype and operational systems.

ATOS-4 Mission Planning System

ATOS-4 Project

Automated mission planning systems are essential components of an advanced MCS. The ATOS programme addresses the problem of automating off-line mission planning and on-line mission re-planning in several distinct studies. ATOS-4 is one of them, which mainly addresses the problem of developing a generic infrastructure for automated mission planning systems integrated with the other components of an advanced MCS, and demonstrates it in the context of an earth observation mission, ERS-2.

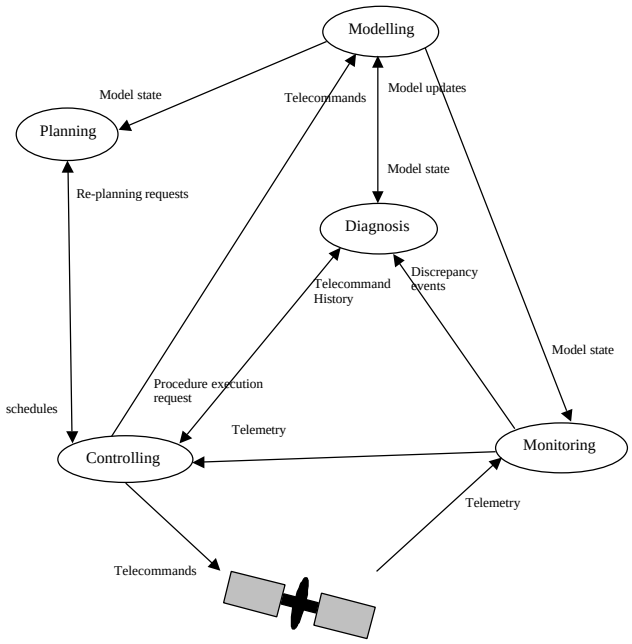


Figure 1: ATOS-4 MCS

The ATOS-4 project has developed a prototype of an Advanced Technology MCS which integrates and automates the application functions of operations schedule planning and generation, schedule execution and control,

spacecraft health monitoring, real-time and predictive model-based monitoring, anomaly diagnosis and recovery. The project's goal was to produce a logical model for an integrated model-based MCS, and to provide the basic components of standard libraries that can be used for the development of such a system.

The ATOS-4 mission planning facility provides two main modules fully integrated with the other components of the MCS: an off-line planning system and an on-line re-planning system. To address the need for reusability of these components, both of them are delivered as part of a C++ library.

Planning Interfaces

ATOS-4 is a prototype Mission Control System, which is composed of five main components, as illustrated on the diagram of Figure 1, which communicate with each other through a communication infrastructure.

The Planning component, itself composed of the planner and re-planner applications, and of a collection of data editors, interacts in operations with only two other components, the Modelling component and the Controlling component. These interactions are further detailed below.

Planning and Modelling

The key feature of ATOS-4 is the use of an Object-Oriented model of the mission to support the operations. The mission model includes structural and behavioural information about the mission, which are needed to support predictive monitoring, i.e. simulation of the future state of the mission, diagnosis, and mission planning. The Modelling application provides services either to query the static information included in the mission model, or to trigger a simulation of specific states of the model, which can in turn be queried.

The Mission Planning System holds its own representation of the information included in the mission model, which it populates by querying the model for the data needed. The planner accesses the mission model to configure its activity database from static information held in the model about the mode transitions of the instruments. It uses model simulation to derive resource profiles by extraction of parameter values stored in the model, and get the starting state of the mission from which planning of activities must start.

Planning and Controlling

The end result of the planning process is a set of schedules, which can be executed by a dedicated Controlling application. Executable schedules consist of a partial ordering of procedure execution requests, with attached start time window, precondition, and expected result. Several types of relationship between execution requests are handled by the Controlling application, namely precedence relationship and causal dependencies. In the planning process, schedules are always generated as increments to schedules that have been prepared in a previous planning session, and chosen for execution. The Controlling application monitors the execution of the selected schedules, checking the validity of the precondition of the execution request before releasing them, and checking the expected result of the requests after execution. If the Controlling application is not able to execute the schedule as expected, i.e. if the expected precondition or effects of a procedure execution request is not valid, or if the release of a procedure execution request is delayed outside its start time window, a re-planning session is started.

In a re-planning session, the Planning component receives from the Controlling component a description of the state of all activities in the schedule. It uses it to create an updated partial schedule from the initial schedule in execution, from which the re-planning session is started.

ATOS-4 Planner

The central problem of mission planning is the generation of sequences of procedures which implement user and operational requests. With respect to this problem, the goal of the ATOS-4 Project has been to experiment with the use of partial-order planning and constraint-based reasoning methods to derive a generic model for a mission planner. The sections below describe the elements of the planning logical model, and the planning process.

Services. The first step in the planning is to generate primary activities from user or operational requests selected from the pool of requests available for planning, on the basis of their preferred execution time window, and their priority. Services are defined in the planning database, one for each type of request accepted by the planner. A service defines a static mapping of a request, with preferred timings and arguments, to a set of activities on the plan.

Activities. Two types of activities are handled in ATOS-4: transitions between instrument modes, or maintaining an instrument in a given mode.

Activities are characterised by

- A conjunction of preconditions on the model and plan state.
- Effects on the model and on the plan state.
- Constraints on existing resources (use as well as provision).
- An actual procedure, executable by the controlling application.

The planning makes use of the required enabling conditions of the primary activities to generate secondary activities whose effects match those conditions. The resource constraints are propagated during planning, and solved when all required activities have been generated on the plan.

Resources. The planner handles several types of resources:

- State resources (e.g. in eclipse).
- Consumable resources (fuel).
- Reusable resources (link).

Initial profiles for these resources are sampled at intervals depending on orbital event timings. The initial resource profiles are generated from the mission model by simulation and query. These profiles are then affected by the activities generated in the course of planning.

Planning algorithm. The planning algorithm is divided in two consecutive steps.

- The planning itself, i.e. generation of the secondary activities required to enable the primary activities and completion of the instrument timelines. This is performed through partial-order planning, based on the UC-POP algorithm. Each time a new activity is put on the plan, all the constraints attached to it (resource, temporal, etc.) are posted and propagated.
- An additional scheduling step, using constraint-based reasoning to solve the constraints derived during the planning.

Planning steps. The steps followed by the ATOS-4 Planner can be summarised as follows:

1. Get a model instance representing the state of the mission at the start of the plan, produced by model simulation.

2. Generate the resource profiles by simulation and query of the model.

3. Read the user requests and generate the corresponding primary activities in the order of priority of the requests, using the service mechanism. If a primary activity implementation results in resource conflicts that cannot be solved, the primary activity is descoped.

4. Generate the secondary activities required to enable the primary activities and complete the instrument timeline. If the secondary activities cannot be created for a given primary activity, the primary activity and all derived secondary activities are descoped.

5. Map the activities generated to procedure execution requests.

ATOS-4 Re-Planner

The ATOS-4 re-planner has internally the same structure as the ATOS-4 planner. The difference between the applications lies in the on-line nature of the re-planner, and on a specific interface for receiving re-planning requests.

Re-planning can be triggered in several situations:

- Failure of a procedure execution, e.g. because the procedure could not be started in its execution time window.
- Introduction of a new request for procedure execution to be implemented with high priority, typically a request generated by the Diagnosis system.
- Report on direct execution of an immediate recovery procedure triggered by the Diagnosis system, by-passing the planner.

A re-planning request consists of a description of the state of the schedule being executed, including actual execution time for the executed procedures, as well as the identification of the procedure executions that have failed, new procedures to be inserted, and procedures whose executions have been triggered by another source.

The re-planner can then deduce the whole set of procedures that could not be executed from the nature of the dependencies between the activities on the plan, and produce an updated plan and the corresponding schedule, which is directly transmitted to the controlling application.

Implementation

The ATOS-4 system has been implemented on UNIX SUN workstations as an extension of ESA's Spacecraft Control and Operation System II (SCOSII), with C++ as main implementation language.

The Kappa development environment from Intellicorp has been used for many of the knowledge-based elements of the ATOS-4 MCS, in particular for the mission model.

ILOG Solver and ILOG Scheduler have been used in the constraint-based reasoning module of the Planning component.

Lessons Learned

The main objectives of the ATOS-4 study were to understand better the problems of integrating a large and complex automated system with advanced functionality for mission control, and to evaluate the use of such a system for various types of mission.

The ATOS-4 system has been configured for evaluation purpose to cover part of the needs of ESA's earth observation mission ERS-2.

Configuration and trials of the mission planning system have revealed the following problems:

Mission Modelling. The use of a multi-purpose Mission Model results in consistency and efficiency problems.

The ATOS-4 Mission Model covers the needs of three different applications: simulation, planning, and diagnosis. Both planning and diagnosis use abstractions of the simulation model. For instance, the planner derives the initial resource profiles for a plan from mission simulation. An active mission model is executed, and the planner extract samples of model variables at intervals from orbital events. These samples are then used to build the resource profiles. This allows relatively complex physical properties of the model, such as the battery charge, to be used to provide to the planner a profile for a resource. This approach stresses the need for controlling the consistency of the simulation model and its abstractions, which is

essential for the development and maintenance of the model. In the current implementation, the use of mission simulation for planning also appears to be computationally too costly.

Procedure Modelling. The modelling of a complex procedure by an activity is difficult. The procedures planned automatically should be restricted to command sequences.

The notion of procedure in ATOS is directly derived from the notion of automated operational procedure. As such, it is a program in an operational language that can include not only sequences of activities, but more complex constructs such as loops, etc. There are two types of procedures in ATOS: the time-tagged procedures, which are restricted to sequences of time-tagged activities to be executed on-board, and the standard procedures, which cover on-board and on-ground activities, and have to be executed through direct commanding (i.e. the satellite has to be visible from a commanding groundstation). Most of the procedures planned by the planner are time-tagged procedures (in many cases they include only one activity, corresponding to an instrument operation). One exception is the case of the diagnosis procedures, which perform checks on the systems and collect data for analysis. The ATOS-4 Mission Model includes information about the constraints related to the execution of atomic activities. When these atomic activities are combined into more complex structures that are not sequences, it becomes more difficult to model even simple properties that are needed by the planner, for instance minimum and maximum duration as a function of the arguments of the procedure.

Note that this notion of procedure does not exist in operational systems such as ERS or Envisat. For instance the Envisat FOS MPS plans atomic activities and sequences only.

Control of the Planning Algorithm. The performance of the planning algorithm does not allow handling of large numbers of dependencies between the activities and the states of the system.

This problem is partly due to the implementation itself. Nevertheless, it is likely that using the same approach in a realistic case will result in major performance problems. The key issue here is to find a way to control the planning mechanism so that the user gets an answer in a reasonable amount of time. The ATOS-4 planner incrementally updates the plan by considering in turn requests according to their priority. Requests that cannot be met are descoped from the plan. The re-planner ensures the availability of an updated plan within the interval between two commanding pass by interrupting the planning at a configurable interval between the pass and providing a solution. This approach is not elaborate enough for handling real cases.

Planning Constraints. The representation of the mission planning constraints through preconditions and effects would require the extension of the expressiveness of the language used to represent them, and therefore an extension to the planning algorithm which would make it ineffective, at least in its current implementation.

In ATOS-4, the activity preconditions and actions are limited to conjunctions of predicative forms. They express conditions that must hold before and when the activity is executed, and effects during and after the execution. They do not cover delays between related events. They do not cover either the case when the condition that enables an activity, or at least makes it sensible, must take place in the future of the activity. These cases would have to be handled, as well as disjunctive preconditions and conditional effects as they appear in UC-POP.

From ATOS-4 to Envisat FOS MPS

The limitations of the ATOS-4 planner, and the specific interfaces it requires for integration to the rest of the ground segment component, reduce considerably its direct applicability for planning a full-size earth observation mission. The Envisat Flight Operations Segment Mission Planning System, described here below, is an example of a planner for such a mission.

Envisat-1 Mission

ESA's new earth observation mission Envisat-1 will be launched in 2001. It has more options available than its ERS predecessors, including 9 instruments and a more complex Data Management System, and these factors have an important impact on the complexity of the Mission Planning.

FOS MPS and Distributed Planning

The FOS MPS is the final link of a chain of planning systems that contribute towards the overall planning of the Envisat payload and ground segment. Each of these planning systems is responsible for the performance of certain planning actions, the results achieved being passed down the line for further consideration and manipulation. These planning functions recognize three distinct planning levels – the amalgamation of these three sub-missions (listed below) forming the overall mission.

- The Global Mission - the ongoing operations that represent the long term mission strategy.
- The Background Regional Mission - more specialised than the Global Mission in that it defines a utilization strategy for regional instrument modes.
- The Regional Mission - specific requests that will tend to have a higher priority compared with the other types of Mission. These activities are not continuous in nature, and cannot be determined in conjunction with predefined events.

The FOS MPS must merge the content of the Regional Mission and Background Regional Mission information, provided by the Mission Configuration Facility planning component, with the Global Mission Plan, generated by the

FOS MPS in order to produce the consolidated Mission Plan. Further checking will be performed upon this final plan, to ensure that the planned operations are conflict free, and that they do not exceed the imposed satellite constraints.

Once it has been confirmed that the plan is acceptable, the command schedules needed to drive the spacecraft itself, the Station Computer (STC), and the Reference Measurement System (RMS) are generated.

Design Constraints

The Envisat FOS Mission Planning System consists of a suite of applications that cover the overall functionality described here above. The central application of this suite is the Plan Engine, responsible for the merging of events generated from the three sub-missions, and for detecting and solving conflicts between elements of the sub-missions and resources available.

The following key factors have influenced the design approach.

Specification of the planning requirements. The planning requirements are usually specified as rules which identify a situation and an action to be taken when the situation occurs on the plan.

Dynamic of the planning requirements. Planning requirements may be modified in the course of the mission, to adapt to the evolution of the mission, or to optimize its usage.

Use of external routines. The planning of several instruments is actually performed by external routines provided as C libraries by the customer, which have to be integrated in the Plan Engine software.

User Interaction. The requirements on the interaction between the user and the Plan Engine impose to consider the planning as a set of intermediate steps, which can be selected individually by the user.

Rule-Based Approach

Starting from a planning logical model similar to the ATOS-4 one, the elements described above have led to the following design for the Envisat FOS MPS:

- replacing in the planning model the planning algorithm for generation of secondary activities by a rule-based system.
- dropping provisionally the constraint-based reasoning element of the system, which is not justified by the nature of the problem.

The proposed use of rules to support the planning algorithm of the Envisat FOS MPS aims at keeping the planner more configurable with respect to changes to the mission specification rules and constraints. It also allows giving more control to the user on the planning process via control of the sets of rules to be executed on the plan. Its drawback is the hard-coding of the dependencies between activities in rules, which prevents the system from finding solutions that could be produced by a conventional non-linear planning algorithm.

The rule-based component of the Envisat FOS MPS is delivered as a C++ library of standard conditions and actions, which can be used to create rules. The rule condition and action evaluations are directly written in C++ in the code of the classes.

Although the idea of a generic rule-based planning system seems attractive and elegant, implementing a generic rule language to implement the planning steps would be rather risky. The dangers are

1. To produce a rule language whose expressiveness would not cover all the needs of the Envisat FOS MPS.

2. To produce a complex language which would make difficult the maintenance of the system.

3. That the design of a generic language would require a significant effort, especially in the generalization of concepts, which is not required by the final goal of the project, i.e. the design and implementation of the Envisat FOS MPS.

The alternative is to implement a simpler rule mechanism based on a C++ library including a generic rule class and a set of condition and action classes on the plan and on the Mission Planning Database. Specific conditions or actions that cannot be expressed using the basic set available can be created by specialization of the condition or action classes, leaving to the programmer the full expressiveness of C++ to code them.

In doing so, we avoid the development of a dedicated operational language for planning rules, while ensuring a minimum query/action structure in the way planning steps are implemented.

Implementation

The Envisat FOS Mission Planning System is being implemented on SUN ULTRA workstations, with C++ as implementation language. Release 2.0 of the Envisat FOS MPS has been delivered to ESA in October 1999, and has been accepted by the Agency.

It is expected that the system will be extended in the future to accommodate other missions of the same nature as Envisat, and to integrate alternative planning algorithms.

Conclusion

This paper aimed at giving an overview of two different approaches to mission planning systems, both driven by the ideal of software re-use and genericity. In each case, the ultimate goal is of course different. The ATOS-4 planner aims at demonstrating the feasibility of automating on-ground operations by integrating advanced applications to a prototype Mission Control System. The Envisat FOS Mission Planning System is an operational system, whose development is also driven by requirements for safety,

stability and maintainability, as well as by operational considerations.

There are several factors which hinder the use of the ATOS-4 approach for an operational mission such as Envisat. The most important factors are interfacing to other systems inside and outside the FOS, and the limitations and characteristics of the internal planning approach itself.

The interfaces depend mainly on operational issues related to the organisation of the mission, and are constraints on the system. For instance, the ATOS-4 planner takes advantage of its integration with the MCS and the power of the controlling application, which can interpret schedules as graphs of activities. The Envisat Flight Operations Control Center is a traditional one, and the Envisat FOS MPS is therefore restricted to schedules of time-tagged command calls. In the same way, the operational concept limits the decisions that can be taken by the planner, and constrain them to operator supervision.

As far as planning is concerned, the ATOS-4 planning approach cannot be considered as a fully generic approach that would be applicable to all areas of Mission Planning. Mission Planning covers a range of problems, which cannot be covered by a single unique approach. Even in the restricted scope of earth observation missions, many complex problems are not addressed in ATOS-4 (e.g. geographic target decomposition versus selection of one temporal solution).

Many of the Mission Planning requirements of a mission such as Envisat are related to the generation of primary activities from external sources, covered partly by the service mechanism and partly by the rule-based system. ATOS-4 concentrates on the generation of secondary activities based on causal dependencies. As such, the areas of Envisat where ATOS-4's approach could be directly applied are limited, e.g.:

- The timeline completion, whereby the gaps between partial plans of primary activities are connected to make a full plan.
- The implementation of highly dependent activity sequences. This is performed in Envisat by specific external routines.
- The derivation of schedules of operational procedures, which is not required for Envisat.

If the Envisat FOS MPS had to be extended in scope for re-use in other missions, the first two applications would surely benefit from extensions in this direction. On the third point, recent evolutions of the Mission Control Systems have focused on the integration of automatic procedure execution to controlling applications, which make the use of a planner is this area more likely in the near future.

From a more general point of view, the current evolution of AI planning systems towards providing solutions to operational planning problems would benefit from making these systems easily available for evaluation in the scope of operational missions. Providing guidance on the potential use that can be made of them with respect to the mission requirements would be an additional step towards the application of these techniques in operations. Working towards more open systems, in order to facilitate their integration to operational infrastructure is another major factor that would make possible the use of advanced techniques in an operational context.

Acknowledgement

The ATOS-4 Project was funded by ESA's Advanced Systems and Technology programmes ASTP, and its successor ASTE. These programmes have been established to promote the development of new technologies for the space domain, and supports European industry in implementing marketable products based on these.

References

[1] J. Wheadon et al., ATOS-4, Advanced Technology in Spacecraft Mission Operations, Proc. Int. Sym. Ground Systems for Space Mission Operations. SPACEOPS '98, Tokyo, June 1998.

[2] J. Wheadon, ESSOPE - Mission Operations, Their Planning Execution and Re-planning, ESA Publication PFF Vol 4 No 1, March 1994.

[3] D. Weld, An Introduction to Least Commitment Planning, AI Magazine, 15(4), pages 27-61, Winter 1994.

[4] P. Van Hentenryck, Constraint Satisfaction in Logic Programming, Logic Programming Series, The MIT Press, 1989.

[5] M. Niézette et al., Envisat FOS Mission Planning System Architectural Design Document, Issue 2 Revision 0, May 15th 1998.

Comments on the Paper "Mission Planning System for Earth Observation Missions"

Pete Bonasso (r.p.bonasso@jsc.nasa.gov)

Texas Robotics and Automation Center Laboratories (TRACLABs)
1012 Hercules
Houston, TX 77059

Introduction

The paper in question essentially discusses two mission planning systems, one which uses at its core UC-POP, a result of several decades of AI research in automated plan generation, and the other which resorts to encoding human plan knowledge in the form of rules to be executed by an rule-base system (RBS). The orientation of the former system, the ATOS-4 planner, is as an experimental prototype, whereas the latter, the Envisat mission planning system, is in the context of a near fielded application.

The concern that leapt to my mind after reading this paper was simply how do the authors expect to carry out a real planning application without a planner? A rule-base system (RBS) can only guarantee that it will fire all rules that apply when they apply, but it can guarantee nothing about finding a plan in the space of action sequences. In my on line interchange with the first author, I came to realize that several issues bear on this question:

- existing software can limit what any planner can do
- the need to extend the planning approach
- the reality of human computer interfaces (HCI)
- the trade-off between a theoretical loss of power and an approach that will just get the job done for most intents and purposes

All four of these issues point to hard questions that need to be answered in the future, both for the AI community and for those who would apply AI results.

Existing Software

I learned in my interchange with the first author that apart from the basic logical model, the Envisat system was fully developed from scratch, yet some modules for the generation of primary activities for specific instruments had been developed by the space engineers as separate libraries. These procedures are not easily reduced to sequences of actions with start and end times, but can include arbitrary types of code. This makes it more difficult to model even simple properties that are needed by the planner, for instance minimum and maximum duration as a function of the arguments of the procedure.

In many applications, particularly control applications, one is usually constrained by the software that has been hand-coded for particular instruments. In a very large project we under took to control life support systems [Schreckenghost et al 98], we were constrained to build a hybrid control system "on top of" a suite of legacy control software. As a result we were not able to realize the theoretical efficiencies of the hybrid system.

Extending the Planner

The authors learned through their endeavors that the UC-POP system they might use from ATOS-4 for Envisat was limited in terms of the temporal constraints that exist between the depending activities in both environments. In UC-POP, secondary activities can only be created before or during the main activity. An attempt was made to analyze the dependencies existing between the Envisat mission activities; the temporal dependencies that can exist between them were far wider in scope. Basically a precondition for enabling an activity could be created by other activities anywhere in the plan, within constraints with respect to the initial activity. Many dependencies include fixed or variable delays (e.g., the secondary activity must have been executed at least one hour before the initial one). This capability would have to be added. The planning algorithm itself would have to be extended to handle threads to dependencies with future activities.

Here then we see a real limitation of a planning algorithm for a particular application. If the team were supported by a nearby research group, as is the case in several CMU projects where AI is applied in the nearby community, the Envisat team might have considered effecting the needed changes. But such was not the case, and the team had little experience in the use and modification of planners, and certainly not with the specific UC-POP code. As the first author put it in one exchange with me

> The technical constraints on the design in general were such that it was not clear that we could use the ATOS

> approach and still take advantage of it. The rule-based approach was considered safer.

The Reality of HCI

The Envisat team was more familiar with RBS techniques, than with planning software. This was especially true with regard to developing interactive human computer interfaces. It was not clear to them that they could extend the AI planning approach such that it would have accommodated the user interactive requirements and have it working on budget and schedule.

The HCI requirements in any application are as critical as any theoretical aspects associated with the type of problem. But here I suspect there might have been ways to build an HCI to UC-POP, and that the real issue was the limited budget. For instance, we have used an HTN planner at NASA-JSC since 1994 [Bonasso et al 97]. It came with an CLIM interface which was limited in what it could provide the user. Our solution was to make the HCI a separate application executable on a wider variety of workstations, and network it to the planner using a client-server message passing system.

Theory versus Practicality

There's no denying that UC-POP is sound and complete. It's not clear what one can say about a planner built in an RBS. My claim is that the UC-POP system will find a plan when the RBS planner may not. So it stands to reason that one should make an effort to "extend" the planning system rather than "settle" for the less complex, but fault prone RBS.

But it's not that simple. Although I have no personal experience with UC-POP, I understand that it is usually difficult to modify systems which emerge from an academic environment. As well, the ATOS team was not allowed to use the Lisp version, and had to recast the algorithm in C++. They did some of the extensions mentioned above for ATOS, but there was little time to effect all the needed changes for Envisat.

Further, there may be something to be said for what an experienced RBS team can do to provide an intelligent system on time and within budget, and still come up with a better than average reasoning system. The Envisat system basically schedules activities for satellites. The system will always produce a solution. It will simply de-scope requests and move activities around until it finds one. In the worst case this will be the degenerate solution: switching all the instruments to their safe mode. The team is wary of some of the pitfalls:

> We try to control things in the rule-based approach by splitting the rule set in modules that are activated separately. In practice, we only have to hope that our testing of these modules will ensure a correct functioning of the system, as per any other application. The modules are combined in finite sequences, so the risk is limited to individual modules. The risk actually exists for some modules only.

Conclusion

Nevertheless, my gut feel is that as earth observing satellites become more advanced, and as we begin to need to control more than one of them at a time, the task will quickly overcome an RBS approach. So what is to be done? I believe some answers lie within the AI research community but most must come from those who seek to apply AI technology.

Making major AI systems accessible via a remote HCI must have more of a priority in AI research. If we wish to see our solutions used in real world applications, we must make the development of APIs to get at function, value and representation an equal partner with efficiency and theoretical soundness. Of course that requires us to convince our funding groups of that need. Pointing out applications like Envisat can be a first step in that argument.

The rest of the issues seem to me to be resolvable if the engineering team makes the case for a budget which looks to the future. For instance, getting help in the form of consultation by universities might uncover solutions to extending a given planning algorithm. Or perhaps UC-POP isn't the planner for this problem. In my opinion, with the future in mind, it behooves the Envisat team to investigate other planning approaches. For example, the Envisat temporal constraints might be more readily handled with an interval planner such as the one used in Remote Agent [Pell et al 97]. Also, some planners exist that can accommodate user functions and legacy software. Two of these are SIPE as used in Cypress [Wilkins et al 95], and AP as used in 3T [Elsaesser and Slack 94]. AP in particular allows the integration of user functions in its operator preconditions, effects and task nets.

And finally, an effort ought to be made to convince management to include a person skilled in AI planning on the applications team. So many large applications require planning techniques that it should be worth the effort in the long run.

All of these suggestions require more time and resources. But I believe the answer is to convince the user of the importance of employing the very best technology has to offer rather than settling for good enough.

References

[Bonasso et al 97] R. Peter Bonasso, R. J. Firby, E. Gat, David Kortenkamp, D. Miller and M. Slack, "Experiences with an Architecture for Intelligent, Reactive Agents",

Journal of Experimental and Theoretical Artificial Intelligence, 9(2), 1997, pp. 237-256.

[Elsaesser & Slack 94] Chris Elsaesser and Marc G. Slack. 1994, Integrating Deliberative Planning in a Robot Architecture, in Proceedings of the AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service, and Space (CIRFFSS '94).

[Pell et al 97] Barney Pell, Erann Gat, Ron Keesing, Nicola Muscettola and Ben Smith. 1997. Robust Periodic Planning and Execution for Autonomous Spacecraft, in Proceedings of the 1997 International Joint Conference on Artificial Intelligence.

[Schreckenghost et al 1998] Schreckenghost, D; Bonasso, P.; Kortenkamp, D.; and Ryan, D. Three Tier Architecture for Controlling Space Life Support Systems . IEEE Symposium on Intelligence in Automation and Robotics. May,1998.

[Wilkins etal 95] David E. Wilkins and Karen L. Myers and John D. Lowrance and Leonard P. Wesley. 1994, "Planning and Reacting in Uncertain Dynamic Environments", Journal of Experimental an Theoretical AI (7).

Acting on information: A plan language for manipulating data

Keith Golden
NASA Ames Research Center
M/S 269-2
Moffett Field, CA 94035-1000
kgolden@ptolemy.arc.nasa.gov

Abstract

Information manipulation is the creation of new information based on existing information sources. This paper discusses problems that arise when planning for information manipulation and introduces an action language, ADLIM, that addresses these problems. Topics include

- How to represent information in a way sufficient to express the effects of actions that modify the information.
- How to concisely represent actions that copy information, or produce new information that is based on existing information sources.
- How to generate a pipeline of information-processing commands that will produce an output containing the desired information.

1 Introduction

A big part of NASA's job is information management. Satellites, unmanned spacecraft, planetary rovers and observatories, for all their complexity, can all be viewed as remote sensors; their sole purpose is to gather data, which they transmit across a vast, interplanetary network, where it is processed, delivered to the scientific community, and archived. Just as controlling spacecraft is a complex task that can benefit from automation, so is managing the data that spacecraft returns. Currently, automation usually consists of writing some scripts to handle expected cases. Some problems with this approach are lack of flexibility in the face of changing requirements or one-time requests, lack of robustness in the face of errors or unanticipated events, and poor data tracking. The latter is a serious problem, since much of the information about various data files and how they are inter-related is stored in the heads of operators, who will eventually go on to other missions, or forget.

We are developing a system, called IMAGEbot[1], to automate information manipulation tasks, such as generating mosaics or converting file formats, and to automatically record in a database the vital meta-information about files that have been produced, such as how they were generated, or which parts of the telemetry logs correspond to which science data. IMAGEbot will robustly respond to errors, such as disks filling up, and will be able to provide greater functionality than hand-generated scripts, since it will generate its own scripts based on user-supplied goals.

This work builds on experience with the Internet Softbot (Etzioni & Weld 1994) and other information-gathering agents. However, the information manipulation problem is beyond the grasp of current softbots. There has been considerable work on developing planners for *gathering* information, but almost no work on planners for *manipulating* information. That is not because manipulating information is not important, but because representing actions that manipulate information is hard. For example, in the Internet Softbot, the `copy` action was never cleanly or accurately represented, and it was impossible to represent Unix pipes (which redirect the output of one command to the input of another) or actions like `tar` (which creates an archive of a collection of files) and difficult to achieve image processing tasks.

We present an action language, ADLIM, that is capable of representing such actions easily and concisely. ADLIM stands for Action Description Language for Information Manipulation, and is an extension of Pednault's ADL (Pednault 1989). It is intended for problem domains, such as space science missions, where information manipulation tasks, like image processing and data archiving and distribution, are commonplace.

The next section discusses how actions and goals for information manipulation are represented in ADLIM. Section 3 discusses a problem that comes up when trying to represent actions that copy or modify information and shows how it is a generalization of the frame problem. Section 4 discusses temporal projection in ADLIM and illustrates it with a simple example from a planetary rover domain. Section 5 discusses related work.

2 Information manipulation

Since the purpose of ADLIM is to represent actions that process data inputs and produce data outputs, inputs

[1] IMAGE stands for Information Manipulation, Archiving, Gathering and Extraction

and outputs are explicitly declared in action descriptions. Every variable is declared as an input, output, parameter or quantified variable. Inputs and outputs are distinct from other variables in that an input is not guaranteed to exist after the action is executed, and an output does not exist before the action is executed.

An action that has no inputs and one or more outputs is called an *information source*. An action that has one or more inputs and no outputs is an *information sink*. An action that has one or more inputs and one or more outputs is an *information filter*. An information filter processes the inputs, producing the outputs. Contrary to the behavior of a physical filter, it does not necessarily remove anything from the input, and may add something or change it completely.

Effects

UWL (Etzioni *et al.* 1992) and SADL (Golden & Weld 1996) represent information-producing actions using the annotation **observe**. For example, to represent that executing the action `ls /bin` reveals the name of every file in `/bin`, the SADL encoding would be:

$$\forall f\, \exists n \textbf{ when } (\mathsf{in.dir}(f,\ \texttt{/bin})) \textbf{ observe}(\mathsf{name}(f, n))$$

However, the encoding using the **observe** annotation does not actually represent the effects of `ls`, but rather the combination of `ls` with a program to interpret its output and produce a set of knowledge base updates. Such a program is called a *wrapper*. Since a filter works on the syntactic output of a program, not the semantic interpretation, using **observe** throws away vital information that a planner needs to reason about the effect of a filter: the syntax. Instead, we describe the syntax of the output directly. We divide conditions that can be sensed into two categories: simple observables and fluents.

Simple observables are functions or relations whose value is independent of the situation in which they are evaluated. For example, the relation contains(`"foobar"`, `"foo"`) is easily answerable by examining only the syntactic representation of "`foobar`" and "`foo`". Rather than providing a sensing action for a simple observable, we provide a function that returns its value (if all arguments are bound), or provides additional constraints for its arguments (if some arguments are not bound).

Fluents are functions or relations whose value depends on the situation. For example, the relation in.dir(`foo,/bin`), which means the file `foo` is in the directory `/bin`, will be true in some situations and false in others. A sensing action can be described using conditional effects, where the antecedent refers to the situation in which the action is executed, described using fluents, and the consequent refers to the contents of the output, described using simple observables. E.g., the effect of `ls` discussed above would be

$$\forall f\ (\mathsf{parent.dir}(f) = \texttt{/bin}) \rightarrow \mathsf{contains\text{-}line}(\mathsf{name}(f),\ out)$$

This translates to "For each file in directory `/bin`, there is a line in the output that is equal to the name of the file." The $\rightarrow$ is used instead of **when** (which is used in SADL) to indicate a conditional effect. The reason for this notation will become clear in the next subsection. We will use LHS to designate the expression on the left hand side of the $\rightarrow$ and RHS to denote the expression to the right of the $\rightarrow$. Note that we use function composition as a shorthand. The above goal is equivalent to

$$\forall f, n\ \ (\mathsf{parent.dir}(f) = \texttt{/bin} \wedge n = \mathsf{name}(f)) \rightarrow \mathsf{contains\text{-}line}(n,\ out)$$

Two things to note are that $\mathsf{name}(f)$ is now on the LHS and n is universally quantified ($\exists$ was used in the SADL example). $\forall$ and $\exists$ are equivalent when the variable represents the value of a function, since there is alway exactly one possible value. We use $\forall$ here because it's simpler. Other examples in the paper can be transformed in the same way: replace each nested function with a new universally quantified variable and add an equality constraint between the variable and the function to the LHS.

Goals

Information manipulation goals, like effects, must be explicit about the syntax of the output. Consider the goal of outputting the result of a query to a file. Merely ensuring that the file contains the information is not sufficient. For example, a list of file names should not be mixed up with a list of userids, since it may be difficult to tell which is which. Furthermore, if the file is to be read by another program, there will be exact formatting requirements.

Suppose our goal is to to produce a killfile, which is a list of email addresses that we don't want to receive email from, each on a separate line. Let's say we don't want to receive email from anyone who has sent us email containing the string "`MAKE MONEY FAST`." We might express this goal as:

$$\forall em\ \mathsf{email\text{-}received}(em) \wedge \mathsf{contains}(\mathsf{subject}(em),\ \texttt{"MAKE MONEY FAST"}) \rightarrow \mathsf{contains\text{-}line}(\mathsf{sender}(em),\ \texttt{killfile})$$

This has the same form as the conditional effects discussed in Section 2, but the meaning is slightly different. Whereas, in effects, the statement $A \rightarrow B$ indicates that if A is true before the *action* is executed then B will be true after the *action* is executed, in goals it means that if A is true before the *plan* is executed, then B must be true after the *plan* is executed. In SADL goals, A would be represented using **initially** and B would be expressed using **satisfy** (Golden & Weld 1996). One problem with the above goal is that including all known email addresses in the killfile would also satisfy the goal. To express more restrictive goals, we also allow expressions of the form $A \overset{\Leftrightarrow}{\rightarrow} B$, which is equivalent to $(A \rightarrow B) \wedge (\neg A \rightarrow \neg B)$.

Information pipelines

An information pipeline is one or more information sources followed by a sequence of filters, possibly terminated by an information sink, in which the output of each action is directed to the input of a following action. The effects of an information pipeline can be represented in the same way as the effects an individual action. The LHS refers to the conditions that are true when the first action is executed and the RHS refers to the contents of the output and the conditions that are true after the last action is executed. This representation can be generated automatically from the individual action descriptions, using the same process that is used in planning (see Section 4). In this manner, a precise description of of the contents of the output of a pipeline is generated, relating the meaning of the information to its form. This description can be archived, along with the time and circumstances under which the output was generated, for later use by IMAGEbot. Thus, if IMAGEbot has a goal, or subgoal, of generating a file with particular properties, and it has generated such a file before, it can use the previously generated file instead.

The same approach can be used to describe files or other information resources not generated by IMAGEbot. It can then use these as information sources for in its plans.

3 The copy problem

The representation of information filters presents a challenge. A filter creates a new object, such as a text file, which is based on an existing object. Although the input and output of a filter are distinct objects, they have much in common. The output may be a copy of the input, with some changes.

We call this the *copy problem,* which can be understood by considering the effects of a photocopier. Initially, we have an original of a document and a blank sheet of paper. After running the photocopier, everything that was true about the original is still true (the frame problem). Additionally, most everything that was true about the original is now also true about the formerly blank page (the copy problem). In some ways, the copy will differ from the original – for instance, it may be of lower quality, or the original may be color whereas the copy is black and white, but if the original contained, say, a tax form, or the first 500 digits of π, then so will the copy. Listing all of these conditional effects explicitly would be impractical.

The copy problem is just a generalization of the frame problem; if the original and the copy are the same object, then we have a restatement of the frame problem. But we are interested in the case where the objects are different.

There have been many solutions to the frame problem, from the STRIPS assumption to logical formalisms such as (Reiter 1991). None of these directly meets our needs, since they are solving a different problem. What we need is to *explicitly* state that an output is identical to an input unless stated otherwise. For example, when creating a compressed copy of a file, one should be able to declare that the size and compression of the files are different, but in all other respects they are identical. We refer to such declarations as *generalized frame effects* (GFEs): The effect $\mathsf{frame}(source, dest)$ in an action a means that, for any predicate or function p, the value of $p(dest)$ after a is executed is the same as the value of $p(source)$ before a is executed, unless contradicted by another effect of a. *Source* and *dest* need not be the same type, but predicates defined for *source* must also be defined for *dest*. Given such declarations, it is straightforward to formalize them using one of the elegant formal solutions to the frame problem.

We require that each *dest* be a newly created object and the target of only one GFE. However, a single output can comprise many distinct objects, each of which may be the target of a separate GFE (see the description of `tarc` below). Given incomplete information, not all the consequences of the GFE will be known, but anything known about *source* will be known about *dest*, and all prior positive knowledge will still be valid. However, negative knowledge, including LCW knowledge, may be invalidated.

One might imagine dispensing with the inputs and outputs and representing all actions as destructive. Then the STRIPS assumption could be used to preserve attributes of the files that don't change. This would require that all files that need to be preserved be explicitly copied (Chien *et al.* 1997). However, doing so merely pushes the frame problem into the action $\mathsf{copy}(f_1, f_2)$, since for any proposition $p(f_1)$ that is true before the copy, $p(f_2)$ should be true afterward. Furthermore, this approach is only applicable in cases where a single input is mapped to a single output. It will not help when modeling the effects of an action that generates mosaics (combining many images into one).

Example

To illustrate the use of frame effects, consider the Unix command `tar`, for creating and extracting tar files. A tar file is an archive (**tar** stands for **t**ape **ar**chive) of some collection of files. It is really little more that a concatenation of all the files, with some extra information to indicate where one file begins and another ends and the relative pathnames of the files. We refer to the portion of a tar file representing a particular file as a file record, and we use the predicate $\mathsf{contains}(t, r)$ to indicate that tar file t contains file record r. There is no magic in $\mathsf{contains}$. In fact, it's a simple observable, since, given the complete contents of a tar file and a file record, testing whether the file $\mathsf{contains}$ the record is a matter of a linear search through the tar file for contents that exactly match the record.

We represent the two functions of the `tar` command with two different actions. The `tarc` action *creates* a tar file from the contents of a directory, descending the directory hierarchy recursively. To avoid representing

the recursion explicitly, `tarc` does not refer to directories, but to pathnames. A file is (recursively) contained within a directory if the pathname of the directory is a prefix of its own pathname. The parent.dir function can also be defined in terms of pathname. The output of `tarc` is a newly created tar file, containing a file record for each file reachable from the directory. Each file record is identical to the original file, except that it has only a relative pathname, and it is not located on any machine.

```
action tarc (path dp, exec-context ec)
 output: tarfile out
 precond:  pathname(currentdir(ec)) = dp
 effect:
   ∀ file(f),path(lp)
       (pathname(f) = concat(pd,"/",lp) ∧
       host-loc(f) = currenthost(ec))
         → ∃ file-record(fr)
               frame(f, fr)∧
               host-loc(fr) = nil ∧
               contains(out, fr)∧
               pathname(fr) = lp
 exec: "tar cf -"
```

The `tarx` action *extracts* information from the tarfile and creates the corresponding files and directories in a new location. For each file record in the tarfile, a new file is created, identical to the file record, except that it has a new pathname and host location. Although these action descriptions omit some minor details, they are essentially complete. The key to their brevity is the frame effects, which stand for a huge number of statements.

```
action tarx (path dp, exec-context ec_in)
 input: tarfile in
 precond:  pathname(currentdir(ec_in)) = dp
 effect:
   ∀ file-record(fr),path(lp)
       (pathname(fr) = lp ∧ contains(in, fr))
         → ∃ file(f)
               frame(fr, f) ∧ ¬contains(in, f) ∧
               host-loc(f) = currenthost(ec_in) ∧
               pathname(f) = concat(dp,"/",lp)
 exec: "tar xf -"
```

It is common practice in Unix to use `tar` to copy large file hierarchies from one machine to another. In Section 4, we provide a short example showing how the frame effects allow properties to be preserved across copy operations using `tar`.

4 Reasoning about plans

Our definition of a planning problem is the standard one: Given descriptions of allowable actions, a goal and the initial state, produce a plan that, when executed starting from the initial state, will move the world to a state satisfying the goal. Regardless of the specifics of planning algorithm used, a planner following this definition must be able to answer one of the following questions:

- If I execute this action, how will the state change? (progression)
- If I want my goal to be satisfied after I execute this action, what needs to be true beforehand? (regression)

We discuss regression in ADLIM. We do not discuss progression, which is somewhat more complicated, but we note that since part of an ADLIM goal refers to the initial state, progression affects not just the initial state, but also the goal. We conjecture that under some circumstances (such as information gathering) that forward planning will be more efficient than backward planning, and that a mixed strategy is likely to work the best.

Regression

Goal regression means determining the conditions that need to be true in the initial state for an action or action sequence to achieve a given goal. We will use $\mathsf{R_a}(\Gamma)$ to represent the result of regressing Γ through action a, and $\mathsf{R}_{\{a\}_1^n}(\Gamma)$ to represent the result of regressing Γ through the action sequence $a_1; a_2; \ldots; a_n$. Regression closely follows (Pednault 1986).

Regression of the empty plan succeeds iff, in the initial state, the LHS implies the RHS.

$$\mathsf{R}_{\{\}}(\Phi \rightarrow \Psi) = (\Phi \Rightarrow \Psi)$$

Regression of a plan consists of successively regressing each action, starting with the last.

$$\mathsf{R}_{\{a\}_1^n}(\Gamma) = \mathsf{R}_{a_1}(\mathsf{R}_{a_2}(\ldots \mathsf{R}_{a_n}(\Gamma)))$$

Conjunction, disjunction, quantification and negation are handled in the usual manner. Namely, $\mathsf{R}_a(\Gamma_1 \wedge \Gamma_2) = \mathsf{R_a}(\Gamma_1) \wedge \mathsf{R_a}(\Gamma_2)$, $\mathsf{R}_a(\Gamma_1 \vee \Gamma_2) = \mathsf{R_a}(\Gamma_1) \vee \mathsf{R_a}(\Gamma_2)$, $\mathsf{R_a}(\neg\Gamma) = \neg\mathsf{R_a}(\Gamma)$, $\mathsf{R_a}(\forall x\Gamma) = \forall x\mathsf{R_a}(\Gamma)$ and $\mathsf{R_a}(\exists x\Gamma) = \exists x\mathsf{R_a}(\Gamma)$.

Given a goal of the form $\Phi \rightarrow \Psi$, regress the Ψ and leave the Φ alone (since it already refers to the initial state).

$$\mathsf{R_a}(\Phi \rightarrow \Psi) = \Phi \rightarrow \mathsf{R_a}(\Psi)$$

Finally, to regress a single literal: φ is true iff the action makes it true, or if it was true previously and the action doesn't make it false.

$$\mathsf{R_a}(\varphi) = (\Sigma^a_\varphi \vee (\varphi \wedge \Pi^a_\varphi))$$

where Σ^a_φ means action a enables φ (makes it true) and Π^a_φ means a preserves φ (doesn't make it false). Informally, $\Sigma^a_{p(c)}$ is true iff a has an effect $p(c)$, or if $p(o)$ is true, and a has an effect of the form $\mathsf{frame}(o, c)$ and doesn't have an effect $\neg p(c)$. Π^a_φ is equivalent to $\neg\Sigma^a_{\neg\phi}$.

Example

To illustrate goal regression, consider the following highly simplified example. Suppose our goal is to store the images downlinked from a planetary rover in the directory `/images`, with a compression quality of 95%. Achieving this goal requires no sensing.

$\forall$ image j $\exists$ file f, filename fn
fromdownlink(j)$\wedge$format(j)=JPEG $\rightarrow$
pathname(f) = concat("/images/", fn)$\wedge$
copy-quality(f, j)≥ 0.95

We will regress this goal over "lifted" actions, as a planner would, but we gloss over certain subtleties of constraint reasoning, skolemization, and the like. The last action in the plan will be `tarx`, to extract files from a tar file, so we, regress through `tarx` first. We will call that action a_1, and rename all variables from the action by prefixing them with "a_1." The goal pathname(f) = concat("/images/", fn) is satisfied by a_1 if concat("/images/", fn) = concat($a_1.dp$, "/", $a_1.lp$). Note that one solution for this constraint is $a_1.dp$ = "/images". If we provide the background knowledge that fn, being a filename, cannot contain "/", and $a_1.dp$, being a pathname, cannot be the empty string, then that solution is unique. However, even without that background knowledge, $a_1.dp$ = "/images" is the only solution that will be found as part of a correct and unambiguous plan. Any other solution either leaves the suffix of parameter $a_1.dp$ unspecified or adds an unsupported constraint to the prefix of $a_1.lp$.

The predicate copy-quality does not appear in the effect of action a_1, but there is an effect frame($a_1.fr$, $a_1.f$), which will transfer the copy-quality of $a_1.fr$ to f. Thus, the goal copy-quality(f, j)≥ 0.95 will be satisfied if copy-quality($a_1.fr_j$, j)≥ 0.95 is true in the prior state, where $a_1.fr_j$ is a new existentially quantified variable, within the scope of j, that is used in place of the $\forall$ variable $a_1.fr$. Finally, the LHS of the conditional effect and the precondition of the action must be true. Thus, the goal is now:

fromdownlink(j)$\wedge$format(j)=JPEG $\rightarrow$
$a_1.dp$ = "/images" $\wedge$
pathname($a_1.fr_j$) = fn $\wedge$
contains($a_1.in$, $a_1.fr_j$) $\wedge$
copy-quality($a_1.fr_j$, j)≥ 0.95 $\wedge$
currentdir($a_1.ec_{in}$) = $a_1.dp$.

Regressing through the `cd` action satisfies the currentdirectory goal. Then regress through `tarc`, which we will designate a_2. The contains condition will be satisfied if $a_1.in = a_2.out$. The pathname condition is satisfied if the LHS is true. Once again, the copy-quality condition can be satisfied by resorting to the frame effect of a_2, resulting in the goal copy-quality($a_2.f_j$, j)$\geq$ 0.95. The resulting goal agenda is:

fromdownlink(j)$\wedge$format(j)=JPEG $\rightarrow$
$a_1.dp$ = "/images" $\wedge$ $a_1.in$ = $a_2.out$ $\wedge$
copy-quality($a_2.f_j$, j)≥ 0.95 $\wedge$
pathname($a_2.f_j$) = concat($a_2.pd$,"/",fn) $\wedge$
host-loc($a_2.f_j$) = current.host($a_2.ec$) $\wedge$
currentdir($a_2.ec$) = $a_2.dp$

The constraint on $a_1.dp$ specifies a parameter of `tarc`, the directory to extract the tar file into. The constraint $a_1.in = a_2.out$ specifies that the output of `tarx` must be directed to the input of `tarc`. Once again, regressing through `cd` satisfies the currentdirectory goal. Now suppose there is an action `get-downlink`, which gets all the JPEG images from the downlink and stores them in a directory `/downlink/jpg`, ensuring that each image has a quality of 0.99. Call that action a_3. The effect of a_3 would be

fromdownlink($a_3.g$)$\wedge$format($a_3.g$)=JPEG $\rightarrow$
copy-quality($a_3.c$, $a_3.g$)$= 0.99$ $\wedge$
pathname($a_3.c$) =
concat("/downlink/jpg/", filename($a_3.c$) $\wedge$
host-loc($a_3.c$) = current.host($a_3.ec$)

The resulting goal agenda would be

fromdownlink(j)$\wedge$format(j)=JPEG $\rightarrow$
$a_1.dp$ = "/images" $\wedge$ $a_1.in$ = $a_2.out$ $\wedge$
$a_2.pd$ = "/downlink/jpg/" $\wedge$ $a_2.ec$ = $a_3.ec$ $\wedge$
fromdownlink($a_3.g$)$\wedge$format($a_3.g$)=JPEG

The last two conditions are implied by the LHS of the goal, so they are automatically satisfied. The remaining conditions are parameter choices, which are trivially satisfied. Thus, the plan `get-downlink;cd /downlink/jpg; tar xf - | '(cd /images; tar cf -)'` achieves the goal. In practice, downlinks are not initiated directly by the planner, and the goal would involve moving the files to a different machine rather than just a different directory. That could be accomplished by inserting an `rsh` (remote shell) command between the `tarx` and `cd` commands.

5 Conclusions

We presented ADLIM, the Action Description Language for Information Manipulation, which can concisely represent actions that copy all or part of an input to an output. We observed that this is a generalization of the frame problem, which has not been noted before in the planning literature, and presented a solution, using frame effects. We showed how information gathering can be accomplished in this framework.

Future work

We are building a constraint-based planner for ADLIM, but many issues still need to be resolved. Constraint reasoning in ADLIM is especially challenging, since most constraint solvers assume that variable domains are static, whereas, in ADLIM, domains may be completely or partially unknown. We are working on a constraint solver, using procedural constraints (Jönsson 1997), that can cope with unknown domains.

Related work

We have already discussed the relation between ADLIM and UWL (Etzioni *et al.* 1992), ADL (Pednault 1989) and SADL (Golden & Weld 1996). Collage (Lansky & Philpot 1993) and MVP (Chien *et al.* 1997) both automate image manipulation tasks, a motivating problem

for ADLIM. However, they don't focus as much on accurate causal models of information manipulation. MVP requires actions to destructively modify their inputs, relying on the STRIPS assumption to preserve properties not listed in the action's effects. Collage relies solely on abstract action decomposition and thus does not need a precise causal theory of the actions.

Representing actions that manipulate information is related to representing sensing actions. (Moore 1985) introduced a theory of knowledge and action, based on a variant of the situation calculus with possible-worlds semantics, which included an analysis of information-providing effects. (Scherl & Levesque 1993) built on Moore's work, providing a solution to the frame problem for knowledge-producing actions. The semantics for ADLIM has been specified following their formalization, but we base our language on ADL, which allows for more tractable planning. Our treatment of knowledge-producing actions using conditional effects follows (Pryor & Collins 1996) and others, but we are unaware of work that treats goals in a similar manner.

There are many other action languages that represent sensing, but none of them have the expressiveness of ADLIM. They either disallow sensing the value of a variable (Levesque 1996; Goldman & Boddy 1996; Pryor & Collins 1996), thus restricting sensors to returning a finite set of possible values, or they disallow the use of conditional effects to describe sensing actions (Kwok & Weld 1996; Levy, Rajaraman, & Ordille 1996; Knoblock 1996; Babaian & Schmolze 1999; Etzioni *et al.* 1992), which is essential for representing information outputs that can be manipulated by other actions.

Acknowledgements

I would like to thank Jeremy Frank, Ari Jonsson, David Smith, Ellen Spertus, Richard Washington and the anonymous reviewer for helpful comments.

References

Babaian, T., and Schmolze, J. G. 1999. PSIPLAN: Planning with ψ-forms over partially closed worlds. Unpublished.

Chien, S.; Fisher, F.; Lo, E.; Mortensen, H.; and Greeley, R. 1997. Using artificial intelligence planning to automate science data analysis for large image database. In *Proc. 1997 Conference on Knowledge Discovery and Data Mining*.

Etzioni, O., and Weld, D. 1994. A softbot-based interface to the Internet. *C. ACM* 37(7):72–6.

Etzioni, O.; Hanks, S.; Weld, D.; Draper, D.; Lesh, N.; and Williamson, M. 1992. An approach to planning with incomplete information. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, 115–125.

Golden, K., and Weld, D. 1996. Representing sensing actions: The middle ground revisited. In *Proc. 5th Int. Conf. Principles of Knowledge Representation and Reasoning*, 174–185.

Goldman, R. P., and Boddy, M. S. 1996. Expressive Planning And Explicit Knowledge. In *Proc. 3rd Intl. Conf. AI Planning Systems*.

Jönsson, A. 1997. *Procedural Reasoning in Constraint Satisfaction*. Ph.D. Dissertation, Department of Computer Science, Stanford University.

Knoblock, C. 1996. Building a planner for information gathering: A report from the trenches. In *Proc. 3rd Intl. Conf. AI Planning Systems*.

Kwok, C., and Weld, D. 1996. Planning to gather information. In *Proc. 13th Nat. Conf. AI*.

Lansky, A. L., and Philpot, A. G. 1993. AI-based planning for data analysis tasks. In *Proceedings of the Ninth IEEE Conference on Artificial Intelligence for Applications (CAIA-93)*.

Levesque, H. 1996. What is planning in the presence of sensing? In *Proc. 13th Nat. Conf. AI*.

Levy, A. Y.; Rajaraman, A.; and Ordille, J. J. 1996. Query answering algorithms for information agents. In *Proc. 13th Nat. Conf. AI*.

Moore, R. 1985. A Formal Theory of Knowledge and Action. In Hobbs, J., and Moore, R., eds., *Formal Theories of the Commonsense World*. Ablex.

Pednault, E. 1986. *Toward a Mathematical Theory of Plan Synthesis*. Ph.D. Dissertation, Stanford University.

Pednault, E. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. 1st Int. Conf. Principles of Knowledge Representation and Reasoning*, 324–332.

Pryor, L., and Collins, G. 1996. Planning for contingencies: A decision-based approach. *J. Artificial Intelligence Research*.

Reiter, R. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., ed., *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*. Academic Press. 359–380.

Scherl, R., and Levesque, H. 1993. The frame problem and knowledge producing actions. In *Proc. 11th Nat. Conf. AI*, 689–695.

Commentary on Acting on information: A planning language for manipulating data

Patricia Johnson

NASA Goddard Space Flight Center
Code 583
Greenbelt, MD 20771
Pjohnson@pop500.gsfc.nasa.gov

Introduction

This commentary discusses the paper: Acting on information: A planning language for manipulating data. The planning language is an action description language used to manipulate information, such as images from a spacecraft, and the paper describes how it works. Examples are given using mathematical expressions. The author helped clarify some of the symbols and how they are used or referred to.

Observations

My synopsis of this paper is the ADLIM action description language is used by the planning tool IMAGEbot to manipulate spacecraft information, though the type of data or information probably does not matter, and how ADLIM can convey data descriptions along with the manipulation. The planning tool IMAGEbot is briefly discussed as to its role in automating the information manipulation and the storage of vital meta-information that corresponds with the manipulation output. But the main focus of the paper details how the ADLIM language works in relation to similar products and keys on problem areas that arise when planning for information manipulation. One such problem area is how to emulate the tar (copy) function. The paper provides comparisons to other similar language products that do not handle this function well.

To demonstrate the problem areas, the author steps the reader through examples showing how ADLIM represents and reasons with actions. The examples used are very practical especially dealing with copy actions like tar to show how the action language uses frame effects, action goals, procedural constraints, and temporal projection to sense and manipulate data robustly and generally. The paper describes each of these properties culminating with the conclusion.

What really caught my attention was in the Introduction section describing how scripts used to manipulate data do not convey information about the data such as its inter-relationship among various pieces of data, its meta-information, and the tracking of the data. If ADLIM can truly convey this type of information along with the manipulation action it could be a very useful language tool indeed.

Comparison to EOS

With ADLIM the action language uses its various properties to include information about the data and the tracking of the data. With EOS, the science data for the Terra spacecraft depends on data type descriptor files called metadata that identifies and categorizes each specific data product used throughout the science data processing. The metadata file provides file origin, content description and layout, quality, condition, and how to decode, interpret, and process the data. To process any EOS science data type, the user must obtain the corresponding metadata file in order to know how to deal with the data. The paper indicates ADLIM can possibly provide similar data descriptions with the action language.

Data manipulation within EOS is primarily by software with scripts to automate the processing and to control where the data goes after each step. Any errors or changes to any data format or manipulation process may require a software change. This makes the EOS system very rigid when dealing with permanent changes, emergency changes, and special requests. The use of the ADLIM action language instead of software and scripts could alleviate this restriction.

One common area to both EOS and ADLIM is the use of relational databases. The author explained that relational databases are quite easy to model in a language like ADLIM since it is at heart a relational language; i.e., functions can be easily mapped to relations. Plus, a database is essential for storing data and a relational database provides a means to retain with data its inter-relationships with other data.

Conclusion

Of course, switching to ADLIM for existing spacecraft systems would be costly and time consuming. But the author states that the versatility and structure of ADLIM and the planner tool IMAGEbot would allow it to be integrated into existing 'legacy' systems without requiring changes to the systems because ADLIM can model the user interfaces or APIs of those systems. Future spacecraft systems in the pre-design phase have the option of selecting the best method for managing information and hopefully IMAGEbot and ADLIM will be among the selections.

Automatic Planning for Autonomous Spacecrafts Constellations

Eric Bornschlegl
European Space Agency
ESTEC, P.O Box 299
2200 AG Noordwijk ZH-NL
ebornsch@estec.esa.nl

Christophe Guettier
Axlog Ingénierie
19-21, rue du 8 mai 1945
94110 Arcueil, F
christophe.guettier@axlog.fr

Jean-Clair Poncet
Axlog Ingénierie
19-21, rue du 8 mai 1945
94110 Arcueil, F
jean-clair.poncet@axlog.fr

Abstract

Autonomous agents are a challenging concept for future unmanned operations. Previous space and aeronautic missions highlight the lack of on-board intelligence to increase the decision making capability and to efficiently react to unexpected situation or events. Many agent-based system approaches exhibit good properties, as demonstrated in flight during the Deep Space1 mission. This paper presents a multi-agent system framework with on-board planning, based on constraint solving models, applicable to a constellation of satellites

Introduction

A constellation of satellites (formation flying) will require more cost effective operation and significant increase in performances, pointing accuracy, reactivity and orbital control. It requires a global management of operations and interconnections, as well as an optimization of the utilization of communication links either between the ground and space segments or between satellites. This requires to deal at the same time with several related system functions layered in different level of granularity. For example, a global configuration and operation modes management of the constellation has to be compliant with the diverse local modes switching or state transitions, involving cooperative or reflexive policies.

The use of distributed functions over a spacecraft constellation (such as planetary observations) makes mission planning more complex. Today's technology enables us to safely process on-board each satellite, a certain number of functions traditionally performed on the ground (such as attitude & orbit, state transition and configurations, planning & scheduling). By joining distributed systems theory and emerging artificial intelligence results, recent researches carried out in space (Ber98; MP98) and aeronautics (Yav94) domains emphasized the benefit of using Multi Agent Systems (MAS) (HJ96). In practical examples, MAS appears as a constructive approach that can tackle underlying difficulties like uncertainty, completeness (HJ96) and many applicative requirements (Yav94) such as:

- *Affordability*: Modern systems can make use of components off-the-shelf which requires a better modularity of software and hardware.
- *Performances*: Autonomous systems have to execute missions under real-time constraints requiring powerful observations, rapidly analyzing input data and selecting the right alternative.
- *Flexibility*: Systems are expected to adapt their behavior as required by changes in environment. This may require the system to dynamically reconfigure. It must then have the ability to build a good representation of its environment.
- *Availability*: Systems must be able to handle and complete requests while reacting under real-time constraints to environment modifications.
- *Survivability*: Systems that evolve in an hostile environment must deal with possible failures. Thus they must use fault tolerant mechanisms and possess dynamic reconfiguration capabilities.
- *Safety*: Agent deployment must guarantee a safe behavior in respect to both other agents and human beings.

MAS allows the design of global intelligent behaviors modeled through symbolic and logical representations (HJ96; WJ94). For instance, it is possible to formally specify how several agents can collaborate to perform a global "goal oriented" mission or to perform dedicated specific actions using a limited set of both discrete and continuous resources (payloads, energies, processing elements, ...). The use of different forms of logic, symbolic reasoning and arithmetic inference provide various ways to model different problems. However benefit of these results is limited due to highly combinatorial explosion and complex collaborative behaviors. Thus decision making and operational capability is restricted if the implementation is only based on simple heuristics.

Those combinatorial problems have been widely investigated in the Constraint Programming (CP) community. Stemming from logic programming, integer and mathematical programming, Constraint Logic Programming (CLP) languages are recognized as powerful tools to

cope with difficult and large combinatorial problems (DHS90; CL96; GH99). Underlying models of the approaches presented below could be expressed using a CLP language based on the simplex algorithm (Col87) and other results from integer and linear programming (such as branch and cut (PR91)) but the solving method would be restricted to a linear existential part of the constraint language. In addition, memory space explosion prohibits the use of $0-1$ modeling techniques. In contrary, CLP offers a higher compositionality to express and solve complex NP-Hard problems by separating declarative and operational semantics defined on finite parts of R and N with all their classical operators. On the declarative side, those constraint languages enable us to model several related complex problems with a slight loss of completeness. On the operational side, dedicated search operators can be specified in order to design global search strategies (CL96). Furthermore, addressing task scheduling and multiple resource allocation problems, recent results emphasized the efficiency of those approaches (DHS90; VSD95; Jou95; VSD95; CL96; GH99). In this paper, we consider a spacecraft constellation as a multi-agent system. In order to maximize the final system efficiency, we investigate how planning under operational constraints can be modeled and related to the MAS behavior through a constraint programming approach. This necessitates considering at the same time several NP-hard sub-problems such as motion planning, collaboration constraints, observation and orbit allocation. The CLP properties offer a more compositional, generic and flexible way to separate model's expression from search strategy.

Background

Formation Flying Mission, (for example, a deep space or earth observation mission), can be characterized by a set of global goals to achieve with the delivery of mission data products. A goal is for example interferometry or stereovision from in-flight combined observations with multiple sensing resources (such as cameras distributed on several satellite platforms , ...).

Space mission

We will consider a spacecraft constellation which for an operator submits mission goals with timing constraints. The constellation is composed of a fixed number of spacecraft (four for instance). Spacecraft fly in formation, very close from each other (a few tens of meters). Each spacecraft is represented by a platform with Inter-Satellite Link (ISL) capability, and by a payload (synthetic aperture radar, spectrometer, ...).

In order to insure flexibility, availability and performances requirements, the constellation can be divided into sub formations that will simultaneously achieve distinct observations from different orbits. Spare satellites might also be needed. Thus separating and/or combining observation maneuvers within the formation are part of the global mission plan. Consequently, flexibility of the platforms depends on the MAS global & local planning capabilities that control their roles (operation and configuration in hard real-time, from predicted or unpredicted discrete events).

A Multi-agent approach

Global operation planning over the whole formation must always be safe for the mission, while local satellite control insures survivability. In our approach, when the local control task fails or an unexpected event occurs, an emergency procedure is triggered to secure the formation and a new plan is generated. It appears to be a suitable technique to dynamically combine and allocate various discrete event control locations levels and criticalities within a global planning.

Spacecraft constellation as a category of MAS We assume that the constellation can constitute a single formation flying or be split into several ones. The constellation is represented as a MAS, composed of a set of collaborative cognitive agents, each one associated with a spacecraft. At any time, only the leader spacecraft of a formation can generate a global mission plan while the others are considered as reactive agents. Thus one formation can be seen as a reactive centralized architecture. However, a set of formations has to satisfy collaboration constraints defined below, so that when the constellation is divided into several formations, the associated MAS behaves as an hybrid deliberative architecture (HJ96).

Action planning We assume that mission autonomy is mainly provided by the on-board system. We are considering in this section only actions and underlying constraints affected by the mission goals. Actions under consideration are discrete actuations for orbit manoeuvres, observation, separation and/or reconstitution of formations.

Collaborative Policy At the low level, to achieve an increased survivability, the collaborative behavior of the MAS framework includes emergency actions that place the constellation in a secured state. If this topic is not under the scope of this paper, it necessitates the computation of a new plan as the previous plan may be no longer feasible after an emergency has occured. At a higher level, a long term collaborative behavior must also be considered by the plan. It specifies how plans from separated formations are compliant. At a minimum, the collaborative policy states that every spacecraft of a given formation obey the leader, and that formations have to be assembled in a given orbit, in a specified time window.

Coordination constraints Chaining orbit changes and observations requires scheduling on-board activities in a compliant way with the formation plan. Thus, we have a local and a global level of coordination. In our model, we consider reconfiguration tasks and action execution supported by operational modes. Coordination constraints are solved at the planning level.

Model-based planning

In our approach, we consider the planning function as solving a set of constraint-based combinatorial problems expressed as MAS or operational models like collaborative policy, altitude or observations.

CP and model-based computing A Constraint Logic Programming language can be viewed as an extension of Logic Programming where unification is replaced with constraint satisfaction. Logical predicates can be constraints interpreted in a mathematical algebra (JL87; VSD95) which is over the finite domains in our context : $\{\mathcal{P}(R), +, -, *, >, =\}$. Such a language enables predicates composition through logical operators and quantifiers. This leads to a more understandable, compositional and modular problem representation.

Model based computing is a practical way to take advantage of those CLP properties (Jou95; AG98). The modeling method (Jou95; GH99) extracts and adapts invariant of each problem by a recursive decomposition until a tractable expression can be formalized. Those formal expressions involve mathematical variables and constrained predicates that represent respectively the problem combinatory and its invariants. So doing, the resulting global problem is represented with different related and heterogeneous models. Relations between models are conjunctions of constraints that maintain the global consistency by propagating local solutions between models. The distinction between the problem formulation and the solving facilities allows to find solutions for various goals automatically.

Model based computing & constellation planning In the following we decompose the planning problem in different sub-problems. Thus, the spacecraft position and velocity are decomposed through *path planning*, *altitude* and the *timing* models. Therefore by combining solutions with classical orbitography, it is possible to extract attitude, positioning and speed along the plan. In fact combination of numerical computation and symbolic reasoning is essential in autonomous systems (WN96). For instance, we still have to prepocess orbitography functions before solving the planning problem. Moreover, the planning also satisfies *collaborative* and *coordination* models.

Path planning using graphs

Each orbit is interpolated by a set of navigation points. Transfers are allowed between the different orbits in specific conditions defined on transition points. The very simplified orbit interpolation described above allows us to project orbits in an oriented 1-graph $G = [X, U]$, where the set of vertices X represents navigation points and the set of edges U represents orbit's arcs.

Event based navigation model

We define a time horizon $h \in N$ that bounds the maximum length of a plan in event time. We duplicate the graph according to h and event time to model trajectory evolution through time defining a set of graphs $\{G_0, G_1, ..., G_h\}$, each one connected to its immediate immediate neighbor in the event time by adding to each vertex an edge to its corresponding vertex at the next event time occurrence. The costs of additional edges are fixed and equal zero.

At each vertex x of the graph G, we associate the set $\omega(x)$ of connected edges and a function d: $\forall x \in X, d_x() : \{0, ..., h\} \rightarrow Q$ where $d_x(j)$ represents the operational time on vertex x in the event time j. If $d_x(j) = 0$ the formation is not on the vertex x at the event time j, otherwise the formation is on the vertex x at operational time $d_x(j)$ in event time j.
Identicaly each edge u of the graph G is associated to a function e_u: $\forall u \in U, e_u() : \{0, ..., h\} \rightarrow \{0, 1\}$ where each $e_u(j)$ represents the use of the edge u in the event time j. If $e_u(j) = 1$ the path uses the edge u in the event time j, otherwise the edge u is not used in the event time j. This model is solved by instantiating $d_x(j)$ and $e_u(j)$ for each time event j. The graph G^h is extended using additional edges and vertex in order to define a circuit between two points. This allows us to express paths constraint along the graph, as shown in the following models.

Path consistency model

On each vertex of the graph, the first Kirshoff law must be verified, stating that the sum of the incoming flow must equal the sum of the outgoing flow (constraint 1). A flow of capacity less or equal to 1 insures a path consistency along the graph edges. However, this well-known model (GM95) in Operation Research has to be extended to fit the event time representation. For each vertex $x \in X$ we respectively define sets $\omega^+(x)$, $\omega^-(x)$ of incoming and outgoing edges.

$$\forall x \in X, \sum_{u \in \omega^+(x)} \Big[\sum_{j=0}^{h-1} e_u(j)\Big] = \sum_{u \in \omega^-(x)} \Big[\sum_{j=0}^{h-1} e_u(j)\Big] \quad (1)$$

This constraint defines the global flow over the extended graph G^h and also states the connectivity of the path over the graph.

Transition Constraints

Now, the operational times have to be consistent with the flows by defining a supplementary constraint (2) for transition over edges in U:

$$\begin{aligned} &\forall\ u = \overrightarrow{xx'} \in U,\ \forall j \in \{0, ..., h-1\}, \\ &e_u(j) = (d_x(j) \neq 0) \wedge (d_{x'}(j+1) \neq 0) \end{aligned} \quad (2)$$

This constraint states that an edge $\overrightarrow{xx'}$ can belong to the path in the event time j if and only if the path goes through the vertex x at event time j and through the

vertex x' at event time $j+1$. Finally, constraint (3) specifies that obviously, a route cannot use more than one edge for each time event:

$$\forall j \in \{0, ..., h-1\}, \ \sum_{u \in U} e_u(j) \leq 1 \tag{3}$$

Altitude model

For each time event j and each spacecraft of the constellation we associate a discrete altitude slot as a global resource using a vector $h(j) \in \{0,1\}^n$ of $0-1$ variables. The constant n is the number of possible slots. At the time event j, a spacecraft will be located at the altitude slot k if the kth component is set to $h_k(j) = 1$. Thus, we impose $\forall j, \ \sum_{k=1}^{n} h_k(j) = 1$ to insure the spacecraft belongs to a unique altitude slot at any time event. Each slot represents an interval which corresponds to a minimal perigee and a maximal apogee such that two orbits belonging to two distinct slots cannot overlap . The model also gives the travelling duration $\rho_u(j) \in Q$ required to cross an edge u according to the initial and final altitudes at the time event j. Solving the altitude model necessitates finding the appropriated slot for each time event while satisfying the constraints defined hereafter. First, when the spacecraft does not change its altitude, the constraint (4) expresses the travel duration across an edge thanks to a linear function. Second, when the spacecraft altitude changes, the constraint (5) approximates the travel duration across an edge using a linear combination of both starting and destination durations.

$$\begin{aligned} &\forall j \in \{0, ..., h-1\} \ / \ h(j) = h(j+1) \Rightarrow \\ &\exists \overrightarrow{xx'} \in U / d_x(j) > 0 \ \wedge \ d_{x'}(j+1) > 0 \\ &\wedge \ \rho_{\overrightarrow{xx'}}(j) = h(j) \times \theta_{\overrightarrow{xx'}} \end{aligned} \tag{4}$$

$$\begin{aligned} &\forall j \in \{0, ..., h-1\} \ / \ h(j) \neq h(j+1) \Rightarrow \\ &\exists \overrightarrow{xx'} \in U \ / \ d_x(j) > 0 \ \wedge \ d_{x'}(j+1) > 0 \\ &\wedge \ \rho_{\overrightarrow{xx'}}(j) = \lambda_{\overrightarrow{xx'}} h(j+1) \times \theta_{\overrightarrow{xx'}} + \lambda'_{\overrightarrow{xx'}} h(j) \times \theta_{\overrightarrow{xx'}} \end{aligned} \tag{5}$$

In both constraints, the constant vector $\theta \in Q^n$ represents the time duration required to cross the edge for each possible altitude. The resulting time duration is given by the variable $\rho(j)$. The two constant scalars $\lambda, \lambda' \in Q \times Q$ allow to approximate the time duration for changing orbits. Those constants can be preprocessed using traditional orbitography. This model illustrates the principles of modeling a set of elements (edges between different orbital locations) by taking advantage from problem's regularities and symetries instead of dealing with explicit set of elements. Nevertheless additional edges or more complex constraints can be defined to model non-regular orbital behaviors.

Timing and Observability constraints

For a given event time j, the value of $d_x(j)$ depends on the operational time $d_{x'}(j-1)$ on the preceding vertex x' in the path and the necessary time to go from x' to x. Then, if $u = \overrightarrow{x'x} \in \omega^+(x)$ is the edge that go from x' to x in G, we can state that if this edge belongs to the path, then the operational time $d_x(j)$ is defined as $d_x(j) = \rho_{\overrightarrow{x'x}} + d_{x'}(t-1)$, otherwise $d_x(j) = 0$. By extending the preceding relation to the whole edges in $\omega^+(x)$, we define $d_x(j)$ using a maximum on $(\rho_u + d_{x'}(j-1)) \times e_u(j-1)$ in the preeceding event time for all $u \in \omega^+(x)$.

For an observation zone, we have to specify that the path must include the travel of this zone in a specific operational time window. Moreover, this zone may be observed from several different orbits (i.e. several vertices of the graph G), and possibly with discrete sequences of operational time windows. In the following, a vertex x of G corresponds to a possible observation of a location in the operational time interval $[t_{min}, t_{max}]$ if it belongs to a triplet $\delta = (x, t_{min}, t_{max})$. A discrete observation window of a specific location from various observation points can then be represented as a set Δ of δ involving the same vertices x: $\Delta = \{\delta = (x, t_{min}, t_{max}) \mid x \in X, \ (t_{min}, t_{max}) \in Q^2\}$

According to constraint (6), a necessary observation of a zone Δ requires that the path contains at least one travel onto vertex x / (x, t_{min}, t_{max}):

$$\begin{aligned} &\forall \Delta, \ \exists \delta = (x, t_{min}, t_{max}) \in \Delta, \\ &\exists j \in \{0, ..., h\} \ / \ (d_x(j) \geq t_{min}) \wedge (d_x(j) \leq t_{max}) \end{aligned} \tag{6}$$

Collaboration model

To maximize the use of the flying formation, according to the availability and performance requirements, the model allows us to divide the formation into several sub-formations . We assume that a plan is composed with a set of several separation steps between the constellation and formations and one assembling step at the end of the mission. The set of spacecrafts is denoted $\mathcal{S}$, while a formation is denoted as $\mathcal{F}$. A leader spacecraft of the constellation is denoted $S \in \mathcal{S}$. A spacecraft $s \in \mathcal{S}$ is separated (resp. assembled) at the event time j^s_{sep} (resp. j_{ass}). In the constraints exposed hereafter we extend our notation to introduce quantification on spacecrafts.

Separating and assembling spacecrafts

We consider a single separation date for each spacecraft along a plan horizon. Before separation, the spacecraft is controlled by a leader and thus follows the same route at the same altitude according to constraint(7):

$$\begin{aligned} &\forall s \in S, \forall \ j \in [0, \ldots, h] \ / \ j \leq j^s_{sep} \ \exists x \in X \ / \ d^S_x(j) > 0 \\ &\Rightarrow \ d^s_x(j) > 0 \ \wedge \ h^s(j) = h^S(j) \end{aligned} \tag{7}$$

In order to maximize safety between spacecrafts differents routes always corresponds to different altitude

slots. Thus we relate collaborative model to altitude to fulfill anti-colliding requirements:

$$\forall s \in \mathcal{S}, \forall\ j \in\ [0, \ldots, h],\ j > j^s_{sep} \Rightarrow h^s(j) \neq h^s(j) \quad (8)$$

Solving the collaboration problem requires us to instantiate the variable j^s_{sep} for each spacecraft. At least, with constraint (9), all the formations have to join in orbital rendez-vous defined by the variable j_{ass}:

$$\forall s \in \mathcal{F}, \forall s' \in \mathcal{F}',\ h^s(j_{ass}) = h^{s'}(j_{ass})$$

$$\wedge\ \forall\ x \in X,\ d^s_x(j_{ass}) = d^{s'}_x(j_{ass}) \quad (9)$$

Consistent behavior inside a formation

The collaborative behavior requires us to solve constraints inside the same formation. After a separation, a spacecraft of a formation remains at the same altitude slot, this leading to constraint (10):

$$\forall s \in S, \forall\ j \in\ [0, \ldots, h],\ j > j^s_{sep} \Rightarrow h^s(j) = h^s(j+1) \quad (10)$$

Moreover, if two spacecrafts belong to the same formation, they have the same separation date as well as the same route and altitude after a separation.

Coordination model

The planning function involves a coordination model for a given agent or for a whole formation. For each vertex of the graph and for each time event, a spacecraft is in a given operational mode. If the modes are different between two vertices of the same edge, then a reconfiguration task is required. The operational mode of a spacecraft $s \in \mathcal{S}$, at the time event $j \in [0, \ldots, h]$ for a given vertex $x \in X$ is denoted $M^s_x(j)$.

Local coordination

Along the plan, some of the modes are imposed by the vertex type, others have to be solved by the planner. In any case, the time taken to travel across the edge must be greater than the time taken to locally reconfigure a given spacecraft (constraint 11):

$$\forall \overrightarrow{xx'} \in U\ /\ e_{\overrightarrow{xx'}}(j) = 1\ \wedge\ M_{x'}(j+1) \neq M_x(j)$$

$$\Rightarrow\ d_{x'}(j+1) - d_x(j) \geq C(M_x(j), M_{x'}(j+1)) \quad (11)$$

The predicate $C(M, M')$ corresponds to the time taken to reconfigure the spacecraft between mode M and mode M'. It is statically computed and remains in the agent knowledge.

Global coordination

As separating and assembling actions are executed commonly we state that all the operational modes have to be equal to a specific mode during the separation. An analog constraint fit for the assembling action. Using this approach it is possible to represent complex configuration and compatibility problems. Even those are not within the scope of this paper.

Solving the planning problem

On the modeling baseline, many kinds of goals can be investigated. It is possible to search for a plan that satisfies all the model constraints, to complete an existing partial plan or to optimize an objective function corresponding to an applicable trade-off in several ways (for example insuring availability by maximizing the number of spare spacecraft or minimizing the overall completion time).

Automatic Solving Using CLP Language

To solve the goal, a global control strategy has to be designed. We use Concurrent Constraint operators over Finite Domains (CC(FD)) (VSD95) as well as domain heuristics. To insure the consistency between model solutions, and to reinforce the concurrency between the solving processes, CC(FD) offers the powerful control operators *Ask* & *Tell*. The satisfaction operator *Tell* states a constraint to the solver and the entailment operator *Ask* checks if a constraint is already satisfied. At the search process level, the *Tell* operator is used to exchange partial solutions between models through relations. When two partial solutions are not consistent, the system generates a backtrack event. The *Ask* operator is used for the synchronization of the global search. Finally using those operators, global solving strategies including domain heuristics are designed to control the composite global search over models. An experiment has been developed using the *Claire* language (CL96) and the Finite Domain *Eclair* library (GS99) based on CC(FD) operators. Activated from a simulation framework (running on a Ultra Sparc Workstation), this preliminary implementation can handle up to 6 spacecrafts.

Conclusion

The pertinence of using Constraint Model Based Programming for specifying and solving the complex problem of a Multi-Agent plan has been shown. We give a better alternative to heuristic-based behavior of traditional multi-agent planners and open a new way to tackle complex cooperative behaviors. Moreover, this work highlights the feasibility of the approach for solving spacecrafts formation plans, by taking advantage of a more complete specification and by allowing the concurrent use of different models. However, models have to be validated according to their level of granularity and their quality of representation. Continuation of this work should focus on search strategies, satisfying properties like anytime and real-time.

References

[AG98]L.Alenius, V.Gupta *Modeling an AERcam: a case study in modeling with concurrent constraint languages*, Proceedings of the CP'97 Workshop, Modeling and computation in the concurent constraint languages, Pisa 1998.

[Ber98]D.E.Bernard & al., *Design of the Remote Agent Experiment for Spacecraft Autonomy*, Proceedings of the 1998 IEEE Aerospace conference, Aspen CO., 1998.

[Col87]A.Colmerauer, *Opening the Prolog III universe*, , Bytes, August 1987.

[CL96]Y.Caseau, F.Laburthe, *Cumulative Scheduling with Task Intervals*, In Proc. of the Joint Int. Conference and Symposium on Logic Programming, M. Maher ed., the MIT Press 1996.

[DHS90]M.Dincbas, P.Van Hentenryck, H. Simonis, *Solving Large Combinatorial Problems in Logic Programming*, Journal of Logic Programming, Vol. 8, p.75-93, 1990.

[GH99]C.Guettier, J.F.Hermant, *A Constraint Based Model for High Performance Real Time Computing*, In Proc. of the Int. Conf. on Parallel and Distributed Computing Systems, Florida, 1999.

[GM95]M.Gondran, M.Minoux, *Graphes et algorithmes*, ed. Eyrolles, 1995.

[GS99]S.de Givry, P.Saveant, J.Jourdan, J.Mattioli *Eclair Reference Manual*, Tech. report TSI-99-905, Thomson-CSF Corporate Research Laboratory, Corbeville, France, 1999.

[HJ96]J.M.P.O'Hare, N.R.Jennings, *Foundations of Distributed Artificial Intelligence*, Sixth-Generation Computer Technology series, 1996.

[JL87]J.Jaffar, J-L Lassez, *Constraint Logic Programming*, In Proc. of the 14th ACM Symposium on Principles of Programming Languages, Munich, January 1987.

[Jou95]J.Jourdan, *Concurrence et coopération de modèles multiples dans les langages de contraintes CLP et CC* PhD thesis, Université Denis Diderot, Paris VII, 1995.

[MP98]N.Muscettola, P.Pandurang Nayak, B.Pell, B.C.Williams*Remote Agent: To boldly go where no AI system has gone before*, NASA Ames Research Center, 1998.

[PR91]M.Padberg and G.Rinaldi, *A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems*, SIAM Review, Vol. 33, No. 1, pp 90-100, March 1991.

[VSD95]P.Van Hentenryck, V.Saraswat, Y.Deville, *Design, Implementation and Evaluation of the Constraint Language CC(FD)*, In Constraint Programming: Basics and Trends, A. Podelski Ed., pp. 68-90, 1995.

[WN96]Brian C.Williams, P.Pandurang Nayak, Immobile Robots: AI in the New Millenium, *AI Magazine, 1996.*

[WJ94]M.Woodridge, N.R.Jennings, *Intelligent Agents: Theory and Practice*, Knowledge Engeneering Review, October 1994.

[Yav94]A.Yavnai. *Distributed Decentralized Architecture for Autonomous Cooperative Operation of Multiple Agent System*, in Proc. of the IEEE 1994 Symp. on Autonomous Underwater Vehicle Technology.

Commentary, regarding "Automatic Planning for Autonomous Spacecraft Constellations"

Mark Boddy
Honeywell Technology Center
MN 65-2200
3660 Technology Drive
Minneapolis, MN 55418
mark.boddy@honeywell.com

Summary of paper

In the paper "Automatic Planning for Autonomous Spacecraft Constellations," the authors discuss a specific approach to the problem of integrating relevant lower-level concerns (primarily continuous dynamics of various sorts) with higher-level planning functions for spacecraft constellations. Despite the title, the paper does not describe a planning system, discussing instead a specific proposal for the kinds of models on which planning should be done.

The main point of the paper is that using constraint-based models, in particular CLP, for the representation and solution of high-level autonomous control problems—"planning under operational constraints," in their words—has specific benefits for spacecraft constellations. In particular, this approach permits the integrated, explicit representation of low-level, continuous constraints derived from orbital dynamics (for example), mixed discrete/continous representation for various types of resources (power, data storage space, etc.), and discrete constraints representing specific goals, or the assignment of discrete domain elements (individual spacecraft, e.g.) to satisfy some need. Furthermore, the ability to integrate these different types of constraints, coupled with the ability to compose constraint problems, means that a local solution of the planning problem for one spacecraft is simply a subset of the overall solution. Finally, there are all the usual advantages claimed for CLP approaches (modelling generality, declarative semantics, separating search from problem statement, and so forth).

This work does not involve "planning" at all, in the classical AI sense of the term. The models constructed reflect constraints upon valid plans, but there is no search through the space of valid plans, rather an assumption that any solution of the CSP will be acceptable. I do not regard that as a problem, rather as a potential misunderstanding to be cleared up, and as a place where further work is needed. The main contribution I see in this paper is that it argues for the use of CLP methods within multi-agent systems, and provides some examples to validate the proposed approach.

The authors provide three examples of constraint classes. The first class covers the expression of the dynamics of individual spacecraft, specifically motion planning in an orbital environment, with timing and positional constraints applied to specific actions (observations). The second class of constraints discussed are for coordinating multiple spacecraft within a formation, for example in coordinated orbital manouvers. The third class, called a "coordination model" by the authors, covers changes in operational modes both for single spacecraft and synchronized across multiple spacecraft. This last class looks very much like a special case of a more general expression of resource usage and interdependent operational states.

Related issues

There are three areas of current work which intersect with this in interesting ways, illuminating or extending some aspect of the proposed approach.

Hybrid systems

The way in which I am most sympathetic to this work is that it has as a basic thesis the notion that hybrid constraint representations should form the basis for constructing, analyzing, and executing models of the operations of complex systems.

Necessary as abstraction is, there are many places in the real world where an abstraction that moves from continuous to discrete with increasing abstraction is not helpful. Of course, there are historical reasons, especially within AI, why there has been a bias in term of abstracting towards something like a propositional representation. Furthermore, these abstractions worked, sort of, for robots wandering down hallways, because the problems were stated such that all the relevant continuous detail could be buried in the lower level, with the occasional exception of duration. While it

does matter whether the robot can fit through a given door, or navigate successfully down a corridor, precise clearances are not important. Locations can frequently be abstracted to a graph of discrete locations.

For spacecraft, as in many other domains, these simplifications will cause problems. Setting a state variable corresponding to whether a spacecraft is "at" Jupiter or not is not very helpful. It would be much better to know the orbital elements, especially for planning out some sequence of operations (close enounters with natural satellites, for example). Similar arguments can be made for decision-making involving transition times (rotating an instrument platform), actions with their own continuous dynamics, or resource models involving material or energy flows.

In domains where continuous dynamics are important in this way, there can be an interesting inversion, in which the most sophisticated continuous models are actually employed at the highest level of abstraction. My imperfect understanding of mission planning for spacecraft is that the orbital models used in preflight mission planning are considerably better than any used for enroute autonomous navigation. The same inversion applies in other domains. In chemical manufacturing, for example, medium-range production planning (including planning the purchase of raw materials) usually involves the use of very detailed kinetic models of the reactions involved in chemical processes. These models are frequently tailored to individual pieces of equipment in a given plant. This level of planning abstracts away individual activities and operations (the discrete part of the problem), and concentrates on getting right the continous dynamics and resulting material balances. Once that has been done, the planning and scheduling of individual operations can be accomplished (within their own sets of continuous constraints). Finally, the individual activities are turned over to operations and executed using control models that are rarely more complex than is required to take a target setpoint (temperature, pressure, flowrate) and determine a gradient and the resulting control action.

Over the past few years, work on hybrid systems, particularly within the areas of automatic control and real-time systems, has grown and evolved to the point that it makes sense to talk about a "hybrid systems community." The results from this community have primarily been in the areas of synthesizing or analyzing controllers (or "reactive systems," depending on your background) for particular problem domains. While in no way belittling the importance or relevance of this work, I will at the same time claim that it is of little use for high-level decision-making, in particular the kinds of decisions generally made by or made using systems doing what we are used to calling planning and scheduling. The basic problem is that the explicit, relatively long-term predictive simulation models employed for planning and scheduling require different techniques than those used either for traditional control theory, which only requires a gradient (or at most a limited predictive horizon), or for system verification, in which the system's behavior can be verified in a forward simulation. The property required for effective planning and scheduling is something Tom Dean once (about 15 years ago) called reasoning about time "from the side:" the ability to view and manipulate an entire chronicle at once.

Planning for real-time systems

There are two requirements for planning for real-time systems where failures are possible. First, timing constraints must be taken into account within nominal plans. CLP approaches, among others, should have no problem with this. Second, there are timing constraints imposed on the system's response to failure. One shortcoming of the straight CLP approach as presented here is that the system generates a plan, which it assumes can be executed. In practice, the course of events is rarely that predictable. The answer proposed in this paper is essentially a "safing" behavior, where the constellation as a whole goes into a safe mode, waiting for a new plan to be generated.

As has been repeatedly observed, there are situations in which this form of fault detection and recovery will have consequences ranging from inconvenient to disastrous. One instance is orbit insertion (one of the Cassini scenarios for autonomous planning). Another arose recently on Galileo, where safing was temporarily disabled during the IO fly-by.

There are several possible approaches to this problem, most of which have been investigated previously, none of which is a clear favorite, or for which the last chapter has been written.

- High level planning with low-level reactive behaviors. This approach involves conventional classical planning on top of an abstract action representation. These actions are further decomposed within a reactive planning and execution architecture such as RAPS or TCA, assuming that both execution failures and the pixels-to-predicates problem can largely be handled at the lower level. This approach has the problems with abstraction discussed above.

- State-based planning with explicit contingencies. In these systems, plans consist of graphs (or tables) of states with timed transitions corresponding to either actions or other events. Correctness is defined in

terms of avoiding failure, and in a secondary sense in terms of eventually achieving a specified goal. The essentially Markov nature of these representations present difficulties for goal-directed behavior (as opposed to proving that a goal is reachable), and to the explicit management of resources over time. The state-based representation itself is potentially subject to combinatorial explosion and does not lend itself to the expression of continuous state parameters. All of these shortcomings are currently being addressed by various research groups, but I don't think any of them have been finally and generally resolved.

- Contingent "classical" "planning." The scare quotes around "classical" are because this category could also include systems like C-Buridan, which employs a probabilistic action semantics. Those around "planning" are because it is possible as well to build an explicitly contingent scheduler, as Bresina and others did for the APT, for example. The problem with this approach is the need to reason about (and generate a plan representing) all relevant contingencies.

- More complex architectures. Remote Agent, as far as I can tell, employs a complex combination of these techniques. HSTS constructs plans on an abstract model which still includes some continuous information. The executive takes the abstract actions in that plan and expands them, assuming that that expansion will not introduce infeasibilities into the plan (it does, sometimes). Finally, execution failures are detected and local responses to those failures are generated in Livingstone, which attempts to keep the system in a safe state until some more general recovery action can be planned for.

Something like the set of tradeoffs involved in RA will be required of any autonomous embedded system. The simpler architectures do not provide enough opportunity to exploit useful domain structure. Some questions raised by the approach proposed in this paper include whether the abstractions employed by RA are the right ones, and how easily RA can be adapted for coordinated multi-agent operations.

Multi-agent systems

The authors argue that constraint-based representations form a good basis for multi-agent systems. The system they actually describe involves one agent generating a plan for an entire formation, within the overall constellation. Coordination between formations is discussed in terms of a "collaborative policy," with little indication that there is any active coordination between formations. Well and good, and it is already clear in this situation why a hybrid constraint representation makes sense, and why the compositionality inherent in a CSP formulation is a good thing.

However, there are two ways in which I do not believe the authors have pushed this as far as they could and should. First, as discussed above, their "planning" isn't really planning. A more complex form of search over something more like a classical planning representation will add additional requirements and complexity, in ways that would make the current argument stronger. Second, the authors have taken a restrictive view of what it might mean to have a constellation of autonomous spacecraft. In particular, the level of autonomy provided to the individual spacecraft in a formation is quite limited. This is somewhat like describing your hand as a multi-agent system because you have five fingers, each equipped with their own sensors, actuators, and reflexes.

It would be more interesting, and consistent as an extension to the approach proposed here, to have the spacecraft negotiating cooperatively in the execution of some task, by exchanging commitments expressed as constraints, possibly within some global plan. In private communication, one of the authors has framed this more general view as a problem of distributed solution of the global CSP, i.e.,. coming up in a distributed framework with an equivalent answer to the one that would be found if the entire CSP was solved in one centralized model. This misses the point, to a large extent. The interesting questions for distributed systems involve situations in which, for reasons having to do with (for example) communication bandwidth or divisions of authority, individual agents are attempting to come to some kind of global solution that satisfies their own local constraints (possibly optimizes a local objective function), within some global set of constraints on behavior (related to a global plan, for example).

CLP and its limitations

One note of caution regarding what is proposed here: I do not, personally, believe that CLP solvers are ready for the kind of industrial-strength use that is required in a production environment involving broad application and extensive re-use and adaptation. There are two obstacles to be overcome before that happens. First, these systems must become sufficiently stable. CPLEX (an LP package now owned by Ilog) is a good example of a solver that is sufficiently mature to be generally useful in a production environment. While I expect some argument on the point, I do not believe that any more general CLP/CP system is, as of yet. I do believe that they eventually will be, though in what

form is still uncertain.

Second, these systems must be sufficiently *and appropriately* expressive. CPLEX satisfies this requirement, though in an odd way. There is a large community of people who are very, very good at rendering real problems in the sometimes very unnatural form of a linear program. This translation (and for that matter, the interpretation of the solver's output) is essentially a black art, but one for which a graduate curriculum has been developed.

There is no such curriculum broadly available for model translation and interpretation of solver results for more general CPs. Such a curriculum is even more necessary for CPs than for LPs. The range of modelling choices is considerably larger, rendering the choice of an appropriate model more difficult and more critical. That flexibility is exploitable in other ways, for example in the construction of combinations of solvers and modelling primitives designed for particular classes of applications (e.g., Ilog Schedule). Such systems are even less mature, and their application to real problems is still a black art. Despite claims to the contrary, applying such tools to a complex real-world problem still requires considerable sophistication in understanding both the underlying machinery and how to work around their expressive and computational limitations. This is the same set of difficulties on which "expert system shells" foundered, more than a decade ago.

None of this should be taking as detraction from the point of view that constraint-based representations are a fruitful avenue to explore. My qualification has to do with the assertions that these representations obviate the modelling problem, or that the current set of available tools are mature enough for general use. Nor am I asserting that the authors claim either of these things, though others certainly have.

What next?

This paper provides a useful first step in considering the application of CLP/CP representations to multi-agent systems in general and spacecraft constellations in particular. There is considerable additional work required, some of which it appears the authors are actively pursuing but did not have room to put in the paper. I look forward to further developments.

Automated Planning for Spacecraft and Mission Design

Ben Smith
Jet Propulsion Laboratory
California Institute of Technology
benjamin.d.smith@jpl.nasa.gov

George Stebbins
Jet Propulsion Laboratory
California Institute of Technology
george.stebbins@jpl.nasa.gov

Abstract

Mission design engineers identify a spacecraft design and mission plan that best achieves the mission objectives while staying within cost, mass, and operability constraints. It is often easiest to evaluate a spacecraft design in the context of a detailed mission plan. Generating plans by hand is labor-intensive. We present an AI planning system that automatically generates and evaluates mission plans for specified spacecraft designs. This system has been applied to design problems from a number of NASA missions.

Introduction

The job of mission design engineers is to identify a spacecraft design and mission plan that best achieves the mission objectives while staying within cost, mass, and operability constraints. We observe that it is often easier to evaluate a spacecraft and mission design in the context of activity plans for key mission scenarios. Just as a simulation allows designers to better understand how the design artifact would behave, a plan helps mission designers to understand how a specified spacecraft design will execute a given mission scenario. For example: How many observations will it take? What are the resource margins? How much slack time is there for contingencies?

We have developed an automated planning system that takes as input spacecraft parameters (e.g., spacecraft slew rates, battery capacity) and mission parameters (e.g., observation requests, frequency of communication passes, trajectory). The planner generates a mission activity plan that achieves the mission goals while obeying the constraints imposed by the given mission and spacecraft design (which are a function of the mission and spacecraft parameters).

This technology enables mission engineers to quickly evaluate several designs. Engineers can evaluate several candidate designs against a given mission scenario by generating plans for each design and automatically evaluating them against objective criteria. Engineers can also use this system for "what-if" evaluations. They can see how a given designs performs in the context of a mission scenario, and then modify the design or mission to improve performance. For example, a spacecraft may be limited to ten science images per orbit because of insufficient on-board data storage, even though there are opportunities for many more. The engineer increases the memory parameter and generates a new plan to see if the spacecraft can now take more science images.

This system has performed design evaluations for several NASA missions: the Solar Interferometry Mission (SIM), LightSAR, and Pluto-Kuiper Express. The remainder of this paper will describe the system in more detail, provide some example trade studies, and discuss the key scheduling issues and algorithms.

System Architecture

The system takes as input spacecraft and mission design parameters and a set of scenario goals. From these inputs an automated planner produces a plan of spacecraft activities that accomplish the scenario goals in a way that is consistent with the spacecraft and mission design. The next subsection describes this process in more detail. The resulting plan is then evaluated with respect to user-specified objective criteria. The overall architecture is shown in Figure .

Automated Planning

The core of this system is an automated planning and scheduling system. We used the Aspen [1] planner, which has a number of reasoning capabilities we find necessary for generating spacecraft mission plans. However, the architecture makes no assumptions about the planner, so one could easily substitute a different planning system.

An automated planner, such as Aspen, takes as input a set of goals and an initial state. It then derives a set of actions that will achieve the goals from the initial state. A domain model specifies the available actions, states, resources and constraints among them. For example, a "take-science-image" action may require that the spacecraft be in the low-vibration state, occur at least 30s after turning on the instrument, and requires 20Mb of on-board storage and 10W of power. These constraints are specified in a declarative language specific to the planning system. The above constraints would be specified in Aspen as shown in Figure 2.

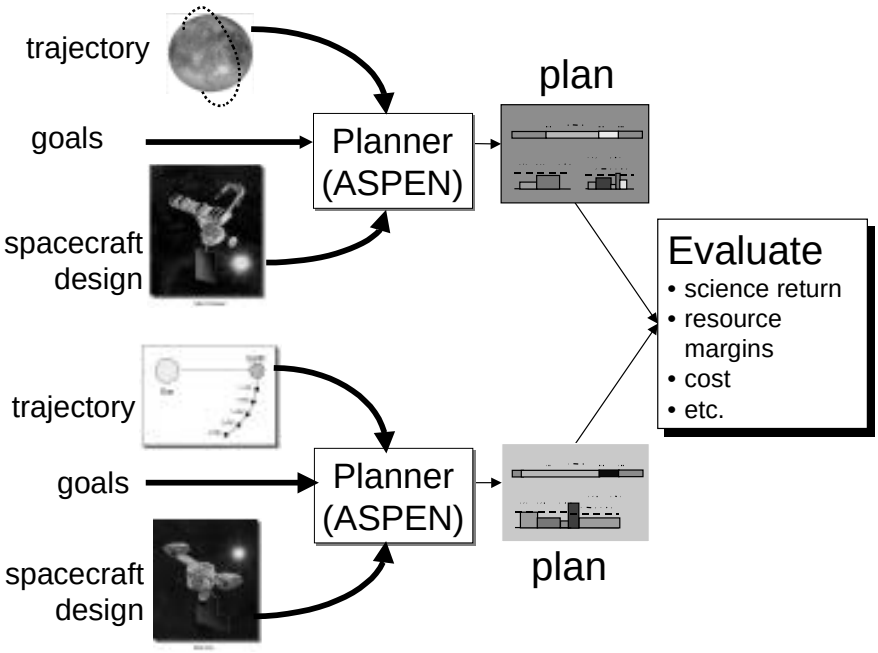


Figure 1: Architecture

Aspen generates a plan that achieves the goals from the initial state while obeying the constraints in the domain model. It does this by a combination of sub-goaling, goal-expansion, and conflict resolution. Sub-goaling achieves a desired state by identifying an activity that achieves the state and inserting it into the plan. Goal-expansion takes a high-level goal and expands it into a pre-defined set of sub-goals. Conflict resolution identifies constraint violations in the plan and resolves them. For example, the plan may contain more "take-science-image" activities than will fit onto the on-board storage. Aspen might resolve this conflict by removing some of the images or inserting a "downlink" activity that will free up more onboard storage. Aspen uses an algorithm called iterative repair [6] to perform conflict resolution.

The domain model specifies the constraints for a given spacecraft design. The goals specify the mission scenario. Each spacecraft design needs a different model, and each mission scenario needs a different set of goals. However, the spacecraft and mission designs are often similar in many ways. To avoid generating new models and goals for each design, we parameterize them with appropriate design variables. The user simply specifies values for each of the design variables, which results in an appropriate model and set of goals. This allows the designer to explore the parameter space quickly and easily. It also opens the possibility of searching that space automatically for an optimal design. This is an area for future research.

```
Activity TakeImage {
    int target-id;
    constraints=
        starts_after end_of turn-instr-on
            by [30, infinity];
    resources =
        storage use 20,  Mb
        power use 10,  Watts
};

Resource storage {
    type=depletable;
    capacity=100;  Mb
};

Resource power {
    type=non-depletable;
    capacity = 130;  Watts
};
```

Figure 2: Model Fragment

```
Activity TakeImage {
    int target-id;
    int PowerAmount, StorageAmount, InstrOnDelay;
    constraints=
        starts_after end_of turn-instr-on
            by [InstrOnDelay, infinity];
    resources =
        storage use StorageAmount,  Mb
        power use PowerAmount,  Watts
};
```

Figure 3: Parameterized Activity

Some typical spacecraft design parameters are resource capacities (battery, on-board data storage, fuel), operability constraints (how long does it take to warm-up the instrument, how much data storage does an image require), and hardware options (use cold-gas thrusters or a reaction wheel). These variables can all be expressed as parameters in the domain model.

The Aspen domain language allows parameters in constraints and activities. An example of a parameterized activity is shown in Figure 3. The parameters are in italics.

The user specifies values for the design variables, and the parameters in the model are set accordingly. The mission scenario is specified by a set of goals. Goals are a set of activities that must appear in the plan, with constraints on their start and end times, duration, and other parameters. The user specifies one set of goals for each mission scenario. The goals have parameters that can be design variables.

Plan Evaluation

Plans are evaluated with respect to user-specified evaluation criteria. A new evaluation function must be written for each criteria. Some typical evaluation criteria are resource margins, resource usage, and science return.

Applications

Pluto-Kuiper Express

The planning for mission design (PFMD) concepts were first applied to a trade study for the Pluto-Kuiper Express mission. The objective was to compare two spacecraft designs, one with a fixed instrument and one with a mobile instrument platform (scan platform). The fixed instrument requires the spacecraft to change attitude in order to acquire different targets, which can potentially restrict the data take opportunities depending on spacecraft slew rates and conflicts with attitude constraints imposed by other spacecraft activities. The scan platform allows the instrument to move independently of the spacecraft which could potentially result in higher science return, but is more expensive. Because the science return depends on how the science goals interact with the spacecraft operations constraints it is difficult to compute the relative science return without generating an activity plan. Plans were generated automatically for each design and the resulting plans evaluated with respect to cost and science return. Detailed results can be found in [5].

Space Interferometry Mission

The planning for mission design system supported an orbit trade study for the Space Interferometry Mission (SIM). The question was whether to use an inexpensive but highly constraining low-Earth orbit, or a more expensive but less constraining Earth-trailing orbit.

SIM will use a space-borne interferometer to take images of distant stars with much higher resolution than is possible with existing telescopes. One of the key scenarios in this mission is a "grid campaign" where the spacecraft images the entire celestial sphere over a period of about a month. To minimize the length of the grid campaign, and thereby maximize the science return, the images must be ordered to minimize the angular distance between adjacent targets. To avoid damage, the interferometer must not be pointed within a certain angular distance (the exclusion angle) of bright bodies in the solar system, such as the Sun, Earth, moon, Mars and Jupiter. Over time targets move in and out of exclusion angles relative to the spacecraft as determined by the spacecraft orbit and celestial mechanics. For a fixed trajectory, each target can therefore be imaged only during time windows when it is not in the exclusion angle. In general the more exclusion windows there are the longer the optimal tour becomes, but it is difficult to say how much longer without actually solving them.

We used the Aspen planner to generate a grid campaign for the Earth-trailing and low-Earth orbit cases, and for different exclusion angles. The objective was to determine whether Earth-trailing campaigns, which have fewer exclusion windows than Earth-orbit campaigns, were sufficiently faster to justify the more expensive orbit. The results, shown in Table I, supported the decision to use an Earth-trailing orbit. Targets is the total number of image targets in the campaign, scheduled is the number of targets that could be taken (some targets are never visible, or their widows overlap so that there is only enough time to take some of them). The plan duration is the total duration of the grid campaign as planned, and time/target is the plan duration divided by the number of scheduled targets.

LightSAR

The LightSAR mission is an Earth-orbiting satellite with a synthetic aperture radar (SAR). The SAR footprint is a rectangular swath over the Earth's surface. The objective is to image specified regions of the Earth (say Greenland) within certain time windows (e.g., March to June). To image a region, one must select a set of rectangular swaths that cover the region. The available swaths and the time at which each swath can

Trajectory	targets	scheduled	plan duration	time/target
Earth trail	1164	1141	25.23 days	31.84 min
Earth orbit	1164	987	26.78 days	39.07 min

Table 1: SIM Plan Evaluation Summary

be taken depend on the spacecraft orbit and the SAR beam angle (there are several adjacent beams with incidence angles separated by a few degrees). For example, one might be able to image a given strip of Greenland from 7:00 am to 7:05 am on Beam 5, or from 7:15 am to 7:23am on Beam 3. Each swath results in many megabytes of data, which reside on the on-board recorder until it can be downlinked. The planning problem is to select beams that cover the desired regions within the specified time windows without exceeding the on-board storage.

The design questions are how the on-board storage constraints and downlink opportunities impact the science return. The storage capacity and downlink opportunities limit the number of swaths per orbit, and thus the total science return, but in a manner that is hard to predict. By generating plans for various storage capacities, available downlink stations, and goal distributions we can understand that relationship and pick values that provide the best balance between science return and cost.

These plans take weeks to generate by hand, but only minutes with an automated planner. Planning technology makes it feasible to explore relationships like this and thereby improve the design in ways that would otherwise not be possible. We have generated plans for the baseline design, and are beginning to explore this relationship in more detail.

Planning Challenges

The planning problems described above are uniquely challenging. The overall problem has many constraints that require a powerful planning system like Aspen. For example, SIM must also consider battery and power constraints and the need to periodically "decondition" the reaction wheel (bleed off excess momentum); LightSAR has interferometry pairs, which are pairs of SAR images that must be take exactly 10 days apart, and a number of miscellaneous constraints on the instrument, data recorder, and downlink activities. The core problem is often a combinatorial optimization problem. For example, the core SIM problem is an instance of the traveling salesman problem with time windows [4], and the LightSAR problem is a kind of constrained bin-packing problem [2]. These often require specialized algorithms to solve effectively (e.g., [2,4]). General-purpose planners often solve them poorly or slowly because they do not have these specialized data structures or algorithms.

We addressed this problem by mapping an abstraction of the planning problem to the core combinatorial optimization problem, and solving that core problem with a special-purpose solver. The additional constraints are expressed in the core problem as a general feasibility constraint into which the specialized solver has no visibility. The solution to this problem then guides the planner in solving the overall planning problem. In our experience this approach yields high-quality solutions within reasonable computational bounds (a few minutes to a few hours, depending on the problem).

Conclusion

It is often easier to evaluate a spacecraft design in the context of a mission scenario. We have developed a planning system that automatically generates mission plans for specified designs. The system can generate scenario plans in minutes that would take designers weeks to generate by hand. This allows designers to more quickly explore the design space and to see interactions between spacecraft design and operations that would be difficult to identify by other means.

This system has been applied to design problems for a number of spacecraft missions and has met with enthusiasm from the mission design engineers.

Acknowledgements

This paper describes work performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract from the National Aeronautics and Space Administration.

References

1. Fukunaga, A.; Rabideau, G.; Chien, S.; and Yan, D. "Toward an Application Framework for Automated Planning and Scheduling," in *Proceedings of the 1997 International Symposium on Artificial Intelligence, Robotics, and Automation for Space*, Tokyo, Japan (1997).
2. Garey and Johnson. Computers and Intractability: A guide to the theory of NP-completeness. W.H. Freeman and co., New York (1979)
3. Rabideau, Knight, Chien, Fukunaga, and Govindjee, "Iterative Repair Planning for Spacecraft Operations in the ASPEN System" in *Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation for Space (ISAIRAS)*, Noordjwijk, The Netherlands, June 1999.
4. Solomon, M. "Algorithms for the vehicle routing and scheduling problems with time window constraints." Operations Research **35**(2) (March-April, 1987), 254-265.

5. Sherwood et al., "Design for X (DFX): Operations Characteristic Spacecraft Design Analysis Tool" in *Proceedings of the First NASA Planning and Scheduling Workshop*, Oxnard, CA, 1997.

6. Zweben, M.; Daun, B.; Davis, E.; and Deale M., "Scheduling and Rescheduling with Iterative Repair," in *Intelligent Scheduling*, Morgan Kaufman, San Francisco (1994).

Comments on
Automated Planning for Spacecraft and Mission Design

Peter Norvig

NASA Ames Research Center
Moffett Field California 94035
Pnorvig@arc.nasa.gov

Abstract

The paper *Automated Planning for Spacecraft and Mission Design* by Smith and Stebbins presents the idea of using automated scheduling in the design phase rather than the operations phase of a spacecraft mission. This raises a number of interesting questions.

Introduction

The use of automated planning and scheduling tools for spacecraft has traditionally been restricted to the operations phase: the flight operations team determines a set of goals for the spacecraft and use a planner to generate a plan to achieve the goals. The Smith and Stebbins paper presents the idea of using a planner in the design phase to evaluate potential spacecraft designs. This paper comments on the proposal, in terms of the nature of the simulation environment in which plans are evaluated, the choices for design optimization techniques, and the very need for mission plans.

Simulation Environment

In their introduction, the authors say how simulations and plans are both used to gain information about a spacecraft design. Clearly, one is not much good without the other: only with a high-fidelity simulation of a realistic plan can we gain information. If we can achieve certain guarantees about the results of simulations, then simulation can become a more integral part of the design process, as proposed, for example, in NASA's Intelligent Synthesis Environment (ISE) program. Without the guarantees, simulation can only be a heuristic aid.

Note that the simulation may miss important interactions. For example, it may consider only the topological connections of spacecraft components, and thus fail to point out flaws resulting from thermal or electrical interference between two components that are placed too close together. The designer needs to know what sort of interactions the simulation will catch and what sorts it will not.

A simulation run that verifies nominal operation is useful, but is not sufficient for verifying a spacecraft design. We want our spacecraft to be robust under failure, and that requires simulations of low-probability failure events. One approach is to explicitly introduce hardware failures of various kinds into the simulation of a mission, and report the results over a suite of failure conditions. A more ambitious approach is to have a model of failure probabilities and run the simulation with the possibility of failure at every time step. The accuracy of this approach can be improved by using likelihood weighting to artificially over-sample the low-probability failures.

Optimization

One exciting possibility mentioned in the paper is to use the planner and simulation environment to guide a search through the space of spacecraft designs in an attempt to optimize the design. This raises the question of what communication is allowed between the planner, simulator, and optimizer modules.

It is certainly possible to do optimization without any communication between modules other than their advertised inputs and outputs. Pick a point in the space of design parameters, evaluate it by running the simulator on one or more missions, and measure the resulting performance. Repeat on other design points (guided by the performance measure of previous points) until a sufficiently high-scoring design is found.

However, it may be possible to do much better if the modules are allowed to communicate more deeply. For example, suppose the planner is forced to make a decision because of a resource limitation imposed by a design parameter (for example, insufficient power to perform a maneuver). The planner could record this limitation, and suggest to the optimizer that a future trial be attempted in which the power parameter is set just high enough to allow the maneuver. Alternatively, if the planner is capable of representing conditional plans, it could do so at this point, with a condition based on the designed power level. Equivalently, the simulation could be split into two paths with different power level values, and both paths

could be followed. One would need to take care in introducing conditionals or splits, so as not to end up with an exponential run time.

Another question is what kind of optimization technique to use. There are a wide variety of options: hill-climbing, simulated annealing, genetic algorithms, tabu search, linear programming, etc. Which is best? Does the ability of the Aspen planner to introduce special-purpose combinatorial solvers come into play in the choice?

Who Needs Mission Plans?

If a spacecraft has been designed and verified using this approach of simulation and planning, we end up with a different view of what the spacecraft does. Rather than looking at the spacecraft as being capable of executing the set of individual commands that are determined by the flight operations team to achieve the mission plan, we can begin to look at it as capable of achieving mission goals, within the design space parameters. This opens the way for an operations phase that matches the design phase in the way automated planners are used to command the spacecraft.

This also raises the question of whether the planner should be located onboard or on the ground. If an onboard planner has the same inputs as a ground-based planner, then uplinking a goal rather than a sequence of commands can be seen as a form of data compression: the goal is a compression of the possibly large conditional plan that the goal generates. If an onboard planner has access to additional or more up-to-date state information that the ground-based planner does not have, then we have a situation where the plan that would be generated and executed onboard is different from what the ground operators anticipated. For this to be acceptable would require great confidence in the level of testing and analysis done in the design phase.

There is a compromise between onboard and ground-based planning, in which a certain level of control is allocated to onboard processes – a kind of heightened safe mode in which the spacecraft can recover from a variety of problems – and the remainder is determined on the ground. This approach too would depend on thorough testing and analysis, starting in the design phase as Smith and Stebbins envision.

Forecasting Telecommunications Support Boundaries for Satellite Constellations Using Deterministic Scheduling

David Joesting, CSOC/Honeywell
Gus Larsson, CSOC/Honeywell
7515 Mission Drive
Lanham, MD 20706
E-mail: david.joesting@csoconline.com
gus.larsson@csoconline.com

Abstract

The NPAS is an advanced mission planning tool set utilized by CSOC personnel to forecast space and ground network loading in NASA's telecommunications commitment process for new LEO/NEO customers. This is accomplished by maintaining baseline network user telecommunication models, incorporating new customer support requirements, and generating near operationally valid planning schedules. Subsequent analyses are conducted to assure knowledge of support variations due to changes in customer requirements, available network resources, and inter-satellite orbital phasing. Recent studies have involved constellations of "inexpensive" satellites that collectively gather patterns of information through their planned dispersion. Results to date have been favorable with some recommendations for improvement in storage capacity made. Expectations are for an increase in multi-satellite proposals for both scientific investigation and commercial exploration.

Overview

The Network Planning and Analysis System (NPAS) is a National Aeronautics and Space Administration (NASA) developed resource used in advanced mission planning which forecasts upper and lower limits of expected low earth orbit (LEO) or near earth orbit (NEO) satellite telecommunications tracking support through sensitivity variations in deterministic resource scheduling. Collectively LEO and NEO orbits at their closest approach to the Earth's surface vary from several hundred miles to several Earth diameters in distance.

As a part of the Space Operations Management Office (SOMO) Consolidated Space Operations Contract (CSOC), the NPAS team's evaluations are an integral part of the commitment process for accepting new customers on both NASA's Ground Network (GN) antennas and Space Network (SN) relay antennas through aperture and data flow assessments. Incorporating new telecommunications requirements supplied by the requesting customer into baseline models of committed/potential customers, the team constructs composite simulation models that reflect the expected load on a given network for the support phases in question. Individual customer supports can be defined as multiple services and, if necessary, broken down to the satellite instrument level to allow a complete space-to-ground assessment. Evaluations may cover multiple phases of customer support: launch and early operations, nominal operations, special in-orbit maneuvers, end-of-life, etc., as requested by the customer.

Other than antenna loading/aperture availability, additional constraints are considered as necessary. These include data flow and latency requirements that are a function of ground site data traffic. To reflect the reality of NASA schedule generation, establishment of a new project's priority by the appropriate NASA resource management is often one of the first steps in any analysis. Based on the complexity of the support requirements, this may include a priority scheme that is multi-leveled.

Resultant assessments are used by the appropriate level of NASA/CSOC management in reviewing and approving the Project Service Level Agreement (PSLA) which documents the agreed level of support over the service life of the customer.

Methodology

Due to the number of unknowns, completing a meaningful advanced mission planning evaluation is seldom straightforward. The performing analyst typically must often read beyond the supplied telecommunications requirements and ask for more detailed information concerning support duration ranges, timing or patterns of support, antenna or site priorities, radio frequency incompatibilities, etc. A formal analysis is then performed using analytic simulation models that assist in defining expected bounds of spacecraft tracking support for assumed network configurations. Inadequate network resources, particularly for constrained sets of ground

antennas, may necessitate recommendations to search for alternatives, particularly commercial providers, or modification of the required support.

The team's collective experience has shown that regardless of the care taken in planning a network, unforeseeable events or factors will ultimately affect the makeup of the network or modify the currently projected users' requirements. Any attempt to forecast more than eight years for the NASA Space Network or five years for the Ground Network with any certainty are efforts in futility. Nevertheless, planning in distant timeframes should be done by applying the range of possible network resources against extremes in the user spacecraft population so as to identify support boundary situations.

In many cases, the object of a planning analysis is first to define a minimal set of supporting antennas and their earth based location (either on the ground or in geostationary orbit.) The second step is to identify the maximum set of antennas that may be required under a worst-case scenario. Worst-case here can mean several things based on the customer's requirements, but most typically reflects a lack of available resources on the requested primary support antennas due to commitments to higher priority users.

The shortcomings of forecasting the future must be appreciated and reflected in analyses by incorporating possible tracking support bounds as appropriate. For example, given any set of missions with uncontrolled orbits, over time their inter-spacecraft phasing will drift to unknown deviations. The importance of considering this orbital variability, although not linear, will vary directly with the number of supported spacecraft but inversely to the number of the supporting antenna apertures.

For any customer phase to be analyzed, it is important to be able to model each user satellite's telecommunication science/telemetry, command, and tracking support requirements to a degree that allows generation of a near operationally valid planning schedule. In addition, a certain level of support that is close to optimal is highly desirable, although due to total uncertainties it would be an unnecessary overkill if excessive resources are required to attain more than a three to five percent variation from that value.

Baseline requirements and resource models need to be maintained for rapid responses. Experience has shown that customer requirements sometimes change unexpectedly based on either changing engineering needs or science goals. Projects are slow to update written requirements, thus direct contact with knowledgeable user project team members on a periodic basis is important for accurate baseline models. In keeping with the possibility of future changes in support, network sensitivity to possible requirement changes should be evaluated for specific support levels, if not for the customer, then for management insight.

Modeling Parameters and Constraints

User requirements are translated into NPAS model parameters with the intent of identifying constraint situations between network resource users and allowing priority considerations (if any) to resolve such conflicts. Requirements for a single spacecraft may themselves be quite extensive when all tracking, telemetry and control (TT&C) support constraints are considered. The NPAS can model user support down to the individual on-board instrument level and can accurately simulate all up and down link dataflows. Most constraints revolve around the need to satisfy the science and telemetry needs of a spacecraft over any given interval, such as a day. Many spacecraft have the capability to perform command uploading at the same time (and for same or shorter duration) as telemetry downloading. The NPAS model supports complex service assignments of almost any configuration through the use of prototype events (Levine and Joesting 1997).

With regard to individual TT&C supports, many spacecraft have some flexibility in their support duration. For example, they may allow supports as small as 8 minutes but would prefer supports that are 15 minutes long. If shorter supports are all that is available, then the number and proximity of such supports would be higher and closer together, respectively, in order to assure the same total amount of dataflow. Other users are not as flexible in their support duration, and as such, may find larger variations in available network support under different situations.

Other modeling parameters that may be defined are quite extensive. This includes minimum separation requirements between services of the same or different spacecraft, special sunlight or darkness requirements on the spacecraft of its Earth sub-point, user defined ground or political boundaries, spacecraft antenna pointing limitations, and mutual interference between spacecraft.

Special Forecasting Considerations

One important uncertainty to consider in any customer feasibility assessment is the orbital phasing differences

between users of the same network resources. Unless a project controls its satellite through powered maneuvers on a regular basis, LEO/NEO orbits will drift over time due to many factors such as solar radiation pressure, atmospheric drag, etc., such that their actual position cannot be estimated more than a month or two into the future. Such changes in phasing between any two, or more, users can affect their ability to be supported on the same resource. Analyses have shown that this effect is more significant for ground based antennas than for space based relay systems due to the shorter view periods and lack of overlapping coverage; however, for low priority SN users, this can still affect the expected bounds of support.

The CSOC team handles the coverage variability uncertainty by using a Monte Carlo simulation with random variations applied to those orbital parameters that are most likely to change. These orbital parameters include the mean anomaly, the argument of perigee, and the right ascension of the ascending node. Each mission is analyzed to determine those parameters that are most likely to vary, and appropriate bounds are established for random variation in the simulation. The total number of trials in any individual Monte Carlo simulation is a function of the number of varied orbital parameters. However, at a minimum, one hundred deterministic planning schedules normally are generated for each phase model in order to establish the range of possible schedule satisfaction and patterns of support sensitivity to the varied parameters. Support commitment analyses should include consequences of worst-case phasing situations and occurrence timing among network users.

The key to generating "valid planning profiles" is a set of scheduling functions that closely match the operational characteristics of the user community. Classic forecasting tools have used modified greedy algorithms to approximate the network load. This does not in itself guarantee a near operationally valid schedule for any user. For example, regardless of network reliability, some users actively manage their support schedules, anticipating a loss of service on any subsequent support. For those users who actively manage the science download of their spacecraft's on-board recorder, it is often the case that they do not wish to have the recorder exceed more than 50% of its capacity at any given time.

The NPAS team has built an integrated collection of scheduling algorithms using heuristics for optimization to allow complex, meaningful planning schedules to be generated for user spacecraft as required, down to the individual instrument level if necessary. Our methods do not guarantee an optimal schedule as our goal is to forecast for each variation modeled a representative support profile that is within 3% of an operational schedule. Over time the tools have produced spacecraft planning schedules that compare favorably to realized support (Stern, Levine, and Pitt 1994).

Due to user community characteristic differences in radio frequency, data format, etc., on NASA's GN, the amount of effective utilization that can be forecast on these resources is significantly less that what the Air Force can achieve on its similar capability RTS Network. A maximum utilization reaching only about 50% (including antenna turn-around time) is all that can be committed to for any given NASA X-/S-band GN antenna due this lack of uniformity. Of course, more load can be added to the sites during the actual schedule period when that time comes in the future. But to analyze a user's requirements at a medium to low priority one or more years in advance requires the effective utilization not to be exceeded if those users are to actually receive their stated support.

Actual loading on the SN is limited by the inherent problem of scheduling and resolving conflicts when users generate their schedules on their own and submit them for inclusion on a user-by-user priority basis. For the resultant actual schedule, conflict resolution is in many cases a manual process with tradeoffs between users made in an unpredictable fashion. The NPAS algorithms have are designed to simulate the individual user's initial requests. In addition when requirements are not met due to priority conflicts, special automatic conflict resolution/priority override options are available to assure that at least minimum levels of support are met. The need to model these overrides is seldom obvious to the modeling analyst at first and typically requires an "iterative build" for any particular new customer's phase model. Not only do schedules generated by NPAS have the ability to accurately forecast network loading, comparisons of individual NPAS "plans" to actual SN schedules have shown similar patterns of support (Simons and Larsson 1994). It must be noted that it is impossible to forecast the actual schedule of any user, such as the Hubble Space Telescope, without knowing the actual targets of observation/science requirements over the period in question. These are unknown even to the project more than several weeks in advance.

Satellite Constellation Studies

Recent studies have been devoted to forecasting the load incurred by constellations of numerous "inexpensive" satellites. These types of projects involve deploying a large number of spacecraft that each consist of minimal

hardware and thus limits how much data can be stored on-board. Such a scenario normally means that data will need to be downloaded to a receiving antenna almost whenever one is in range of a particular spacecraft. This can be difficult, however, when the number of satellites in the constellation is sufficiently large as compared to available antenna resources. The objective is to produce a schedule such that minimizes the data loss experienced over all spacecraft in the constellation.

One such proposed constellation is the Nanosat constellation, which can consist of over 100 identical spacecraft, each weighing 10 kg, deployed into highly elliptical orbits around Earth. The apogees range from 12 to 60 Earth radii, and the perigee of each is 3 Earth radii. This results in orbital periods ranging in length from 1 day for the lowest altitude flyer to over 10 days for the spacecraft with the highest apogee. Contact with ground stations can occur only when the spacecraft is within 5 Earth radii of the station.

In the study, each of the spacecraft in the Nanosat constellation was modeled as having 864 megabits of on-board memory into which data is stored at a rate of 1 kilobit per second. This of course means that the recorder will be filled to capacity after 14400 minutes (10 days) if data cannot be downloaded at some interim point. For the outermost flyers, whose orbital periods exceeded 14400 minutes, this was indeed the case, and for those spacecraft we could not avoid losing some amount of data in each orbit.

However, through the use of NPAS playback requests the analysts were able to ensure the full requirements satisfaction of those spacecraft whose fulfillment was not a geometric impossibility. For each spacecraft in the constellation, one generic playback request was created to manage the filling and dumping of the on-board recorder. The requests were prioritized to allow spacecraft with higher altitudes to schedule before those with lower altitudes could.

NPAS playback schedule requests assisted here both in the realistic modeling of actual user equipment but also in managing optimal resource usage for planning purposes. The result was that the data loss was minimized for those spacecraft whose recorder capacity was exceeded between successive station views; the remaining 96 spacecraft in the constellation, through efficient station and spacecraft resource management, were able download 100% of their recorded data. Recommendations to the project were to increase the on-board data storage capacity of the few outer satellites if 100% of science data is required.

A second recent assessment request involved support of "clusters" of identical SN users in the 2001 timeframe. Using the multiple access return (MAR) resources of the SN Tracking Data Relay Satellite System (TDRSS), multiple sets of 5 users each desired to receive from 100 to 50 percent continuous coverage based on their type of operation. Although the TDRSS contains 5 MAR services per satellite (for each of 4 nominal first generation TDRSS satellites in operation), an additional 14 shared MAR services were proposed. Each are currently limited to 5 but can actually support 19 MAR services. This "shared augmentation" of services would be a fairly simple modification of the existing ground control equipment.

Four sets of these clusters of users would reside in a LEO nominally the same as a low inclination Space Shuttle orbit. 2 sets would be 50% users and 2 sets would be 100% MAR users. One set in each group would be "closely" located and one set in each group dispersed in orbit around the Earth. In addition, another 2 sets of 5 each would be dispersed on the ground. These users would have only real-time observation and transmit capability, thus minimizing cost.

This analysis was modeled in a straightforward manner using the NPAS scheduling option that maximizes requested support (also minimizes hand-over.) In addition, the supplemental shared 14 MAR resources were constrained using a special network concurrent shared resources option. Results were very favorable with all users receiving 98% or more of their stated support.

Conflict Explanation Utility

One recent addition to the NPAS toolbox is the Conflict Explanation Utility, referred to simply as the Explainer. It was designed primarily to assist NPAS modeling analysts determine how exactly a new set of requirements affects the performance of existing elements in a model. This can be particularly helpful when the introduction of new requirements has an indirect effect on a seemingly unrelated requirement that already exists in the model.

In its most basic form of operation, the Explainer is designed to assist NPAS analysts determine if a particular schedule request attained its maximum geometric satisfaction. If it happens that the geometric maximum is not reached by the request, the utility helps to determine what may be hindering the scheduling of the request. Many times the schedule degradation is due to higher priority requests that consume resources required by the analyst’s input request. Sometimes, however, the poor performance is due to competing requirements within the

same request (for example, restricting the spacecraft to schedule at only one station, but that spacecraft never has available coverage at the station). Lastly, the utility also alerts the user to simple conditions which appear to affect the scheduling process, for example, a minimum support time that is too large, or a requested service or antenna type may not be available at a requested station, and so on.

The utility is most helpful when new requirements are added to an existing model. This occurs often and it is not always clear why a new load on one resource affects the satisfaction of a seemingly unrelated requirement, for instance, a mission that is of lower priority but in a different orbit and that requests a different resource. This utility is designed to unravel the chain reaction that occurs when the scheduling of one request bumps another to a different resource, which in turn bumps another request to a still different resource, and so on. For example, it could easily happen that the introduction of a new load at high priority on the TDRSS station at location 174 degrees West longitude affects a medium priority request seeking to schedule a TDRSS station at location 41 degrees West longitude. It can be a time-consuming effort for an analyst to trace back through the scheduling process to determine why this seemingly unrelated scheduling task is affected in this situation. The Explainer simplifies this process by automating the conflict backtracking so that the NPAS analyst can report to the customer with confidence the cause-and-effect relationship seen.

Furthermore, the Explainer can optionally suggest how a request might be modeled differently to better utilize existing resources. This targeted request is not necessarily the request initially input by the analyst (the request that originally could not meet its satisfaction) nor the request for the new set of requirements: it could easily be one of the intermediate requests that played a part in the chain.

An NPAS analyst invokes the Explainer once a schedule run is made and a poorly performing schedule request is identified. Since the scheduling of this input request may have been affected by higher priority requests, the Explainer reads the modeling information pertaining to both the input schedule request and all higher priority requests to build lists of applicable constraints. An example of such a constraint would be if this request was intended to schedule only when the spacecraft was in-sunlight. Then the geometric visibility data for all these requests is processed by the Explainer and schedules with full tolerance (that is, all valid scheduling choices are included in the schedule, even if this results in far more events than desired) are created. Note that “optimal” events are identified during this process, and these are used to determine what the geometric maximum satisfaction for each schedule request should be. The process to this point is what is called the “Geometric State” construction; it describes what could be scheduled under ideal circumstances and also includes a list of alternate selections.

Following this, the “Schedule State” is determined. This begins when all requests of higher priority than the input request, in addition to the input request itself, are fed into the scheduling engine. As events are scheduled, the reason why a particular event has been chosen over another is noted and stored. For a higher-priority event, the reason would most likely be due to normal intra-requirement constraints such as scheduling only when the spacecraft is in-sunlight, selecting the longest pass, etc. If a scheduled event happens not to be one of the optimal events identified during the Geometric State construction, then the system tries to determine whether this is due to constraints imposed by the request upon itself (for example, a minimum separation from an event it previously scheduled) or if it is due to a higher priority request consuming the resources required by the optimal request. In the latter case, a “lookup” is performed to determine what higher priority request events are directly conflicting with the optimal event for the current request. Following this, one link of a dependency chain is built linking the event scheduled by the current request to that scheduled by the higher-priority request, for each interfering higher priority event. Any other reasons why the current event was chosen (for example, spacecraft is in-sunlight, the station was the most desirable) are still noted for the event, but the dependency chain is another part of the explanation why this particular event was chosen.

The Schedule State therefore is the result of the process in which the cause-and-effect relationships are determined for events of all requests that can affect the NPAS analyst’s input request. Once this has been completed, The Explainer can begin to determine why the analyst’s input request did not attain its maximum satisfaction. There are two cases here: one in which a non-optimal event was scheduled, and one in which no event could be scheduled in a given schedule period. In the former case, the descriptions associated with each event node are simply reported to the user. In the latter, the Explainer follows all links back through the dependency chain for each possible event in the period. (A schedule period here might be one orbit or one day, for example).

Once all non-optimal events and non-scheduled periods are examined for the request, the software determines which intra-request constraints or higher-priority requests had the most significant impact on the analyst's input

request. Through the dependency chain backtracking, these impacts could of course be either direct or indirect.

The system can also suggest possible remedies, within given parameters, which can result in the input request attaining a more favorable satisfaction. The parameters allow the analyst to instruct the system to ignore certain facets of the model that should not be modified, like the network architecture or the scheduling of unmovable events. The suggestions take into account knowledge of how the scheduling software operates internally. For instance, the utility might recommend that a station priority list be reordered for some request, or that visibility events should be sorted in AOS/LOS order instead of by station priority.

To summarize, the Explainer is helpful to the NPAS analyst because it provides a full accounting of the schedule process, indicating why a particular event did schedule or why none could be scheduled for a given time span. The analyst can use this information simply as something to report to the customer, or it can be a starting point for further model optimization. The Explainer assists in this regard as well as it can suggest different ways that requests in the model can be modified to maximize the satisfaction of the input request.

Neural Network Application

In response to SN customer representative/management requests for rapid determination of the ability of baseline network configurations to support additional users (such as during a meeting in which the requirements are originally proposed), the NPAS team devised a neural network (NN) application. The underlying neural network application is a NASA COTS product, NETS (Baffes 1989). In constructing a neural network, the NPAS team identifies several types of customer support requirements that may be requested. The executing end user would choose the requirements closest to those desired and the resultant expected mission satisfaction and total network loading (both as percentages) would be given.

To date, we have determined that the best accuracy across the multiple SN baselines is to generate a separate NN for each baselined network model in each model timeframe. The design of the system is such that the core of the system is of minimal size, and the menus and all associated options are dynamically constructed at run time from flat database files. The benefit here is that new scenarios can be created by the end user and easily imported into the utility without having to recompile the software. This also allows for the rapid construction of a new NN if any of the component baseline models are changed.

Due to the inherent CV problem with the GN (covered before), we have not yet perfected the application of NN in this area. There is currently too much flux on the composition of the GN (for example, the recent addition of the commercial DataLynx service for NASA Code Y support) to justify fielding such an application.

Conclusion

Given the trend of higher numbers of satellites on minimal cost networks, proven and adaptable tools are required to provide proper feasibility assessments for advanced mission planning of telecommunications tracking. Providing for the inherent differences in ground-based and space-based antenna systems is necessary, as is the ability to combine such networks when required. A knowledgeable team to analyze input requirements, ask additional questions to bound true problems, appropriately model, and analyze the phases of support is requisite for meaningful customer commitments.

References

Levine, A., and Joesting, D. 1997. Load Forecasting of a Space-based Telecommunications Network: NASA's TDRSS. In workshop notes of the International Workshop on Planning and Scheduling for Space Exploration and Science, Oxnard, CA.

Stern, D., Levine, A., and Pitt, K. 1994. Accuracy Analysis of TDRSS Demand Forecasts. In proceeds of the Third Symposium on Space Mission Operations and Ground Data Systems, 331-318, Greenbelt, MD: NASA Conference Publication 3281.

Simons, M., and Larsson, G. 1994. Analysis of Space Network Loading. In proceedings of the Third Symposium on Space Mission Operations and Ground Data Systems, 1071-1077, Greenbelt, MD: NASA Conference Publication 3281.

Baffes, Paul. NETS User's Guide, Version 2.0. Software Technology Branch, Lyndon Johnson Space Center, September 1989.

Satellite Telecommunications Forecasting

Commentary on Paper by D. Joesting & G. Larsson

Ken Galal
NASA/Ames Research Center, MS 244-19
Moffett Field, CA 94035-1000
kgalal@mail.arc.nasa.gov

Abstract

In an effort to adequately plan for the tracking and communications support of upcoming space missions, a number of methods and tools have been devised to help forecast the loading on existing networks. The need for such accurate forecasting is compounded, given the popularity of proposed mission concepts that call for the placement of entire constellations of satellites in Earth orbit. Joesting and Larson, in their paper, *Forecasting Telecommunications Support Boundaries for Satellite Constellations Using Deterministic Scheduling*, discuss some of the challenges that must be faced, and present some of their recent experience and tools used in tackling the forecasting problem. Capable methods are indeed necessary to establish the nature of the scheduling problems that lie ahead, and to suggest ways of re-designing missions to minimize conflicts. Additionally, new technologies must be developed and widely implemented to automate the station acquisition and telemetry downloading process, if we are to meet the spacecraft telecommunications needs of the future.

The Forecasting Problem

As pointed out by Joesting and Larson, the forecasting of long range telecommunications schedules of low Earth orbiting (LEO) satellites is complicated by imperfect modeling of the forces that perturb spacecraft orbits over time (e.g. variable drag forces that result from solar induced variations in atmospheric density) and the resulting growth in orbital position uncertainty associated with attempts to propagate spacecraft orbits beyond a few weeks. These orbit uncertainties, when coupled with uncertainties in spacecraft true anomaly for pre-launch forecasting, typically require statistical means to derive time-averaged station coverage profiles.

Thus Joesting and Larson describe the use of the Network Planning and Analysis System (NPAS), which relies on a Monte Carlo simulation to generate statistical data concerning spacecraft coverage profiles and station scheduling conflicts. This approach is similar to that implemented in the program LEO4CAST [1] developed by the Jet Propulsion Laboratory (JPL) for station resource allocation studies. Other forecasting methods that have been proposed include a deterministic technique described by M. Lo of JPL, which relies on dynamical systems methods to estimate long-term station view periods [2]. Relative to simulated data gathered from long-duration orbit propagations, Lo's method has been shown to be accurate to within 0.2% for circular orbits and within 15% for low eccentricity orbits ($e < 0.05$). While offering advantages in terms of much lower computational cost, Lo's method, however, does not directly address station conflict assessment between multiple spacecraft.

Another useful software tool described by Joesting and Larson is the Conflict Explanation Utility, which can provide insight into the various factors that influence the availability of station coverage for a spacecraft. When used together, all these tools can help NASA assess the adequacy of its network resources to meet future telecommunications demands. These tools can also suggest changes to a mission's characteristics (e.g. different orbit, antenna size, on-board data storage capacity, station complement, etc.) to improve its chances of obtaining the communications resources it needs.

The potential for such pre-emptive changes to the design of a spacecraft is appealing, in that they could result in a decrease of overall network loading, and lead to a more efficient use of available resources. However, for such changes to be practical, they must occur very early in the design process. As such, it is important that prospective space projects be given access to these forecasting tools and allowed to iterate on their mission design, preferably with occasional access to guidance from NASA scheduling experts. To that end, a web-based tool, accurately configured with an up-to-date station complement and mission set, would be most useful.

The Constellation Challenge

Constellations, consisting of multiple inexpensive spacecraft in orbit, have been advocated lately as a low-cost way of enabling global measurement from space with minimal risk posed to the overall mission as a result of an individual spacecraft failure. While these constellations

offer intriguing possibilities, they also have the potential of quickly overwhelming existing communications resources. Aside from adding a large number of spacecraft to future mission sets, constellations composed of inexpensive spacecraft will likely be limited in on-board storage capacity, power, and antenna size --- attributes that exacerbate the scheduling problem.

But beyond the station-scheduling problem, there is also the hurdle of cost for the support of such large numbers of spacecraft. Given traditional operations methods, the resources required to maintain these constellations, in terms of station scheduling, commanding, and data reception, could dampen enthusiasm for such missions. As an example, the cost of operations support for the recent Lunar Prospector mission was largely driven by the need to maintain 24 hr operator support to ensure the nearly continuous tracking requirement for downlink of science telemetry. During a one-month period arbitrarily chosen for study, over 30 data transfer anomalies were counted which involved problems with station acquisition, data transfer, or workstation errors. In most cases, without the active intervention of spacecraft controllers and station personnel, significant amounts of data would have been lost.

The Role of Automation

To help get around the scheduling challenge imposed by the future use of spacecraft constellations, a greater reliance on automation is necessary. Already, spacecraft operators have implemented ground stations that can automatically acquire and track a satellite without operator intervention [3]. Such automation has been easier to implement on missions with a dedicated ground station and a limited number of common spacecraft. For large networks like NASA's GN and SN that support a wide variety of missions, automation has proven to be more challenging, due to the large disparity in tracking requirements and spacecraft characteristics. However, in recent years, there have been some successful demonstrations of "lights out" station operation using NASA's Low Earth Orbit Terminal with a 3-m antenna system, and the Deep Space Terminal with a 34-m antenna [4].

Such progress has indeed been promising; however, further steps will probably be necessary to meet the heavy telecommunication demands of spacecraft constellations. Perhaps the ultimate goal should consist of enabling spacecraft to independently request support from automatic ground stations. Using GPS to establish their position relative to an onboard database of available ground stations, future spacecraft could compute view periods at candidate sites along with their required contact duration, and request service directly from a station. Priority for a given s/c request could be decided by each station, based on the saturation level of a spacecraft's storage capacity, or according to the relative importance of the data collected, as defined by scientists and operators. Of course such a system would not eliminate contention over limited telecommunications resources, and it would require solutions to non-trivial technological hurdles. It may ultimately, however, be the direction of things to come, given the complexity and costs associated with more traditional alternatives.

References

[1] Fox, G. and Borden, C. 1994. Low-Earth-Orbiter Resource Allocation and Capacity Planning for the DSN using LEO4CAST, Technical Report, TDA PR 42-118, Jet Propulsion Laboratory.

[2] Lo, M. 1994. The Long-Term Forecast of Station View Periods, Technical Report, TDA PR 42-118, Jet Propulsion Laboratory.

[3] Wertz, J. and Larson, W. eds. 1996. *Reducing Space Mission Cost.* Torrance, Calif. : Kluwer Academic Publishers.

[4] Paal, L., Golshan, N., Fisher, F., Law, E., Veruttipong, W., and Stockett, M. 1999. Deep Space Terminal Demonstration, Technical Report, TDA PR 42-136, Jet Propulsion Laboratory.

Approximate Planning in the Probabilistic-Planning-as-Stochastic-Satisfiability Paradigm

Stephen M. Majercik and **Michael L. Littman**
Department of Computer Science
Duke University
Durham, NC 27708-0129
{majercik,mlittman}@cs.duke.edu

Abstract

ZANDER is a state-of-the-art probabilistic planner that extends the probabilistic-planning-as-stochastic-satisfiability paradigm to support contingent planning in domains where there is uncertainty in the effects of the agent's actions and where the scope and accuracy of the agent's observations may be insufficient to establish the agent's current state with certainty (Majercik & Littman 1999). We describe ZANDER and then discuss an approximation technique we are developing that will help us to scale up our SSAT-based technique to large planning problems. We report results using this approximation algorithm on random SSAT problems and discuss issues that arise in the application of this algorithm to SSAT encodings of planning problems.

Introduction

Planning is a critical activity in space exploration efforts. Uncertainty in the various domains—especially partial observability—makes it impossible to rely on a simple straight-line plan. One approach to dealing with this uncertainty is to construct a plan as if the environment were deterministic, and then replan quickly if the plan fails. If the uncertainty in the environment is quantifiable, however, we can use that knowledge to construct a plan with contingencies so that there is less likelihood of having to replan.

Our work focuses on the latter approach. ZANDER is a probabilistic planner that extends the probabilistic-planning-as-stochastic-satisfiability paradigm to support contingent planning in domains where there is uncertainty in the effects of the agent's actions and where the scope and accuracy of the agent's observations may be insufficient to establish the agent's current state with certainty (Majercik & Littman 1999). ZANDER solves a planning problem by converting it into an instance of stochastic satisfiability (SSAT), a type of satisfiability problem in which some of the variables have probabilities attached to them. A solution to the SSAT problem yields a plan that has the highest probability of succeeding. Initial results have been very encouraging—ZANDER operates at state-of-the-art speeds on planning problems drawn from the literature—and we are now developing techniques based on this paradigm that will allow us to scale up to large, real-world problems. A natural approach is an approximation technique and, in this paper, we describe efforts to develop such an algorithm for solving the SSAT encodings of planning problems generated by our planner.

We envision two possible applications of our planning technique (Brooks 2000). First, our planner has the potential to accept a list of requests for use of a particular spacecraft, organize these requests into a viable sequence of activities, and convert these requests into input for other elements of the uplink data system. Second, our planner could be used as part of an on-board fault protection system in a spacecraft. Given a fault condition, the planner would observe the condition of the spacecraft, plan more diagnostic tests if necessary, create a contingent plan to correct the fault and, finally, replan any interrupted or unexecuted activities and continue the sequence.

In the remainder of this section, we describe our domain representation, the SSAT framework, and how contingent planning problems can be encoded as SSAT problems. After a brief description of ZANDER, we describe our approximation algorithm, report results on a large number of random SSAT problems, and discuss issues that arise in applying this algorithm to SSAT encodings of planning problems. We conclude with an assessment of the strengths and weaknesses of this approach.

Probabilistic Planning Representation

ZANDER uses a propositional representation for planning problems. A planning problem is described using a finite set P of n distinct *propositions*, each of which may be `True` or `False` at any (discrete) time t. A *state* is an assignment of truth values to P. A probabilistic initial state is specified by a set of decision trees, one for each proposition. *Goal states* are specified by a partial assignment G to the set of propositions; any state that extends

G is considered to be a goal state. Each of a finite set A of *actions* probabilistically transforms a state at time t into a state at time $t+1$ and so induces a probability distribution over the set of all states. In this work, the effect of each action on each proposition is represented as a separate decision tree (Boutilier & Poole 1996). For a given action a, each of the decision trees for the different propositions are ordered, so the decision tree for one proposition can refer to both the new and old values of previous propositions. The leaves of a decision tree describe how the associated proposition changes as a function of the state and action. A subset of the set of propositions is the set of *observable propositions*, each of which has, as its basis, a proposition that represents the actual status of the thing being observed. The planning task is to find a plan that selects an action for each step t as a function of the value of observable propositions for steps before t. We want to find a plan that maximizes (or exceeds a user-specified threshold for) the probability of reaching a goal state.

Stochastic Satisfiability

In the deterministic satisfiability problem, or SAT, we are given a Boolean formula and wish to determine whether there is some assignment to the variables in the formula that results in the formula evaluating to `True`. Papadimitriou (1985) explored an extension of SAT in which a random quantifier is introduced. The stochastic SAT (SSAT) problem is to evaluate a Boolean formula in which existential and random quantifiers alternate:

$$\exists x_1, \mathrm{Y} x_2, \exists x_3, \ldots, \exists x_{n-1}, \mathrm{Y} x_n (E[\phi(\mathbf{x})] \geq \theta).$$

In words, this formula asks whether there is a value for x_1 such that, *for random* values of x_2 (choose 0 or 1 with equal probability), there exists a value of $x_3 \ldots$ such that the *expected* value, or probability of satisfaction, of the Boolean formula $\phi(\mathbf{x})$ is at least a threshold $0 \leq \theta \leq 1$. In our SSAT problems, we will allow *blocks* of existential and random quantifiers to alternate. Furthermore, we will allow *annotated random quantifiers* such as $\mathrm{Y}^{0.2}$, which takes on value `True` with probability 0.2 and `False` with probability 0.8. The specification of an SSAT problem consists of the Boolean formula $\phi(\mathbf{x})$, the probability threshold θ, and the ordering of the quantifiers. In what follows, we will refer to existentially quantified variables in the SSAT formula as *existential variables* and randomly quantified variables as *randomized variables*.

Encoding Planning Problems

In an SSAT formula, the value of an existential variable x can be selected on the basis of the values of all the variables to x's left in the quantifier sequence. This suggests a way of mapping contingent planning problems to stochastic satisfiability: encode the contingent plan in the variable ordering associated with the SSAT formula. By alternating blocks of existential variables that encode actions and blocks of randomized variables that encode observations, we can condition the value chosen for any action variable on the possible values for all the observation variables that appear earlier in the ordering. A generic SSAT encoding for contingent plans appears in Figure 1.

The quantifiers fall into three segments: a plan-execution history, the domain uncertainty, and the result of the plan-execution history given the domain uncertainty. The plan-execution-history segment is an alternating sequence of existential-variable blocks (one for each action choice) and randomized-variable blocks (one for each set of possible observations at a time step).

The domain uncertainty segment is a single block containing all the randomized variables that modulate the impact of the actions on the observation and state variables. These variables are associated with random quantifiers; when we consider a variable that represents uncertainty in the environment, we want to take the probability weighted average of the success probabilities associated with the two possible settings of the variable.

The result segment is a single block containing all the non-observation state variables. These variables are associated with existential quantifiers, indicating that we can choose the best truth setting for each variable. In reality, all such "choices" are forced by the settings of the action variables in the first segment and the chance variables in the second segment. If these forced choices are compatible, then the preceding plan-execution history is possible and has a non-zero probability of achieving the goals. Otherwise, either the plan-execution history is impossible, given the effects of the actions, or it has a zero probability of achieving the goals.

An attractive feature of this planning technique is that it is straightforward to add additional constraints to the SSAT encoding of the planning problem. This means, for example, that human judgments about activities that *must* be performed can easily be enforced.

ZANDER

We briefly describe ZANDER, our SSAT-based probabilistic planner. Details are available elsewhere (Majercik & Littman 1999). ZANDER must find an assignment *tree* that specifies the optimal existential-variable assignment given all possible settings of the observation variables. The most basic variant of the solver follows the variable ordering exactly, constructing a binary tree of all possible assignments. Each node in the tree contains a variable under consideration, and each path through the tree describes a plan-execution history, an in-

$$\overbrace{\exists x_{1,1},\ldots,\exists x_{1,c_1}}^{\text{first action}}\overbrace{\text{Ɜ}w_{1,1},\ldots,\text{Ɜ}w_{1,c_2}}^{\text{first observation}}\cdots\overbrace{\text{Ɜ}w_{n-1,1},\ldots,\text{Ɜ}w_{n-1,c_2}}^{\text{last observation}}\overbrace{\exists x_{n,1},\ldots,\exists x_{n,c_1}}^{\text{last action}}$$

$$\overbrace{\text{Ɜ}^{\rho_1}z_1,\ldots,\text{Ɜ}^{\rho_{c_4}}z_{c_4}}^{\text{random outcomes}}\overbrace{\exists y_1,\ldots,\exists y_{c_3}}^{\text{the state}}(E[\phi(\mathbf{x})]\geq\theta).$$

c_1 = number of variables it takes to specify a single action (the number of actions),
c_2 = number of variables it takes to specify a single observation,
c_3 = number of state variables (one for each proposition at each time step), and
c_4 = number of chance variables (one for each possible stochastic outcome at each time step).

Figure 1: A contingent planning problem can be encoded as an instance of SSAT.

stantiation of the domain uncertainty, and a possible setting of the state variables. An observation variable is a branch point; the optimal assignment to the remaining variables will, in general, be different for different values of this variable.

The solver does a depth-first search of the tree, constructing a solution subtree by calculating, for each node, the probability of a satisfying assignment given the partial assignment so far. For an existential variable, this is a maximum probability and produces no branch in the solution subtree; the solver notes which value of the variable yields this maximum. For a randomized variable, the probability will be the probability weighted average of the success probabilities for that node's subtrees and will produce a branch point in the solution subtree. The solver finds the optimal plan by determining the subtree with the highest success probability.

We use three pruning techniques to avoid checking every possible truth assignment. Whenever an existential or randomized variable appears alone in an active clause, *unit propagation* assigns the forced value to that variable. Whenever an existential variable appears always negated or always not negated in all active clauses, *variable purification* assigns the appropriate value to that variable. *Thresholding* allows us to prune plans based on a prespecified threshold probability of success (i.e. find a plan whose probability of success meets or exceeds the threshold probability).

An SSAT Approximation Algorithm

ZANDER performs at state-of-the-art speeds on problems drawn from the literature (Majercik & Littman 1999). This is encouraging since there are a number of potential improvements to ZANDER that have shown promise for scaling up to larger problems (better data structures to optimize the application of heuristics, more compact and efficient SSAT encodings, encoding domain knowledge, memoization for contingent planning, using learning to accelerate the solution process, and more sophisticated splitting heuristics). In order to scale up to even larger problems, however, it may be necessary to develop approximation techniques. In this section we describe one possible approach.

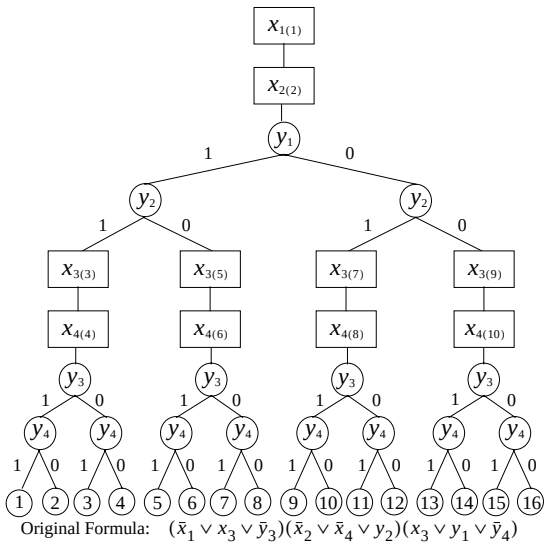
Our approximation algorithm randevalssat takes an SSAT formula and returns an approximation of its value, along with the policy that approximately produces this value. The algorithm uses a *policy tree* representation. Policy trees include multiple copies of each existential variable—one for each possible assignment to the randomized variables that precede it in the quantifier ordering—thus emphasizing the fact that the value of each existential variable can be a function of the values of the preceding randomized variables. Figure 2 shows the policy tree for the following SSAT instance:

$$\exists x_1,\exists x_2,\text{Ɜ}y_1,\text{Ɜ}y_2,\exists x_3,\exists x_4,\text{Ɜ}y_3,\text{Ɜ}y_4,$$
$$E[(\overline{x_1}\vee x_3\vee\overline{y_3})(\overline{x_2}\vee\overline{x_4}\vee y_2)(x_3\vee y_1\vee\overline{y_4})]\geq\theta.$$

Existential variables, and their copies, are called *decision variables* and are shown as rectangular *decision nodes* in the policy tree. To indicate that the value of each instance of a copied variable can be set independently, these variables are renumbered in the policy tree (parenthesized subscripts).

Each leaf of the policy tree represents a partial assignment consisting of an assignment to all randomized variables in the SSAT formula. The *probability of a leaf* is the product of the probabilities of the outcomes along the path from the root to the leaf. A *policy* is an assignment of Boolean values to the decision variables in the policy tree. Given a policy, all the root-to-leaf paths in the policy tree represent complete assignments to the variables in the formula, each of which is either satisfying (value 1) or unsatisfying (value 0). The *value of a policy* is the weighted sum of the probabilities of the satisfied leaves. The *value of a policy tree* is the maximum over all policies of the policy values.

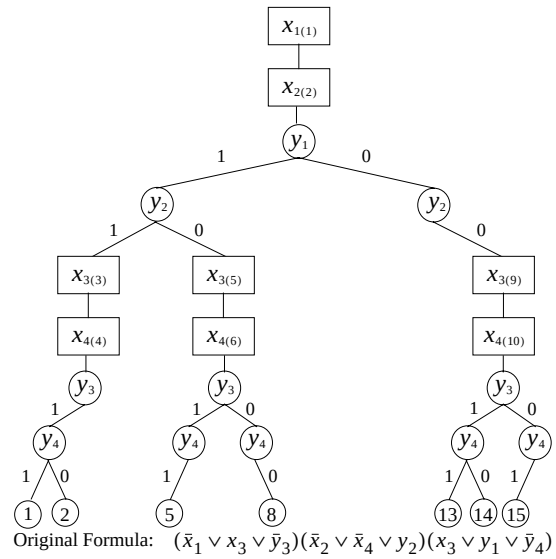
A systematic search for the policy with the highest value would solve the SSAT problem, but this is a doubly exponential process: the size of the policy tree is exponential in the number of randomized variables and systematically searching for the best policy is exponential in the number of decision



Simplified Formulae at Leaves Given Random Variable Assignments

Leaf 1:	$(\bar{x}_1 \vee x_3)$	Leaf 9:	$(\bar{x}_1 \vee x_7)(x_7)$
Leaf 2:	$(\bar{x}_1 \vee x_3)$	Leaf 10:	$(\bar{x}_1 \vee x_7)$
Leaf 3:	Satisfied	Leaf 11:	(x_7)
Leaf 4:	Satisfied	Leaf 12:	Satisfied
Leaf 5:	$(\bar{x}_1 \vee x_5)(\bar{x}_2 \vee \bar{x}_6)$	Leaf 13:	$(\bar{x}_1 \vee x_9)(\bar{x}_2 \vee \bar{x}_{10})(x_9)$
Leaf 6:	$(\bar{x}_1 \vee x_5)(\bar{x}_2 \vee \bar{x}_6)$	Leaf 14:	$(\bar{x}_1 \vee x_9)(\bar{x}_2 \vee \bar{x}_{10})$
Leaf 7:	$(\bar{x}_2 \vee \bar{x}_6)$	Leaf 15:	$(\bar{x}_2 \vee \bar{x}_{10})(x_9)$
Leaf 8:	$(\bar{x}_2 \vee \bar{x}_6)$	Leaf 16:	$(\bar{x}_2 \vee \bar{x}_{10})$

Figure 2: A policy tree (a) represents the set of contingent choices in an SSAT problem.



Assignments Responsible for Leaves in Partial Decision Tree

Leaf 1:	$y_1 = 1$	$y_2 = 1$	$y_3 = 1$	$y_4 = 1$
Leaf 2:	$y_1 = 1$	$y_2 = 1$	$y_3 = 1$	$y_4 = 0$
Leaf 5:	$y_1 = 1$	$y_2 = 0$	$y_3 = 1$	$y_4 = 1$
Leaf 8:	$y_1 = 1$	$y_2 = 0$	$y_3 = 0$	$y_4 = 0$
Leaf 13:	$y_1 = 0$	$y_2 = 0$	$y_3 = 1$	$y_4 = 1$
Leaf 14:	$y_1 = 0$	$y_2 = 0$	$y_3 = 1$	$y_4 = 0$
Leaf 15:	$y_1 = 0$	$y_2 = 0$	$y_3 = 0$	$y_4 = 1$

Figure 3: Randomized local search can be applied to a partial policy tree, obtained by sampling, to provide an approximate answer for an SSAT instance.

nodes in the policy tree. Instead, the algorithm randevalssat uses stochastic sampling to limit the size of the policy tree constructed and randomized local search to find the best policy for that reduced tree. This approach is similar to that of Kearns, Mansour, & Ng (1999) for choosing approximately optimal actions with high probability in infinite-horizon discounted Markov decision processes.

The algorithm constructs a *partial policy tree* by choosing a set W of w assignments to the randomized variables proportional to their probability and independently of the policy. These sampled assignments select out a set of leaves from the full policy tree, with the probability of an assignment given by its frequency of selection in the random sample. The top of Figure 3 gives a partial policy tree derived from the sample in the bottom of Figure 3 and the policy tree of Figure 2. The value of a policy is then estimated as the proportion of the w sampled leaves that are satisfied by the policy. A direct application of Chernoff bounds shows that $w = O(\frac{1}{\epsilon^2}\log(\frac{1}{\delta}))$ samples are sufficient to be sure with probability $1-\delta$ of having an estimate no further than ϵ away from the true value (Littman, Majercik, & Pitassi 2000). Note that this number does not depend on n, m, k, or how the SSAT formula or the policy was created.

As illustrated in Figure 2, the effect of a policy on a leaf can be summarized by a Boolean formula, called a *path formula*, for that leaf. Each random sample can potentially produce a different path formula, and the union of all the path formulas produces a *treeSAT* formula, which can be viewed as both a collection of clauses and a collection of path formulas. A satisfying assignment to the treeSAT formula corresponds to a setting of the decision variables that satisfies the original SSAT formula along all paths in the partial policy tree. In general, this will not be possible; instead randevalssat use randomized local search to search for a treeSAT assignment that maximizes the number of path formulas that are satisfied. In our experiments, this was done by hillclimbing on an objective function that counts both clauses satisfied and path formulas satisfied. A satisfied path formula is weighted as heavily as the number of clauses in the original

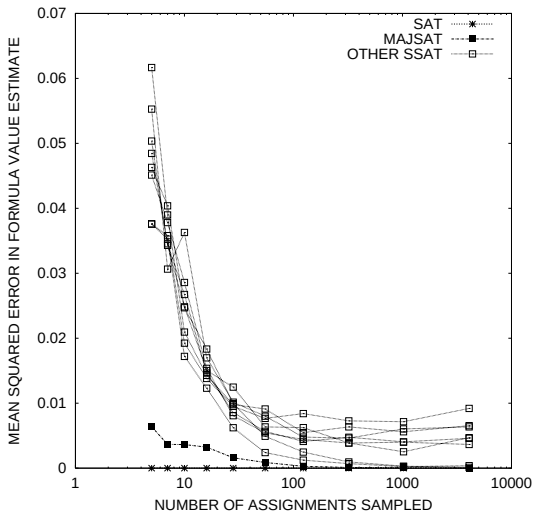


Figure 4: As the number of sampled assignments increases, the accuracy of the randomized local search algorithm increases. Because of local optima in the search space, increasing the number of sampled assignments does not drive the error to zero.

SSAT formula, so that satisfying all the clauses in a path formula contributes more to the treeSAT formula's value than satisfying the same number of clauses scattered over more than one path formula.

Experiments

The performance of randevalssat for computing the value of SSAT instances was tested on 27 sets of 100 random formulas, generated under the fixed-clause model using a modified version of `makewff` that guarantees a formula with exactly n variables. For each problem a SAT instance (all existential variables), a MAJSAT instance (all randomized variables), and all possible instances that contain an equal number of existential and randomized variables in alternating blocks of the same size were constructed. For each of these problems, 10 different partial policy trees were created by sampling randomized variable assignments and constructing the tree paths specified by the samples. The number of assignments sampled w ranged from 5 to 4086 on an approximate logarithmic scale; the larger the number of samples, the greater the similarity of the partial tree to the full tree. Performance was measured by comparing the estimate of the value of the partial policy tree to the exact value of the full policy tree (and, hence, the formula).

Figure 4 shows results for randevalssat on problems with 12 variables, 24 clauses, and 3 literals per clause (results for all 27 sets of problems were similar). The graph shows mean squared error in the value estimate as a function of the number of sampled assignments for all problem types. The mean and variance of the squared error decrease as the number of assignment samples increases. We note that the nonzero error in the limiting case, where the number of assignments sampled is sufficient to construct the full policy tree with high probability, is due to the use of randomized local search. If a more time-consuming systematic search of the partial policy tree is employed, the error is driven to zero as the number of sampled assignments increases (Littman, Majercik, & Pitassi 2000).

Application to Planning Problems

Once we have translated a probabilistic planning problem into an SSAT instance, it would seem to be straightforward to apply the stochastic sampling algorithm to the SSAT problem to find an approximation of the optimal contingent plan and an approximation of its probability of success. The situation is complicated, however, by the fact that we use randomized variables to describe observations. This means that a random sample of the randomized variables describes an observation sequence as well as an instantiation of the uncertainty in the domain, and the observation sequence thus produced may not be *observationally consistent*. Informally, an observation sequence is observationally consistent if there exists a sequence of actions and an instantiation of the environment that could possibly produce that observation sequence. For example, assuming perfect sensors, it would be observationally inconsistent to observe at time step t that a valve driver is permanently failed and at time step $t+1$ that it is operational.

Applying the stochastic sampling algorithm directly to planning problems can result in the generation of observationally inconsistent paths in the partial policy tree. And these paths, which are unsatisfied paths regardless of the setting of the action variables chosen, need to be treated differently from observationally consistent paths, which have the potential to become satisfied paths, depending on the values chosen for the decision variables. If the algorithm includes observationally inconsistent paths in its estimate of the probability of success of a given policy, it will tend to underestimate this probability. The algorithm needs to either avoid generating observationally inconsistent paths in the first place, or ignore them in its calculations.

Considerations of efficiency suggest that the first strategy is to be preferred whenever possible, and this suggests two alterations to the stochastic sampling algorithm in order to make it applicable to SSAT encodings of planning problems:

- Sample tree paths only from P, the set of paths that are observationally consistent for some action sequence and instantiation of the environment.

- Adjust the evaluation of policy trees as follows: Instantiating the decision variables in the policy tree selects $P' \subseteq P$, the set of paths in P that are observationally consistent for that setting of decision variables and some instantiation of the environment. Let $S \subseteq P'$ be those paths in P' that are satisfied paths. Then, the probability of success of the contingent plan represented by that policy tree is $\frac{|S|}{|P'|}$.

There are various algorithmic issues that need to be addressed. First, we need to find an efficient way of constructing P and P'. One possibility is to use user-supplied information to prune at least some of the observationally inconsistent paths when constructing P. A second possibility is to construct encodings such that unit propagation can be used to efficiently detect observational inconsistencies.

Second, the algorithm returns a partial assignment strategy, or policy, that specifies how each decision variable should be set given the settings of the decision and observation variables that precede it in the quantifier ordering, but this is only specified for those situations represented by paths in the random sample used to construct the partial tree. The algorithm implicitly assumes that performance on missing branches is the same as the average performance over all paths in the partial tree, but doesn't actually find a plan that achieves this performance. One possible strategy for addressing this issue would be an iterative approach that alternates planning and evaluation. In this approach, each iteration would perform an evaluation of the current policy (perhaps by simulating executions of the policy) and use that evaluation to guide the construction of a partial policy tree that would lead to a better policy. This is similar to approaches that identify the most severe problems in an imperfect plan and then attempt to correct them (Drummond & Bresina 1990; Onder & Pollack 1997).

Summary

The stochastic sampling algorithm randevalssat appears to be a promising approximation method for SSAT encodings of planning problems. Its primary strength is the use of sampling to convert the problem to a lower complexity problem and its use of randomized local search to solve that problem efficiently. A feature of this process is that it does not necessarily completely discard the probabilities of the original problem (as would, say, a conversion that merely rounded off the probabilities of the randomized variables to 0 and 1 and set their truth values accordingly). It would, for example, be possible to modify the algorithm such that the probabilities of the randomized variables are used to direct the construction of the partial policy tree. A more substantial modification, discussed above, would be to iteratively build the partial policy tree by using the solution of the partial policy tree in a given iteration to construct a better partial policy tree (and solution) in the next iteration.

This algorithm has some weaknesses. First, randevalssat does not return an answer whose correctness is "easy" to certify; this is to be expected, given the complexity of SSAT problems. Second, there are SSAT problems for which the algorithm needs provably large samples (Littman, Majercik, & Pitassi 2000). Third, randevalssat is subject to the same pitfalls as other randomized local search algorithms; in particular, it may become stuck in local optima. Allowing the algorithm to restart after a period of no progress helps minimize, but does not erase, this problem. Finally, the memory requirements of the algorithm can be prohibitively large. This weakness can be partially overcome by more memory-efficient implementations, but problems that require a large number of samples to produce an accurate answer will inherently generate large treeSAT formulas.

Acknowledgments: The first author acknowledges the support of a NASA GSRP Fellowship.

References

Boutilier, C., and Poole, D. 1996. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1168–1175. AAAI Press/The MIT Press.

Brooks, R. 2000. Personal communication.

Drummond, M., and Bresina, J. 1990. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 138–144. Morgan Kaufmann.

Kearns, M.; Mansour, Y.; and Ng, A. Y. 1999. A sparse sampling algorithm for near-optimal planning large markov decision processes. Submitted.

Littman, M. L.; Majercik, S. M.; and Pitassi, T. 2000. Stochastic Boolean satisfiability. *Journal of Automated Reasoning*. To appear in a special issue on Satisfiability in the Year 2000.

Majercik, S. M., and Littman, M. L. 1999. Contingent planning under uncertainty via stochastic satisfiability. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 549–556. The AAAI Press/The MIT Press.

Onder, N., and Pollack, M. E. 1997. Contingency selection in plan generation. In *Proceedings of the Fourth European Conference on Planning*.

Papadimitriou, C. H. 1985. Games against nature. *Journal of Computer Systems Science* 31:288–301.

Comments on
Approximate Planning in the Probabilistic-Planning-as-Stochastic-Satisfiability Paradigm

Paper by Stephen M. Majercik and Michael L. Littman

Comments by Robert N. Brooks Jr.

Telecommunications and Mission Operations Directorate
Mission Services and Applications
Service System Development Engineer
Jet Propulsion Laboratory
4800 Oak Grove Dr.
Pasadena, CA 91109
Mail Stop: 301/280
818-393-7856
Robert.N.Brooks@jpl.nasa.gov

First, a brief description of the commentator's interpretation and understanding of this paper. It describes a methodology for bounding non-deterministic problems. It appears this is done by solving for deterministic states and then using those solutions to "limit" the extent of the non-deterministic states to be considered, thus making the "whole problem" easier to solve by raising it to a higher probability of being knowable. Optimization of a resulting sequence is not a requirement on this tool.

The first question would be about where in an uplink process this tool would be useful? As with any planning tool it can be of value during early planning phases but has any thought been given to other types of uplink tasks it may be used for? Would this type of technology be used primarily on the ground by a flight team or could it be used as part of a spacecraft's onboard flight software, perhaps in responding to faults encountered during flight?

Some discussion is provided about the performance of the algorithm but it really only consists of comparisons and general discussion. There is a need for data that could provide a more quantitative assessment? For instance, a sequence of activities consisting of N activities with an average of M commands per activity and with a total duration of T days required X minutes (or hours) to run to completion and the resulting sequence satisfied some percentage Y of the requirements levied on it. It would be useful to see even some quick and dirty quantitative analysis of performance. The comparisons to other similar tools is valuable but it's actual numbers that help a user understand the product. Also, if it takes the program longer to run to completion than the duration of the sequence it's trying to build then it's of questionable value as a useful tool during the later stages of sequence development.

Because this is a technical paper about an extremely technical topic, it contained a great deal of jargon, much of which is specialized jargon. Though this is a method for accurately expressing the concepts contained in the paper, it also makes the work difficult to understand by those not intimately familiar with the specific technical field. The commentator had to work at not becoming lost in the words and the notation. It is suggested that the authors consider "wordsmithing" some of the text to make it more readable (and, hence, more meaningful) to someone who doesn't necessarily have this type of technical background. This should be especially considered for a few of the most important concepts and techniques. Doing so would make the paper more "intellectually available" to a wider audience.

The commentator's experience dictates that any tool used as a planner for sequencing a spacecraft must provide the ability to override its decisions while maintaining its internal fidelity. This ability to override is useful when a flight team needs to force things to be done differently than a tool would do it because they're responding to an anomaly or, more likely, some user demands that their activity be performed in a manner or at a time that the planner thinks is wrong. Maybe the deviation from the standard rules is a one-shot-deal or maybe it's in response to some special circumstances that suddenly arose ("Oh, oh, there's a supernova happening right now over there in the sky and we have to violate this pointing constraint and go look at it"). The paper did not address this issue. Therefore, some level of flexibility should be made available with this tool? The way spacecraft operations are done today and are being planned for the future makes flexibility a requirement on any planning tool.

First published by the authors under the same title in AIPS 2000, p. 300-307
Copyright (c) 2000, American Association for Artificial Intelligence. Reprinted with permission from AAAI

Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling

Steve Chien, Russell Knight, Andre Stechert, Rob Sherwood, and Gregg Rabideau

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
{firstname.lastname}@jpl.nasa.gov

Abstract

The majority of planning and scheduling research has focused on batch-oriented models of planning. This paper discusses the use of iterative repair techniques to support a continuous planning process as is appropriate for autonomous spacecraft control. This allows the plan to incorporate execution feedback - such as early or late completion of activities, and over-use or under-use of resources. In this approach, iterative repair supports continuous modification and updating of a current working plan in light of changing operating context.

Introduction

Traditionally, much of planning and scheduling research has focused on a batch formulation of the problem. In the batch approach, time is divided up into a number of planning horizons, each of which lasts for a significant period of time. When one nears the end of the current horizon, one projects what the state will be at the end of the execution of the current plan. The planner is invoked with a new set of goals and this state as the initial state (for example the Remote Agent Experiment operated in this fashion (Pell et al, 1997)) .

This approach has a number of drawbacks. In this batch oriented mode, typically planning is considered an off-line process which requires considerable computational effort and there is a significant delay from the time the planner is invoked to the time that the planner produces a new plan.[1] If a negative event occurs (e.g., a plan failure), the response time until a new plan may be significant. During this period the system being controlled must be operated appropriately without planner guidance. If a positive event occurs (e.g., activities finishing early), again the response time may be significant. If the opportunity is short lived, the system must be able to take advantage of such opportunities without a new plan (because of the delay in generating a new plan). Finally, because the planning process may need to be initiated significantly before the end of the current planning horizon, it may be difficult to project what the state will be when the current plan execution is complete. If the projection is wrong the plan may have difficulty.

To achieve a higher level of responsiveness in a *dynamic planning* situation, we utilize a *continuous planning* approach and have implemented a system called CASPER (for Continuous Activity Scheduling Planning Execution and Replanning). Rather than considering planning a batch process in which a planner is presented with goals and an initial state, the planner has a current goal set, a plan, a current state, and a model of the expected future state. At any time an incremental update to the goals, current state, or planning horizon (at much smaller time increments than batch planning)[2] may update the current state of the plan and thereby invoke the planner process. This update may be an unexpected event or simply time progressing forward. The planner is then responsible for maintaining a consistent, satisficing plan with the most current information. This current plan and projection is the planner's estimation as to what it expects to happen in the world if things go as expected. However, since things rarely go exactly as expected, the planner stands ready to continually modify the plan. From the point of view of the planner, in each cycle the following occurs:

- changes to the goals and the current state first posted to the plan,
- effects of these changes are propagated through the current plan projections (includes conflict identification)
- plan repair algorithms[3] are invoked to remove conflicts and make the plan appropriate for the current state and goals.

[1] As a data point, the planner for the Remote Agent Experiment (RAX) flying on-board the New Millennium Deep Space One mission (Muscettola et al 1997) takes approximately 4 hours to produce a 3 day operations plan. RAX is running on a 25 MHz RAD 6000 flight processor and uses roughly 25% of the CPU processing power. While this is a significant improvement over waiting for ground intervention, making the planning process even more responsive (e.g., on a time scale of seconds or tens of seconds) to changes in the operations context, would increase the overall time for which the spacecraft has a consistent plan.

[2] For the spacecraft control domain we are envisaging an update rate on the order of 10 seconds real time.

[3] In this paper we do not focus on the state/resource representation or the repair methods (Rabideau et al. 1999).

At each step, the plan is created by using iterative repair with:

- the portion of the old plan for the current planning horizon;
- the updated goals and state; and
- the new (extended)planning horizon.

Even though our intent is to make the planning process very responsive (on the order of seconds), there still remains a synchronization process between planning and execution. Specifically, there are several issues in integrating planning with real time execution - below we list these issues and how they are addressed in our approach.

- When to replan? Our approach replans when the current plan projection predicts a problem with the current plan (i.e. when the plan combined with the current state is infeasible).
- What to do (execute) during planning time? If feedback from the world combined with the current plan indicates that the current plan has a flaw (e.g., the plan will not execute or does not achieve goals), what gets executed during the time that the planner is replanning? *Our approach attempts to minimize the amount of time for replanning to minimize the chance that a conflict appears in the portion of the old plan that gets executed.*
- How much time should the planner be given to replan? The longer the planner is given the more likely it will be able to resolve all of the problems. But the longer the replan time, the greater the problem of "What to execute in the meantime?" from above. *Our approach attempts to minimize the amount of time given to replan, but we believe this criteria can only be determined in a domain-specific fashion*[4].
- How to ensure the planner does not change activities that are already in execution? *Our approach uses a commitment mechanism to represent activities that would not be changeable by the time that the planner would complete its current cycle of reasoning. When an activity overlaps with this window (i.e. the activity is scheduled to begin very soon) it is committed. This means that the planner is forbidden from altering any aspect of this activity (such as by moving the activity or altering the activity parameters). Thus far we have focused on time-based commitment strategies (e.g., commit any activities scheduled to begin in the next T time units), however, our architecture supports more complex commitment strategies (such as it being dependent on the class of activity and allowing parameter changes later than activity moves, etc.).*

In addition to increasing the responsiveness of planning, the continuous planning approach has additional benefits:

- The planner can be more responsive to unexpected (i.e., unmodeled) changes in the environment that would manifest themselves as updates on the execution status of activities as well as monitored state and resource values.
- The planner can reduce reliance on predictive models (e.g., inevitable modeling errors), since it will be updating its plans continually.
- Fault protection and execution layers need to worry about controlling the spacecraft over a shorter time horizon (as the planner will replan within a shorter time span).
- Because of the hierarchical reasoning taking place in the architecture there is no hard distinction between planning and execution – rather more deliberative (planner) functions reside in the longer-term reasoning horizons and the more reactive (execution) functions reside in the short-term reasoning horizons. Thus, there is no planner to executive translation process.

In conjunction with this incremental, continuous planning approach, we are also advocating a hierarchical approach to planning, however we do not describe this approach here due to space constraints, for details see (Chien et al. 2000).

An Architecture for Integrated Planning and Execution

Our approach to integration of planning and execution relies on three separate classes of processes.

- **The Planner Process(es)** - this process represents the planner, and is invoked to update the model of the plan execution, to refine the plan, or when new goals are requested.
- **The Execution Process(es)** - this process is responsible for committing activities and issuing actual commands corresponding to planned activities.
- **The State Determination Process(es)** - this process is responsible for monitoring and estimating states and resource values and providing accurate and timely state information.
- **The Synchronization Process** - this process enforces synchronization between the execution, planner, and state determination processes. This includes receiving new goals, determining appropriate timeslices for planning and locking the plan database to ensure non-interference between state updates and the planner.

We describe planning, execution, and state determination as sets of processes because often these logical tasks will be handled by multiple processes. For example, spacecraft attitude control execution might be handled by one process, data management by another, etc. However, for the purposes of this paper (e.g., integration of planning and execution), the only relevant issue is that our synchronization strategy can be applied to a multiple process scheme for planning, state determination, etc.

[4] However, an interesting area of research is to automatically determine this via empirical feedback and domain analysis.

The overall architecture for the continuous planning approach is shown in Figure 1. We now describe how each of the four basic components operates.

The planner process maintains a current plan that is used for planning (e.g. hypothesizing different courses of action). It responds to requests to replan initiated by the execution processes, activity commitments from the execution module, state (and resource) updates from state estimation, and new goals (from external to the system). All of these requests are moderated by the synchronization process that queues the requests and ensures that one request is complete before another is initiated. The planners copy of the current plan is also where projection takes place and hence it is here that future conflicts are detected. However, as we will see below, requests to fix conflicts occur by a more circuitous route.

The execution process is the portion of the system concerned with a notion of "now". The execution module maintains a copy of the plan that is incrementally updated whenever the planner completes a request (e.g., a goal change, state change, or activity change). This local copy includes conflict information. The execution module has three general responsibilities:

1. to commit activities in accordance with the commitment policy as they approach their execution time;
2. to actually initiate the execution of commands (e.g., processes) at the associated activity start times
3. to request re-planning when conflicts exist in the current plan

The execution module performs 1 & 2 by tracking the current time and indexing into relevant activities to commit and execute them. The execution module also tracks conflict information as computed by the projection of the planner and submits a request for replanning to the synchronization module when a conflict exists.[5]

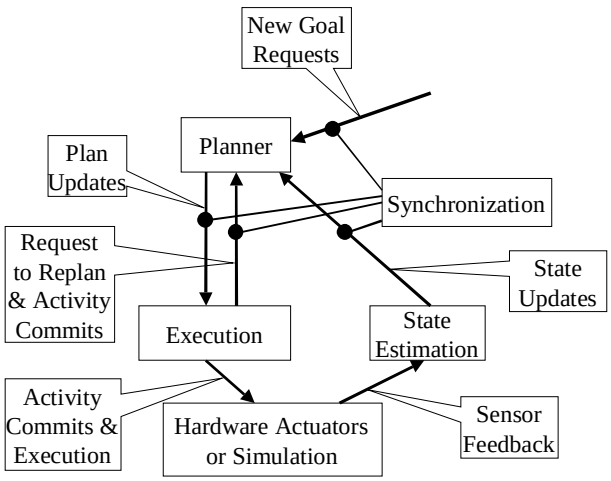


Figure 1: CASPER Architecture

The state estimation module is responsible for tracking sensor data and summarizing that information into state and resource updates. These updates are made to the synchronization module that passes them on to the planners plan database when coordination constraints allow.

The synchronization module ensures that the planner module(s) are correctly locked while processing. At any one time the planner can only be performing one of its four responsibilities: (re)planning, updating its goals, incorporating a state update, updating the execution module's plan for execution, or updating commitment status (otherwise we run the risk of race conditions causing undesirable results). The synchronization module serializes these requests by maintaining a FIFO task queue for the planner and forwarding the next task only when the previous task has finished.

The execution module also has a potential synchronization issue. The planner must not be allowed to modify activities (through replanning) if those activities might already have been passed on to execution. We enforce this non-interference by "commit"-ing all activities overlapping a temporal window extending from now to some short period of time in the future (typically on the order of several seconds). We ensure that the planner is called in a way that each replan request will always return within this time bound and we enforce that the planner never modifies a committed activity. This ensures that the planner will not complete a replan with an activity modified that is already in the past. Additionally, we use the synchronization process to ensure that the Execution module does not commit activities while the planner is replanning. This prevents the planner from modifying activities that have been committed subsequent to the planner call (but still in the future).

ST4 Spacecraft Landed Operations Scenario Validation

Space Technology 4 / Champollion (ST4) is a mission concept for outer solar system exploration involving landing on a comet and returning a sample to Earth. Our scenario focuses on operations of the lander which drills for samples, analyzes them and takes science images. After several days on the surface, the lander will bring a sample back to the orbiter for return to Earth.

In order to test and evaluate our integrated planning and execution approach, we have constructed an ST4 simulation, which accepts relatively high-level commands such as: MOVE-DRILL, START-DRILL, STOP-DRILL,

[5] In our implementation replanning is initiated by the execution module because this allows for the notion of urgency information (e.g. closeness of the conflict to current execution) to be incorporated in the decision to replan. If we did not wish to incorporate this information, the planner module could make this request directly to the synchronization module.

TAKE-PICTURE, TURN-ON <device>, etc. The simulation covers operations of hardware devices. In this test scenario the planner has models of 11 state and resource timelines, including drill location, battery power, data buffer, and camera state. The model also includes 19 activities such as uplink data, move drill, compress data, take picture, and perform oven experiment.

The nominal mission scenario consists of three major classes of activities: drilling and material transport, instrument activity including imaging and in-situ materials experiments, and data uplink. Of these, drilling is the most complex and unpredictable.

The nominal mission plan calls for three separate drilling activities (each consisting of a number of lower-level operations). Drilling rate and power are unknown a priori, but there are reasonable worst-case estimates available. Drilling can fail altogether for a variety of reasons.

In order to validate the effectiveness of our continuous planning approach we have performed a number of empirical tests to measure CASPER performance in terms of:

1. responsiveness - the ability to deal with execution feedback in a timely fashion;
2. robustness - the ability to produce executable plans despite run-time variations in state, resource, and activity durations; and
3. plan effectiveness - a measure of the overall goodness of executed activities (with respect to achieving plan goals).

We assessed these performance metrics using a stochastic version of the ST4 simulation described above. This simulation had a number of random variables, which are described below.

- Compression - we model the compression for science data as a normal random variable with a mean of 0.9 and a standard deviation of 0.25*0.9. This has the effect of forcing the planner to respond to buffer over-runs (as described above) and buffer under-runs (to optimize the plan).
- Drilling Time - we model the amount of time to drill in minutes as a random variable with mean of 30 and standard deviation of 3.
- Drilling power - we model the actual power consumption from drilling in watts as a normal random variable with mean 40 and standard deviation 4.
- Oven Failure - we model oven failure occurrence as Poisson distributed with each oven having a 50% chance of failure over the entire mission horizon.
- Data Transmission Rate: we model the time to transmit data in kilobits per second as a normal random variable with a mean of 100 and a standard deviation of 10. This is intended to model the variability in communications to the orbiter.
- Oven Warming and Cooling Times: we model the amount of time to heat up the sample and for the oven to cool down in minutes as random variables with means of 30 and 120, and standard deviations of 3 and 12, respectively. This is intended to model the unknown thermal properties of the samples.

In our tests we compare the CASPER continuous planning repair approach to two alternative approaches:

1. Batch planning with no feedback - in this approach an operations plan is generated from the initial state and this plan is executed. No feedback from execution is used.
2. Batch replanning on failure - in this approach an operations plan is generated from scratch. When an activity fails, the execution system halts execution and replans from scratch (rather than modifying the existing plan as in the CASPER approach). No activities are executed while the planner is replanning.

In all cases, we compare the approaches using models with best guess nominal estimates for times and resource usage, as well as pessimistic 1-sigma estimates.

In order to assess the responsiveness of the system, we measured the average amount of time from the receipt of an update that required replanning to the time when a conflict free plan is available (see Tables 1, 2 and 3: Time to Correct Plan). In order to assess the robustness of the system, we track the number of times when an invalid activity is commanded (see Tables 1, 2 and 3: Number of Invalid Commands). In order to assess the plan

Overall Performance	# invalid commands	# achieved science goals	time to correct plan (in seconds)
CASPER	2.365	20.063	1.134
Batch planning	54.769	2.194	0
Batch replanning	17.977	6.722	20.125

Table 1 Overall Performance Comparison Averages

Best Guess Performance	# invalid commands	# achieved science goals	time to correct plan (in seconds)
CASPER	2.909	24.677	.978
Batch planning	61.341	2.699	0
Batch replanning	22.112	16.878	17.107

Table 2 Best Guess Performance Comparison Averages

Pessimistic Performance	# invalid commands	# achieved science goals	time to correct plan (in seconds)
CASPER	1.821	15.448	1.291
Batch planning	48.197	1.690	0
Batch replanning	13.842	10.566	23.144

Table 3 Pessimistic Performance Comparison Averages

effectiveness, we measure the science return of executed activities (as measured by number of samples drilled and analyzed in situ where the data was successfully transmitted to the orbiter) 24 science goals are originally submitted to the system, and we report the number completed successfully. (See Tables 1, 2 and 3: Number of Achieved Science Goals).

In our setup, CASPER was running on a Sun Sparcstation Ultra 60 with a 359 MHz process with 1.1 GB Memory. During each of 1000 runs, the simulator updates the plan an average of 18,000 times. (Most of these are battery power level updates.) On average, only 86 updates result in conflicts that should be handled by the planner/scheduler.

We observe that CASPER outperforms batch planning and batch replanning in the ST4 domain in terms of spacecraft commanding and achieving science goals. Note that batch planning requires no time to correct an updated plan because it does not replan, and therefore is superior to CASPER in terms of the amount of time required to correct a plan. However, batch planning suffers considerably due to incomplete data transmissions and spoiled experiments where samples where placed into inappropriately configured or failed ovens. Batch replanning performs much better, but the replan time translates into missed opportunities to plan and schedule science goals. Also, more invalid commands are executed due to the time it takes to replan. CASPER does execute some invalid commands due to the fact that it takes some time to correct an invalid plan, but CASPER achieves far more science goals.

Discussion, Related Work, Conclusions

While the current prototype has been tested on a range of cases in which state updates require replanning, we have focused on execution feedback that cause conflicts in the plan. In the case of the failed oven, buffer over-use, and activity completion time problem, the state update (when propagated through the plan) causes a conflict. There are other cases in which a state update enables a plan improvement. For example,

- battery power usage might be lower than expected enabling insertion of an additional sample activity;
- content-dependent compression might perform better than expected allowing storage of additional experiment data; or
- drilling might be faster than expected again allowing for additional science activities.

In each of these cases, the planner needs to be aware of the potential for improvement in the current plan and be triggered to attempt to take advantage of the fortuitous situation. In related work (Rabideau et al. 2000), we have been developing plan optimization techniques for representing soft constraints (preferences) and improving plans with respect to these preferences (e.g., do more science). Our approach to optimization is an anytime, incremental approach, thus the timeslices for the planner can be used to attempt to improve the plan if there are no conflicts in the plan.

A second issue is that in the current prototype, the planner can only respond to unexpected changes on activity boundaries. This is a significant limitation when there are activities that have extremely long durations. This limitation is because the planner does not have a model detailed enough to predict the resultant state if activities are interrupted in mid-execution. It would be useful if the planner could incorporate a model that could represent interruptible activities and act appropriately. Currently such phenomenon must be modeled by breaking the activity into smaller activities.

While we have tested our prototype on a range of realistic scenarios, we would like to have a larger set of missions and concepts to test against. Because CASPER is currently being used for autonomous rover applications, we are in the process of adapting rover simulations for similar testing. Additionally we anticipate having access to several other spacecraft simulations. We intend to further test and validate our approach against these missions.

Another interesting area for future work is investigating more powerful commitment strategies. One could easily envisage problems in which different classes of activities would have different possibilities for interruption or might be terminatable with sufficient lead-time. Enabling the planner to represent these contexts and handle them appropriately would be desirable.

The high-speed local search techniques used in our continuous planner prototype are an evolution of those developed for the DCAPS system (Chien et al. 1999) that has proven robust in actual applications. Many others have used iterative algorithms for general problems such as traveling salesman as well as scheduling systems(such as GERRY/GPSS (Zweben et al. 1994)).

The OPIS system (Smith 1994) can also be viewed as performing iterative repair. However, OPIS is more informed in the application of its repair methods in that it applies a set of analysis measures to classify the bottleneck before selecting a repair method. With iterative repair and local search techniques, we are exploring approaches complementary to backtracking refinement search approach used in the New Millennium Deep Space One Remote Agent Experiment Planner (ARC 1999).

Excalibur (Narayek, 1998) represents a general framework for using constraints to unify planning and scheduling constraints, uncertainty, and knowledge. This framework is consistent with the CASPER design, however in this paper we have focused on a lower-level. Specifically, we have focused on re-using the current plan using iterative repair and specific locking mechanisms to avoid race conditions.

Work on the PRODIGY system (Cox & Veloso 1998) describes goal alteration from environmental feedback. These changes would be modeled in our framework via task abstraction/retraction and decomposition for

potentially failing activities. Other PRODIGY work (Veloso, Pollack, & Cox 1998) has focused on determining which elements of world state need to be monitored because they affect plan appropriateness. In our approach we have not encountered this bottleneck, our fast state projection techniques enable us to detect relevant changes by noting the introduction of conflicts into the plan.

Work on CPEF (Continuous Planning and Execution Framework) (Myers 1998) uses PRS, AP, and SIPE-2, also represents a similar framework to integrating planning and execution. CPEF and CASPER differ in a number of ways. First, CPEF attempts to cull out key aspects of the world to monitor (as is necessary in general open-world domains). They also suggest the use of iterative repair (they use the term conservative repairs). And their taxonomy of failure types is very similar to ours in terms of action failure and re-expansion of task networks (re-decomposition). However, in this paper we have focused on lower level issues in synchronization and timing.

Work in the O-Plan system has also addressed rapid replanning (Drabble et al. 1997). They describe an approach that generally invokes the planner with the current plan in a repair mode from the current state. In this way their approach and the CASPER one are very similar. However, we have focused on lower-level timing and synchronization issues necessary for execution and planning on a shorter timescale.

3T system (Bonasso et al. 1997) has also examined issues of integrating planning and execution. Again, they present a framework consistent with our architecture but we have focused on lower-level timing issues.

This paper has described an approach to integrating planning and execution for spacecraft control and operations. This approach has the benefit of reducing the amount of time required for an onboard planning process to respond to changes in the environment or goals. In our approach, environmental changes or inaccurate models cause updates to the current state model and future projections. Additionally, the planner's current goal set may change. In either case, if these changes matter (e.g., the current plan no longer applies) they will cause conflicts in the current plan. These conflicts are attacked using fast, local search and iterative repair methods

Acknowledgements

This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. Portions of this work were supported by: the Autonomy Technology Program, NASA Code SM and by the Mission Data Systems Project, NASA Code S, and by the JPL Telecommunications and Mission Operations Technology Program, Mission Services Element. Comments and feedback from Bob Rasmussen, Kim Gostelow, Ton Starbird, Dan Dvorak, Erann Gat, Glenn Reeves, and Ed Gamble were very helpful in formulating and refining the ideas presented in this paper.

References

NASA Ames & JPL, Remote Agent Experiment Web Page, http://rax.arc.nasa.gov/, 1999.

R. P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, M. Slack, "Experiences with an architecture for intelligent, reactive agents. Journal of experimental and theoretical artificial intelligence 9(2).

S. Chien, G. Rabideau, J. Willis, and T. Mann, "Automating Planning and Scheduling of Shuttle Payload Operations," *Artificial Intelligence Journal*, 114 (1999) 239-255.

S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau, "Using Iterative Repair to Improve Responsiveness of Planning and Scheduling," Proc. AIPS-2000, CO, April 2000.

M. Cox & M. Veloso, "Goal Transformation in Continuous Pannning," in Proceedings of the AAAI Fall Symposium on Distributed Continual Planning, 1998.

B. Drabble, J. Dalton, A. Tate, "Repairing Plans on the Fly," Working Notes of the First International Workshop on Planning and Scheduling for Space, Oxnard, CA 1997.

N. Muscettola, B. Smith, S. Chien , C. Fry , K. Rajan, S. Mohan, G. Rabideau , D. Yan, "On-board Planning for the New Millennium Deep Space One Spacecraft," *Proceedings of the 1997 IEEE Aerospace Conference*, Aspen, CO, February, 1997, v. 1, pp. 303-318.

K. Myers, "Towards a Framework for Continuous Planning and Execution", in Proceedings of the AAAI Fall Symposium on Distributed Continual Planning, 1998.

A. Nareyek, "A Planning Model for Agents in Dynamic and Unicertain Real-Time Environments," in Integrating Planning, Scheduling, and Execution in Dynamic and Uncertain Environments, AIPS98 Workshop, AAAI TR WS-98092.

B. Pell, D. Bernard, S. Chien, E. Gat, N. Muscettola, P. Nayak, M. Wagner, and B. Williams, " An Autonomous Spacecraft Agent Prototype," *Autonomous Robots*, March 1998.

G. Rabideau, R. Knight, S. Chien, A. Fukunaga, A. Govindjee, "Iterative Repair Planning for Spacecraft Operations in the ASPEN System," Int Symp on Artificial Int. Robotics and Aut. in Space (ISAIRAS), Noordwijk, The Netherlands, June 1999.

G. Rabideau, B. Engelhardt, S. Chien, "Using Generic Preferences to Incrementally Improve Plan Quality," in Proc. 5th Int. Conf. on Artificial Intelligence Planning and Scheduling (AIPS-2000), Breckenridge, CO. April,2000.

S. Smith, "OPIS: An Architecture and Methodology for Reactive Scheduling," in *Intelligent Scheduling*, Morgan Kaufman, 1994.

M. Veloso, M. Pollack, M. Cox, "Rationale-based monitoring for planning in dynamic environments," Proceedings Artificial Intelligence Planning Systems Conference, Pittsburgh, PA, 1998.

M. Zweben, B. Daun, E. Davis, and M. Deale, "Scheduling and Rescheduling with Iterative Repair," in *Intelligent Scheduling*, Morgan Kaufman, San Francisco, 1994.

COMMENTARY ON

USING ITERATIVE REPAIR TO IMPROVE THE RESPONSIVENESS OF PLANNING AND SCHEDULING

Commentary Author: Mario Merri

ESA/ESOC, Robert-Bosch-Str 5, D-64293 Darmstadt, Germany. Fax: +49 (6151) 903010, E-mail: mmerri@esoc.esa.de

Paper Authors: Steve Chien, Russell Knight, Andre Stechert, Rob Sherwood, and Gregg Rabideau

Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109, {firstname.lastname}@jpl.nasa.gov

Method

As recommended by the workshop organising committee, we have adopted an "open ended" commentary process. This translated by an informal exchange of e-mails between the commentator and the authors of the paper.

In the following, the main findings in the commentary process are reported. Firstly, I briefly introduce my understanding of the paper assigned to me. Subsequently, I summarise the dialogue that took place between the paper authors and myself. Wherever possible, I have tried to use the exact wording of the authors' replies.

Main Findings

Understanding of the Paper

The paper presents an innovative approach to modify almost in real-time the planning of a mission. Actually, the terminology used is to "repair" the planning following the occurrence of events. In this context, I feel that we are more in the field of on-board autonomous system than in that of mission planning proper, although the boundary between them is clearly difficult to define.

The approach presented in this paper is in alternative to the more traditional batch planning where only a very limited reactivity to negative or positive unforeseen events is possible. On the contrary, the proposed dynamic planning is claimed to allow a faster response to unpredicted event with the consequence of, on one hand, having a safer spacecraft, and, on the other hand, maximising the scientific return and the resource utilisation. In fact, the on-board planner starts from a current set of goals, a plan, a current state, and a model of the expected future state. At any time, an incremental update to the goals, current state, or planning horizon may trigger a request for re-planning with the constraint to maintain a consistent and satisfactory plan with the most current information available. Synchronisation between planning and execution is achieved by defining a time window where activities are committed and can no longer be modified by the planner.

Furthermore, the system proposes a hierarchical approach to planning with several levels. Starting from an abstract long-term planning, shorter and shorter planning horizons are considered which include greater and greater details, until finally we arrive to the short time planning. This approach aims at limiting the use of on-board computational and at maximise the system responsiveness by requesting detailed planning only on the short term.

Commentary

General: I have found the topic of the paper very interesting. The paper is also quite clear and reads fine.

Testing: I have pointed out to the author that one of the known critical issue for these types of autonomous system is the complexity of testing. In fact, as the system will have to make decisions in isolation, it is clearly extremely

important to have the system undergoing an extensive testing campaign. It is also well know that testing a mission planning system might be extremely complicated as it is often difficult to cover all possible cases. In this context, I have asked the authors of the paper to provide details on how the testing had being planned, set up and performed in their case.

The author's response was that "testing is planned on series of testbeds, but testing for our system is intended to be similar as for other planning systems:

1. test case generation - based on scenario analysis, etc.

2. regression testing

3. the above can be run on progressively higher fidelity environments

It is true that our approach, because it relies more on environmental feedback, is more likely to have greater execution variability.

However, precisely because of this reliance, it is less reliant on model accuracy."

Safety-Critical Operations: My next question concerned safety-critical operations. Usually, operational staff would like as much as possible to be in control of the execution of safety-critical operations. On the other hand, technology provides today tools and methods to achieve reasonably-safe levels of autonomy. These are clearly two distinguished schools of thought. More specifically, my question to the authors was if, in their opinion, they felt that their system was adequate also for the execution of safety-critical operations or if their concept foresaw that such operations were only handled under strict supervision from ground

The author's response was that "this depends on the ability to characterize system behavior via testing and analysis, and on the risk-aversness of the mission. We see our approach as being comparable in riskiness to traditional batch planning autonomy."

Ground Observability of Timeline Execution Status: We then continued discussing on the dualism between fully autonomous systems versus ground controlled ones. I felt that an autonomous system would stand more chances to be accepted even by "conservative" operational staff if it had the capability to provide operational staff with relevant information on the execution of the autonomous timeline. In this context, I asked if any mission intending to make use of their autonomous planning tool had put observability requirements on the execution status of the timeline. That is to say, if missions have specified mandatory information to be downlinked to ground so that operational staff could know exactly what is going on on-board. Furthermore, in case of positive answer to the previous question, I asked how do they handle these observability requirements.

The author's response was that they "are not that far on the technological maturity to have encountered that issue. In general, our execution and timeline analysis is not restricted by our representation, that is to say that if they could build monitors and state determination software to provide this observability in conventional flight software, we can incorporate such software in our approach. Thus, we do not introduce any additional observability problems."

Autonomy Valid Only for Simple Planning Problems?: The complexity of the planning problem clearly plays a fundamental role in the outcome of the planning system. Furthermore, complex planning problems require complex planning systems that are clearly more prone to errors also considering that it is more difficult to exhaustively testing them. My observation was that, for simple planning problems, the behaviour of the planning system, and thus the status of the spacecraft, is mainly deterministic (see also observability point above). Instead, if the mission planning problem is a complex one, the overall status of the spacecraft is somehow more stochastic, which, in case of problems, might bring to unrecoverable situations.

The author's response was that "Yes - this is certainly true. Indeed, it is precisely these situation that one has two options:

1. make the decision making onboard, with the ability to respond in known patterns of behavior to execution feedback. (our approach).

2. introduce abstraction, worst-case reasoning, and reduce efficiency in order to increase stability/predictability (common approach)"

How Complex is your System?: As follow on of the discussion reported to the point above, I have asked the authors to comment on how their system scale for increasingly complex planning problems and what is their operational concept to recover from timeline errors.

The author's response was that they "have demonstrated the iterative repair on what we would call medium-size problems, and are able to get on the order of 100+ search operators in the planner (i.e., move add, delete activity) on the order of CPU seconds, however, it is always possible that problems from execution require wholesale changes (e.g. beyond the computational ability of the planner to resolve). [Their] approach to dealing with this has been two-fold.

1. develop anytime approaches, which are able to adopt a feasible but poor (quality-wise) solution and improve it as computational resources are available.

2. use abstraction to enable larger-scale changes to take place with fewer search steps."

Conclusions

My personal experience on this exercise was certainly positive as it allowed a direct discussion with experts on the hot topic of autonomous mission planning. Although I am not in a position to express an unconditioned acceptance of all the authors' replies, I certainly feel that this field is very important and requires further investigation. It is clear that, as the space frontier is moving farther and farther from the Earth, this technology becomes more and more required.

As a final remark, I think that the format proposed by this workshop worked quite well and, despite the limited time available, it was possible to establish some interesting technical discussion on stimulating topics.

SOFIA's Choice: Automating the Scheduling of Airborne Observations *

Jeremy Frank
Caelum Research Corp.
NASA Ames Research Center
Mail Stop N269-1
Moffett Field, CA 94035-1000

Why SOFIA's Choice Matters

This paper describes the automated scheduling of observations for an airborne observatory. The problem is to construct flight plans in support of astronomical observations of very distant objects in order to optimize the science output of the observatory. The resulting problem contains a large set of prioritized observations to choose from, and a wide range of complex constraints governing legitimate choices and orderings.

This problem is quite different from scheduling problems which are routinely solved automatically in industry. For instance, the problem contains many interacting complex constraints over both discrete and continuous variables, and the consequences of making some decisions are other decisions to make later. Furthermore, new types of constraints may be added as the problem changes over time. As a result of these features, this problem cannot be solved by traditional scheduling techniques. The problem resembles other problems in NASA, from observation scheduling for rovers and other science instruments to vehicle routing; consequently, it is worthwhile to determine how to address this problem.

In this paper we describe the observatory and the problem of planning flights in support of astronomical observations. We discuss why this is a difficult problem to solve using traditional scheduling techniques, then discuss an approach based on dynamic constraint satisfaction techniques which can address this problem.

SOFIA: The Observatory

The Stratospheric Observatory for Infrared Astronomy (SOFIA) is NASA's next generation airborne astronomical observatory. The facility consists of a 747-SP modified to accommodate a 2.7 meter telescope. Employing a suite of optical, infrared, and sub-millimeter instrumentation, the observatory spans operational wavelengths of 0.3 to 1600 microns. SOFIA supersedes NASA's Kuiper Airborne Observatory (KAO) - a modified C-141 with a 0.9 meter telescope. SOFIA is expected to fly an average of 140 science flights/year over it's 20 year life time, double the previous rate of the KAO. The combination of a factor of nine in telescope collecting area and an approximate factor of two in aircraft flight rate establishes SOFIA as NASA's premier observatory for innovative astrophysical instrumentation throughout the broad wavelength range of the facility. More details on SOFIA can be found in (Becklin 1997) and (Erickson & Davidson 1997).

Building upon the KAO program, SOFIA will routinely provide astronomers with a research platform above 99% of the earth's water vapor. The SOFIA telescope is mounted aft of the wings on the port side of the aircraft and is articulated through a range of 20 to 60 degrees of elevation. Most flights will originate and terminate at Moffett Field, CA; therefore, it is necessary for the observatory flight plans close in on themselves. This typically requires an astronomical observing plan covering both Galactic and extra-galactic targets.

In this paper, we are concerned primarily with the problem of scheduling flights in support of the General Investigator (GI) program. GIs are expected to propose single observations, and many observations must be grouped together to make up single flights. The SOFIA science staff is expected to have 3 - 4 facility science instruments to support GIs. The scope of the flight planning problem for supporting GI observations with the anticipated flight rate for SOFIA makes the manual approach for flight planning daunting. There has been considerable success in automating the scheduling of jobs in a wide variety of industries with many different types of constraints. However, these problems are typified by relatively simple, homogeneous constraints, and the successful approaches depend on these simple representations. In this paper, we describe an approach to solving more complex scheduling problems which can address the difficult problem of planning flights.

Planning for a Single Flight

The basic problem for an airborne observatory like SOFIA is the *Single Flight Planning Problem* (SFP). This problem consists of constructing a good flight plan for a single flight on a given day. The problem input consists of the set of *observations* that have been requested, the constraints peculiar to the *flight environment*, and the *objective function*.

*I would like to thank Ari Jónsson, David Smith, Sean Casey, and Jesse Bregman for their help in writing this paper.

The Observation Requests

An observation request consists of the name and coordinates of the object to be observed, the amount of time requested, the relative importance of the observation, and a set of constraints on the observation. We assume that the amount of time is fixed and also that it is strictly less than the maximum duration of the flight. The importance or priority of the observation is a summary of several different factors. Some observations are naturally more interesting to the science community than others. Additionally, due to the limited duration of flights, it may be necessary to observe a target many times, and thus important to finish a sequence of observations on a target than to start a new observation. Observation requests have the following constraints:

Ordering Constraints Some operations must be performed in a pre-specified order. For example, instruments may need to be calibrated by observing particular objects before the primary observation of interest is performed. In addition, the telescope may need to be tuned at the beginning and periodically during the flight by observing objects with particular characteristics. High-precision tuning may require observing the same object at multiple elevations, for instance. These requirements impose ordering constraints on the observations that must be obeyed [1].

Astronomical Constraints Some objects may only be visible from certain positions on the earth at certain times of day, resulting in an earliest start time and a latest end time for completing a given observation request. Astronomers may also provide explicit constraints on particular observations so that the data is of high quality. For example, the astronomer may require that the object be sufficiently far away from the moon or the sun, or that airmass or atmospheric water vapor be below a certain threshold. These constraints also dictate when a target may be observed. In particular, minimizing airmass requires observing at a higher altitude, and minimizing water vapor can be accomplished by observing at higher altitudes or by observing further north (Horn & Becklin 2000).

Aircraft Constraints SOFIA has complex constraints simply because it is an airborne observatory. Most objects appear to move through the sky as time passes. Because the telescope has little horizontal flexibility, the aircraft must fly a curved trajectory in order to keep objects in view. The wind speed, aircraft speed, and time and position an observation is started dictate the trajectory and final location of the aircraft at the end of the observation. Object visibility windows are further constrained by the limits on the telescope's angle of elevation. Even though an object may be visible from the ground, it may not be visible from the aircraft, either because it is too low or too high for the telescope to view. An object may sometimes have multiple windows of visibility during a single flight. For example, it may pass above and then below the maximum telescope elevation, requiring a choice of when to observe.

The aircraft must normally return to the airport it took off from. Flight time is limited by fuel, requiring all observations to be done with enough time for the aircraft to return, from wherever it is, to the airport. Finally, the aircraft's altitude is constrained by its weight, but the weight decreases over time as fuel is consumed, so the aircraft can generally climb an additional 2000 ft every two hours. These factors can interact with constraints on airmass or water vapor to further limit possible observing times.

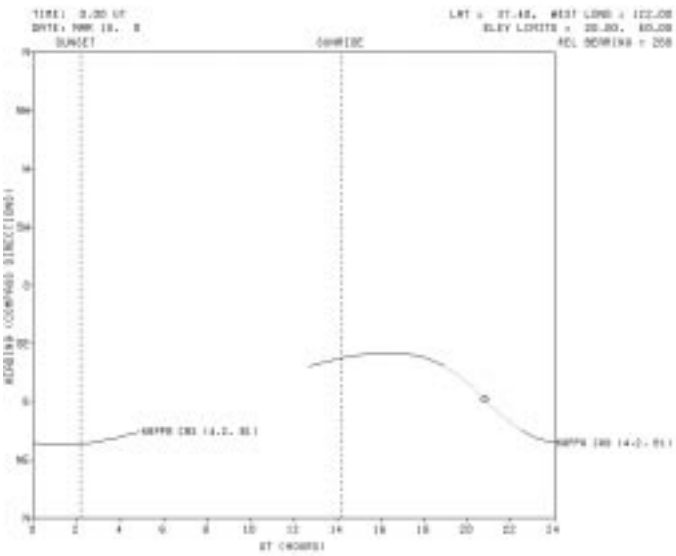


Figure 1: Visibility of an object during March 18, 2000 from Moffett Field. Note that time is given in Universal Time, and that sunset and sunrise are marked.

Many of these constraints are demonstrated in Figure 1. This figure shows visibility and heading information for an object viewed from Moffett Field during March 18, 2000. The y axis shows the heading the aircraft must fly to keep the object in view. The direction changes over time, indicating that the aircraft must constantly turn to keep the object in view. The curves on the plot indicate when the object is in view. No curve indicates the object is below the telescope's 20 degree minimum elevation, while the dotted curve indicates the object is above the 60 degree maximum elevation. Notice that the object passes below the minimum elevation and then returns to view then passes above the maximum elevation and again returns to view. During any 9 hour period, there are at most two windows of visibility for this object.

The Flight Environment

In this section we discuss constraints derived from the environment on the day of the flight. One important example of such constraints are airborne warning zones, commercial flight routes, and other administrative restrictions on where the aircraft can fly. Some flight environments may have fewer such restrictions;

[1] While calibrations and setup operations are not strictly observations, we represent them this way for convenience.

for example, if the aircraft flies out of Hawaii or New Zealand, there will be fewer such restrictions than flights over Nevada. Bad weather may constrain observations as well. While cloud cover is usually not an issue at the altitudes where observing is likely to occur, turbulence can affect the performance of the observations, and may increase observation time or have other effects on flights. Wind speed and direction can also have an effect on a particular flight. The aircraft's ground speed is directly affected by wind, and wind patterns change over time. Flight planners must take these effects into account when doing planning.

The Objective Function

The final component of the SFP is the objective function, which is used to compare two candidate flight plans. Within the confines of the single flight planning problem, the objective function can range in complexity. A good flight contains as many high-priority observations as possible; hence a good objective function might be to simply sum the priorities of the observations which are performed. Astronomers may also *prefer* rather than require that observations be done at various water vapor levels, that targets be observed when they are far from the moon or other heavenly bodies, and so on. All of these preferences can then be added to the objective function, resulting in a fairly complex measurement of the quality of a flight plan.

The Statement of the Problem

With all the components in place, we can now state the SFP: given a set of observations to perform, a date to perform them, a description of the environment on that date, and the objective function, select a take-off time, a subset of the observations, a start time for each observation, and specify any dead-legs (i.e. flying without observing). The resulting flight plan must not violate any of the constraints and should optimize the objective function. Figure 2 shows an example of a flight plan.

The Complexity of Flight Planning

Many efficient scheduling techniques rely on special encodings of problems in order to deliver good computational results. These techniques work only for simple constraints, such as equality and inequality or simple combinations of resource and precedence constraints. For example, many clever encodings and algorithms exploiting them can be found in a recent text on scheduling techniques (Brucker 1998). While these techniques are very efficient, they have limited application, and a great deal of sophisticated modeling may be necessary to pose problems in the correct form.

As we have seen, any instance of the SFP is composed of a large number of complex, heterogeneous constraints over both continuous and discrete variables. Even relatively simple versions of the SFP contain geometric constraints, precedence constraints, mutual exclusion constraints and temporal constraints, all in the same problem. While it may be possible to encode parts of the problem in order

Figure 2: A flight plan. Each observation leg is marked with +s and labeled, while dead legs are simple lines. Note that no restricted areas are indicated in this figure; this flight plan is likely to cross restricted areas over Nevada.

to take advantage of efficient algorithms, it is unlikely that we can find a good encoding which will serve for all instances of the problem. Furthermore, over the lifetime of the observatory, other constraints may be required to represent a flight planning problem. For example, new instruments and newly discovered objects may impose new constraints. The addition of these constraints may invalidate any specialized representations and algorithms.

The SFP presents another challenge. In particular, it is not known beforehand how many observations will be performed on any given flight, nor is it known how many dead-legs will be required. Each choice made during the planning and scheduling of the flight may explicitly create other choices; for example, a decision to observe an object at a particular time and place could be preceded by a dead leg to reach that position, or observing another object beforehand. Traditional scheduling techniques require *all* of these choices to be explicitly represented, along with constraints which are triggered once the choice is made. Encoding all of this information results in large representations. The problem only becomes worse as the scope of the problem grows; thus, while it may be possible to encode a small SFP, it may not be possible to encode a larger one because the encoding will not fit into the computer's memory.

Automatic Generation of Flight Plans

In this section, we describe an approach to representing and solving the SFP. This approach is based on the Constraint-Based Interval Planning (CBIP) paradigm, which is described in (Smith, Frank, & Jónsson 2000) and which was successfully employed in the Remote Agent Planner (**?**).

Constraint Satisfaction and Optimization Problems

Constraint Satisfaction Problems or CSPs are an approach for representing many problems, including scheduling problems (both (Beck & Fox 1998) and (Nuijten 1994) both represent and solve scheduling problems as CSPs). A CSP consists of a set of variables, each of which has an associated domain of legal values it can take on in a solution, and a set of constraints, which restrict the legal assignments to sets of values. These constraints can be extensional, in which all the legal assignments are listed, or intentional, in which constraints are written as a mathematical relationship. A constraint is *satisfied* by an assignment if the values assigned to the variables are permitted by the constraint. A *solution* to a CSP is an assignment of each variable to a single value in its domain such that all the constraints are satisfied.

CSPs contain no mechanism to express a preference between two solutions that satisfy all of the constraints. A *Constraint Optimization Problem* or COP is a CSP which includes a mapping from a solution to the real numbers. This mapping encodes the preferences between solutions which satisfy all the constraints. It should be clear from the discussion of the SFP above that we can pose the SFP as a COP.

The representation of a CSP may be unwieldy due to the large number of constraints required to encode the conditional effects of all of the choices. An alternative representation is the *Dynamic Constraint Satisfaction Problem* or DCSP. A DCSP is a sequence of CSPs, in which each CSP is a modification of the previous CSP in the sequence. A CSP C is said to be a *relaxation* of a CSP D if C has fewer constraints, fewer variables or more combinations of assignments permitted in its constraints and a *restriction* if C has more constraints, more variables or fewer combinations of assignments permitted in it's constraints. These ideas are formalized in (Jónsson & Frank 1999). DCSPs provide a way to formally characterize how a problem changes over time, and require less space since the impact of choices need not be encoded in a single representation of the problem.

Procedural Constraints

The notion of a constraint is very general, and many real-world problems can be represented very naturally with simple constraints. However, sometimes representations of problems using simple constraints can become large and unwieldy. For instance, an enormous amount of space is required to explicitly represent all of the possible time-location pairs when an object is visible. It is often more efficient to represent a constraint with a mathematical relation such as $\leq$. *Procedural constraints* (Jónsson 1996) generalizes this concept by formalizing the notion of a procedure which enforces a relation among the variables of a constraint. A special form of procedural constraint called an *elimination procedure* is permitted to get rid of any element of a domain that is provably not part of any solution to the problem, given the information currently at hand. For example, an elimination procedure might eliminate all observations as candidates for the next observation on a flight because the aircraft is almost out of fuel. Procedures also are used to formalize the concept of *decision variables*. If a procedure is able to assign values to a set of variables $V - D$ given a consistent assignment to the variables in D, then there is no reason to search the values of variables in $V - D$. The variables in D are called the decision variables, since those are the only variables which require making decisions. Continuous variables can be handled by making sure they are not in the set of decision variables (Jónsson & Frank 1999).

Constraint-Based Interval Planning

CBIP is a general planning framework that uses a model to specify the domain in which the planning is to take place. The elementary notion in this framework is that of an *interval*, which denotes an activity or state which is maintained over a period of time. Each interval is described using a set of DCSP variables. The temporal aspects of a interval are described using variables that represent the start time, the end time, and the duration of the interval. Actions and states can have parameters that further specify the action or fluent, so the state value described by the interval is also described by a set of DCSP variables. An interval consists of the name of an action or state, and some parameters of that action or state. The model consists of *planning schemata*, which specify relations between intervals, enforcing preconditions, effects, enabling conditions, mutual exclusions, etc. These schemata then give rise to constraints between the different DCSP variables that describe the different intervals. A set of connected intervals, i.e, a candidate plan, gives rise to an instance of a constraint network. We employ procedures as described in the previous section to enforce some of the more complex constraints.

The CBIP Representation of the SFP

In this section we describe how to represent the SFP in the CBIP framework [2]. Let us assume that the problem consists of a set of observation requests, and our task is to construct a flight for a particular day such that the sum of the priorities of the observations performed exceeds a certain threshold. The durations of the observations are fixed, but we will permit dead legs in this problem. For simplicity, we ignore restricted zone constraints and artificially restrict the bearings of dead legs to the 4 cardinal directions, and restrict dead leg flight duration to 5, 10, 15 or 20 minutes.

Let us suppose that as part of the SOFIA domain we want to represent calibration operations for the instruments on board the aircraft. We would then write a planning schema for a Calibrate action. In the schema below, variables are preceded by a ?. The schema might look like this:

Calibrate(?loc,?Target,?Instrument,?NextOp):-
$CalibrateTime(?dur, ?Target, ?Instr)$
$LocationAfterCalibrate(?loc, ?endloc, ?dur)$

[2] We have considerably simplified the presentation, and the resulting flight plans would require post-processing before final approval.

contained-by	Status(?Instr,ON)
contained-by	Calibration(?Target)
contained-by	Observable(?Target)
meets	Calibrated(?Instr)
$Eq(?Next, Obs) \rightarrow$	meets Observe(?any,?endloc,?any)
$Eq(?Next, DLeg) \rightarrow$	meets DeadLeg(?endloc,?any)

The line $CalibrateTime(?dur, ?Target, ?Instr)$, indicates that the instrument, duration of the operation, and target used to perform the calibration are mutually constrained. Subsequent lines specify temporal relationships between the calibration event interval and other intervals. For example, contained-by Status(?Instr,ON) indicates that the instrument must be turned on for an interval encompassing the calibration interval. Finally, the last two lines specify disjunctions over intervals that may be related to the calibration event interval. For example, the value of the ?NextOp variable indicates whether an observation interval or a dead-leg interval follows the calibration interval.

As planning decisions are made, variables and constraints arising from new intervals are added to the constraint network, and variables and constraints stemming from intervals that are no longer part of the plan are removed from the constraint network. Figure 3 shows how a partial plan translates to an induced DCSP.

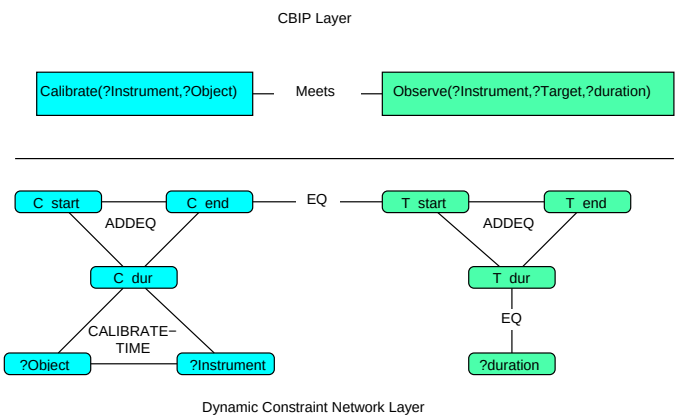


Figure 3: A partial plan and the induced DCSP. As commitments are made to steps in the plan, new intervals are generated and new variables and constraints are added to the DCSP.

At this point we mention how domain specific constraints are integrated seamlessly into the description of the planning domain. For example, $CalibrateTime(?dur, ?Target, ?Instr)$ may be a very complex relationship between the duration, target, and instrument. However, once $CalibrateTime$ is written as an elimination procedure, it can be specified as part of a plan axiom as described above. This gives modelers the flexibility to decide how sophisticated and powerful an elimination procedure they wish to provide. We also observe that it is now possible to make restrictions such as those we demand on consecutive dead legs. In this case, we would simply write into the plan schema for dead legs that they must be followed by an observation leg. As a final point, we observe that it is relatively straightforward to write a procedural constraint which handles the flight plan threshold criteria; as each new observation leg is posted, we can update the inputs to the optimization function and compute the new value of the flight plan.

Finding Plans

Finding plans in the CBIP framework involves solving the DCSP which is created by specifying the initial state of the particular planning problem. This is normally accomplished using *backtracking search*, which constructs a solution to a DCSP by selecting a variable from the remaining unassigned variables in a problem, then trying each possible value in turn. If at any point a constraint violation is detected, the procedure returns to the previous variable binding, and tries another value. If all the values of a variable are tried without success, then the procedure also returns to the previous variable and tries another value.

We pose a particular instance of the SFP by specifying an interval for each observation which could be performed on a particular night, and specifying intervals reflecting the takeoff and landing legs of the flight plan. The commitments to variable bindings will establish which observations are made, in order, after takeoff and throughout the flight. Plan axioms can specify that certain intervals exist, but this requirement can be satisfied by either creating a new interval, or making use of a pre-existing interval in the current plan; these choices are also reflected as variable binding decisions. For instance, suppose that the planner must decide what to do about an observation after a calibration. Either the planner can insert a new observation into the plan, or determine that the observation following this calibration is some observation already in the plan, thereby satisfying the axiom. To ensure that the planner does not try and make all observations, we permit the planner to "disable" an observation by eliminating it's interval. The optimization criteria will prevent the planner from creating a legal flight plan with no observations.

This conceptually simple algorithm is guaranteed to solve a DCSP or demonstrate that no solution is possible. The worst-case running time for this procedure is the product of the sizes of the domains of all the variables, which is exponential in the number of variables. Consequently, backtracking is only possible for DCSPs with discrete domains. Backtracking can be easily modified to solve DCOPs by saving the value of the best solution found, and searching over all solutions instead of halting after finding the first solution satisfying the constraints. In essence, this is like imposing a new bound on solution quality each time the old bound is improved upon. Figure 4 shows a simple backtracking algorithm for solving the SFP by complete search. Note the step in which the next CSP is generated; this is accomplished by consulting the plan schemata.

The performance of this algorithm depends dramatically on functions which select the next variable to choose, select the order in which to try values, perform fast inference to eliminate values from the domains of unbound variables, and decide which variable binding decision is responsible for a constraint violation. For instance, in the SFP, there is often

```
procedure FindSFP(P)
    generate next CSP, P'
    if a constraint is violated return fail
    if problem solved return success
    select an uninstantiated variable V
    for all values of this variable v ∈ dom(V)
      if FindSFP(P' ∪ V = v) = success
        return success
    end for
    replace P' with P if necessary
    return fail
end
```

Figure 4: Finding a simple flight plan.

a choice concerning which observation to make for a given observation leg. Since the goal is to exceed a bound on the priority, a good choice might be to select the remaining observation with the highest priority to try next. However, if this observation is too long or takes the aircraft in the wrong direction, other observations will not be possible and a poor quality flight plan will result. Consequently, modifying this choice by checking the direction the aircraft must fly may lead to better flight plans in less time. It should also be clear that using procedural constraints to eliminate bad choices for variables can save time, since the algorithm does not need to guess these values. These modifications are critical to good algorithm performance; for results on traditional scheduling problems, see (Beck & Fox 1998) and (Nuijten 1994).

It should be clear from the discussion of the constraints that flight planners must analyze several tradeoffs when scheduling flights. For example, an obvious tradeoff concerns whether to try a high-priority observation first, or to try an observation of lower priority which may be easier to schedule. Another tradeoff concerns when to schedule a particular observation. If the observation is constrained by a minimum water vapor threshold, for instance, then this may be satisfied by flying higher or further north (Horn & Becklin 2000). However, these both require making observations later in the flight, and care must be taken to ensure that the aircraft can still return home. Tradeoffs such as these drive the construction of both variable and value ordering heuristics, which are necessary to ensure good algorithm performance.

Related Problems

The SFP is similar to other problems important for both NASA and industry. For example, the Vehicle Routing Problem (VRP) is the problem of delivering packages to various destinations in an urban area(Kilby, Prosser, & Shaw 1999). This problem does not have the complex geometric constraints that the SFP features, but ordering constraints, package size, fuel constraints, truck capacity and distances all interact to make for a complex scheduling problem. Scheduling operations for planetary rovers and interplanetary vehicles requires sequencing science observations and their enabling activities such as sample acquisition, as well as operations to aid in navigation as well as re-charging operations. Furthermore, the total number of operations must be within the available power budget of the vehicle. Finally, path planning and astronomical navigation feature complex geometric constraints which may require simulation.

Conclusions

The flight planning problem motivated by the SOFIA GI program features many heterogeneous constraints over a mixture of continuous and discrete variables. The resulting problem is further complicated by the fact that the problem includes an uncertain number of steps and other conditional constraints that can be very expensive to encode in a single problem instance. These factors lead us to the conclusion that traditional scheduling techniques which solve simple scheduling problems are, by themselves, likely to be inadequate. We show how to represent the problem as a DCSP with procedural constraints. This problem can then be solved by a complete search algorithm using the procedural constraints to efficiently eliminate assignments. Only by doing so can we ensure that SOFIA's choice will be made correctly.

References

Beck, J. C., and Fox, M. 1998. A generic framework for constraint-directed search and scheduling. *AI Magazine* 19(4):101–130.

Becklin, E. E. 1997. Stratospheric observatory for infrared astronomy. *Proc. of ESA Synmposium* The Far Infrared and Submillimetre Universe, *ESA SP-401* 201–206.

Brucker, P. 1998. *Scheduling Algorithms.* Springer-Verlag.

Erickson, E. F., and Davidson, J. A. 1997. Sofia: The future of airborne astronomy. *Proc. of the Airborne Astronomy Symposium on the Galactic Ecosystem: From Gas to Stars to Dust* 73:707–732.

Horn, J. M., and Becklin, E. 2000. Optimized flight planning for sofia. In *Proceedings of the SPIE*, volume 4014.

Jónsson, A., and Frank, J. 1999. A framework for dynamic constraint reasoning using procedural constraints. In *Proceedings of the Workshop on Constraints and Control, held in conjunction with the 5th International Conference on Principles and Practices of Constraint Programming.*

Jónsson, A. K.; Morris, P. H.; Muscettola, N.; and Rajan, K. 1999. Next generation remote agent planner. In *Proceedings of the Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS99).*

Jónsson, A. 1996. *Procedural Reasoning in Constraint Satisfaction.* Ph.D. Dissertation, Stanford University, Stanford, CA.

Kilby, P.; Prosser, P.; and Shaw, P. 1999. A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints. *Journal of Constraints, to appear.*

Nuijten, W. 1994. *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach.* Ph.D. Dissertation, Eindhoven University of Technology.

Smith, D.; Frank, J.; and Jónsson, A. 2000. Bridging the gap between planning and scheduling. *Knowledge Engineering Review* 15(1).

Comments on "SOFIA's Choice: Automating the Scheduling of Airborne Observations"

Sanford M. Krasner
skrasner@jpl.nasa.gov
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91001

Abstract

The author comments on the above-mentioned paper, also included in these proceedings. The author of these comments is not an authority either on planning or on the SOFIA observatory airplane, so these comments do not address the feasibility of the suggested approaches. Comparison is made to similar problems in the planetary-spacecraft domain. Concerns are raised about the level of overlap required between technical domain experts and planning domain experts, but no recommendations are made. These concerns are a focus of ongoing research in the planning community, and some tools exist to address them.

Commentator's Background, or "What gives you the right…?"

The commentator is not an authority on the planning and scheduling domain, and is not an authority on observatories, so he does not have any comments directly on the feasibility of the planning strategies proposed. The commentator has extensive experience in the development of flight software for planetary missions, and has related experience with the mission designs, mission activities, mission planning and operations. These missions pose analogous planning problems to the SOFIA mission.

Analogies to Planetary Mission Planning Problems

Planetary missions pose planning problems analogous to the SOFIA mission, although the specific constraints and models are different. For instance, on the New Millennium Deep Space 1 (DS1) mission, the science team had to schedule an asteroid encounter sequence. An asteroid encounter is a short time period (usually, hours) during which the mission will try to pack in as many observations as possible. Some constraints on the mission plan:

- Some observations can only be taken at certain distances from the target, or under certain lighting (phase-angle) constraints.
- Since not all instruments share the same boresight, shifting from one instrument to another, or from one target on the asteroid to another requires time, during which other observations cannot be taken.
- Some types of observations (imaging spectrometers, particles and field experiments) require slewing relative to the target, while others (imaging) require staring at a single target.
- There are points during the closest encounter when the spacecraft cannot slew fast enough to track the target.
- Onboard data storage is limited. Storage space can be reclaimed by downlinking data to Earth, but this takes time away from data collection.
- Orbital tours have similar types of constraints, combined with orbital constraints on when particular targets are visible.

Because of these analogous planning problems across different problem domains, it would be desirable to come up with representations and solutions that are not closely coupled to particular domains.

Applicability of MDS Concepts

The commentator is currently working as the Guidance, Navigation, and Control lead for the Mission Data System (MDS) project at the Jet Propulsion Laboratory. MDS is developing a framework to describe planetary mission system in terms of the states of the system, how they are estimated and how they are controlled. States are controlled by imposing goals on the states, which constrain their acceptable values over certain time periods. A temporal constraint network is used to enforce the precedence & simultaneity dependencies among various goals.

Imposition of goals results in a recursive elaboration process, where additional states and subgoals are imposed to support higher level goals. This elaboration establishes a hierarchical decomposition of the mission system, and allows designers to decouple the knowledge of different components of the system. Elaboration allows the software system to "fill in" a high-level mission plan by adding detailed configuration requirements and preparatory

activities. Elaboration also provides a framework for implementing the flight rules that are typically found in a spacecraft design. (Flight rules include requirements on preparations for certain activities, such as turning on and warming up equipment. They also define mutually exclusive activities, such as the inability to point an instrument in two directions at once.) This elaboration, coupled with models of the behavior and constraints on each state, allows an MDS-based software system to capture domain knowledge and to evaluate the feasibility of a proposed plan and its elaboration. If no conflicts (violation of flight rules) are discovered in the elaborated goals on any state, the plan is legal.

Analysis of the system into its constituent states also simplifies the job of extending the system models. New types of activities can be added as new goals on existing states, making use of existing models. Unfortunately, this process does not give any hints on how conflicts can be resolved.

Overlap between Domain Experts and Planner Experts

Apparently, the typical process of designing a planner requires the implements to gather heuristics from the domain experts. The implementers then represent these heuristics, using tools supplied by a planner engine, such as HSTS. Unfortunately, this requires extensive interactions between domain experts and planning experts, and requires the planning experts to become domain experts (or vice versa). I understand that more automated heuristic gathering is an open research area. It would also be desirable not to rework the planner when new observations are added. With current planner designs, addition of new observations requires addition of new heuristics, and redesign of some portion of the planner.

Planner-Imposed Restrictions on Domain Modeling

The available search methods apparently place severe constraints on the types of constraints that can be represented in a planner. In the SOFIA problem, the plane can only fly in cardinal directions (North, South, East, and West) in order to discretize the possible directions. This is an artificial constraint on the system modelled, and requires domain experts to understand the limits on representation imposed by the planning system.

No Surprises, No Silver Bullets

According to the author of this paper, these are not novel observations, and they are active areas of research. I look forward to learning more about the HSTS system and planning, in general, in the interactions at the Planning & Scheduling Workshop.

Toward Scheduling Over-Constrained Remote-Sensing Satellites

Joseph C. Pemberton
pemberto@psrw.com
Veridian Pacific-Sierra Research
Arlington, VA 22209

Abstract

The high demand for satellite imagery and the relative scarcity of commercial imaging satellites combine to produce scheduling problems in which one or more of the resources is over constrained. Over-constrained scheduling problems can be problematic for constraint programming methods because of the overhead associated with propagating constraints and the time spent backtracking over a large number of tasks that cannot be scheduled. We present a hybrid approach called priority segmentation that breaks the scheduling problem up into sub-problems based on the priority of the tasks, finds an optimal solution to each sub-problem, and then combines the sub-problem solutions. Our preliminary results indicate that priority segmentation produces solutions that are an improvement over a greedy solution when the problems are highly over constrained. When the problems are only slightly over constrained, the percent improvement over greedy is very small. Our results also indicate that greedy solutions are competitive with more computation intensive approaches under certain problem conditions.

Introduction

Satellite scheduling is the problem of mapping tasks (observations, communications, downlinks, control maneuvers, *etc.*) to resources (sensor satellites, relay satellites, ground stations, *etc.*). Although the term satellite scheduling has been applied to many different aspects of a satellite's operation (*e.g.*, design planning, launch control, lifecycle, *etc.*), our work has largely focused on scheduling mission operations, namely the day-to-day activities of an operational satellite. Mission operation activities include: payload operations (*e.g.*, using a sensor to collect data), bus operations (*e.g.*, maintaining the health and status of the vehicle), and communications operations (*e.g.*, transmitting data between satellites or from satellites to the ground and receiving information or commands from a ground station). The rest of this paper will focus on scheduling of payload operations for remote-sensing satellites – namely deciding which observations to perform.

Satellite mission operation schedules are typically over constrained because the commercial demand for satellite imagery to date has far exceeded the capabilities of commercial satellite systems. For this paper, the term "over constrained" refers to scheduling problems in which the capacity required by the tasks exceeds the total available capacity for at least one resource in the problem.

To further quantify this property, we have adopted the following definitions: The *average constrainedness* of a scheduling resource is the ratio of the sum of the resource capacity required by all tasks to the total capacity available. The *average constrainedness* of a scheduling problem is the maximum average constrainedness over all resources in the problem.

When the average constrainedness of a resource is greater than one, we say that the resource is over constrained. Note that a resource can be over constrained in one part of the scheduling problem and then under constrained in another part. In cases where the level of resource constrainedness differs greatly from one part of a problem to another, it may be necessary to evaluate the constrainedness of the different parts of the problem independently. For this paper, we have focused on problems where the average constrainedness of a resource does not vary significantly from one part of the scheduling problem to another.

The paper is organized as follows. Section 2 describes the remote sensing scheduling problem and presents a simplified satellite imagery scheduling problem. Section 3 describes a constraint programming approach to solving this class of scheduling problems and discusses why this approach can be problematic when the scheduling problem is over constrained. Section 4 describes a hybrid method that we have used to address over constrained scheduling problems. Sections 5 and 6 describe our preliminary experiments and initial results. Section 7 discusses the results and related work. Finally, Section 8 contains some conclusions and plans for future work.

Remote Sensing Scheduling

The term remote sensing refers to the class of satellite missions in which a satellite is used to make observations. The observations can be of either earth-bound or interstellar objects. Due to the high cost of building,

launching and maintaining remote sensing satellites, it is typically the case that more than one user will share the use of a satellite's sensing capability. It is also typically the case that the demand for use of the satellite will exceed its capacity. This means that the scheduler must choose which subset of the tasks to execute. A good schedule is a tradeoff between the constraints of the problem and the needs of the users.

As an example, a group of farmers in Kansas has banded together to perform a land use analysis of their farmland and the surrounding communities to better assess the impact of the current drought. They have contacted a remote sensing satellite operator and agreed to purchase a set of satellite images that will be taken over the next three days. The farmers set a maximum three-day spacing between images to ensure that any unexpected storm would not have time to affect the crop growth assessment.

The satellite system's ability to perform the farmers' tasks is constrained by several factors. First, the ground location must be visible to the satellite. Second, the quality of the image typically depends on the angle between the satellite and ground (the aspect angle). Typically, the best quality (highest resolution) picture is taken when the satellite is directly over the position to be imaged. For our example, the farmers might have specified a minimum quality for their images. The farmers could have also asked for stereo images, which are created by taking two separate images of the same ground location from slightly different viewing angles. To satisfy the constraints of a stereo image, the satellite must take two images within a few minutes of each other, and it must take both pictures in order to satisfy the stereo requirement.

The physical characteristics of the satellite components can also constrain the execution of tasks. The most common examples are power, momentum, slew, heat and memory. Slew refers to the movement of a satellite sensor that is necessary to set it up before a task is executed. In most cases satellite operators prefer to minimize the amount of slew because the time and power spent slewing the satellite is time and power that could be spent performing revenue-generating tasks. A satellite scheduling system must consider all of these factors at some level in order to produce a schedule that satisfies the problem constraints and also addresses the objective functions (*e.g.*, minimize slew, maximize profit, *etc.*).

Another important source of constraints is users' requirements. Since we have already assumed that the satellite will not be able to perform all tasks in the time available, the scheduling problem becomes an optimization problem. Operators of commercial remote sensing satellites want to maximize their profits; therefore, they are interested in generating schedules that maximize the amount of money that they can collect from their customers. For our example, the farmers might decide to pay only if all of the images are taken because anything less would make their analysis incomplete. Alternatively, they might create a priority ordering for the imaging tasks and agree to a cost scale based on the priority (*i.e.*, to pay more for higher priority tasks).

The remote sensing scheduling problem can be summarized as follows. The tasks are observations to be performed by the satellite(s). Each task has a start time and duration. Tasks require resources (*e.g.*, the satellites camera, power, *etc.*). Resources have finite capacity (*i.e.*, a camera can only take one picture at a time). When it is not possible to schedule every task, the quality of the schedule can be measured by an objective function (*e.g.*, maximize the number of tasks scheduled, maximize total revenue, minimize slew, *etc.*).

A Constraint Programming Approach

We have adopted a constraint programming approach to satellite scheduling problems. We model the satellite-scheduling problem using four basic objects: tasks, resources, events and constraints. Tasks are the activities and operations to be performed. Resources are the people, satellites, sensors, communication channels, *etc.*, that are required to accomplish tasks. Events are used to capture domain-specific occurrences that restrict when tasks can be scheduled (*e.g.*, satellite visibility windows). Constraints are further restrictions on when tasks can be scheduled due to interactions with other tasks and to resource capacity and availability. When the problem is over constrained, we assign a priority value to each task. The priority values can then be used by the scheduler as part of an objective functions (*e.g.*, optimize the priority sum of the scheduled tasks). For commercial imaging tasks, the task priority could be directly related to the revenue of the task.

A task is scheduled by assigning it a resource, start time and duration, which satisfy all of the relevant constraints. Tasks that require more than one resource (*e.g.*, a downlink task requires a satellite and a ground station) will need to have more than one resource assigned to it. More formally, a scheduled task is a tuple:

$$(task,\ \{resource\ set\},\ start\ time,\ duration)$$

In cases where the specific set of resources needed to perform a task are known *a priori*, then the scheduling problem is reduced to finding an execution time and duration for each task. When each task has a fixed duration, then the scheduling problem is further reduced to finding an execution time for each task. If the task start times are also fixed, then scheduling becomes a problem of deciding which tasks to perform. We will discuss the last problem class further in later sections.

We map satellite scheduling problems into constraint programming problems by associating a variable with each task. The values correspond to the start time, end time and resources assigned to the task. In general, our constraint-based scheduler explores the space of possible schedules by choosing a task to schedule and then selecting values for the start time, end time and resource assignments. Our general purpose constraint-based scheduler is built on top of ILOG Solver and

Priority Segmentation:

1. Sort the set of tasks by their priority value.
2. Select the n highest priority tasks not yet scheduled.
3. Find an optimal schedule for the n tasks given the resource constraints and previously scheduled tasks.
4. Lock the tasks that were successfully scheduled and remove the tasks that could not be scheduled.
5. Loop to 2.

Figure 1: Priority segmentation algorithm.

Scheduler, which performs the constraint propagation and directs the search based on the heuristics that we have designed (*e.g.*, the value and variable selection heuristics).

This general approach to scheduling works well for a range of problem parameters. However, we have found that when the number of tasks requested by users exceeds the number of tasks that can be reasonably performed, the performance of the constraint engine suffers. This is due to the fact that it must propagate constraints over a large set of tasks, most of which cannot be scheduled. In addition, the scheduler can spend a lot of time backtracking over a relatively large number of tasks that have very little chance of being scheduled. Thus the runtime of a constraint-based approach can depend more on the number of tasks requested than on the number of tasks that can ultimately be executed. We have found this overhead cost to be intolerable for the time demands of scheduling operational satellites. For this reason, we have developed a hybrid algorithm as a first step toward addressing this problem.

A Hybrid Approach

In this section, we present a hybrid approach for generating solutions to over-constrained satellite scheduling problems.

Priority segmentation reduces the constraint propagation overhead by breaking up the problem into a subset of problems. The idea is to sort the tasks by their priority value, divide them into groups of n tasks (where n is a small integer), and then schedule one group at a time. The algorithm details are shown in Figure 1.

Priority segmentation can be viewed as an application of staged search (Ibaraki 1978) to a constraint-based formulation of scheduling problems. Because the algorithm commits to scheduling decisions before all tasks are considered, it is necessarily a suboptimal algorithm. For large scheduling problems (*i.e.*, greater than 1000 tasks), the computation cost of finding optimal solutions is prohibitive so we must consider suboptimal scheduling algorithms. Suboptimal algorithms must also be considered for problems with real-time constraints on the scheduling. Obviously, we could apply additional optimization methods (*e.g.*, local search) to the output of priority segmentation. These issues are not addressed in this paper and are the subject of future work.

Experimental Design

We implemented priority segmentation using branch-and-bound search to generate the optimal schedules for task subsets. We also implemented a random problem generator to provide us with a set of test problems. For our initial investigation, we made some simplifying assumptions about the remote sensing scheduling problem. For example, we assumed fixed duration tasks with fixed start times. This reduces the over-constrained scheduling problem to deciding which of the tasks to schedule (since the start time and duration cannot change). The objective is to choose a set of tasks to schedule such that the sum of the task priorities is maximized.

There are two justifications for simplifying the problem in this manner. One is to better focus our attention on the over-constrained resource problem. The other is that one of our customers is interested in solving a remote sensing problem where a large number of tasks (*e.g.*, greater than 1000) need to be scheduled in less than 30 minutes. The problem description includes a model to calculate the slew required by the satellite to reposition its camera between tasks. The slew model provides the slew time between a given pair of tasks, which depends on the imaging locations and the start times of the two tasks. The time constraint on the scheduler means that the slew must calculated ahead of time and stored in a table. The size of the slew table is $(cn)^2$ where c is the number of possible start times for individual tasks and n is the number of tasks. We chose to require fixed start times for tasks so that the slew table could fit in the computer's memory.

We generated a set of random remote-sensing scheduling problems with the following characteristics. The satellite system resources were simplified to a single resource (*e.g.*, the camera) which has a capacity of one (*i.e.*, only one picture can be taken at a time). Tasks were generated to simulate different levels of demand for the satellite resource that directly correspond to different levels of average constrainedness. The task duration was fixed at 10 seconds for all tasks. Each fixed start time was randomly selected from a uniform distribution over the schedule period. The test sets did not include any stereo tasks, precedence or logical constraints between tasks. The tasks priorities were randomly selected integers from the range [1, 50] (where 1 is the lowest priority value). We generated problem sets by varying the number of tasks and the scheduling period to control the average constrainedness of the problem (C_{ave}).

Initial Experimental Results

We ran priority segmentation on a set of randomly generated scheduling problems that were built using the simplifying assumptions described above. The priority

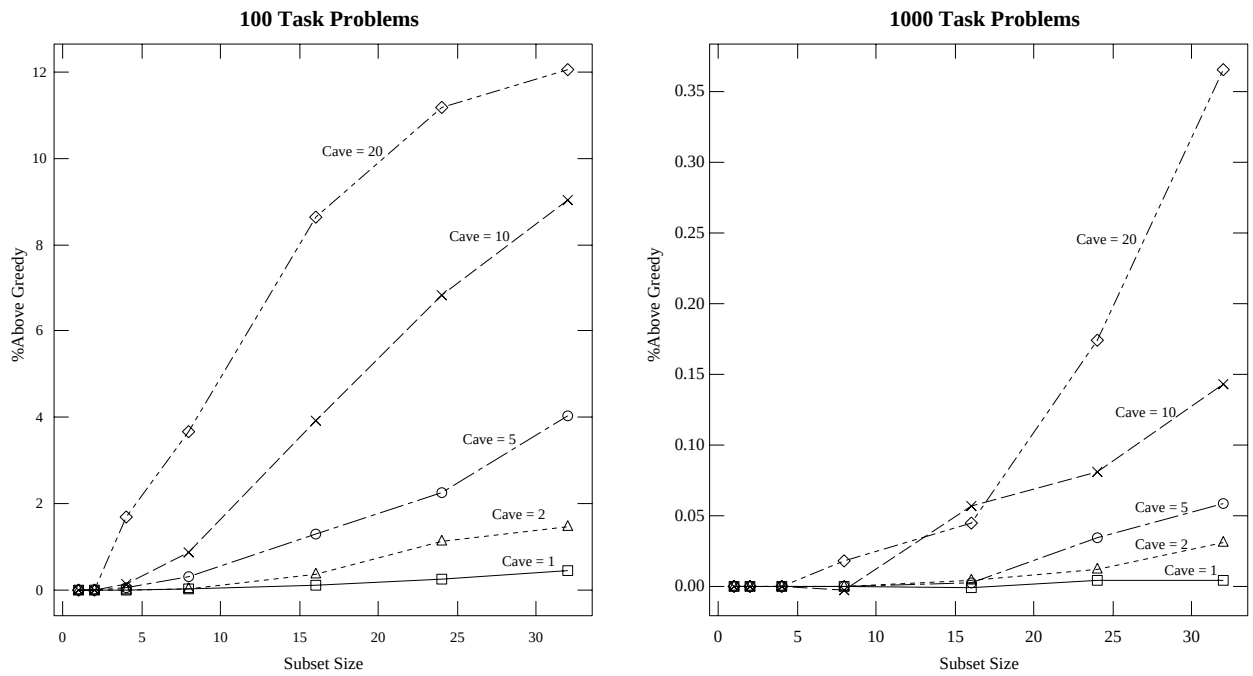


Figure 2: Percent average priority sum improvement relative to greedy (100-task and 1000-task problems).

segmentation subset size was varied as well as the size of the problem (*i.e.*, number of tasks) and the average constrainedness (C_{ave}). For a given problem size, we varied the average constrainedness by choosing the length of the scheduling period so that the average number of tasks per unit time was equal to the desired constrainedness. We performed experiments on 100-task and 1000-task problems.

To provide a baseline for comparison, we implemented a simple greedy algorithm (*i.e.,* consider tasks for scheduling one at a time from highest to lowest priority). Priority segmentation with a subset size of 1 is equivalent to this greedy algorithm. It is also interesting to note that priority segmentation with a subset size of 2 also produces the same solutions as this greedy algorithm. This is because the order in which two tasks are considered for scheduling doesn't affect which one is scheduled.

Figure 2 shows the average solution quality as a percentage above the greedy solution for both the 100-task and 1000-task problems. Each point is the averaged over 100 problem instances. The results show that the average solution quality (*i.e.*, average schedule priority) increases with the size of the priority segmentation subset. For the 1000-task problems, the average solution quality only increases very slightly for smaller C_{ave} values and sometimes the average solution quality is lower than greedy (*e.g.,* when $C_{ave} = 10$ and the subset size is 8). In general, the results shown in the graphs illustrate the trend that we expected, namely a gradual increase in average solution quality as the subset size is increased. The 100-task graph also shows a gradual approach to optimal (12-percent above greedy) for the $C_{ave} = 20$ case as the subset size is increased.

As expected, the time required to perform priority segmentation increases with the size of the problem and the size of the subset. An example of the CPU time for the case where $C_{ave} = 20$ is shown in Figure 3. Notice that the CPU time levels off for the 100-task case when the subset size is 32, in part because priority segmentation is finding optimal solutions.

At first we didn't expect to see the slight decrease in average solution quality that occurred in some of the results. This result can be explained by the fact that priority segmentation with different subset sizes will search different parts of the scheduling space, thus it is possible for a larger subset to result in a lower average solution quality for any given problem instance. We expect that larger priority subsets will generate better solutions on average, although better solutions are not guaranteed.

The results also show that the percent improvement in solution quality over greedy increases with C_{ave} for a given priority subset size. This is because as C_{ave} becomes larger, the set of tasks to choose from for a given resource availability time slot increases. This also increases the expected value of the maximum priority of these tasks. In the limit as C_{ave} approaches infinity, the probability that there is a task with maximum priority for every time slot approaches one. Thus as C_{ave} increases, we expect the optimal schedule score and the average priority segmentation score to increase.

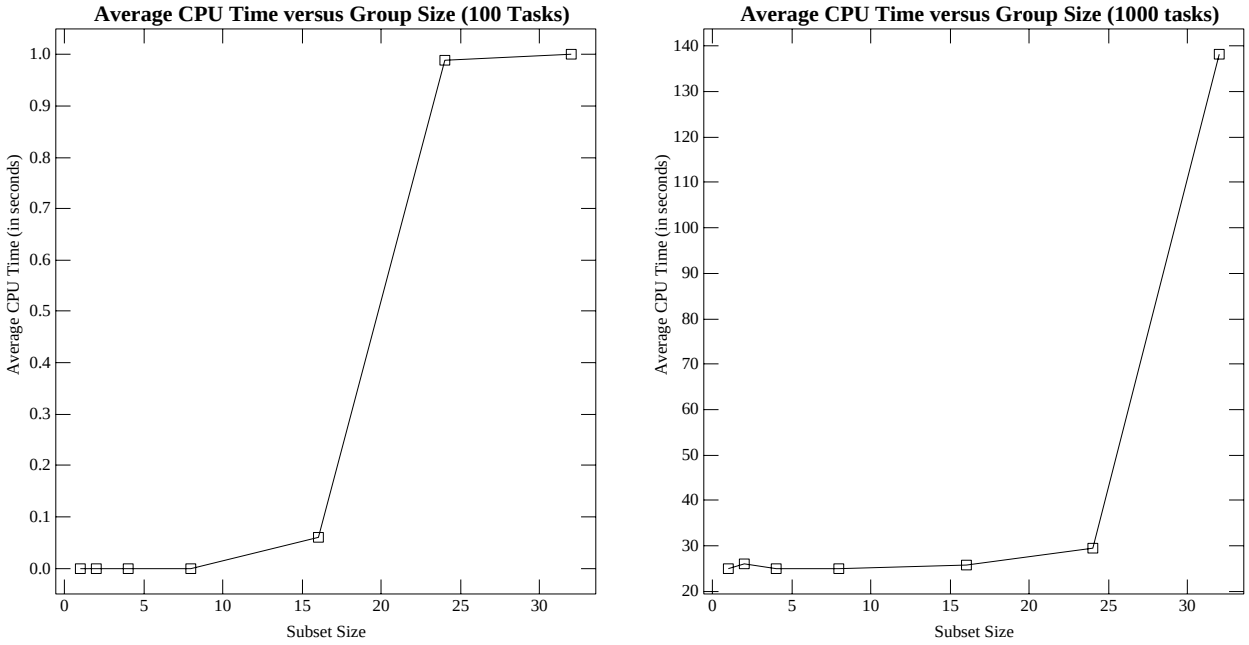


Figure 3: Average CPU time versus priority segmentation subset size ($C_{ave} = 20$, 100-task and 1000-task problems).

Discussion and Related Work

Admittedly, the results presented here are just a first step toward understanding both this class of problems and the priority segmentation algorithm. Although we have used priority segmentation to address over-constrained scheduling problems for at least two of our customers, we are just beginning to evaluate its performance on more general problems. Our initial results indicate that priority segmentation can produce better quality solutions that greedy under certain conditions (*e.g.,* when the average constrainedness is large). Our result also suggest that, for this class of problems, greedy solutions are often very competitive with more sophisticated algorithms. Additional experiments and analysis are necessary to clearly determine the conditions under which priority segmentation is worth the computational effort.

We are aware of other work on over constrained scheduling problems. For example, the work by Bressina *et al.* (*e.g.*, (Bresina, Morris, & Edgington 1997; Drummond, Bressina, & Swanson 1994)) on ground-based telescope scheduling used a dynamic priority weighting scheme that depends on both the value and age of the tasks (*i.e.,* older tasks are given a higher priority). Sobiesk *et al.* (*e.g.*, (Olawsky, Kregsbach, & Gini 1995; Sobiesk, Kregsbach, & Gini 1996)) have investigated a stochastic dynamic programming approach to solving over-constrained product-quality planning problems. We have not yet evaluated either of these approaches on our over-constrained remote-sensing problems. As mentioned above, priority segmentation can be viewed as a variation on Ibaraki's work on hybrid search (Ibaraki 1978) and thus is related to other hybrid search algorithms (Pearl 1984).

Lemaitre *et al.* (Lemaitre & Verfaillie 1997; Bensana, Lemaitre, & Verfaillie 1999) have looked at a similar problem of scheduling imaging tasks for the Spot5 satellite. For their problem, tasks have fixed duration and fixed start times. They present results that compare a variant of depth-first branch-and-bound called Russian Doll Search to branch-and-bound and an implementation using ILOG Solver. Their results show that Russian Doll Search takes significantly less time on average than the other algorithms to generate good quality solutions. Although the ILOG-based approach requires more computation time, they argue that it may still be the better choice because it is a more general approach. Their work helps to justify our simplification of the more general satellite scheduling problem. In addition, their results support our belief that algorithms tailored to exploit the structure of a class of problems can often outperform general purpose approaches.

Conclusions and Future Work

We have presented a hybrid scheduling algorithm called priority segmentation. Our initial results indicate that it can produce better results than a simple greedy algorithm when the average number of tasks competing for a resource at any given time is high (*i.e.*, greater than 10). Our results also suggest that greedy solutions can be competitive for this general class of problems.

We intend to extend the work presented in this paper in several ways. First, we are planning to create a more sophisticated random problem gener-

ator that will allow us to evaluate scheduling algorithms on a wider range of problems (and more realistic problems). Second, we are planning to consider other suboptimal scheduling techniques such as real-time search (Korf 1990). Finally, we intend to implement and evaluate other approaches (*e.g.,* local search, Russian Doll Search (Lemaitre & Verfaillie 1997; Verfaillie, Lemaitre, & Schiex 1996), dynamic programming, etc.) to scheduling satellite resources.

Acknowledgments

This work has benefited from discussions with members of the Veridian-PSR Resource Optimization Group. Greg Heberle wrote the software that generates random remote sensing problems. Jesse Chappell helped with the algorithm implementation. John Calvit provided helpful comments on a version of this paper. This work has also benefited from discussions with some of our customers. We like to thank the reviewer for his comments and suggestions. We would also like to thank David Harrison, William Cheng, Leslie Lamport and the Free Software Foundation for their graphics and text formatting software.

References

Baptiste, P.; Pape, C. L.; and Nuijten, W. 1995. Incorporating efficient operations research algorithms in constraint-based scheduling. In *Proceedings of the First International Joint Workshop onArtificial Intelligence and Operations Research.*

Bensana, E.; Lemaitre, M.; and Verfaillie, G. 1999. Earth observation satellite management. *Constraints* 4:293–299.

Bresina, J. L.; Morris, R. A.; and Edgington, W. R. 1997. Optimizing observation scheduling objectives. In *Proceedings of the NASA Workshop on Planning and Scheduling for Space.*

Drummond, M.; Bressina, J.; and Swanson, K. 1994. Just-in-case scheduling. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94).*

Fox, M. S., and Sycara, K. P. 1990. Overview of CORTES: A constraint based approach to production planning, schedulingand control. In *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management.*

Ibaraki, T. 1978. Depth-m search in branch-and-bound algorithms. *International Journal of Computer and Information Sciences* 7:315–343.

ILOG Corporation, Gentilly Cedex, France. 1998. *ILOG Optimization Suite: White Paper.*

Korf, R. E. 1990. Real-time heuristic search. *Artificial Intelligence* 42(2–3):189–211.

Larson, W. J., and Wertz, J. R., eds. 1992. *Space Mission Analysis and Design.* Kluwer Academic Publishers, second edition.

Lemaitre, M., and Verfaillie, G. 1997. Daily management of an earth observation satellite: Comparison of ILOG Solver with dedicated algorithms for valued constraint satisfaction problems. In *Proceedings of the 1997 ILOG Optimization Suite International Users' Group.*

Olawsky, D.; Kregsbach, K.; and Gini, M. 1995. An analysis of sensor-based task planning. Technical Report 95-51, Department of Computer Science, University of Minnesota.

Pacific-Sierra Research Corporation, Arlington, VA. 1997. *GREAS Application Framework Reference Manual, Version 4.0.*

Pearl, J. 1984. *Heuristics.* Reading, MA: Addison-Wesley.

Sadeh, N., and Fox, M. S. 1990. Variable and value ordering heuristics for activity-based job-shop scheduling. In *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management.*

Smith, S. F. 1994. Reactive scheduling systems. In Brown, D., and Scherer, W., eds., *Intelligent Scheduling Systems.* Kluwer Publishers.

Sobiesk, E.; Kregsbach, K.; and Gini, M. 1996. Overconstrained scheduling using dynamic programming. In *Proceedings of the AI and Manufacturing Research Workshop, National Conference on Artificial Intelligence (AAAI-96).*

Verfaillie, G.; Lemaitre, M.; and Schiex, T. 1996. Russian doll search for solving constraint optimization problems. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-96).*

Commentary on the paper entitled
Towards Scheduling Over-Constrained Remote-Sensing Satellites

Gérard Verfaillie
ONERA/DCSD, Toulouse Center
2, av. Édouard Belin, BP 4025
31055 Toulouse Cedex 4, France
phone: +33 5 62 25 26 32
fax: +33 5 62 25 25 64
email: verfaillie@cert.fr

At first, the paper entitled *Towards Scheduling Over-Constrained Remote-Sensing Satellites*, presented by J. C. Pemberton from *Veridian Pacific-Sierra Research*, reminds us that the problem of the management of an observation satellite (either Earth observation, or space observation) is a difficult combinatorial constrained optimization problem.

As it is clearly explained in the paper, its input data are:

- a scheduling horizon h;
- a set R of user requests;
- associated with each user request $r \in R$, a set $C_u(r)$ of user-defined constraints, which define the conditions in which the user wants his/her request to be achieved;
- a set C_s of constraints expressing the limitations of the satellite: trajectory, manoeuvring ability, power, energy, memory on board, visibility windows, data downloading ...
- an optimization criterion c, which is often a function of the set of the selected requests.

The problem is to find a subset R' of R, which is feasible (the constraints in $C_s \cup [\cup_{r \in R'} C_u(r)]$ are all satisfied) and optimal according c.

In fact, there are three kinds of decision to make:

- about the selection: which requests to select ?
- about their order: in which order to achieve them ?
- about their starting times: at which time to start each of them ?

This problem is close to well-known problems in the *Operations Research* community, as the *Multi-Knapsack* problem, the *Job-Shop Scheduling* problem, or the *Traveling Salesman* problem with time windows: well-known but difficult problems, at least if one looks for optimal solutions or solutions the distance to the optimum of which can be guaranteed.

Generic frameworks, currently used to represent combinatorial constrained optimization problems, such as *Mixed Linear Programming* or *Constraint programming*, can be used to represent it.

Consequently, generic tools dedicated to these frameworks, such as *CPlex* (*Ilog Planner*) for *Mixed Linear Programming* or *Ilog Solver* for *Constraint programming*, can be used to solve it.

But, as J. C. Pemberton said, if these tools allow the observation satellite scheduling problem to be easily represented and small instances to be solved efficiently, they do not allow generally large realistic instances to be solved correctly: for these instances, they do not manage to produce optimal solutions within a reasonable time; if they produce optimal solutions, they do not manage to prove their optimality; what is worse, the quality of the solutions they deliver when interrupted is generally poor.

According to J. C. Pemberton, and I agree with him, it is due to the difficulty of the selection of a small optimal feasible subset of requests among a large set of candidate requests, when the priority levels or the weights associated with each request are very similar. The problem is at once over-constrained and uniform, and it is known that over-constrained uniform optimization problems are very difficult to solve: neither bound computing mechanisms used in the *Linear Programming* tools, nor constraint propagation mechanisms used in the *Constraint Programming* tools manage to make up for the explosion of the search space.

In fact, all the methods available to solve the problem, at least as well as possible, in order to use these satellites as efficiently as possible, belong to the following four families:

- *Heuristic Greedy* algorithms;
- *Local Search* algorithms;
- *Tree Search* algorithms;
- *Dynamic Programming* algorithms.

Algorithms of the first two families are inexact (no optimality guarantee). Those of the last two families are exact (optimality guarantee) when they are not interrupted. In each of these families, a lot of variants can be defined. Between them, a lot of hybrid algorithms can be defined too.

What J. C. Pemberton proposes in this paper is a particular hybrid algorithm: a combination of *Greedy* and *Tree Search* algorithms.

As he observed that *Tree Search*, combined with *Constraint Propagation*, as it is implemented in *Ilog Solver* and *Scheduler*, can solve efficiently small instances, he orders the set of the requests by decreasing priority, he cut it into small subsets, solves optimally the problem that includes only the first subset, then solves optimally the problem that includes only the second subset with the solution of the previous problem locked, and so on.

The algorithm he describes can be viewed as a *Greedy* algorithm, the decisions of which are made on sets of choices rather than on elementary choices. Besides, when the subsets he considers are reduced to be singletons, his algorithm is a simple *Greedy* algorithm. At the contrary, when the set of the requests is not cut, his algorithm is a simple *Tree Search* algorithm. Between both these extrema, any tradeoff between quality and time is possible.

Research is currently active about hybrid search mechanisms, because people think that it is one of the ways of improving the efficiency of the known mechanisms. J. C. Pemberton's proposal is interesting, simple and generic. It could be used on many other problems.

Experiments he carried out show improvements in terms of quality of the produced solutions, when compared with a simple greedy algorithm. But, I would like to take advantage of this commentary to ask people who report experiments about the use of inexact optimization methods to give information both in terms of quality and of time. And the best way of reporting seems to be a quality profile, which shows how the quality of the best solution found so far evolves with time.

Finally, I would like to note that, at least for the simplified problem used in the experiments (fixed request starting times, fixed request durations, one instrument), there exists a very simple exact *Dynamic Programming* algorithm, which can certainly outperform all the other algorithms, in terms of quality (due to optimality) and in terms of time. Unfortunately, it is not the case for the general observation satellite management problem.

First published by the authors in "Computing Valid Intervals for Collections of Activities with Shared States and Resources," AIPS 2000, p. 339-346, Copyright (c) 2000, American Association for Artificial Intelligence.
Reprinted with permission from AAAI.

Merging the Semantics of State Constraints for Collections of Activities

Russell Knight, Gregg Rabideau, and Steve Chien
Jet Propulsion Lab, M/S 126-347
California Institute of Technology
4800 Oak Grove Drive, Pasadena, CA 91109-8099
firstname.lastname@jpl.nasa.gov

Abstract

When scheduling a collection of activities, it is often useful to calculate the valid intervals for the collection with respect to shared resources and states. The constraints of these activities on shared resources and states need to be analyzed to avoid miscalculations due to interactions between the constraints. For shared resources, a combined profile is typically generated and used to compute the valid intervals. We introduce a technique for generating a combined profile for shared state constraints, which is subsequently used to compute valid intervals. The worst-case complexity of our technique is quadratic in the total number of resource and state constraints in the problem. We present empirical evidence indicating that our technique improves performance of our planner on real problems when compared to the performance of the same planner using more naïve techniques.

Introduction

This paper describes an approach to solving the problem of scheduling interdependent sets of activities in a combined scheduling/planning problem. In this approach, we compute a combined profile that encapsulates the requirements and effects of the collection of activities on shared states and resources. This combined profile is then used to determine the least-conflicting placements for the collection allowable in the current plan.

This problem is an important aspect of solving combined planning and scheduling problems. In many approaches to combined planning and scheduling, one alternates between finding activities to satisfy pre- and post-conditions (planning) and finding temporal assignments and resources for those activities (scheduling). Complex activity placement is also an important component of many scheduling problems, as finding temporal assignments for complex activities can be computationally challenging.

This work advances the approach of moving collections of activities whose temporal relationships among themselves are fixed. We observe similar techniques in job-centered scheduling approaches and tabu-search approaches. We proceed by describing our motivation, defining the problem, and describing the solution. Finally, we present empirical evidence in favor of our technique.

Motivation

We believe that reasoning about collections of activities (as opposed to individual activities) is helpful in finding a solution to a combined planning/scheduling problem. We believe this is due to the increased complexity introduced by the backtracking required for reasoning about individual activities. In a sense, we assume that decisions concerning the members of the collection will hold even when shifted "lock-step" in the schedule either forward or backward in time. Although research concerning flexible intervals between members of the collection might be fruitful, it is left as future work. Even the simplest of approaches (e.g. gradient descent) require that we know the least conflicting intervals for a given collection, therefore we wish to compute these intervals.

Problem

We assume some method for partitioning the complete set of activities into separate collections. This partitioning is a research area in its own right and outside the scope of this paper (although we do describe our technique for partitioning activities.)

We discover that naïve intersection of valid intervals for single activities in the collection render both false positives (intervals returned as being valid in fact cause constraint violations) and false negatives (intervals returned as causing constraint violations are in fact valid) with respect to the whole. This can only be due to interactions between the activities; or, more specifically, interactions between the constraints on shared states and resources.

For example, consider a pair of activities that affect a battery (see Figure 1). The first activity a_1 uses 10 amp-minutes, while the second activity a_2 restores 10 amp-minutes. If we schedule a_1 individually, we find no intervals that will not cause an over-subscription of the battery, because activity a_3 has already fully depleted the battery by the end of the current schedule. But, if we schedule these together, we find placements that are valid. The positive effect a_2 has on the schedule makes up for a_1's usage. We wish to handle this sort of constraint-interaction for shared states and resources. We continue with some definitions.

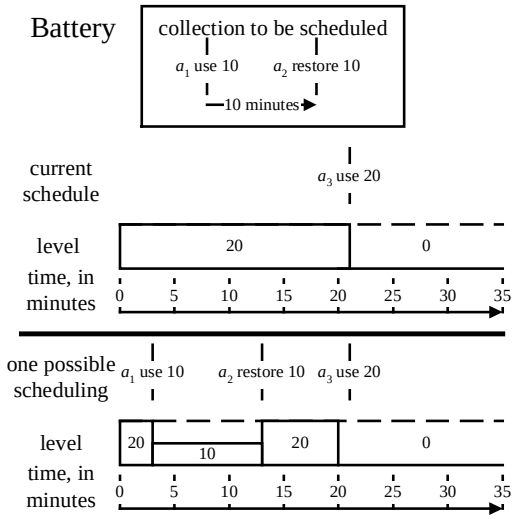


Figure 1 Battery interaction example

Definitions

Activity: consists of a start-time (s), an end-time (e), and a set of timeline constraints or reservations that represent the constraints of the activity with respect to shared states and resources (R). R is conjunctive, therefore we do not deal with multiple resources that could satisfy a single constraint. For simplicity, we assume that times are integers. We also assume that $s < e$.

Timeline: the representation of a shared state or resource over time. All timelines consist of a set of value units that represent the value of the timeline for all time-points in the horizon. All timelines also consist of a set of global constraints. Global constraints and values differ according to the type of timeline.

Depletable Resource Timeline: a timeline representing a resource that is added to or removed from over time. An example is a battery, where usage depletes the battery and recharging restores the battery. Global constraints include the minimum level, the maximum level, and the default or starting value.

Non-Depletable Resource Timeline: a timeline representing a resource that is used for a period of time and then relinquished back at the end of the usage. An example is a power bus, where usage takes up some of the capacity of the bus, and the cessation of usage restores the capacity. Obviously, a non-depletable timeline can be represented by a depletable timeline with a depleting reservation followed by a renewing reservation (reservations are described later), but we include these because the semantics of their reservations aids in describing the semantics of transformed depletable reservations. As with depletable timelines, global constraints include the minimum level, the maximum level, and the default value.

State Timeline: a timeline representing a state that can be either changed or required by reservations. Values are symbols; we use strings. Global constraints consist of a transition graph G whose nodes represent allowed values and whose edges represent allowed consecutive transitions from one value to the next, and a default value.

Reservation: a constraint on a shared state or resource for a specific interval to a specific value. Since reservations are part and parcel with activities, reservations inherit their start- and end-times (s, e) from the activity containing them. A reservation also consists of a reference to a timeline. The type of timeline that the reservation constrains determines the type of the reservation.

Depletable Reservation: consists of a start-time (the end-time is ignored) and a value (v). v is an integer representing the amount of capacity to be used by the reservation. The value of any point on such a timeline is the sum of all reservations at the time-point and previous, as well as the default value. See Figure 2. Note that the timeline models a value over time. The reservations are labeled with their values.

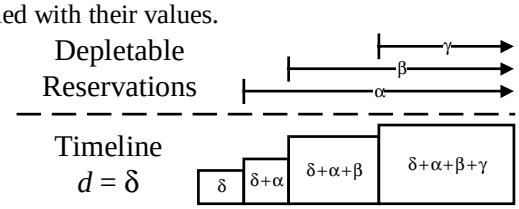


Figure 2 Effect of depletable reservations

Non-Depeletable Reservation: consists of a start- and end-time (s, e, as above) and a value (v). v is an integer representing the amount of capacity to be used by the reservation. The value of any point on such a timeline is the sum of all reservations that include the time-point in their temporal extents, as well as the default value. See Figure 3.

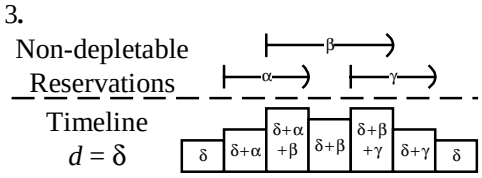


Figure 3 Effect of non-depletable reservations

State Reservation: we address two types of state reservations: changers and users.

A changer reservation consists of a start-time (s, the end-time is ignored) and a value (v). v is a symbol as described for state timelines. The value of any point on such a timeline is the value of the most recent changer reservation. If two or more changers are simultaneous to different values, the resultant value is invalid. See Figure 4.

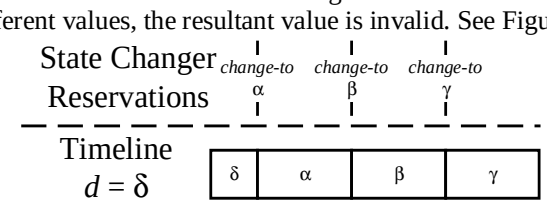


Figure 4 Effect of state changer reservations

A user reservation consists of a start- and end-time (s, e) and a value (v). v is a symbol as described for state timelines. A user reservation constrains the timeline such

that for all points during its temporal extent, the value of the timeline is the same as *v*. Note that user reservations have no direct effect on the value of the timeline.

Conflict: a conflict is a violation of any constraint. These include: 1) over or under subscriptions, where a resource timeline value lies outside of its allowed range, 2) state transition violations, where a state timeline value is immediately followed by a value for which no corresponding transition arc exists in its transition graph, 3) state usage violations, where a state timeline value differs from a user reservation constraining the timeline during the temporal extent of the reservation.

Solution Description

Our approach to computing the valid intervals (*I*) is as follows: 1) gather all reservations of all activities, 2) partition these according to timeline (i.e. for each timeline, create a set that contains all of the reservations for the timeline), 3) compute the valid intervals for each partition (*P*), 4) translate and intersect the valid intervals. We focus on step 3. We know that if the reservations in *P* do not interact, then we can compute valid intervals for *P* by computing those for each individual reservation in *P*. This is accomplished by first computing each set of intervals I_r for each individual reservation *r* in *P*. We then translate I_r by the difference between the start-time of the reservation and the earliest start-time in *P*, and intersecting all of the resultant intervals.

The key point of our approach: transforming *P* into a set of non-interacting reservations *P'* is one way of making the computation of valid intervals tractable.

Even if reservations in *P* interact, the reservations in *P'* do not, thus we can compute valid intervals using simple translation and intersection of the valid intervals for each individual reservation in *P'*.

To obtain *P'*, we merge the semantics of the existing reservations. This is straight-forward for resource reservations, but state reservations are problematic. For example, if *P* consists only of a changer reservation to "on" preceding a user reservation of "on", the *P'* requires a reservation between them that ensures that either no changes occur, or that the last change that does occur accommodates the user. Our solution technique is described more fully in (Knight, R., Rabideau, G., and Chien, S. 2000.) The complexity of our technique is quadratic in the total number of reservations, and scales at a constant factor cost over more naive techniques.

Empirical Evaluation

We now describe an empirical comparison of an aggregate search technique using our informed approach for determining valid placements for collections of activities to the same search technique using a naïve approach. We evaluate two aspects of our algorithm: 1) quality in terms of speed and accuracy, and 2) efficacy in terms of conflict reduction, both in scheduling and combined planning and scheduling.

In our empirical analysis we use four models (and corresponding problem set generators): 1) the EO1 spacecraft operations domain, 2) the Rocky-7 Mars rover operations domain, 3) the DATA-CHASER shuttle payload operations domain, and 4) the New Millennium Space Technology Four landed operations domain.

Within each model and corresponding problem set, we generate random problems that include a background set of fixed activities and a number of movable activity groups. The activity groups are placed randomly. The goal is to minimize the number of conflicts in the schedule by performing planning and scheduling operations.

To solve each problem, we use the ASPEN (Automated Scheduling and Planning Environment) system using an "iterative repair" algorithm, which classifies conflicts and attacks them each individually (Rabideau, G., Knight R., Chien S., Fukunaga A., Govindjee A. 1999). Conflicts occur when a plan constraint has been violated; this constraint could be temporal or involve a resource or state timeline. Conflicts are resolved by performing one or more schedule modifications such as moving, adding, or deleting activities. The iterative repair algorithm continues until no conflicts remain in the schedule, or a timeout has expired.

The scheduler entertains non-conflicting placements when moving activity groups. In the *control* trials the scheduler does so using the naïve algorithm for computing valid placements. In the *experiment* trials the scheduler is using the informed method to compute valid placements. In all cases for each domain, both trials are using the same set of heuristics at all other choice-points (e.g., selection of a conflict or activity group to attempt to repair, where to place within computed valid intervals, etc.). Note that simple (non-aggregate) operations are available in both real domains, although they are of limited comparative utility. Using only non-aggregate operations, the problems are intractable within reasonable time bounds. This is because the distance in terms of sub-optimal moves from one local optima to the next is $O(n)$ and the space to be searched is $O(m^n)$ where *n* is the number of activities in a movable collection and *m* is the number of possible locations given by a naïve calculation of legal intervals for an individual activity. For example, in the EO1 domain, *n* ranges from 23 to 56; in the Rover domain, *n* ranges from 8 to 17. Note that naïve calculation for valid intervals for an individual activity are adequate for that activity.

We forego further description of the domains due to space constraints.

Quality Assessment

To assess the quality of the algorithm, we directly compare the accuracy and speed of the informed search mechanism to the naïve intersection approach and a random placement approach. We assess the accuracy of the competing approaches by comparing the intervals for legal placement that each algorithm returns to the correct intervals. We assess the speed of the three approaches by measuring the

CPU time taken by each algorithm to compute its intervals.

Accuracy is a desirable property in any algorithm to determine "good" placements for aggregated activities. Ideally, a legal interval generator would return exactly those times that are legal placements for the member activity. This would mean that the algorithm would be sound (e.g., all times in the interval returned would be legal) and complete (e.g., all legal times would be returned by the algorithm). Table 1 shows the results of this evaluation on the ST4, EO1, Rover, and DCAPS domains. Because the informed method is complete and sound, it returns all of the correct interval(s) and no incorrect intervals. However, the naïve intersection method has both false positive (soundness) and false negative (completeness errors). As the data shows, it tends to miss the majority of the legal interval (by failing to recognize positive interactions between activities in the collection).

	EO1	Rover	DCAPS	ST4
informed	$0/_{14870}$	$0/_{10030}$	$0/_{5100}$	$0/_{96890}$
naïve	$14880/_{10}$	$4250/_{5780}$	$5700/_{3600}$	$51690/_{45220}$
random	$2742250/2757120$	$16101/_{17104}$	$2100/_{7200}$	$360150/457040$

Table 1 Errors in Calculated Intervals ($^{\text{avg. errors}}/_{\text{avg. interval size}}$)

Speed is another desirable property of a legal times algorithm. Ideally, a legal interval generator would take very little CPU resources to compute. Table 2 shows the average CPU time taken by each algorithm to compute its estimation of the legal intervals. Because the random algorithm simply returns the whole interval it takes effectively zero time. We also observe that the informed algorithm takes more time than the naïve algorithm. This not surprising as it must first transform the basic state and resource reservations into non-interacting reservations, then compute legal intervals for each, and then intersect them. The naïve approach need only perform the latter two steps.

	EO1	Rover	DCAPS	ST4
informed	1.8925	.025	.3528	.0315
naïve	.1506	.025	.1090	.0300
random	0	0	0	0

Table 2 Average time to compute intervals, in seconds

Efficacy Assessment

We assess the efficacy of our algorithm in terms of conflict reduction. We compare the number of conflicts reduced for scheduling operations, and the effect this has on solving combined planning/scheduling problems.

In terms of scheduling, ideally, placing an aggregate at a time recommended by a legal times algorithm should result in an improved schedule (i.e. one with fewer conflicts). Table 3 shows the average reduction in the number of conflicts in the schedule after placement of the aggregate. Note that because having an unplaced activity is a conflict, by default each algorithm gets a score of *n* if there are *n* activities in the aggregate just for placing the activity (i.e. in the absence of any state or resource conflicts introduced or removed). We see that the informed algorithm strictly outperforms the others.

	EO1	Rover	DCAPS	ST4
informed	21.2878	1.9042	25.2549	5.800
naïve	19.9978	1.8339	13.8416	0.9997
random	19.2978	-0.7837	21.4041	3.0785

Table 3 Effectiveness through conflict reduction

To assess the effect that the algorithm has in solving planning/scheduling problems, we examine the number of conflicts over time (in terms of CPU usage) and the total number of problems solved for each domain. If superior, the informed search should result in faster reduction of conflicts and more problems being completely solved.

We generate twenty random problems for each domain and run ASPEN with twenty different random seeds for each combination of problem and technique. Note that we do not guarantee that the problems are solvable.

We evaluate the performance of our technique versus the performance of the naïve technique in terms of the number of iterations to solve conflicts, amount of time to solve conflicts, and the total number of problems solved for each domain.

For the EO1 operations domain, the naïve technique and informed technique perform similarly at first. This is because a number of the conflicts do not involve interacting reservations and hence the naïve technique can solve them. However, many of the conflicts involve interacting reservations. Because the informed technique correctly handles these interactions, it is able to solve these conflicts. Thus in the longer term the informed algorithm is able to solve more conflicts in the schedule.

For the Rocky-7 Mars rover operations domain, the informed technique appears to strictly dominate the naïve technique. Interestingly, conflict count rises before it falls for both algorithms. This is due to the added planning necessary to solve conflicts. Adding activities leads to more conflicts initially, but eventually leads to solutions.

For the New Millennium ST4 Landed Operations domain, the informed technique strictly outperforms the naïve technique. Conflict count rises before it falls as in the rover domain and for the same reason, except the algorithm employing the naïve technique never recovers. Many of the activities in a group interact, therefore the naïve technique often makes mistakes in recommending placements for activity groups. Because of this faulty advice, repair using the naïve approach actually increases the number of conflicts in the schedule.

For the DCAPS domain, the informed technique strictly outperforms the naïve technique. In this domain, almost all of the activities in a group interact, leading to similar consequences as the ST4 domain.

In terms of number of problems solved, we observe that ASPEN employing the informed scheduling technique

is able to completely solve (i.e., remove all conflicts) more problems than ASPEN employing the naïve approach in all five domains (Table 4).

	EO1	Rover	DCAPS	ST4	total
informed	149/400	390/400	387/400	243/400	1169/1600
naïve	60/400	243/400	1/400	0/400	304/1600

Table 4 Problems Solved

These empirical results imply that aggregate reasoning is effective in real domains, both in terms of number of constraint violations repaired and in terms of overall time to reach a desired solution quality, as long as we use an informed scheduling function. We have computed the statistical confidence that the average final number of conflicts using the informed search method is less than the final number of conflicts using the naïve method. For the EO1, ST4, and DCAPS domains, this confidence is greater than 99.9%. For the Rover domain, this confidence is greater than 98%.

Related Work

There are a number of related systems that perform both planning and scheduling. IxTeT (Laborie, P., Ghallab, M. 1995) uses least-commitment approach to sharable resources that does not fix timepoints for its resource and state usages.

HSTS (Muscettola, N. 1993) enforces a total order on timepoints affecting common shared states and resources, allowing more temporal flexibility. We believe that our technique is applicable in this case at a greater computational expense (while still being polynomial), and future research should address this issue.

Both IxTeT and HSTS are less committed representations than our grounded time representation and this flexibility incurs a greater computational expense to detect and/or resolve conflicts.

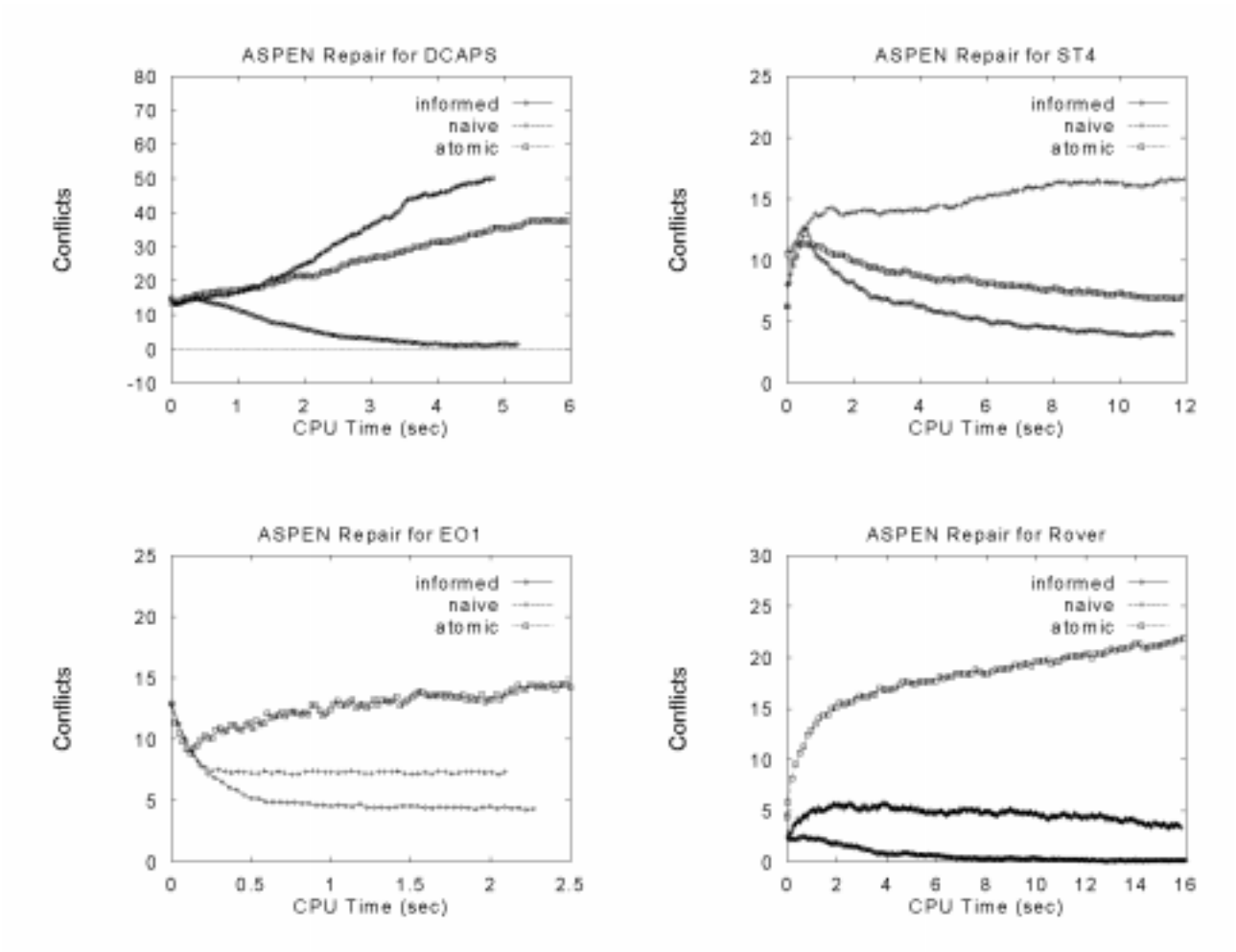


Figure 5 Conflicts over time for domains

O-PLAN (Drabble, B., and Tate A. 1984) also deals with state and resource constraints. O-PLAN's resource reasoning uses optimistic and pessimistic resource bounds to efficiently guide its resource analysis when times are not yet grounded. Like ASPEN, O-PLAN also allows multiple constraint managers which would enable it to perform general reasoning when times are unconstrained and more efficient reasoning in the case where all timepoints are grounded (also enabling aggregate informed search as described in this paper). SIPE-2 (Wilkins, D., 1998) handles depletable/non-depletable resource and state constraints as planning variables using constraint posting and reasons at the same level of commitment as IxTeT.

(Cesta, Oddi S., and Smith S. 1998) apply constraint-posting techniques to satisfy multi-capacitated resource problems at the same level of commitment. Depletable/non-depletable resource constraints are easily transformed to multi-capacitated resource constraints. None of these systems generally consider aggregate operations in their search space.

Conclusion

This paper has described the use of informed transformation techniques to improve the efficiency of scheduling sets of interdependent activities. We describe our algorithm for processing interacting state and resource requirements of a cluster of interdependent activities into a set of independent requirements and use these to search for placements for the activity set. We show empirically that our informed search method outperforms the alternative approach of searching for legal placements on problems from space exploration domains.

Acknowledgements

We wish to thank Stephen Smith of Carnegie Mellon University and Richard Korf of the University of California, Los Angeles for their helpful suggestions concerning this paper. This work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

Cesta, Oddi S., and Smith S. 1998. "Profile-Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems." *Proc. AIPS98*, pp. 214-223.

Dechter, R., Meiri I., and Pearl J. 1991. "Temporal Constraint Networks," *Artificial Intelligence*, 49, 1991, pp 61-95.

Drabble, B., and Tate A. 1984. "Use of Optimistic and Pessimistic Resource Profiles to Inform Search in an Activity Based Planner," *Proc. AIPS94*.

Estlin, T., Chien, S., and Wang, X. 1997. "An Argument for an Integrated Hierarchical Task Network and Operator-based Approach to Planning," in *Recent Advances in AI Planning*, S. Steel and R. Alami (eds.), Notes in Artificial Intelligence, Springer—Verlag, 1997, pp. 182-194.

Fukunaga, A., Rabideau, G., Chien, S., Yan, D. 1997. "Towards an Application Framework for Automated Planning and Scheduling," *Proc. of the 1997 International Symposium on Artificial Intelligence, Robotics and Automation for Space*, Tokyo, Japan, July 1997.

Kautz, H., and Selman B. 1996. "Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search." *AAAI-96*.

Knight, R., Rabideau, G., and Chien, S. 2000. "Computing Valid Intervals for Collections of Activities with Shared States and Resources." *AIPS-2000*.

Laborie, P., Ghallab, M. 1995. "Planning with Sharable Resource Constraints," Proc. IJCAI-95, 1643-1649

Muscettola, N. 1993. "HSTS: Integrating Planning and Scheduling." *Intelligent Scheduling*. Morgan Kaufmann, March 1993.

Rabideau, G., Chien, S., Backes, P., Chalfant, G., and Tso, K. 1999. "A Step Towards an Autonomous Planetary Rover," *Space Technology and Applications International Forum*, Albuquerque, NM, February 1999.

Rabideau, G., Knight R., Chien S., Fukunaga A., Govindjee A. 1999, "Iterative Repair Planning for Spacecraft Operations Using the ASPEN System," *iSAIRAS '99*, Noordwijk, The Netherlands, June 1999.

Sherwood, R., Govindjee, A., Yan, D., Rabideau, G., Chien, S., Fukunaga, A. 1998. "Using ASPEN to Automate EO-1 Activity Planning," *Proc. of the 1998 IEEE Aerospace Conference*, Aspen, CO, March 1998.

Tate A., Drabble, B., and Dalton, J. 1996. "O-Plan: a Knowledge-based planner and its application to Logistics," *Advanced Planning Technology, Technological Achievements of the ARPA/RL Planning Initiative*, AAAI Press, 1996, pp. 259-266.

Vidal, V., and Regnier, P., 1999. "Total Order Planning is More Efficient than we Thought." *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, AAAI Press, 1999, pp. 591-596.

Wilkins, D., 1998. "Using the SIPE-2 Planning System: A Manual for Version 4.22." SRI International Artificial Intelligence Center, Menlo Park, CA, November 1998.

Zweben, M., Daun, B., Davis, E., and Deale, M., 1994. "Scheduling and Rescheduling with Iterative Repair," in *Intelligent Scheduling*, Morgan Kaufman, 1994.

COMMENT

Merging the Semantics of State Constraints for Collections of Activities

Miriam H. Nadel
The Aerospace Corporation
P. O. Box 92957
Los Angeles, CA, 90009-2957
miriam.nadel@aero.org

Introduction

Coupling between activities can create a number of practical problems for scheduling. Aggregation of activities and related constraints can reduce complexity, allowing greater computational efficiency in scheduling. This commentary is divided into

Comment on Motivation

Interactions between constraints on shared states and resources are clearly a significant issue for scheduling. One key point about scheduling activities individually is that scheduling decisions are not commutative. If you determine that you can't schedule a particular activity, you need to consider whether or not to reexamine the activities that are already on the schedule. Attempting to reschedule existing activities quickly causes one to run into the "curse of dimensionality" and processing time often becomes a practical limitation. However, if you don't attempt rescheduling, underutilization of assets is likely. It is only when all of the activities and constraints are viewed simultaneously, that one can identify all of the scheduling options. Therefore, merging activities effectively reduces the dimensionality of the scheduling problem.

Another consideration is that some types of constraints can be recast in the form of additional activities. Use of antennas for satellite ranging, for example, typically requires a number of setup and configuration activities for the antenna, in addition to the actual ranging data collection. Typically, those activities are just lumped with the collection request. This is inefficient, as it doesn't allow insight into situations in which multiple requests from different users require the same antenna configuration. By treating the configuration as a separate activity, which interacts with the collection activity, it may be possible to achieve more efficient schedules.

A final practical motivation arises in a multi-user environment, where political considerations can affect the scheduling process. Prioritization of tasks (done by some external agency, rather than the developers of the scheduling system) is typical in such situations. Being able to aggregate activities could eliminate some of the flak when a lower-priority activity gets scheduled because it has a positive interaction with a higher-priority activity, while some activity with intermediate priority is bumped.

Modeling

The essence of the approach in this paper is the modeling of activities as activities in terms of reservations and the transformation of a set of individual reservations into a related set of non-interacting reservations. The wide range of domain examples is interesting and relevant to many other space-related scheduling applications. The effort involved in translating a domain into an expression of a model could be considerable. In practice, translating scheduling problems into any modeling language is often one of the hardest aspects of utilizing existing algorithms and has been one of the reasons for the limited application of several commercial software packages.

Solution Methodology

The concept of transforming the set of interacting reservations, P, into a set of non-interacting reservations, P' appears to be a powerful method. One area that isn't addressed explicitly, but is of considerable practical importance, is conflict resolution. A number of situations are discussed in which no possible non-conflicting set of intervals exists. This information is often useful to users, who may then redefine their requested activities.

Quality Measures - Accuracy

Random placement is effectively a worst case bound on performance (and a best case bound for flexibility). It would be interesting to call out false positives vs. false negatives when assessing accuracy. Depending on the exact problem, one may be more significant than the other. For example, if you have an undersubscribed system, false negatives may not be a major concern, although they're undesirable from the standpoint of producing an excessively conservative solution. It's harder to think of a situation in which false positives would be insignificant, but that might be the case where some constraint can be violated temporarily. For example, a sensor tasking system may allow violation of a power constraint for a brief period of time.

Quality Measures - Speed

The difference in computational time for the informed and the naïve approaches is considerably different for the domains discussed. The number of activities involved (or, in particular, the activity group size) is clearly one factor in how long it takes to solve a given problem, but this is true for both scheduling algorithms. How many interactions there are between activities is probably predictive of how much longer the informed approach will take for a given algorithm, but this is not necessarily obvious a priori for a given domain.

Of course, computational load is more significant for some problems than for others. A very stressing example is near-real-time tasking of one sensor in response to an observation by another sensor. That case requires detecting an event, recomputing the schedule, calculating commands to slew the sensor to the new location, and transmitting commands to the satellite (possibly via ground relays, depending on satellite location) in a very short amount of time. Similarly stressing timelines arise in other domains, e.g. weapon to target assignment for ballistic missile defense.

Efficacy Assessment

Efficacy assessment is a broader issue than just effectiveness in conflict resolution and the number of problems solved. Because the problems used for the empirical evaluation here were generated randomly, it's not easy to place bounds on how many problems could be solved. That is, we don't know how many solvable problems the informed search method was still unable to solve. This is important because a lot of real world problems are unlikely to solvable. An oversubscribed system can never schedule all of the requested activities and, therefore, other metrics may be more appropriate in those circumstances.

Conclusion

The background of this commenter includes sensor tasking for surveillance applications, as well as scheduling of antenna resources for satellite control applications. The focus of the commentary has, therefore, been on considerations for practical applicability to problems that arise in implementing schedulers in these domains. The commenter hopes that the issues raised in this paper will highlight pertinent issues in broadening the application of the methodology.

Questions?

Contact Miriam Nadel (miriam.nadel@aero.org)

THE XMM MISSION PLANNING PROCESS AND TOOLS

Mario Merri[1], Ian M. Palmer[2], Fabienne Delhaise[1], Nestor Peccia[1]

[1] ESA/ESOC, Robert-Bosch-Str 5, D-64293 Darmstadt, Germany.
Fax: +49 (6151) 903010, E-mail: mmerri@esoc.esa.de, fdelhais@esoc.esa.de, npeccia@esoc.esa.de

[2] Logica UK Ltd, Stephenson House, 75 Hampstead Road, London, NW1 2PL, United Kingdom. Fax: +44 (171) 344 3669, E-mail: palmeri@logica.com

ABSTRACT

The X-ray Multi Mirror Module (XMM) spacecraft, an X-ray observatory that was launched on 10th December 1999, is the largest ESA scientific spacecraft ever built, with a mass of nearly 4 tonnes. One of ESA's cornerstone scientific projects, the objective of the XMM mission is to obtain the most comprehensive survey of X-ray sources, in total approximately 1,000,000 celestial targets are expected to be observed. The XMM payload comprises 3 high sensitivity X-ray imaging cameras, 2 high-resolution X-ray spectrometers and one additional instrument covering the optical wavelengths. The instruments are designed to be operated in parallel so that efficient use of observation time is made.

This paper describes the planning constraints of the XMM mission and explains the concept behind the mission planning process, in particular how the activities of the Science Operations Centre (SOC) and the Mission Operations Centre (MOC) are coordinated to make the most of the observing time available for science taking. A detailed description of how proposals for observing time are gathered from members of the scientific community, how they are evaluated and subsequently inserted in the XMM operations schedule, and how telecommands are generated and uplinked to the spacecraft is included.

Keywords: Mission planning, Proposal handling, Web-based system, Scheduling, Simulated annealing.

Introduction

In common with previous scientific ESA missions, the planning and operation of the XMM spacecraft and payload is achieved through the interaction between two geographically distinct bodies. The XMM Science Operations Centre (SOC) is responsible, among other things, for planning scientific observations and processing and archiving the resultant data. The XMM Mission Operations Centre (MOC) controls the operations of the spacecraft and instruments and ensures that the spacecraft's safety is not compromised. A similar high-level planning strategy was also used for ESA's highly successful observatory mission, the Infrared Space Observatory (ISO) launched in November 1995 and operated until April 1998.

XMM is an observatory mission and as such its telescope needs to be pointed towards x-ray sources. Because of the faint nature of the x-radiation, XMM observations are usually long, in the order of a few hours. The spacecraft orbits around the Earth with perigee and apogee of approximately 7,000 and 114,000 km, respectively. Two ESA ground stations, Kourou (in the French Guyana) and Perth (in Australia) are used to command XMM and to receive its telemetry. Except for a short period around perigee, the spacecraft is always visible from one of the ground stations.

The paper describes the constraints imposed on the planning system by the spacecraft, the instruments and the observation strategy. The remainder of the paper describes in detail the end-to-end mission planning process, from the capture of a scientific proposal through to the uplinking of spacecraft and instrument telecommands for execution. In addition, an outline of the main planning tools and techniques used in the SOC planning subsystems can be found.

Most of the MOC and SOC subsystems have been developed for ESA by a consortium of European Industries lead by Logica (UK) that included Dataspazio (I) and GMV (E).

Definitions

Before the planning process is described in detail, it is first useful to define a few terms used throughout the paper:

Proposal - a proposal is defined to be a collection of observations which can themselves be either scientific, calibration, or engineering in nature. Proposals of scientific observations are the principal proposal type and are created and submitted by the X-ray scientific community.

Observation - an observation is the term given to a single, stable 'pointing' of the -X axis of the spacecraft, to which the instrument boresights are nominally aligned. Thus, there is a single designated target (a point on the celestial sphere) associated with each observation. An observation consists of one or more exposures.

Exposure - an exposure is defined to be a single fixed configuration period of an instrument. It is analogous to the notion of an exposure for an optical camera, for which the term defines the camera settings in terms of shutter speed and aperture size required to record an optical image on film. Onboard XMM, exposures on the different instruments can be made in parallel.

Planning Strategy

As there is almost always visibility from the XMM ground stations of Perth and Kourou during science taking periods, the spacecraft has been designed with limited on-board storage capability for both

telecommands and telemetry. This implies that the operational concept is to have the control loop closed at the ground. In this context, the planning strategy for the XMM operations is one in which:

- all science observations and associated operations are pre-planned;
- all planned and contingency spacecraft and instrument operations are performed under on-line command from the MOC and the SOC in real-time. That is, there is no onboard time-tagged command schedule to be used for nominal scientific operations. XMM does have an onboard time-tagged command buffer, but this is to be used by the MOC only for safety and security critical situations;
- there is limited onboard storage of science telemetry. All scientific observation data is downlinked to ground in real-time as it is obtained.

The science mission planning process is an off-line activity and can be broadly broken down into 2 distinct areas: proposal handling and observation sequence planning. These are described in detail later in the paper.

Planning Constraints

There is a number of planning constraints that must be taken into account by the mission planning systems in order to ensure that instrument and spacecraft safety is not compromised. XMM comprises 3 high sensitivity X-ray imaging cameras (EPIC - European Photon Imaging Camera), 2 high-resolution X-ray spectrometers (RGS - Reflection Grating Spectrometer) and one optical instrument (OM - Optical Monitor). To operate the instruments safely and to obtain good scientific data from the instrument, XMM performs scientific observations only when its altitude from the Earth is above the 40,000 km, i.e. above the radiation belts. The pointing constraints that are imposed in order to operate the spacecraft and its on-board instruments within safe operating limits are as follows:

- the angle between the telescopic line of sight (-X axis) and the sun direction must be $90° \pm 20°$;
- the roll angle must be less than $20°$;
- during scientific observation periods, the telescopic line of sight must be more than $47°$ from the earth limb and $22°$ from the moon limb;
- outside scientific observation periods, the telescopic line of sight must be more than $35°$ from the earth limb and $3°$ from the moon limb.

Avoidance criteria are also likely to be applicable for other solar system objects; these are, however undefined at the present time.

The pointing constraints imposed by the physical limitations of the spacecraft sensors and the instruments are not the only constraints that are observed during the planning process. The following considerations fall into the category of operational constraints:

- time to perform routine space and ground segment maintenance operations needs to be allocated as part of the routine planning cycle;
- de-configuration of the instrument and spacecraft subsystems prior to 'unsafe' events such as solar eclipse periods, entry to perigee passage and so on, needs to be scheduled;
- de-configuration of the instrument and spacecraft subsystems prior to the 'apogee gap' experienced once per orbit by the spacecraft. This gap in ground station visibility arises due to the adoption of the highly inclined 40 degree southern orbit and lasts approximately 1 hour each orbit.

Proposal Handling

The start of the mission planning process is the capture, evaluation and enhancement of proposals of scientific observations from the X-ray community. This is called *proposal handling*.

Procedure

Proposal creation occurs in response to the Announcement of Opportunity (AO) made by the XMM Project Scientist. Representatives from the X-ray scientific community, Principal Guest Observers (PGOs), are then invited to submit proposals for scientific observations to the SOC where they are evaluated by SOC staff in the next stage of the planning procedure. Proposals can be created and submitted directly into a database of unevaluated proposals by any scientist with access to a suitably configured World Wide Web (WWW) browser. For those without access to a browser, the procedure involves submitting the required observation details to SOC staff directly (via e-mail or fax) who can enter the information using a similar tool to that available to the external observers.

All submitted proposals must undergo evaluation prior to being made available for scheduling for the following reasons:

- to remove duplicate observations from being scheduled; the XMM observing time is a valuable resource and in order to meet the mission objective to observe 1 million celestial targets, this must not be squandered. Various tools are available to help the SOC staff responsible perform this activity and to enable them to compare an unevaluated proposed observation with evaluated and approved observation requests;
- to 'enhance' the submitted scientific observations to ensure that the maximum scientific return is obtained each individual spacecraft pointing. Enhancing an observation can be achieved by including additional instrument exposures for instruments other than the designated prime one and by tuning the spacecraft attitude to include objects of interest in the instrument field of views additional to the designated observation target.

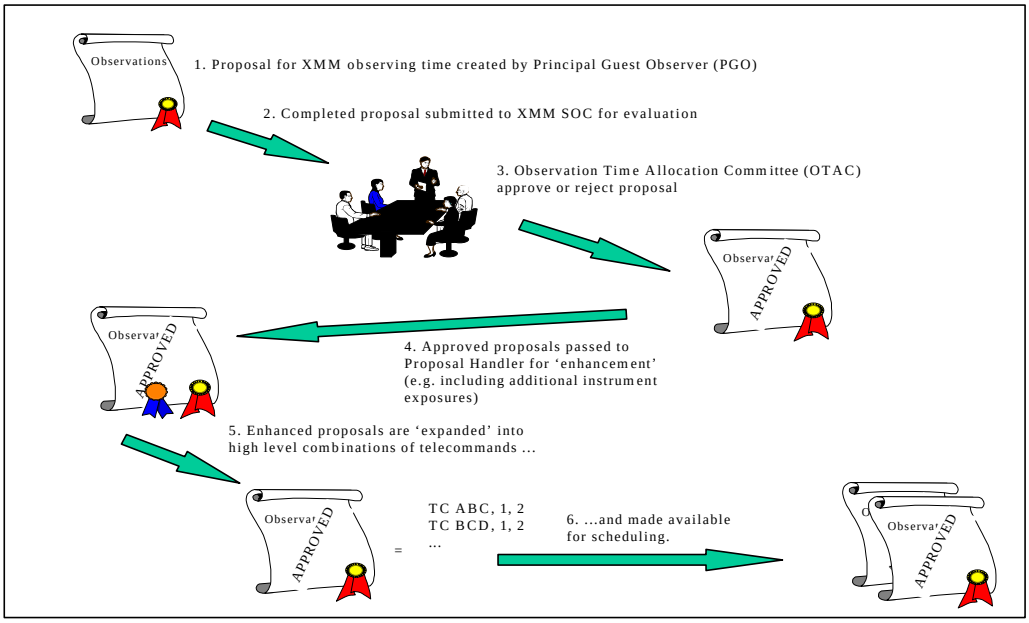


Figure 1: Proposal Handling Stages

The evaluation procedures are carried out by Proposal Handlers, SOC staff members, and by the Observation Time Allocation Committee (OTAC). Approved observations are assigned a scheduling priority that is used at the sequence generation stage of the process to ensure that the highest priority observations are included in the spacecraft operations. The principal stages of the proposal handling process are shown in Figure 1.

Proposal Handling Tools

The Proposal Handling Subsystem (PHS) is the name given to the collection of tools available to support the proposal handling process. These are described below.

Remote Proposal Submission (RPS) - this is a WWW site available to the scientific community and which provides proposal creation, editing and validation functions. The RPS comprises a set of Active Server Pages (ASP) written in Visual Basic and which use an Open Database Connectivity (ODBC) interface to an underlying ORACLE database, which stores the temporary proposal information. By building certain instrument configuration 'rules' into the database and linking the ASPs to the database the tool provides on-line validation at each stage of proposal creation. Following the completion of the temporary proposal details, the PGO can 'submit' the proposal, which effectively places the information under configuration control within the main Proposal Database (PDB) of the SOC. At this stage, the status of all observations in the proposal is set to 'Entry'.

Proposal Editor (PE) - this is the name given to the tool available for use by the SOC staff only. This tool can be accessed via any WWW browser and is based on the RPS in an attempt to ensure the maximum commonality and re-usability of software. The functions of the PE are however, much more sophisticated and enable the Proposal Handler to specify instrument modes and configurations not available to the scientific community. The PE is used by the Proposal Handler to enhance a submitted proposal following approval of the proposal from the OTAC. All modifications to the original proposal are configuration controlled, so that previous edits can be reinstated if required. In addition, the tool provides functions to create new proposals; this enables instrument scientists at the SOC to store 'calibration' and so-called 'engineering' proposals in the PDB for inclusion in the routine mission planning observation sequence. For instance, it is possible to carry out calibration of one instrument by using its internal calibration source, while another one is being used to observe a scientific target. In this manner, less 'down-time' of the instruments is required, thus increasing the scientific efficiency.

Proposal Tools (PT) - there are a number of stand-alone tools which are used by the Proposal Handler to evaluate the scientific merit of an observation and to calculate additional information required to configure the instruments for the series of exposures within the observation. These are collectively called the Proposal Tools and the various functions have been integrated into a single Man Machine Interface (MMI) to provide the Proposal Handler with a uniform procedure to control this activity.

Observation Sequence Planning

Observation Sequence Planning is the term given to the selection of approved observations scheduling them in a time-ordered sequence within a planning period (i.e. a period of time for which spacecraft and instrument operations are being planned).

Procedure

The cycle for generating observation sequence schedules revolves around a 2 week planning period and is based on the concept of

'filling in' a skeleton, or outline, plan with increasing levels of detail at each processing stage, until finally a complete operational plan is created.

A number of different subsystems are involved in the evolution of the skeleton plan into the final sequence of time-ordered telecommands ready for uplinking to the spacecraft. The primary systems involved in this process are:

- Flight Dynamics System (FDS), located at the MOC;
- Sequence Generation Subsystem (SGS), located at the SOC;
- Mission Planning Subsystem (MPS), part of the XMM Control System (XMCS) located at the MOC.

The data flows between these systems and the associated timings are shown in Figure 2. The process is initiated 28 days before the period being planned is due to start, by the production of seven Planning Skeleton Files (PSF) by the Flight Dynamics System (FDS) at the Mission Operations Centre (MOC). Each PSF contains 'windows' of time for a single orbit, in which the SOC Sequence Generation Subsystem is able to schedule observation time. Other periods of time not available for scientific planning purposes (e.g. for spacecraft or ground segment maintenance) are identified in the file, along with orbit related events, such as eclipse information and star-tracker operational characteristics, that must be used during the planning of the observation sequence.

PSFs are received at the SOC by the SGS and are used, together with the database of schedulable observations and the planning constraint information, to produce a set of Preferred Observation Sequence (POS) files and Instrument Command Parameters (ICP) files. The POS is essentially an extended PSF, since it contains all the PSF information, but includes additional data defining the spacecraft attitudes required for each observation in the sequence and the time-related observation and exposure commanding details. The ICP is an auxiliary file used to contain the instrument command parameters. Like the PSF, each POS and ICP pair contains information pertaining to a single revolution. POS and ICP files, 7 pairs of files corresponding to 7 orbits, are sent back to the MOC, 14 days prior to the planing period starting. This is received by the FDS, which checks that the observation sequence defined in the POS is feasible with regard to the spacecraft pointing and operational constraints. The FDS also generates the necessary commands and parameters that are required by the spacecraft Attitude Orbit Control System (AOCS) and Startracker (STR) on-board systems to manoeuvre the spacecraft between the different observation attitudes. The result of this processing is an additional file pair, called the Enhanced Preferred Observation Sequence (EPOS) and the Attitude Parameter File (APF). These are basically extensions of the POS and ICP files described earlier, and again, there is one file pair per orbit being planned. The EPOS and the APF files are submitted to the XMCS Mission Planning System less than one week after reception of the POS/ICPs. There, they are placed under configuration control and validated one final time against the telecommand database before being used to generate the Timeline and the Timeline Summary File (TSF). The timeline is submitted to the XMCS Telecommanding Subsystem (TCS) for uplink to the spacecraft and the TSF is transferred to the SOC as a confirmation of the acceptance of the observation sequence.

Sequence Generation Tools

The Sequence Generation Subsystem (SGS) comprises a set of manual and automatic tools to enable the Mission Planner at the SOC to define the observation sequence. They are briefly introduced below.

Session Manager (SM) - this controls the mission planning session and provides a number of high level functions. For example, an interface is provided to support the introduction of the PSFs from the MOC along with simple viewing functionality to inspect the files. In addition, the SM validates the POS/ICP files produced by the Schedule Editor (see below) against the telecommand database and initiates the transfer of the files to the MOC for further processing.

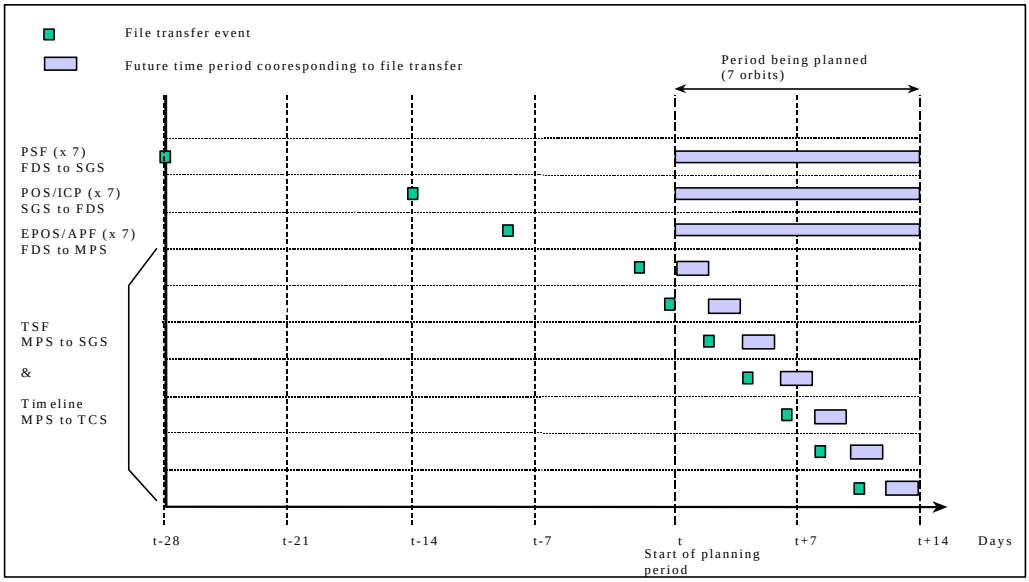


Figure 2: Mission Planning Data Flows

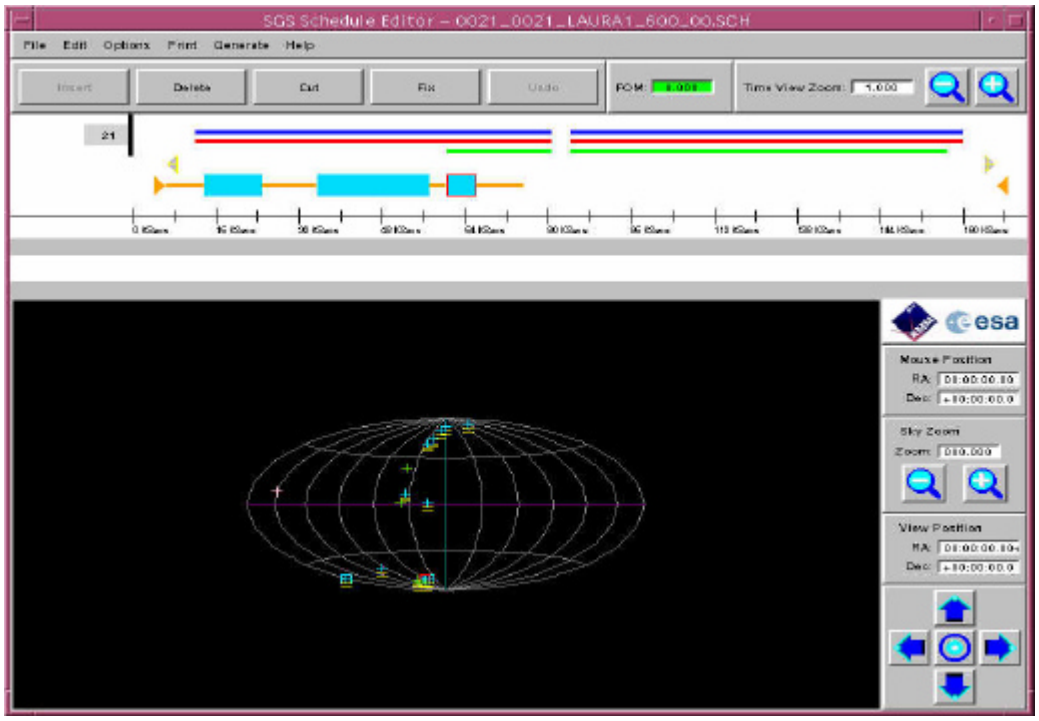


Figure 3: Snapshot of the XMM SGS Schedule Editor

Schedule Editor (SE) - this is the primary planning tool used by the Mission Planner. It is essentially a GUI editor, providing the Mission Planner with visualisations of the planning and observation data and also provides him/her with the functions to create and manipulate an observation sequence, whilst enforcing all planning constraints. The SE comprises 3 main displays: the Timeline View, which is a temporal representation of the planning constraint information and the observation sequence for the current planning period; the Sky View, which is a spatial representation of the celestial sphere (2 projections are provided, Aitoff and Orthographic Sin), showing candidate and scheduled observations; and the List View, which as the name suggests, is a text list of candidate observation details. A snapshot of the main window that includes the Timeline and the Sky views of the SGS SE tool is given in Figure 3. These views give the Mission Planner all the information s/he requires to produce and evaluate an observation sequence. When used in manual mode, the SE enables the Mission Planner to add and remove observations from the sequence, or to change the order of existing observations, simply by selecting observations in any of the views and pressing the button to perform the desired operation.

Schedule Optimiser (SO) - this is an important automatic scheduling tool available to the Mission Planner. It provides the means to automatically create different observation sequence solutions, based on a Simulated Annealing (SA) algorithm, which is a simple extension to the 'hill-climbing' algorithm (see next bullet). The schedule optimiser can be invoked by the mission planner at any time while a schedule is being edited by the schedule editor described above. The optimiser works by taking a copy of the current schedule and then producing incrementally 'better' solutions while it runs as a background activity. Each time an improved schedule is created, it is saved to a different electronic file, which is thus available subsequently for manual editing using the schedule editor. When the mission planner invokes the optimiser, there are a number of 'directives' that can be selected to ensure that certain manual scheduling decisions are not overruled by the automatic scheduling. For example, a time period which the optimiser is to work on can be specified, so that other parts of the schedule remain unaffected. Another example concerns the nature of the automatic algorithm itself; during the optimisation the algorithm might choose to remove an observation already in the schedule and replace it with another observation from the candidate pool. This behaviour can be prevented by manually 'fixing' individual observations in the schedule prior to running the optimiser.

In addition, the 'cost function' of the algorithm (described in the bullet point below) can be manually configured before running the optimiser and the results compared with those produced by other cost functions. One operational example of when a different cost function might be used is if an instrument on board the spacecraft fails, or its performance starts to degrade. In this scenario, the cost function can be easily replaced with another one which prioritises the use of the remaining operational instruments.

Simulated Annealing (SA) algorithm – it works by mimicking the annealing process in a cooling metal, whereby the energy level of the metal's molecules is analogous to the energy level, or 'cost', of a particular observation sequence solution. The solution cost is evaluated by calculating a numeric value based on metrics of the

schedule which are deemed to contribute positively, or negatively, to the quality of the sequence. For example, slew time, the time spent by the spacecraft manoeuvring between observation targets, should be minimised in order to spend the maximum amount of time obtaining quality scientific data. However, this must be offset against the requirement to schedule high priority observations, which might be in a portion of the sky further away from the ideal, minimal slew path. By evaluating how the cost of a solution changes in response to variations, or perturbations, which are randomly introduced, the algorithm can increase the quality of the solution. A number of other controlling influences, or parameters, conditions the decisions taken by the algorithm. Probably the most important of these is the definition of a 'cooling curve'. The cooling curve, as its name suggests, mimics the natural process of reducing temperature over time. This is used by the algorithm to determine the probability that a perturbation that reduces the quality of a schedule will be accepted before continuing with the next perturbation on the schedule. This feature enables the algorithm to (potentially) converge on a globally optimal solution in the domain, or solution space, rather than simply converging on a localised solution, as occurs in other techniques. The rate at which the algorithm 'cools' can impact the effectiveness of the algorithm; if it cools to quickly the potential to find an optimal solution is reduced; cooling too slowly might mean that the algorithm takes a very long time to converge. The definition of the cooling curve is configurable in the schedule optimiser code, enabling the mission planners to experiment with different curves and to determine a suitable one for the XMM planning problem.

Conclusion

The XMM mission planning process is based on the concept of distributed responsibility. The SOC mission planning components are responsible for scheduling observation time such that they meet the safety and operational constraints imposed by the spacecraft systems and instrument payload. Planning information is added incrementally at different planning stages by MOC and SOC subsystems until an operational schedule is produced. In this manner, the complex planning problem is simplified into a number of less complex and more manageable activities. Efficiency of the XMM mission planning process can be assessed at a number of levels. First, the number of data transfers (and system interfaces) has been reduced from that used for the ISO model, thus reducing the number of system interactions and making the overall system more efficient procedurally. Second, where possible within the cost constraints of the project, the SOC planning tools have been designed with useful automatic functions to assist in the more time-consuming aspects of the process. In addition, verification and validation is performed as early in the planning process as possible to reduce the number of instances where re-planning might be required due to incorrect or incomplete data. Third, the mix of automatic and manual tools to achieve good use of the observing time available, is key to the design of the Sequence Generation Subsystem; human operators provided with sophisticated visualisation tools, are capable of creating good scheduling solutions. Together with the automated support from the simulated annealing algorithm, it is envisaged that very good efficiency will be achieved. Finally, the design of the XMM spacecraft systems and instruments ensures that as much useful information as possible is obtained through parallel operation of the instruments. For some instruments, serendipitous data may even be acquired during slews, which are also scheduled via the SGS.

As of the time of writing this paper, the XMM mission planning is being used to plan the actual mission since January 4th 2000. Despite the limited experience that could be accumulated in this short period, the mission planning process and the associated tools are performing as expected with no major problem to report. Currently, most of the XMM operations are executed using the timelines generated using the planning process and tools described in this paper. This confirms that our system can do the job for XMM!

References

ESA 1995: XMM Technical Requirements, PX-RS-0023, Issue 7, XMM Project.

ESA 1997: XMM Mission Planning Concept, XMM-OPS-TN-0002-SMD, Issue 2.

ESA 1997: XMM Combined MCS-SCS Development Software Requirements Document, Issue A4.

University of Aizu 1996: Mathematical Optimisation, Computer Science Education Project.

Commentary on "XMM Mission Planning"

Erann Gat
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
gat@jpl.nasa.gov

Commentary

I find myself in agreement with the review: this paper describes what seems to be a mostly manual scheduling process, and does not get into enough detail to understand the obstacles towards further automation. The reader is left with more questions than answers. For example, the first sentence of the second paragraph of the introduction says, that there are significant differences between the spacecraft that result in differences in operational considerations that must be addressed by the planning system. What are those differences? And how do they impact the planning process?

Section 4 purports to answer some of these questions, but does not address the differences between XMM and ISO. I am also left with many questions about the underlying rationale for some of the constraints. Why must the roll angle be less than 20 degrees? What is the eclipse season? How often must routine maintenance be done?
Why is there no on-board command buffer or on-board data storage?

Some of the data in Table 1 looks like it might contain typographical errors. Is the angular constraint on the reflection grating spectrometer relative to the moon limb really one minute of arc? And there is no unit on the constraint for the photo imaging camera relative to the Earth limb.

One could envision an "ideal" planning system that would schedule the spacecraft based on on-line user requests with no manual intervention whatsoever. Is this possible? If not, how close could one potentially get, and what are the fundamental factors that would prevent further progress? What do you see as the major problems that need to be solved in order to make progress in that direction? (Do you even see that scenario as an ideal situation? If not, why not?).

First published by the author in "A unified dynamic approach for dealing with Temporal Uncertainty and Conditional Planning," AIPS 2000, p. 395-402, Copyright (c) 2000, American Association for Artificial Intelligence.
Reprinted with permission from AAAI.

Dealing with Temporal Uncertainty and Reactivity in a space mission plan

Thierry Vidal
LGP/ENIT
47, av d'Azereix - BP 1629 - F-65016 Tarbes cedex
thierry@enit.fr

Abstract

In the Deep Space One project, temporal uncertainty in a plan is taken into account through Temporal Constraint Networks in which one further distinguishes between controllable and uncontrollable events. Two properties have been introduced to check the validity of such a plan, namely the Dynamic and the Waypoint Controllabilities. This paper presents a possible way to combine both of them for verifying such a plan before execution, using a game-based strategy based on an automaton formalism, and shows how this tool may model as well more complex reactive behaviours and be used as an execution control tool.

Background and overview

Temporal Constraint Networks (TCN) (Schwalb & Dechter 1997) rely on numerical constraint algebras and are well-suited for modelling a temporal plan (Morris, Muscettola, & Tsamardinos 1998) and check its temporal consistency. In realistic applications the inherent uncertain nature of durations of some tasks must be accounted for, distinguishing between *contingent* constraints (whose effective duration is only observed at execution time) and *controllable* ones (which instanciation is controlled by the agent): consistency must then be redefined in terms of the *Dynamic controllability* (Vidal & Fargier 1999) which encompasses the reactive nature of the solution building process in dynamic domains, checking that a solution can be built in the process of time, each assignment depending only on the previous observations, and still needing to account for all the possible remaining ones. Then *Waypoint controllability* (Morris & Muscettola 1999) means there are some points which can be assigned the same time of occurrence in all solutions, which allows to add wait periods in a plan and hence partition it in more tractable subparts. All that is recalled in next section.

Using a Dynamic controllability checking method through a Timed Game Automaton (third section), the paper shows how in planning this space costly process can be confined in subparts using waypoints, and used as a control execution tool (fourth section). Then some hints are given about using the TGA for modelling more complex reactive behaviours (last section).

Contingent TCN and Controllability

Temporal plans can be represented through Temporal Constraint Networks (Schwalb & Dechter 1997) consisting of time-points (graph nodes) related by constraints (graph edges) that might be mere precedence ($\preceq$) relations, or continuous binary numerical constraints defining the possible durations between two time-points x and y by means of temporal intervals: $l_{xy} \leq (y - x) \leq u_{xy}$ or $c_{xy} = [l_{xy}, u_{xy}]$[1]. A TCN is said to be consistent if one can CHOOSE for each time point a value such that all the constraints are satisfied, the resulting instanciation being a *solution* of the TCN. That can be checked in polynomial time through propagation algorithms.

But one may wish to extend the expressiveness distinguishing between two kinds of time-points: the time of occurrence of an *activated* time-point can be freely assigned by the agent, while *received* time-points are those which effective time of occurrence is out of control and can only be observed. This raises a corresponding distinction between *controllable* and *contingent* constraints (*Clb* and *Ctg* for short): the former can be restricted or instanciated *by the agent* while values for the latter will be provided (within allowed bounds) *by the external world* (see (Vidal & Fargier 1999) for details). For instance, in planning, a task which duration is uncertain and will only be known at execution time will be modelled by a Ctg between the beginning time-point which is an activated one and the ending one which is a received one. The introduction of such uncertainty in the STP results in the following definition of the corresponding *Contingent TCN*.

Definition 1 (CTCN) $\mathcal{N} = (V_b, V_e, R_g, R_c)$ *is a Contingent TCN with*
$V_b = \{b_1, \ldots, b_B\}$: *set of the* B *activated time-points,*
$V_e = \{e_1, \ldots, e_E\}$: *set of the* E *received time-points,*
$R_c = \{c_1, \ldots, c_C\}$: *set of the* C *Clbs,*
$R_g = \{g_1, \ldots, g_G\}$: *set of the* G *Ctgs.*

In the following, a *decision* $\delta(b_i)$ will refer to the effective time of occurrence of an activated time-point b_i, and an *observation* ω_i will refer to the effective duration of the Ctg between x_i and e_i.

[1] This is actually the restricted *STP* (Simple Temporal Problem) where disjunctions of intervals are not permitted.

In a CTCN, the classical consistency property is of no use, since it would mean values for Ctgs can be CHOSEN. The decision variables of our problem are only the activated time-points. And hence a solution should be here, intuitively, an assignment of these activated time-points such that all the Clbs are satisfied, whatever values are taken by the Ctgs. This suggests the definition of some *controllability* property. In (Vidal & Fargier 1999), three different levels of controllability have been exhibited (we will barely focus here on the Dynamic one), completed in (Morris & Muscettola 1999) by the Waypoint Controllability.

Definition 2 (Situations and Schedules) *Given that* $\forall i = 1 \ldots G$, $g_i = [l_i, u_i]$,

- $\Omega = [l_1, u_1] \times \ldots \times [l_G, u_G]$ *is called the* space of situations, *and*

$\omega = \{\omega_1 \in [l_1, u_1], \ldots, \omega_G \in [l_G, u_G]\} \in \Omega$ *is called a* situation *of the CTCN.*

Then, for each time t, one can define the current-situation $\omega_{\prec t} \subseteq \omega$ *which is the set of observations prior to t, i.e. such that only Ctgs with ending points* $e_i \preceq t$ *are considered.*

- $\delta = \{\delta(b_1), \ldots, \delta(b_B)\} \in \Delta$ *is called a* schedule, Δ *being the* space of schedules *(i.e. the cartesian product of interval constraints* $(b_i - b_0)$*)*

Then, for each time t, one can define the current-schedule $\delta_{\preceq t} \subseteq \delta$ *which is the sub-schedule assigned so far, s.t.* $\forall b_i \in V_b$ *with* $b_i \preceq t$, $\delta(b_i) \in \delta_{\preceq t}$

In other words, a *situation* is one possible assignment of the whole set of Ctgs, and a *current-situation* with respect to t is a possible set of observations up to time t. And a *schedule* is then one possible sequence of decisions (that might be "controllable" or not), and a *current-schedule* encompasses the notion of reactive chronological building of a solution in plan execution.

Definition 3 (Projection and Mapping)

$\mathcal{N}_\omega$ *is the* projection *of* $\mathcal{N}$ *in the situation* ω*, built by replacing each Ctg* g_i *by the corresponding value* $\{\omega_i\} \in \omega$*. In (Vidal & Fargier 1999) a projection is proved to be a simple TCN corresponding to a STP.*

μ *is a mapping from* Ω *to* Δ *such that* $\mu(\omega) = \delta$ *is a schedule applied in situation* ω*.*

Intuitively, a CTCN will be "controllable" if and only if there exists a mapping μ such that every schedule $\mu(\omega)$ is a solution of the projection $\mathcal{N}_\omega$. In fact this is only the "weakest" view of the problem (called Weak controllability in (Vidal & Fargier 1999)), that assumes that one knows the complete situation before choosing one schedule that will fit. But when a plan is executed, decisions are taken in a chronological way, and for each atomic decision the agent knows the past observations, but the observations to come are still unknown. The Dynamic controllability property defined below takes that into account.

The Waypoint controllability has been further introduced in (Morris & Muscettola 1999), stating that there are some points for which all the schedules share the same time of occurrence, whatever the situation is[2]. Those waypoints serve as "meeting" time-points in a plan, when the agent waits for all the components of a subpart of the plan to be over before starting the next stage. Waypoints can be created during the planning process through the addition of "wait periods" (Morris & Muscettola 1999).

Definition 4 (Dynamic/Waypoint controllability)

- $\mathcal{N}$ *is* Dynamically controllable *iff*
  (1) $\exists \mu : \Omega \longrightarrow \Delta$ *s.t.* $\mu(\omega) = \delta$ *is a solution of* $\mathcal{N}_\omega$
  (2) $\forall (\omega, \omega') \in \Omega^2$*, with* $\delta = \mu(\omega)$ *and* $\delta' = \mu(\omega')$*,* $\forall t$*, if* $\omega_{\prec t} = \omega'_{\prec t}$ *then* $\delta_{\prec t} = \delta'_{\prec t}$
- $\mathcal{N}$ *is* Waypoint controllable *with respect to* $W \subset V_b$ *iff*
  (1) $\exists \mu : \Omega \longrightarrow \Delta$ *s.t.* $\mu(\omega) = \delta$ *is a solution of* $\mathcal{N}_\omega$
  (2) $\forall (\omega, \omega') \in \Omega^2$*, with* $\delta = \mu(\omega)$ *and* $\delta' = \mu(\omega')$*,* $\forall x \in W$*,* $\delta(x) = \delta'(x)$

Dynamic and Waypoint controllability are proven to be exponential in the number of time-points. But the complexity of Waypoint controllability is actually exponential in the maximum number of time-points between two waypoints (Morris & Muscettola 1999), which means the more waypoints one considers in the CTCN, the less complex is Waypoint controllability.

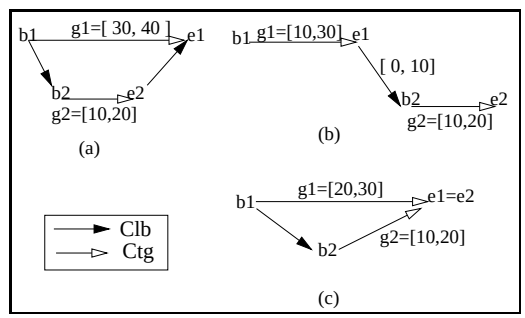


Figure 1: Illustrating controllability

Figure 1 exhibits a threefold example. In the first case (a), one task is constrained to occur *during* the other[3]. For instance that might be a data sending task to an orbiter for a planetary robot that needs to be achieved during a visbility orbiter temporal window, both durations being uncertain. The second example (b) shows two successive contingent tasks, with a maximum (controllable) delay of 10 time units between them. For instance a planetary rover might have to send data within 10 seconds after having correctly directed its antenna towards the orbiter. The third example (c) shows two tasks, one activated after the other, that should finish exactly at the same time, though they

[2]We have chosen to restrict in some sense the original Waypoint controllability definition that allowed the set W to contain received events, since in general those would not satisfy the Waypoint controllability property.

[3]Unlabelled arrows stand for simple precedence, that should simply be $[0, +\infty[$ intervals in the TCN framework.

both have uncertain durations. Obviously this last example cannot be accepted as a valid plan, since no execution can guarantee that the constraint will be met. Interestingly enough, the three examples are all controllable in the weak sense (i.e. condition 1 holds), but only examples (a) and (b) meet condition 2 of Dynamic controllability, which fits what one should expect in realistic planning. Moreover, in example (a), one should get that b_2 must be activated at most 10 time units after b_1 to ensure the satisfaction of the relation whatever values are taken by ω. Last, about Waypoint controllability, in example (b), b_2 shouldn't be set as a waypoint since the time at which one can release the second task depends upon the effective duration of the first one, and hence cannot be set in advance. In (Morris & Muscettola 1999), Waypoint controllability is proven to be equivalent to Dynamic controllability under some restrictive conditions (see page 5), that unfortunately do not hold in case (c) ($W = \{b_1\}$ satisfies Waypoint controllability: actually it lets b_2 depend on the outcome of ω_1 and ω_2), hence Dynamic controllability still has to be checked.

Using Timed Game Automata

The method we present here relies on the *timed automata* model used for describing the dynamical behaviour of a system (Alur & Dill 1994). It consists in equiping a finite-state automaton with time, allowing to consider cases in which the system can remain in a state during a time T before making the next transition. This is made possible by augmenting states and transitions with "continuous variables called *clocks* which grow uniformly when the automaton is in some state. The clocks interact with the transitions by participating in pre-conditions (*guards*) for certain transitions and they are possibly reset when some transitions are taken." Guards may be converted in *staying conditions* (Asarin, Maler, & Pnueli 1995) in states.

Such tools are well-suited (Asarin, Maler, & Pnueli 1995) to *real-time games* for controlling reactive systems, where transitions are divided in two groups (such as constraints in CTCN) depending on which of the two players control it: the *controller* or the *environment* ("Nature") and some states are designated as *winning* for the controller. This extension of the classical discrete game approach, with has the following features: (1) there are no "turns" and the adversary need not wait for the player's next move, and (2) each player not only choose between alternative transitions, but also between *waiting* or not before taking it". Finding a *trajectory* (i.e. a path in the automaton) reaching a winning state defines a so-called *safety game* policy. We propose hereafter our own definition of a Timed Game Automaton that follows those lines and will perfectly fit our purpose.

Definition 5 (Timed Game Automaton)
$\mathcal{A} = (\mathcal{Q}, \Sigma, \Gamma, \mathrm{S}, \mathrm{T})$ *is a timed game automaton (TGA) where*

- $\mathcal{Q}$ *is the discrete and finite set of states* q_i*, with three special cases:*
  - q_0 *is the initial state,*
  - $q_o k$ *is the unique* winning *state,*
  - $q_\dagger$ *is the unique* losing *state;*
- $\Sigma = \Sigma_b \times \Sigma_e$ *is the input alphabet such that any label in* Σ_b *is of the form* b_i *and any label in* Σ_e *is of the form* e_i*;*
- $\Gamma = \Gamma_{sw} \cup \Gamma_{et}$ *is the discrete and finite set of clocks and may be of any cardinality (i.e. one can define as many clocks as one needs), where* Γ_{sw} *is the set of* clocks*, on which are defined two sets of conditions and actions:*
  - $\mathtt{Re} = \{(sw_i \leftarrow 0)$ *s.t.* $sw_i \in \Gamma_{sw}\}$ *is the finite set of all possible clock reset functions,*
  - $\mathtt{Gu} = \{(l_i \leq sw_i \leq u_i)$ *s.t.* $sw_i \in \Gamma, (l_i, u_i) \in \mathbb{Z}^2\}$ *is the infinite set of all possible clock conditions that will be used as guards or staying conditions.*
- $\mathrm{S} : \mathcal{Q} \rightarrow \mathtt{Gu}$ *assigns staying conditions to states;*
- $\mathrm{T} = \mathrm{T}_b \cup \mathrm{T}_e \subseteq \mathcal{Q}^2 \times \Sigma \times \mathtt{Gu} \times \mathtt{Re}$ *is the set of transitions of the form*

  $\tau = < q, q', \sigma, g, r >$ *with a distinction between*
  - $\tau \in \mathrm{T}_b$ *is an* activated *transition* `iff` $\sigma \in \Sigma_b$*,*
  - $\tau \in \mathrm{T}_e$ *is a* received *transition* `iff` $\sigma \in \Sigma_e$*.*

For an activated transition, the controller decides the time of activation by "striking" the clock at any time within the two bounds of the guard, while a received transition will be taken at some unpredictable time within the bounds.

(Vidal 2000) provides a translation algorithm from a CTCN to a corresponding TGA: an event in $\mathcal{N}$ will appear as a translation labelled with this event in $\mathcal{A}$. Similarly, temporal intervals in $\mathcal{N}$ will appear as guards in $\mathcal{A}$. The example of Figure 1(a) gives the corresponding TGA in Figure 2, where on each transition one can view the guard condition and the label above and the clock reset below. Two clocks g_1 and g_2 are used by analogy with the corresponding contingent durations. One can notice that the system is not a priori prevented to receive e_1 before e_2, which would violate a constraint, hence receiving e_1 before e_2 appears as a transition to the losing state in the automaton.

Then (Vidal 2000) provides a *synthesis* algorithm that checks Dynamic controllability, using *controllable predecessors* operator that recomputes from a state q the previous ones, revising the staying conditions so as ensure the losing state cannot be reached any longer. The operator is recursively applied from the initial set of winning states until it reaches a fixed point. If q_0 is in the final set, then the controller can always win the game. This is illustrated in Figure 2: in q_2, $g_1 \leq 30$ is added (and hence the transition to $q_\dagger$ can be removed) and propagated backward to q_1: considering that one may stay up to 20 time units in q_2 because of the uncontrolled clock g_2, then g_1 shouldn't get to more than $30-20=10$ time units in q_1. This condition corresponds to a restriction of the Clb $(b_2 - b_1)$ in the original CTCN.

Modifying the example by replacing the second Ctg by $g_2 = [25, 35]$ would as one should expect it lead the

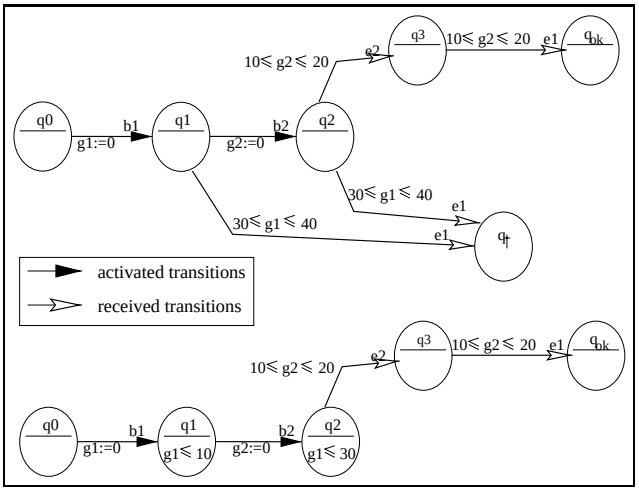


Figure 2: TGA interpretation of a CTCN

synthesis algorithm to fail, since adding the staying condition $g_2 \leq 30$ in q_2 would not propagate back to q_1 considering that one may stay in q_2 up to 35 time units. Hence one would get as a result that the CTCN is not Dynamically controllable.

On the application of TGA in planning

Relevance and conditional nature of the approach

Expressing and reasoning upon temporal uncertainty is needed in most of planning and scheduling applications such as the NASA project Deep Space One (Morris, Muscettola, & Tsamardinos 1998; Morris & Muscettola 1999), where having to deal with contingent durations means basing activation decisions on previous observations, as in Figure 1(b).

A more interesting example is the following. Let us suppose the on-board computation system predicted that a meteorit (that has been detected by a radar) will pass at a minimal distance from the spatial vehicle between 10 and 30 seconds from now. "Now" is the time-point b_1, and the radar will issue event e_1 when the meteorit is at minimal distance (i.e. when the distance starts increasing again). The automatic planning system is given the goal to take a picture of the meteorit close enough to this ideal position, i.e. within 5 seconds before or after e_1. This decision is modelled by time-point b_2. Besides one may easily suppose that the picture should be taken the sooner the better, after what the spatial vehicle is planned to change direction to escape the area in which other meteorits are expected ... Figure 3 shows the resulting synthesized TGA: one shouldn't decide to activate b_2 less than 25 seconds after b_1, but if e_1 is received before, then just activate b_2 within 5 seconds. This is Dynamically controllable, but here one gets two completely different schedules depending on ω_1 being lower or greater than 25, which corresponds to some kind of conditional planning. Moreover, the TGA models a reactive opportunistic behaviour, since the activation of b_2, initially scheduled at 25, might be opportunistically made before on early reception of e_1.

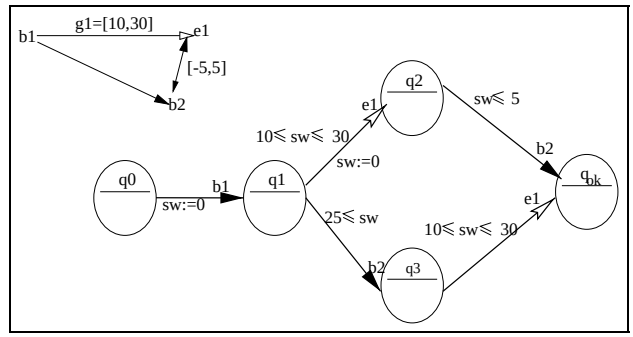


Figure 3: An example of reactive behaviour

Hence the TGA implicitely models reactive behaviours and conditional plans, since two transitions from the same state correspond to a branching (OR node), while in a CTCN time-points are AND nodes. Therefore, the CTCN formalism is a very powerfull tool for describing the *specifications* of a dynamic system (as e.g. a contingent plan), through the constraints it has to meet, expressing rich temporal information in a compact way. Whereas the TGA is a *simulation* model that captures all the possible *execution scenarii* of the plan: it has the advantage of providing efficient and robust techniques to check its "safety", but runs the risk of combinatorial explosion in the number of states.

The TGA as a plan execution control tool

Another strength of the TGA is that it can be directly used as an execution supervisor tool, since the synthesis not only checks Dynamic controllability but also gives schedules of starting times for the planned tasks. Moreover, in constraint networks only a deterministic sequence of decisions is issued, whereas with a TGA one can get a reactive execution supervisor, with disjunctive possible trajectories, which makes it well adapted to stochastic environments.

Complexity and practical efficiency issues

As far as complexity is concerned, the algorithms for building and synthesizing the automaton are respectively linear and in logarithmic time in the number of states, which is in the worst case p^B, where B is the number of activated time-points and p the *degree of parallelism* (i.e. the maximum number of time-points possibly occurring at the same time) (Vidal 2000). Hence the complexity of the approach lies in the possibly exponential number of states developped, which depends upon the network feature p.

This potential combinatorial explosion can be restricted when the plan is mostly a sequence and there are not so many events in parallel, which is often

the case in space mission planning. Moreover, one could use *dispatchable* networks (Morris, Muscettola, & Tsamardinos 1998), that are TCNs in which redundant constraints are removed and only minimal paths are exhibited, so as to optimize propagation during execution. This could restrict as well the number of states produced in the TGA. Besides, automata-based techniques might be improved to reduce the number of states produced, considering that two subsequences containing the same set of events, though in distinct orders, might converge on the same state, or using more complex abstraction views (see (Asarin, Maler, & Pnueli 1995)).

One may also accept an incomplete checking algorithm in the long term (based on incomplete propagations in the CTCN), using the TGA only in the short term, as far as execution runs: the algorithm anticipates the possible losing state deadends and can activate any necessary recovery action in advance.

But the most promising idea seems to be the one that follows ...

An extended framework using TGAs and waypoints

It is argued in (Morris & Muscettola 1999) that choosing cleverly the set of waypoints through addition of "wait" periods in the plan might lead to Dynamic controllability being equivalent to Waypoint controllability. Anyway, trying to design a plan in this way might lead to a high number of waypoints lowering the plan optimality: an opportunistic scenario like the one in Figure 3 would not be possible. In other words, one may compel the executed plan to "play for time" at some points when it is not really needed.

But waypoints might be used merely to restrict Dynamic controllability checking in all subparts of the networks between any pair of waypoints: we need to prove the following property, where $\mathcal{N}_(x, x')$ stands for the restriction of $\mathcal{N}$ to the time-points temporally constrained to be after x and before x'.

Property 1 (Partitioned controllability) *$\mathcal{N}$ is Dynamically controllable iff*

(1) $\mathcal{N}$ is Waypoint controllable wrt $W \subset V_b$, and

(2) $\forall (x, x') \in W^2 s.t.\ x \preceq x'$ and $\nexists x''/x \preceq x'' \preceq x'$, $\mathcal{N}_(x, x')$ is Dynamically controllable

Sketch of proof. The proof is rather straightforward. Dynamic controllability means that a current-schedule will only depend on the corresponding current-situation. For any time t between two waypoints x and x', $\delta(x)$ is set in all schedules, which means it does not even depend on $\omega_{\preceq \delta(x)}$, and any forthcoming decision will not depend on it either. Consequently, $\delta_{\preceq t}$ does only depend on the part of the situation between $\delta(x)$ and t, which is equivalent to the formulation above. $\diamond$

Then, a possible global algorithmic framework to check Dynamic controllability could be to (1) use some heuristic (to be defined at the planning engine level) to decide to introduce wait periods or not while planning, and (2) check Dynamic controllability by building a TGA between any pair of successive waypoints[4]. The idea is to introduce "not so many" waypoints in the plan, so as to still meet in one hand high optimality requirements for the plan, while on the other hand drastically reducing time and space complexity of the TGA approach by only synthesizing automata corresponding to subparts of the whole plan. This decomposition technique hence offers a nice trade-off between expressiveness, optimality and efficiency.

The full expressiveness of TGAs in planning

The expressive power of TGAs is obviously larger than what is used here, and might be of interest for other planning problems.

Generalized conditional planning

A first obvious remark is about the relation between the TGA model and conditional or reactive planning: if the TGA allows to represent the inherent conditional nature of a CTCN, then why not using it for different kinds of branching in planning ? For instance, consider information gathering problems, in which a perception action is included in a plan, and the next steps of the plan depend on the outcome of this action. Or more generally speaking, all cases in which non-determism of actions has to be dealt with. If one wishes to represent distinct evolutions of a plan, then this corresponds to some disjunctions (OR nodes) that are naturally represented in a TGA and may be smoothly merged with temporal contingency branching. Hence TGA might be used in such cases as well. The only difference is that this kind of non-determinism cannot be represented in a compact way in a CTCN, and one can hardly avoid the use of OR nodes in addition to partial order (i.e. AND nodes) in a temporal constraint-based planning graph.

Preprocessed plans and reactive planning

Sometimes a unique plan with branching nodes is not enough to solve a problem. One may need to use a library of subplans to run in reaction to typical events. For instance a planning system may produce a nominal plan together with a number of alternative "recovery" sequences to be runned in replacement when a modeled disturbance occurs, as in (Washington, Golden, & Bresina 1999). Instead of having those subsequences connected to a node of the nominal plan, they may be stored in a library and connected to a type of received and unpredictable event, which defines a more general kind of uncertainty than the one addressed in this paper (not only the time of occurrence of the event is unknown, but the occurrence of the event itself). Receiving this event will force the execution supervisor to

[4]Wait periods will by construction introduce waypoints such that the network is necessarily Waypoint controllable, which hence does not need to be checked.

temporally abandon the current plan to run the corresponding recovery sequence.

Then a more general reactive framework needs to be designed, as in (Vidal & Coradeschi 1999): one can define temporal *chronicles* corresponding to each "abnormal" scenario, with the possibility of having several "faulty" events in elaborated and rather complete scenarios. A purely reactive TGA framework can be designed, where on-line automaton building is processed, in reaction to received events: the system dynamically matches the received event with the chronicles that contain it, and synthesizes the possible next steps in those chronicles in one unique incremental automaton.

By mixing the general off-line planning framework presented in this paper with this purely reactive behaviour, one get a real-time planning system that might be very robust in stochastic environments.

Synchronization constraints

Last, sometimes so-called synchronization constraints that are outside the scope of classical temporal algebras may be useful. Three of them have been defined in (Fargier *et al.* 1998):

Parmin Two "tasks" A and B are related by a parmin if both starts at the same time and the first that is finished terminates the other as well (e.g. task of recording a given sequence of signals from a meteorit and a task waiting for the radar to notify that the meteorit has got out of sight);

Parmaster This is the same as parmin, except that only the first "task" in the relation forces the second one to finish at the same time (e.g. the previous example with the length of the recording sequence not predetermined);

Parmax Two "tasks" A and B are related by a parmax if both starts at the same time and the first that is finished has to wait for the second one to finish as well before next steps in the plan are processed.

The two first ones are interruption-like behaviours, while the third one is an "appointment" constraint. The CTCN model cannot model these constraints: a parmin for instance implies a ternary constraint (Fargier *et al.* 1998) between (1) the expected end of the first task, (2) the expected end of the second task, and (3) the "effective end" of both, that will actually be one of the two previous ones (we recall that in CTCNs only binary constraints are allowed). But interestingly enough, those behaviours are implicitely conditional behaviours, and are very easy to model through a TGA.

Conclusion

This paper has brought to light the advantage of using Timed Game Automata for checking dynamic temporal properties of a plan in the presence of temporal uncertainties. Discussing efficiency and usefullness in practice, it suggests the addition of heuristically and sparingly selected wait periods in the plan to partition it so as to be able to check the Dynamic controllability property locally. The applicability of Timed Game Automata to more general conditional and reactive planning features may as well allow to address more realistic real-time planning problems in stochastic environments.

Acknowledgement

The author is grateful to Paul Morris (NASA Ames Research) for fruitful discussions and his suggestion of the modified Dynamic controllability definition.

References

Alur, R., and Dill, D. 1994. A theory of timed automata. *Theoretical Computer Science* 126:183–235.

Asarin, E.; Maler, O.; and Pnueli, A. 1995. Symbolic controller synthesis for discrete and timed systems. In Antsaklis, P.; Kohn, W.; Nerode, A.; and Sastry, S., eds., *Hybrid Systems II, LNCS 999*. Springer Verlag.

Fargier, H.; Jourdan, M.; Layaïda, N.; and Vidal, T. 1998. Using temporal constraint networks to manage temporal scenario of multimedia documents. In *ECAI-98 workshop on Spatio-Temporal Reasoning*.

Morris, P., and Muscettola, N. 1999. Managing temporal uncertainty through waypoint controllability. In Dean, T., ed., *Proceedings of the 16th International Joint Conference on A.I. (IJCAI-99)*, 1253–1258. Stockholm (Sweden): Morgan Kaufmann.

Morris, P.; Muscettola, N.; and Tsamardinos, I. 1998. Reformulating temporal plans for efficient execution. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*.

Schwalb, E., and Dechter, R. 1997. Processing disjunctions in temporal constraint networks. *Artificial Intelligence* 93:29–61.

Vidal, T., and Coradeschi, S. 1999. Highly reactive decison making: a game with time. In Dean, T., ed., *Proceedings of the 16th International Joint Conference on A.I. (IJCAI-99)*, 1002–1007. Stockholm (Sweden): Morgan Kaufmann, San Francisco, CA.

Vidal, T., and Fargier, H. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence* 11:23–45.

Vidal, T. 2000. Controllability characterization and checking in contingent temporal constraint networks. In *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*. Breckenridge (Co, USA): Morgan Kaufmann, San Francisco, CA.

Washington, R.; Golden, K.; and Bresina, J. 1999. Plan execution, monitoring, and adaptation for planetary rovers. In *IJCAI-99 workshop on Scheduling and Planning meet Real-Time Monitoring in a Dynamic and Uncertain World*, 9–15.

Comments on the Paper "Dealing with Temporal Uncertainty and Reactivity in a Space Mission Plan"

Keyur Patel (keyur.c.patel@jpl.nasa.gov)
Jet Propulsion Laboratory

The author of these comments is not an authority in the area of planning and scheduling or on automaton formalism. The commentator does have significant experience in the area of flight system development, test and mission operations.

The author of the paper has proposed a mechanism for planning and scheduling using Time Game Automata for planning and scheduling problems. The commentator will restrict his comments to only space based missions.

Historically planetary missions pose sequencing problems that are solved in a serial manner regardless of the parallel function of the activities. The author propose an approach that uses waypoints to restrict (control) dynamic controllability. The approach seems to offers a nice trade between expressiveness, optimality and efficiency to help solve on board planning and scheduling problems that have historically been solved on the ground with humans.

The commentator feels that this approach should be developed further and evaluated against parallel activities that occur on a spacecraft that would involve planning and scheduling. The author uses a limited DS-1 example to illustrate the approach. It would be beneficial to take a larger set of DS-1 activities and apply this approach to see if it truly offers an advantage.

SWIM: An AI-based System for Organisation Management

Pauline M. Berry
Artificial Intelligence Center
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
berry@ai.sri.com

Brian Drabble
CIRL
University of Oregan
1269 University of Oregon
Eugene, OR 97403
drabble@cirl.uoregon.edu

Abstract

This paper reports on the development of the Smart Workflow for ISR Management (SWIM) system which has been designed to enable complex systems, business and software, to be controlled within the workflow management paradigm. The system exploits recent advances within the AI community in reactive control, scheduling and continuous execution. SWIM extends the workflow paradigm to respond to the dynamic and uncertain environments by viewing the control processes themselves as dynamic evolving entities. SWIM is being applied to the domain of Information Surveillance and Reconnaissance (ISR), a highly reactive domain where continual and complex requirements for information acquisition, analysis and distribution must be satisfied within a temporal and resource constrained setting. Similar problems occur in the space domain in the development of missions, science experiments, payload check out, etc. The SWIM system comprises two main components: process manager and the dynamic execution order scheduling system. Details of both of these components are provided in the paper.

Introduction

Many applications in the space domain require the close cooperation of a number of individuals and groups e.g. assembly, integration and test, mission management, science experiment planning, etc. These applications can be categorized as a number of agents (human and/or software) working together on a common task. The agents themselves can be within the same group or spread over different continents and time zones. For example, the components of a spacecraft may come from different contractors, e.g. Rockwell, ESA, etc, and their design, development and integration is a complex coordination task. In addition the agents can be working on more than one task, i.e, resources shared across multiple missions, e.g. Galileo, Voyager, etc. SWIM is designed to work at this coordination level and not at the level of controlling values, motors and engines. Coordinating and tasking a group of agents raises a number of key research questions:

- How to identify the best agent for a task? For example, which of my three telescope schedulers is the best for the task?
- If no agent(s) can be found for a task how can it be divided into sub-tasks for which agents can be found? For example, in house testing is no longer available so what is the process to sub-contract the work?
- How is information communicated between agents, when and what should be sent? For example, if the power specification for a probe changes which agents should be notified?
- How can the workload on individual agent be monitored and reallocated with minimum disruption? For example, if the solar panels are three days late which agents should be reassigned?

As described above one of the key questions is dealing with the dynamics of coordinating cooperating and distributed agents. A task can fail for a number of reasons and the reason can be as simple as a missed deadline e.g. the solar panels will be three days late, through to the complete re-tasking of a spacecraft program e.g. we are changing from solar panels to nuclear fusion as the power source. In most cases the failures can be solved by inserting a few new tasks and modifying and/or deleting others. However, these new tasks may cause further knock on effects within the network with some agents becoming overloaded and other sitting idle. This process of task assignment and balancing is often referred to in the literature as process or workflow management. Many domains of interest to the workflow community are characterized by

ever-changing requirements and dynamic environments. However, traditional workflow systems provide only limited reactivity and flexibility. Within the AI community, work on reactive control has led to the exploration of techniques for intelligent process management to meet the requirements of adaptivity for dynamic and unpredictable environments.

This paper describes a revolutionary approach to workflow management using advanced AI planning, scheduling, and reactive control techniques. The system described has been built to manage the highly dynamic processes involved in the intelligence, surveillance and reconnaissance (ISR) domain. The Smart Workflow for ISR Management (SWIM) system comprises two main components: process manager and the Dynamic Execution Order Scheduler (DEOS). The function of the process manager is to select and instantiate processes to deal with *events* occurring in the domain. The function of the DEOS is to develop a coherent schedule for the each of the separate processes identified by the process manager. The schedule assigns resources to each of the activities and tries to maximize the execution window for each activity. Details of the process manager and the DEOS system are provided later in the paper.

The system has been evaluated in the ISR domain to develop plans at the process level, i.e the development of the overall ISR plan, e.g. the development of information needs, collection track generation, etc. This would equate to the process of developing a mission from its inception, through to spacecraft assembly, verification and launch. Changes can occur in many places which require activities being removed, added and in some case modified, e.g. the decision to sub-contract out a component. The first version of the SWIM system was successfully demonstrated to the ISR community in August 1999. The DEOS has been separately evaluated in the air campaign planning domain to generate schedules at the process level, i.e. the development of the overall strategy and at the the lowest level to assign weapon/aircraft to tasks [Drabble 1999].

Motivation

This section provides an overview of the ISR process and shows the links between the problems faced by ISR planner and those faced by mission planners and ground staff.

Intelligence planners are faced with the task of coordinating multiple Information Surveillance and Reconnaissance (ISR) assets to provide as much information about the battlefield as possible and thus

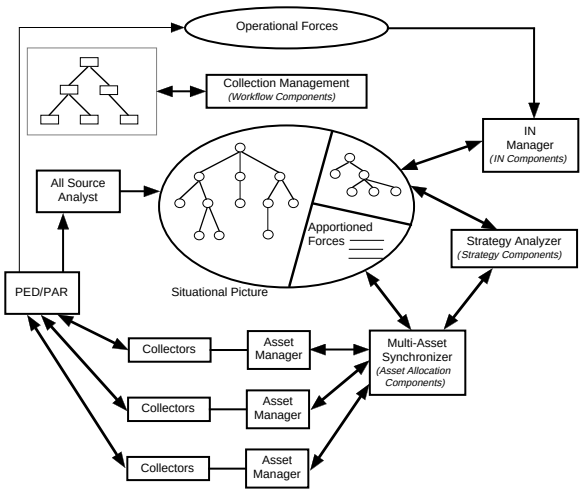


Figure 1: Overview of CPED/PAR Process

increase the effectiveness of fielded forces. The ISR process is driven by Information Needs (INs) which range in complexity from obtaining a complete picture of the electrical power grid of Bosnia to recognizing whether a tank is at a location. Before assets can be tasked with collecting the IN a complex planning process needs to executed. This includes checking if the requested IN is already in a database, is tasked to platform, prioritizing it with other INs, analyzing its overall information gain, etc. A similar process exists in the mission planning domain e.g. a request for an image of the North Atlantic. The INs are divided into different collection tasks depending on time, resources needed, importance, resolution, platform capabilities, risk, etc. The ISR collection (C) tasks cannot be considered in isolation as they are inherently related to processing, exploitation, dissemination (PED) tasks and processing, analysis and reporting (PAR) tasks. This is similar to the case in the space domain where the designers of one component need to be aware of the capabilities and needs of other component developers. The scenario being explored by the AIM project involves the fusion of CPED/PAR tasks as shown in Figure 1.

The development of the SWIM system is funded by the DARPA Advanced ISR Management (AIM) program, which will provide ISR planners with tools for more effective and efficient management of the assets and information in the system. It will address the complete ISR management problem from the strategic development of objectives to the management of individual assets. From a SWIM perspective all these tools (human and/or software) are viewed

as agents that are capable of adding value to the emerging ISR plan. For these agents to work efficiently, they must be organized and managed; that is, a workflow plan that describes how the problem will be solved and identifies the agents necessary to support it must be developed. This will involve the development of intelligent workflow techniques that identify when an agent should be tasked and which tasks are appropriate for the agent to solve. The use of intelligent workflow management together with information discovery and integration are the keys to success in this domain.

ISR Process Modeling Methodology

The AIM process defines the ways in which ISR plans are developed and communicated between agents. Although this planning process involves activities and their coordination, they are described in terms of the activities that take place in the planning process itself (such as plan, analyze, review) rather than containing activities that relate to military effects (such as photograph, intercept). The activities are described using a set of "process verbs" indicating the actions performed at the process level e.g, plan, analyze, review and the process products that are created, modified, used, and authorized within the action. Examples of process products are the documents, reports, orders, letters, and communications (formal or informal) and these are viewed as resources within SWIM. Authority relationships and other conditions can also be modeled and used as an extension of the basic mechanism.

Details of the development of the verb and process product models can be found in [Berry & Drabble 1999]. This framework is general enough to be applicable across various different military domains e.g. logistics, and air campaign planning, as well as many complex business processes and distributed software applications. The *verb/noun(s)/qualifier(s)* (VNQ) model was adopted from a previous modeling exercise that used a similar model to capture air campaign planning processes [Drabble, Lydiard, & Tate 1998]. The SWIM models are encoded in the ACT representation [Myers 1993], which can be directly executed by the Procedural Reasoning System (PRS). It is hierarchical and provides a rich scheme for both the representation of normative processes and the derivation of new processes based on AI reasoning and planning.

The SWIM System

The Smart Workflow for ISR Management SWIM system is a multi-agent framework for performing and managing complex tasks in dynamic and uncertain environments. It provides taskability (i.e., the ability to formulate and execute processes to achieve assigned tasks at different levels of abstraction) and reactivity (i.e., the ability to adapt behavior based on changes in the operating environment). Tasks may be served by different processes and SWIM can select appropriate processes based on context and adapt processes to reflect new environmental or task features. Tasks may also involve long-term commitments that require look-ahead analysis; for this reason, generative planning technology will eventually be used to compose new plans from libraries of process building blocks *(operator templates)*.

SWIM leverages many of the reactive control capabilities from CPEF [Myers 1998], augmenting them with advanced resource allocation, capacity analysis, and scheduling capabilities. CPEF is a novel continuous planning and execution framework embracing the philosophy that plans are dynamic, open-ended artifacts that must evolve in response to an ever-changing environment. Plans are updated in response to new information and requirements in a timely fashion to ensure that they remain viable and relevant, and replaced by alternatives when they are not. SWIM similarly embraces this philosophy, drawing a parallel between plans and workflow processes.

The Procedural Reasoning System (PRS) [Myers 1997], a hierarchical reactive control system, is used as both an executor for workflow processes, and a high-level controller for the overall system. Advanced techniques to effectively schedule tasks onto processing entities are drawn from recent work on the Squeaky Wheel Optimizer (SWO) [Joslin & Clements 1998] and effectively integrated with the process enactment using novel representations that allow process activities to "breath" within temporal windows.

SWIM supports both *direct* models of execution, e.g. actions performed by the system itself, and *indirect* models of execution for which the system supervises execution by a collection of distributed, e.g. a subcontractor for the solar panels. The indirect model of execution is essential for domains where direct control of processing entities is impossible, including many classes of WFM problems. SWIM employs a *procedure library* (encoded in the ACT representation language [Myers 1993]) providing a seamless

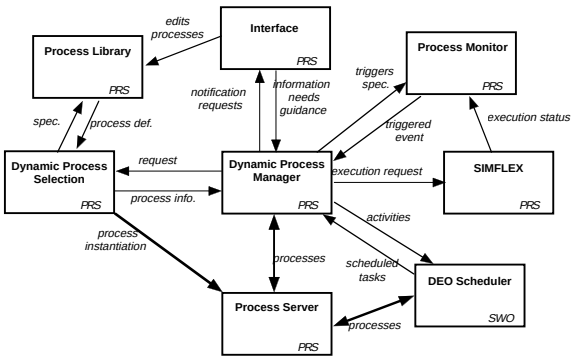


Figure 2: Functional Overview of SWIM

transition between workflow processes (also represented in ACT) and the internal control processes. Elements of the library span multiple abstraction levels and are usable for both process generation and execution, thus supporting smooth transitions between the two capabilities. Process expansion and generation can proceed to arbitrary levels of refinement, with the executor applying additional procedures at runtime to refine tasks to executable activities. Process scheduling and execution operate asynchronously, in a loosely coupled fashion, with agents communicating domain knowledge, activities, requests, and situation information as required. Monitors are automatically created from enacted processes and their constituent activities based on content and context. Future work will define a wide range of monitors, which will be used to support the repair of both specific and generalized types of failure. To date, SWIM has not implemented repair mechanisms but will be building on minimal-perturbation dependency structure methods that have been extended to accommodate generalized failures in CPEF and heuristic techniques.

The SWIM Architecture

The SWIM architecture, shown in Figure 2, is compatible with the WFMC's architectural definition but has distinctive characteristics derived from the field of AI.

Dynamic Process Manager, the hub of SWIM, directing its overall behavior and responsible for managing the instantiation of processes, their dynamic decomposition, their consolidation, the release of activities to the DEO scheduler, the creation of a monitoring plan, and recovery from failure.

Dynamic Process Selector, responsible for the selection, adaptation and evolution of processes. It responds to requests from the Dynamic Process Manager with the appropriate process definition at the required level of abstraction.

Process Library Server, maintains both template processes and process building blocks (or *operator templates*) from which new processes can be generated by the Dynamic Process Selector.

Interface, supports interactions between the user and SWIM and currently includes the ability to edit processes in the Process Library, input requests for information (information needs) and inform the user of critical events and activities.

DEO Scheduler, responsible for the allocation of activities to processing entities (*agents*) over time. It also reasons about windows of opportunity aiding the interaction with higher-level process reasoning.

Process Server, provides a store for multiple processes at a variety of levels of abstraction, e.g. those with scheduled activity information and those without.

Process Monitor, monitors the execution state of the activities and processes using trigger definitions generated automatically by the Dynamic Process Manager.

SIMFLEX (*SImulated FLexible Execution*) a PRS-based simulation environment developed to enable testing, evaluation and demonstration of the dynamic workflow management capabilities of SWIM, [Myers 1998].

Dynamic Process Management

A unique feature of SWIM is the inclusion of an advanced procedural-based reactive controller. The PRS-based controller is organized around an interpreter that runs a tight control loop of *sensing* to detect key changes in the environment e.g. the design, development and launch of Cassini, or sets of assigned tasks, *deliberation* to determine how to respond to sensed changes, and *acting* to execute relevant responses. This approach involves predefined procedure libraries describing processes that can be performed to achieve some goal, or that serve as appropriate responses to designated events (for example, [Firby 1994; Georgeff & Ingrand 1989; Myers 1996; Musliner, Durfee, & Shin 1993; Howe & Cohen 1991]). The bodies of these procedures employ rich operations and control constructs that provide a highly expressive method for representing activity. As such, procedural reactive control is particularly well suited for the *activity-based* paradigm for workflow, although it could readily accommodate artifact and communication-based models

through the introduction of appropriate ontological constructs into the basic process description languages.

DEO Scheduler

The basic concept behind a DEO is to generate schedules quickly and to update them on the fly as new requirements and changes occur. A DEO schedule uses an expressive formalism that breaks down the tasks into the constituent parts. These parts reflect a natural breakdown of the task from a user perspective. In the case of the ISR domain the task is broken down into 4 sub-tasks forming a structure called the PAER model:

- **P**lan: Time to plan the task, e.g. to time taken to identify which telescope scheduler should be used. Once a plan has been identified it is inserted in the slot for other tasks to examine and check. This allows other tasks to identify why the particular scheduler was chosen and the product(s) it will generate, i.e. the telescope schedule.
- **A**cquire: Time to acquire the information necessary to carry out the task, e.g. the observation requests, weather information, etc.
- **E**xecute: Time to carry out the alloted task, e.g. time take to generate the schedule.
- **R**eport: Time to file or report the results of the task, e.g. sending e-mail to the scientists, database update, etc.

Each task is represented by a task specification block (TSB) composed of the PAER sub-tasks. The TSB can "breath" as changes in the domain are reflected as changes in one of the TSB's sub-blocks. For example, if an agent chosen for a task develops a problem during it's **A**cquire phase e.g. the computer collating the observation requests crashes, then the **A**cquire sub task will expand. Alternatively, a second agent may be scheduled with the task inheriting the results from the failing action. An example of a partial PAER network is shown in Figure 3.

The more common reason for a TSB to change is due to a "knock on" effect from another TSB. For example, a change in the **R**eport block of one task may caused the **A**cquire block of a dependent block to expand. By creating a dynamic link between sub-blocks it becomes possible to quickly identify the impact of a change and to identify an appropriate set of repairs. Failures may also cause new TSBs to be added to the schedule to deal with repairs. The dependency links usually reflect an interchange of information between the tasks. The information

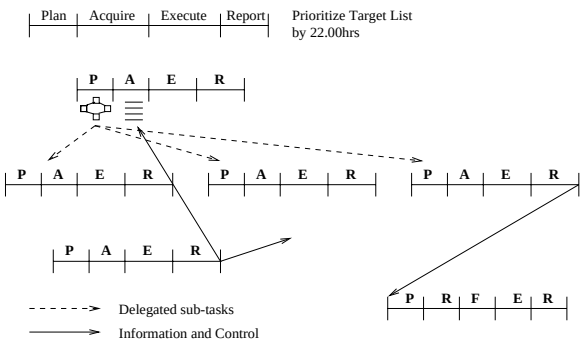


Figure 3: PAER tasking structure

takes the form of products from a task e,g. reports, orders, etc, which reflect new information created or updated by the task. Interactions between the different TSB's is handled by the squeakywheel algorithm which is described later in this section.

A TSB can be generated in response to decision made by other parts of the schedule, e.g. the use of a particular test chamber might dictate that certain additional components are needed. If the test chamber develops a fault and a different test chamber is used then the request for additional components can be removed. The scheduler dynamically launches a new TSB to deal with the chamber substitution and provides feedback to the user of the changes. This approach also has the advantage of hiding unnecessary information, e.g. the staff operating the replacement test chamber do not need to know the details of the failure, just the time at which the spacecraft should be available. By using the DEO model to break the scheduling problem into smaller pieces it allows large scheduling problems to become tractable and maintains the necessary dependencies between the subproblems. Similar task breakdown structures have been developed for other domains including, USAF mission planning and CD manufacturing.

Squeaky Wheel Optimization (SWO) The insight behind SWO is that in any real-world problem it is impossible to capture all associated constraints and that in most cases the context in which the constraints apply cannot be easily determined. SWO uses a priority queue to determine the order in which tasks should be released to a greedy scheduling algorithm. The priority queue is determined by how difficult the task is to deal with that is, the higher the task is in the queue the harder it is to handle it correctly and not by some external priority identified by the user. On each iteration of the algorithm, SWO quickly creates a schedule and

then examines it to identify the parts that were handled badly, for example, task was completed too late or by an unsatisfactory agent. Any task that "squeaks" is promoted up the priority queue, with the distance it is promoted determined by the the extent of the problem. The new priority queue is then used to generate another schedule that is analyzed for problems. This process continues until no significant improvement in the schedule is noted over several iterations. SWO is extremely fast with each iteration taking less than a few seconds, even for large problems.

Summary and Further Work

We have presented an overview of a system for dynamic workflow management. The SWIM system and its generic tasking and process models are applicable across a wide number of applications involving distributed agents (human and/or software) working cooperatively to solve a task. The SWIM system is work in progress and an initial version of the system has been successfully demonstrated in August 1999 to the ISR community. The initial demonstration tackled problems varying from lost assets to large scale re-plans, resource updates and changes in mission objectives. SWIM provides the foundation for workflow enabled reactive control that includes an agent-based architecture, rich modeling and representation of processes and their constituent actions, flexible integration of process instantiation, task allocation and execution, and highly reactive scheduling techniques. Additional experiments have shown that the DEOS scheduler (using a variation of the PAER task model) was able to generate and update schedules for 2500 tasks in less than 5 seconds. Further development of the SWIM system is planned with the main foci being the full integration of the DEOS scheduler, automatic process creation and evolution, advanced failure recovery and repair techniques, and the maintenance and use of the dynamic dependencies between tasks at different levels of abstraction.

In introducing a system such as SWIM there are a number of factors which need to be considered. One of the main concerns is the availability of adequate information on the tasks being performed and the capabilities of the agents being tasked. For example in the design phase it may not be case that a task's duration can be accurately specified. However, it is the case that the system can still perform well with such data, but an answer of between 1 and 60 days may not be useful to a program manager. The current simple models of agent capability need to take into account a number of more realistic concerns, e.g. the change over time between tasks, people's skills changing over time, the combining of skills among agents, etc. It is expected that these simplifying assumptions will be relaxed as more experiments are carried out.

References

Berry, P. M., and Drabble, B. 1999. The AIM Process Modeling Methodology. Technical report, AI Center, SRI International, Menlo Park, CA.

Drabble, B.; Lydiard, T. J.; and Tate, A. 1998. Workflow Support in the the Air Campaign Planning Process. In Myers, K., and Ferguson, G., eds., *Proceedings of the Workshop on Interactive and Collaborative Planningheld as part of Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98)*.

Drabble, B. 1999. Task Decomposition Support to Reactive Scheduling. In *5th European Conference on Planning (ECP-99)*. New York, NY: Springer Verlag Press.

Firby, R. J. 1994. Task Networks for Controlling Continuous Processes. In *Second International Conference on AI Planning Systems*. Menlo Park, CA: AAAI Press.

Georgeff, M. P., and Ingrand, F. F. 1989. Decision-Making in an Embedded Reasoning System. In *Eleventh International Joint Conference on AI*.

Howe, A. E., and Cohen, P. R. 1991. Failure Recovery: A Model and Experiments. In *Nineth National Conference on Artificial Intelligence*.

Joslin, D. E., and Clements, D. P. 1998. Squeakywheel Optimization. In *Fifteenth National Conference on Artificiall Intelligence*. Menlo Park, CA: AAAI Press.

Musliner, D. J.; Durfee, E. H.; and Shin, K. G. 1993. CIRCA: A Cooperative Intelligent Real-time Control Architecture. *IEEE Transactions on Systems, Man, and Cybernetics* 23(6).

Myers, K. L. 1993. *The ACT Editor User's Guide*. AI Center, SRI International, Menlo Park, CA.

Myers, K. L. 1996. A Procedural Knowledge Approach to Task-Level Control. In *Third International Conference on AI Planning Systems*. AAAI Press.

Myers, K. L. 1997. *User's Guide for the Procedural Reasoning System*. AI Center, SRI International, Menlo Park, CA, version 1.8 edition.

Myers, K. L. 1998. Towards a Framework for Continuous Planning and Execution. In *Proceedings of the AAAI 1998 Fall Symposium on Distributed Continual Planning*. Menlo Park, CA: AAAI Press.

Commentary on
ÒSWIM: An AI-based System for Organisation ManagementÓ

Robert D. Rasmussen

Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, California 91109
robert.d.rasmussen@jpl.nasa.gov

Abstract

This is a short commentary on a paper appearing in this workshop by Pauline M. Berry of SRI International and Brian Drabble of the University of Oregon. Their paper reports the development of a workflow management system using AI techniques for reactive control, scheduling and continuous execution. While developed for application in information surveillance and reconnaissance, the authors claim applicability to a variety of domains, including space.

Background

The commentary in this short report comes from the perspective of someone steeped in interplanetary space missions. This undoubtedly distorts the perspective that can be brought to general applications of planning and scheduling technologies for space, because in the experience of this commentator the emphasis so far for AI technologies in this narrower realm has been on increased autonomy and other improvements directed toward the flying vehicle.

The greatest challenge in commenting on the subject paper has been to stretch this point of view to accommodate the authorsÕ visionÑsomething that might be more readily achieved by others. The following commentary is offered in that spirit.

Setting a Context

Discussion of applications for planning and scheduling in the space community has generally involved the following key objectives:

- Automation of routine functions for reduced operations cost
- Reduced development cost and schedule through higher level programming and model based reasoning
- Reduced complexity and greater efficiency in the deployment of limited resources
- Remote system autonomy in uncertain environments and continued operation in the presence of faults
- Improved mission return through in situ analysis and response

In many cases, these are old problems that are well understood. They generally involve the minutiae of a complex, tightly coupled system with many parts (like a spacecraft, rover, or ground station), and organizing all their many constraints and interactions within a context of competing demands on the system. In a sense, one can view the management process for such a system as omniscient, in that an attempt, at least, is made to understand everything and be ready for anything. The principle issues in applying automated planning and scheduling techniques in this domain are in marshalling all the details needed to describe a system, in making it easy for a team of diverse talents to do so and to operate the resulting system, in getting these complex technologies to meet embedded, real-time performance demands, and in convincing a very conservative culture to place their trust in a technology whose products often defy simple explanation. Answers may not be easy, but at least the criteria they need to meet are evident.

When we move into fresher territory and there are no clear solutions to the problems at hand, short of applying AI techniques, a host of difficult issues arise. Most unsettling is the fading option of total understanding and control. We accept this reluctantly, but not without a reserved notion that, even in these cases, we retain a perimeter around the problem that we can police. This is possible because we have still tended to focus on fairly narrow pursuits, such as the operation of a single vehicle in an uncertain environment, or at worst a few systems, most in relatively certain environments. Moreover, weÕve tended to assume that uncertainty was largely resolvable, once encountered. Discussion of planning and scheduling issues in space applications has therefore tended to center around a comparatively disciplined set of approaches.

A Place for SWIM?

SWIM, on the other hand, having addressed the world of information surveillance and reconnaissance for DARPA, has grown out of quite different situation in which the set of assets to be managed is potentially vast, each asset or objective may itself be a complex system with idiosyncratic behavior (e.g., it can include people), goals

are often only subjectively defined, constraints may be soft, and so on. In some ways, this is a much harder problem. Consequently, it seems forced to relinquish a level of rigor that we have assumed for space applications.

Given this disparity, it was at first difficult to grasp how one might bridge the apparent gap between the types of problems SWIM purports to address and those more typically bandied about when space is the subject. However, following some dialog on this issue, the authors have offered in their paper a number of examples tailored to space applications that clarify the potential role for this sort of approach. These examples encompass everything from planning procurements to conducting tests—not the usual sort of thing that comes up in debates over AI applications in space.

Put in this light, the workflow management approach described by the authors, while not of the prevailing kind we generally expect, would nevertheless provide some welcome discipline in processes that often appear to sustain themselves on much more tangled fodder. That is, rather than offering less rigor when considered for unintended applications, SWIM appears to offer more rigor than usual when applied as intended. The introduction of a system like SWIM to a forum of space practitioners, therefore, opens a new avenue of discussion largely unexplored—especially in the unmanned sphere where autonomy is foremost in the common dialog.

With this insight it is worthwhile to explore which aspects of space applications might have the right tenor for this approach. It is hard to imagine, frankly, whether some of the processes we presently suffer could comfortably tolerate the scrutiny such a system would impose. One can certainly see opportunities in project development and test, or in the complex logistics of something like a space station.

More tractable, however, may be some real technical challenges just on the horizon that will arise when present ambitions for ubiquitous presence throughout the solar system begin to be met. These systems begin to exhibit characteristics similar to those described in the paper (though smaller in scope). One can imagine, for instance, several aerobots drifting through the atmosphere of Venus, where it is the collective behavior of several vehicles that must be managed, while for each vehicle individually, behavior is potentially erratic, objectives are fuzzy, at best, and assignments may change from one vehicle to another as atmospheric whims dictate.

Closer to home, fleets of earth observing and communication platforms grow harder to manage every year as numbers increase in bounds. As it becomes necessary to address these units as one large coordinated asset, workflow management systems may become invaluable.

It really doesn't take too much effort to find examples where this sort of thing could become important for space.

A Brief Assessment

SWIM may very well be a contender in this arena. Making such an assessment from the paper, however, is difficult since most of the interesting questions and issues that one might wish to explore cannot be summarized in a short paper. The authors properly motivated the work, and provided a competent summary of the general structure, but SWIM is a large system with many components, each of which could be given only terse treatment.

Details on results so far were likewise short. The work has been performed mainly for the information surveillance and reconnaissance community, so more detail about this application domain would probably have been hard to extrapolate to space. In any event, a practical assessment by the customer of this work would be more illuminating as an indicator of proper bearing. In this vein, the authors did offer a brief discussion of some of the concerns they have in introducing such as system and clearly indicate that much work remains.

What can be said is that where insight into the philosophy behind the approach could be gained, the ideas presented seem to be on track. Recognizing the importance of a continuous planning and execution framework, for instance, was gladly received.

It should also be mentioned that incorporated components, such as the Procedural Reasoning System, the Squeaky Wheel Optimizer, and others, will get a good workout in this architecture, and if nothing else will offer further validation for these items.

Conclusion

It is this commentator's opinion that there is room for systems such as SWIM in the discussion of planning and execution technologies for space. Furthermore, there are elements of SWIM which clearly have potential application in more commonly discussed space applications. There may even be straightforward modifications of SWIM to these applications that can exploit this potential. This would be worth exploring.

Regardless of the technology inside, though, the real key for any such architecture is successful adaptation to a real application fielded in the real world with real users. This should be on the road map for any serious technological endeavor. It will be interesting to see how SWIM evolves in this regard.

Designing and Evaluating an On-line On-board Autonomous Earth Observation Satellite Scheduling System

Gérard Verfaillie
ONERA/DCSD, Toulouse Center
2, av. Édouard Belin, BP 4025
31055 Toulouse Cedex 4, France
phone: +33 5 62 25 26 32
fax: +33 5 62 25 25 64
email: verfaillie@cert.fr

Éric Bornschlegl
European Space Agency, ESA/ESTEC/ESD
Keplerlaan 1
2201 AG Noordwijk, The Netherlands
phone: +31 71 565 34 87
fax: +31 71 565 42 95
email: ebornsch@estec.esa.nl

Abstract

Managing an Earth observation satellite consists in building at regular intervals a feasible optimal sequence of image acquisitions over a given time horizon. Up to now this management was performed off-line and on the ground by semi-automated mission control centers, under the supervision of human operators. But the next generation satellites shall be managed autonomously on-line and on-board.

In this paper, after some background about Earth observation satellites and their management, we present the potential advantages and drawbacks of an evolution towards an on-line and on-board management. Then, we present two versions of a scheduling algorithm, both based on *Dynamic programming*: the first dedicated to off-line management and the second to on-line management. Finally experiments, using simplified models, show that an on-line management, performed on-board and using consequently limited computing resources, may outperform an off-line management, performed on the ground.

Background about Earth observation satellites

Orbit features

Earth observation satellites use circular polar sun-synchronous phased orbits. Combined with the natural Earth rotation, this kind of orbit allows any point of the Earth surface to be flown over several times during the orbit cycle under the same altitude and illumination conditions.

Depending on the orbit parameters, a revolution around the Earth takes about one hour and a cycle about ten days. Thus, depending on the latitude, the time between two successive flights over a given point of the Earth surface ranges between several hours and several days.

Image acquisition

Depending on the mission, satellites are equipped with various observation instruments (either optical or infrared instruments, radars ...). These instruments generally offer various spectral bands and acquisition modes. With an optical instrument, acquisition is possible only during the illuminated part of each revolution.

Image acquisition on both sides of the satellite ground track is generally possible using either mobile mirrors in front each instrument, or satellite manoeuvering ability. Satellites of the next generation will offer manoeuvering ability according the roll, pitch and yaw axes, and thus the maximal flexibility for image acquisition.

Data delivery

Data associated with an image acquisition can be, either directly down-loaded, when the satellite is within the visibility window of a ground station, or recorded on-board and down-loaded within one of the next ground station visibility windows.

Uncertainty about accomplishment

With an optical instrument, the possible presence of clouds may invalidate the image acquisition. In that case, the image has to be acquired another time, using other acquisition opportunities.

User requests

Users order image acquisitions, with associated requirements about the acquisition conditions: possible modes and spectral bands, minimum and maximum pitch and roll angles, minimum and maximum sun angle, either mono or stereo acquisition, time window, possible cyclic acquisition and associated constraints, grounds stations to which data may be down-loaded ... Priority levels or weights are generally associated with each user request.

Physical and technological constraints

The main physical constraint is associated with the instrument resources: at any time, only one image can be acquired using one instrument; moreover a transition time is generally necessary between two successive image acquisitions (changes in acquisition mode, in mirror orientation, or in satellite attitude).

The other constraints are associated with the energy and memory resources. Energy and memory are consumed during and between image acquisitions and are renewed via solar arrays and data down-loading. One of the ways of taking these constraints into account consists in defining for each of them a maximum consumption by revolution or set of revolutions, that guarantees that everything that is consumed will be renewed.

Managing an Earth observation satellite

Global view

Globally speaking, managing an Earth observation satellite consists in building over a given time horizon a sequence of image acquisitions that:

- satisfies all the physical and technological constraints associated with the satellite and its payload,
- satisfies the requirements associated with each user request,
- is optimal according to a criterion that is a function of the set of selected requests.

Scheduling horizon

The scheduling horizon can range from one part of a revolution to several revolutions over one or several days. Large horizons allow the global constraints such as energy or memory constraints, as well as the several accomplishment opportunities associated with each user request, to be better taken into account. But large horizons induce very large combinatorial problems and force to reason about to too uncertain data (accomplishment of selected requests, arrival of new requests . . .).

Selection assessment criterion

Various assessment criteria may be used. The most used is utilitarist: a weight is associated with each user request and the objective is to maximize the sum of the weights of the selected requests. But an egalitarist criterion might be used: a priority level is associated with each user request and the objective is to minimize the maximum level of all the not selected requests. Subsequently, we will assume an utilitarist criterion.

A constraint optimization problem

Thus, managing an Earth observation satellite implies to solve a constrained optimization problem. When compared with classical *Operations Research* problems, this problem lies somewhere between the *Job-Shop Scheduling* problem (tasks to perform, not sharable instrument resources, transition times between tasks) and the *Multi-Knapsack* problem (cumulative energy and memory resources)

Potential representation frameworks

It can be cast in any of the classical frameworks used to represent constrained optimization problems, such as *Integer Linear Programming* (*ILP*; the criterion is linear and constraints can be linearized) or *Constraint Programming* (*CP*).

Potential solving methods

Any of the main associated methods can be used to solve it: either optimal methods like *Branch and Bound* or *Dynamic Programming*, or approximate methods like *Local Search* or *Greedy Algorithms*.

For example, the specific management problem of the French satellite *Spot5*, described in (BLV99), has been cast using the *ILP* and *CP* frameworks (*Cplex* and *Ilog Solver* tools), as well the *Valued Constraint Satisfaction Problem* framework (*VCSP*; an extension of the *CSP* framework to deal with over-constrained problems (SFV95)). It has been dealt with using associated *Branch and Bound* algorithms, as well as *Local Search* or *Greedy* algorithms.

Towards an on-line on-board management system

Current management systems

Earth observation satellites are currently managed on the ground and off-line: image acquisition and data downloading sequences over a given horizon are determined in the frame of semi-automated mission control centers, under the supervision of human operators and then up-loaded using satellite visibility windows.

On-line versus off-line management

That means that all the events that may occur between the sequence up-loading and its execution and may affect the user demand (arrival of urgent requests), the satellite state (information about component failures), or the environment (new meteorological forecasts), are not taken into account.

Only an on-line management, performed either on-board or on the ground via an inter-satellite communication network, can take them into account. But it is sure that such a management may induce strong temporal constraints on the reasoning task and consequently severe limitations of the search.

On-board versus on the ground management

Managing on-board has undeniable advantages: direct availability of the information about the satellite state, independence of visibility windows and of inter-satellite communication networks; complete automation and staff cost reduction . . .

It has also some known drawbacks: limited computing and memory capacities available on-board, high reliability requirements on the hardware and on the software, knowing that they will run without any human supervision . . .

Evaluating before deciding

Before choosing, it may be interesting to run simulations, in order to explore and to assess several trade-offs. That is what we did within a simplified framework.

A simulation framework

A simplified satellite

For the sake of rapid prototyping, we chose to consider a simplified satellite, inspired from the *Spot* family, but methods and algorithms could be extended to many more complex cases.

This satellite has only one optical instrument. We assume that it has only one degree of freedom according the roll axis for image acquisition, provided either by a mobile mirror in front of the instrument or by the satellite attitude control system. Consequently, a fixed acquisition window can be

associated with each image acquisition within a given satellite revolution. We assume also that the transition time between two image acquisitions is a function of the attitude difference of the mirror or of the satellite, necessary to acquire both images. We assume finally that solar arrays and on-board recorders are such that there is neither energy nor memory constraint to check.

An off-line scheduling algorithm

We assume that the off-line computing of an optimal feasible sequence is performed before the illuminated part of each revolution for the whole of this part, from the north to the south pole, on the basis of the information available at this time. This can be easily performed by using an *Inverse Dynamic Programming* algorithm (LC78) based on the following recurrent equations:

$$\begin{array}{lll} if & R_{bef}(r) = \emptyset & \\ then & \mathbf{E}G_{opt-bef}(r) = & 0 \\ else & \mathbf{E}G_{opt-bef}(r) = & max_{r' \in R_{bef}(r)} \\ & & [\mathbf{E}G_{opt-bef}(r') \\ & & +p_e(t,r').g(r')] \end{array} \quad (1)$$

$$\begin{array}{ll} \mathbf{E}G_{opt} = & max_{r \in R} \\ & [\mathbf{E}G_{opt-bef}(r) \\ & +p_e(t,r).g(r)] \end{array} \quad (2)$$

In these equations:

- R is the current set of requests that can be performed over the whole of the illuminated part of the revolution, and $R_{bef}(r)$ is the set of requests that can be performed before performing r, taking into account transition times;
- $\mathbf{E}G_{opt}$ is the optimum expected gain that can be obtained over the whole of the illuminated part of the revolution, and $\mathbf{E}G_{opt-bef}(r)$ is the optimum expected gain that can be obtained before performing r;
- $p_e(t,r)$ is the estimated value at time t of the accomplishment probability of request r, and $g(r)$ is the gain associated with its accomplishment.

An on-line scheduling algorithm

We assume that:

- before the beginning of the illuminated part of each revolution, an optimal feasible sequence has been computed using the above algorithm;
- following any event, like either user request arrivals or removals, or new meteorological forecasts, a new feasible sequence is computed, but resources and time are no more necessarily sufficient to compute an optimal sequence over the whole of the remaining illuminated part of the revolution;
- consequently, using a kind of *anytime* approach (BD94), only an optimal feasible sequence of length h, that is involving h image acquisitions, is searched for, by beginning with $h = 1$ and incrementing h each time it is possible, that is each time no new event interrupts the current reasoning

This can be performed by using an *Inverse Dynamic Programming* algorithm based on the previous equations slightly modified:

$$\begin{array}{lll} if & [R_{bef}(r) = \emptyset] \vee [h = 0] & \\ then & \mathbf{E}G_{opt-bef}(r,h) = & 0 \\ else & \mathbf{E}G_{opt-bef}(r,h) = & max_{r' \in R_{bef}(r)} \\ & & [\mathbf{E}G_{opt-bef}(r', h-1) \\ & & +p_e(t,r').g(r')] \end{array} \quad (3)$$

$$\begin{array}{ll} \mathbf{E}G_{opt}(h) = & max_{r \in R} \\ & [\mathbf{E}G_{opt-bef}(r, h-1) \\ & +p_e(t,r).g(r) \\ & +\hat{\mathbf{E}}G_{opt-aft}(r)] \end{array} \quad (4)$$

In these equations:

- $\mathbf{E}G_{opt-bef}(r,h)$ is the optimum expected gain that can be obtained before performing r, by selecting at most h requests;
- $\hat{\mathbf{E}}G_{opt-aft}(r)$ is an estimation of the optimum expected gain that can be obtained after performing r, function of the satellite state (time, either mirror or satellite attitude ...) when ending request r's acquisition;
- $\mathbf{E}G_{opt}(h)$ is an approximation of the optimum expected gain that can be obtained over the whole of the illuminated part of the revolution, computed when searching for an optimal feasible sequence of length h.

Note that:

- everything that is computed when searching for an optimal feasible sequence of length h is reused when searching for an optimal feasible sequence of length $h+1$;
- the only benefit of an increase in the optimal sequence length is a possible better choice of the next image to acquire; the basic question in an on-line context is indeed: *What to do next ?*
- in our current implementation, everything is computed again from scratch in case of any event modifying the problem data: user request arrivals or removals, new meteorological forecasts ... but more sophisticated implementations could reuse at least one part of the previous computing;
- in the context of this very simplified satellite, the gain that results from the computing of an optimal feasible sequence of only length h is not obvious; it will be clearer in the context of more realistic satellites, where *Dynamic Programming* on large horizons will be no more practicable, off-line as well as on-line.

Experiments

All the experiments we carried out compare an off-line scheduling with an on-line scheduling, both on the illuminated part of one revolution and both performed either on the ground or on-board.

Modeling the user request flow

To carry out experiments, we need a model of the user request flow. We assume a set of N randomly generated user requests, known before the beginning of the illuminated part of the revolution and a flow of N_a added requests and N_r removed requests, randomly generated too, within the illuminated part of the revolution.

Modeling the environment

We need a model of the cloudy cover. This model takes into account the fact that indeterminism about the cloudy cover decreases with the distance t between the current time and the image acquisition time: when t is large, the accomplishment probability is approximated via weather statistics; let $p_s(r)$ be this approximation; when t decreases, the accomplishment probability evolves, either up to 1 with a probability $p_s(r)$, or down to 0 with a probability $1 - p_s(r)$.

Modeling the environment forecasts

We need a model of the meteorological forecasts. This model takes into account the fact that the distance between the accomplishment probability and its estimation by meteorological services decreases with t: the lower t, the more reliable the meteorological information. We assume that off-line and on-line schedulings use meteorological information provided respectively t_{off} and t_{on} hours before image acquisition.

Modeling the computing power

Finally, we model the difference between the computing power on the ground and on-board by a ratio $cpr_{on/off}$ between the latter and the former.

Experimental results

We carried out systematic experiments with:

- $t_{off} = 48$ and $t_{on} = 48$ (first graph), 24 (second graph) or 1 (third graph);
- $N = 150$ and $N_a = N_r = 0, 10, 20, 30, 40$, or 50;
- $cpr_{on/off} = 1/100, 1/10, 1$, or 10.

Each parameter configuration has been tested on 500 randomly generated scheduling problem instances. Each time, what has been measured is the sum of the gains associated with the user requests that have been selected, accomplished, and not removed from the current satellite order book: G_{off} for the off-line management and G_{on} for the on-line one. Each bar in the graphs of Figure 1 shows the mean gain (or loss) $(G_{on} - G_{off})/G_{off}.100$ that can be obtained by (or results from) managing the satellite on-line and eventually on-board.

We see on these graphs that:

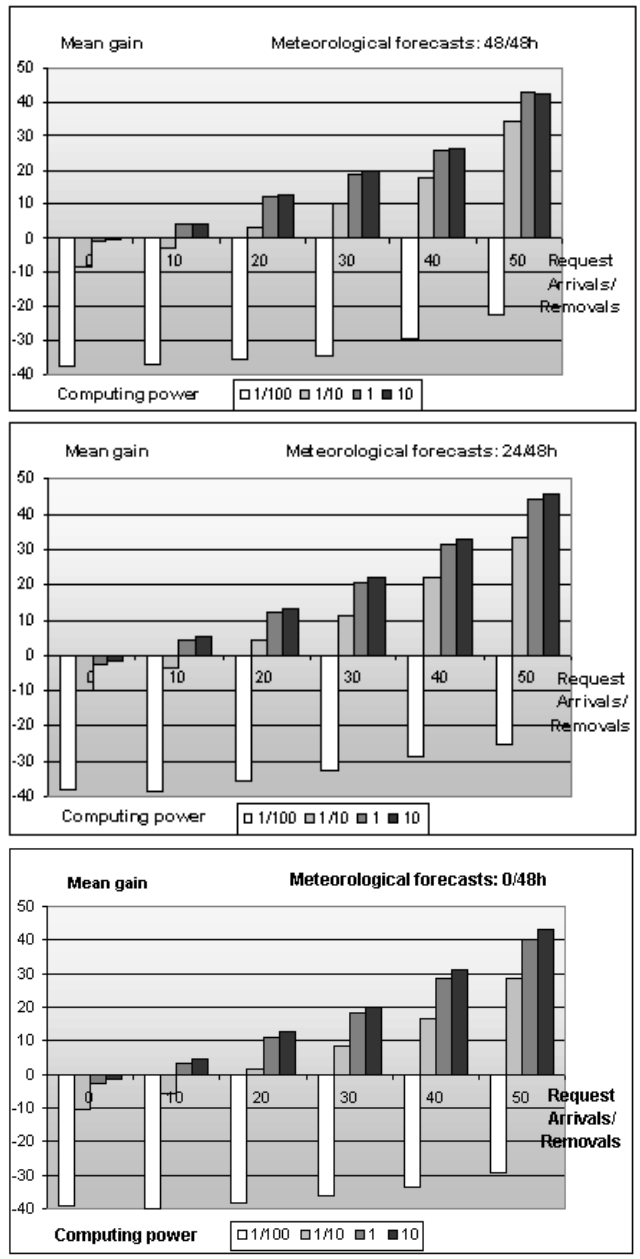


Figure 1: Mean gain (or loss) obtained by (or resulting from) managing the satellite on-line and eventually on-board: $(G_{on} - G_{off})/G_{off}.100$.

- to have at one's disposal more reliable meteorological information has no decisive influence on the results: the three graphs have globally the same shape; but the meteorological forecasting model we used may be questionable;
- the more dynamic the satellite order book, the more interesting an on-line management;
- a minimum computing power on-board is necessary to manage the satellite correctly: with a ratio $cpr_{on/off}$ equal to $1/100$ between the computing power on-board and on the ground, the gain is always negative.

Conclusion and future directions of work

The first lesson of this work is that an on-line and on-board Earth observation satellite management system is feasible and can outperform an off-line and on the ground one, in the context of a dynamic user demand (numerous requests known little time in advance), even if the computing power available on-board is lower than the one available on the ground. That means merely that, in this context as in other contexts, a limited search performed on updated data can outperform a systematic search performed on not updated ones.

The second lesson is that, at least in the context of Earth observation, building a schedule on a variable horizon, by using either *Dynamic Programming* or *Branch and Bound* methods, may be a good answer to the question of the balance between deliberation and reactivity: no frontier between them, more reactivity when it is necessary to decide quickly, more deliberation when time is available for that.

Future directions of work may concern:

- the extension of this work to more realistic models of satellite, user demand, environment, and forecasting;
- the design of more sophisticated on-line algorithms, that can, for example, reuse previous solutions or reasoning, when events slightly modify the problem data;
- the implementation on-board and the integration with the other components of the satellite management (executive, attitude and orbit control system, power management, memory management, communication management, fault diagnosis, identification and recovery ...), following the work performed for the *Proba* satellite (BDS98);
- the use of *reinforcement learning* techniques (SB98) to learn for example good approximations of the optimum expected gain that can be obtained after a given image acquisition (see section).

Acknowledgements

We would like to thank Simon Bretin, Nelly Lecubin, and Olivier Samain, from *SupAéro* (French Engineering School in Aerospace, Toulouse), who implemented the simulation framework that allows us to obtain these experimental results.

References

M. Boddy and T. Dean. Deliberation Scheduling for Problem Solving in Time-Constrained Environments. *Artificial Intelligence*, 67(2):245–285, 1994.

E. Bornschlegl, P. David, and H. Schindler. On-Board Software and Operation for Proba Small Satellite Autonomy. In *Proc. of the International Conference on Data Systems in Aerospace (DASIA-98)*, Athens, Greece, 1998.

E. Bensana, M. Lemaître, and G. Verfaillie. Earth Observation Satellite Management. *Constraints : An International Journal*, 4(3):293–299, 1999.

R. Larson and J. Casti. *Principles of Dynamic Programming*. Marcel Dekker Inc., New York and Basel, 1978.

R. Sutton and A. Barto. *Reinforcement Learning*. MIT Press, Cambridge, Massachusetts, 1998.

T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems : Hard and Easy Problems. In *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 631–637, Montréal, Canada, 1995.

Commentary: Designing and Evaluating an On-line On-board Autonomous Earth Observation Satellite Scheduling System

2nd International NASA Workshop on Planning and Scheduling for Space
16 – 18 March 2000
Peter R. Glück, Jet Propulsion Laboratory, California Institute of Technology
Pasadena, California, USA

Abstract

The paper reviewed [1] presents a simple but useful demonstration of an engineering trade study between an off-line (ground) and on-line (on-board) autonomous scheduling system. Of particular interest are the algorithm used and the results pertaining to the need, if any, for examining alternate observation permutations as well as the required computing power.

Review

The abstract mentions development of a "feasible optimal sequence." It is important to realize that the sequence built is not optimal in most cases. The goal is to optimize the observations, but this is rarely accomplished because of the uncertainties in actually executing the observations. Worst case (2 or 3 sigma) estimates of execution times must be used to guarantee that the specified data is collected. For example, the nominal time to turn or slew the spacecraft to the observing attitude may be only a few seconds, but in order to guarantee that the observation will occur at the proper time, this value may be padded to a minute or more. This results in substantial inefficiencies.

The author claims that an Earth-observing satellite may have an orbit of "about one hour." In fact, astrodynamics requires that any sustainable orbit be at least 1.5 hours in duration. A typical Earth-observing satellite (e.g., Landsat, TOPEX) at 828 km altitude has an orbital period of 1.7 hours (101.5 minutes). There is also an implication that current satellites do not maneuver in all three axes (roll, pitch and yaw). Actually, many current spacecraft can easily maneuver around all three principle axes.

An important factor in planning which is mentioned is the time required to repoint the instrument for the next observation. This is very often a source of substantial inefficiencies in spacecraft operations, especially when it is necessary to reorient the entire spacecraft. In order to guarantee the proper orientation, a worst-case estimate of the time required is usually used. In a nominal case, then, much time is wasted waiting for the appropriate imaging time to arrive. A system that could accurately estimate on-board what the current repositioning rate is would be able to substantially reduce this inefficiency.

The author notes that "the only benefit of an increase in the optimal sequence is a possible better choice of the next image to acquire." This is a keen insight and key attribute of the algorithm described. The algorithm seeks to maximize the sum benefit from each set of observations considered. Thus, for k = 1, it simply picks the single observation with the highest benefit function value. For k = 2, the observations with the two highest benefit functions are selected since these, naturally, give the largest sum. This reasoning continues for any arbitrary k. Increasing k, in the absence of any new information, adds observations of declining benefit. Since it is not possible to find an unselected observation that is of greater benefit than any of those already selected, there is no overall benefit to changing the order of observations (again, assuming that no new information has become available). Hence, the only time to re-evaluate the next image to acquire is when new information arrives.

The experiment performed demonstrated that the benefits of on-line scheduling are realized when the observation requests are most dynamic. This makes sense intuitively. It also makes sense that the trade of off-line vs. on-line is ineffective if the computing power of the ground system is much greater than that of the on-board system. The interesting result is that the on-board system must not be more than an order of magnitude slower than the ground system for there to be a positive benefit to performing the work on-board.

Acknowledgment

The work described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

[1] Verfaille, G. and Bornschlegl, E. , 2000, "Designing and Evaluating an On-line On-board Autonomous Earth Observation Satellite Scheduling System," *Proceedings, 2nd International NASA Workshop on Planning and Scheduling for Space*, San Fransisco, CA.

First published by the authors under the same title in AIPS 2000, p. 177-186
Copyright (c) 2000, American Association for Artificial Intelligence. Reprinted with permission from AAAI

Planning in Interplanetary Space: Theory and Practice*

Ari K. Jónsson **Paul H. Morris**
Nicola Muscettola **Kanna Rajan**
NASA Ames Research Center, MS 269-2
Moffett Field, CA 94035-1000,
{jonsson,pmorris,mus,kanna}@ptolemy.arc.nasa.gov

Ben Smith
Jet Propulsion Laboratory
Pasadena, CA 91109-8099
smith@aig.jpl.nasa.gov

Abstract

On May 17th 1999, NASA activated for the first time an AI-based planner/scheduler running on the flight processor of a spacecraft. This was part of the Remote Agent Experiment (RAX), a demonstration of closed-loop planning and execution, and model-based state inference and failure recovery. This paper describes the RAX Planner/Scheduler (RAX-PS), both in terms of the underlying planning framework and in terms of the fielded planner.

Introduction

During the week of May 17th 1999, the Remote Agent became the first autonomous closed-loop software to control a spacecraft during a mission. This was done as part of a unique technology validation experiment, during which the Remote Agent took control of NASA's New Millennium Deep Space One spacecraft (Muscettola *et al* 1998; Bernard *et al* 1999). The experiment successfully demonstrated the applicability of closed-loop planning and execution, and the use of model-based state inference and failure recovery.

As one of the components of the autonomous control system, the on-board Remote Agent Experiment Planner/Scheduler (RAX-PS) drove the high-level goal-oriented commanding of the spacecraft. This involved generating plans that could safely be executed on board the spacecraft to achieve the specified high-level goals. Such plans had to account for on-board activities having different durations, requiring resources, and giving rise to subgoal activities, all while satisfying complex flight safety rules about activity interactions.

In this paper, we describe the Remote Agent Experiment Planner/Scheduler from both the theoretical and the practical perspectives. The architecture of the planning system is as shown in Figure 1. The *domain model* describes the dynamics of the system to which the planner is being applied – in this case, the Deep Space One spacecraft. A *plan re uest*, consisting of an initial state and a set of goals, initializes the *plan database*. The *search engine* then modifies the plan database to generate a complete valid *plan*, which is then sent to the execution agent. The *heuristics* and *planning e perts* are not part of the core framework, but they are an integral part of the planning system that flew on board Deep Space One. The heuristics provide guidance to the search engine while the planning experts provide a uniform interface to external systems, such as attitude control systems, whose inputs the planner has to take into account.

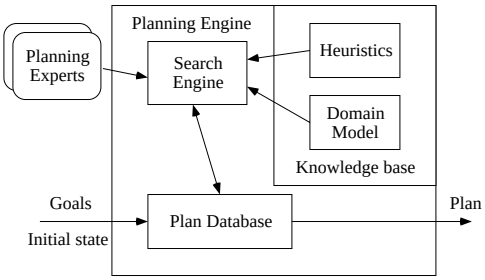


Figure 1: The Planner/Scheduler architecture

Aadditional information about the theoretical and practical aspects of RAX-PS can be found in (Jónsson *et al* 2000).

Theory

The RAX-PS system is based on a well-defined framework for planning and scheduling that, in many ways, differs significantly from classical STRIPS planning. For instance:

- Actions can occur concurrently and can have different durations.
- Goals can include time and maintenance conditions.

In this section, we will describe the PS framework from a theoretical perspective.

Tokens, Timelines and State Variables

To reason about concurrency and temporal extent, action instances and states are described in terms of temporal intervals that are linked by constraints. This approach has been called constraint-based interval planning (Smith, Frank, & Jónsson 2000), and has been used by various planners, including IxTeT (Ghallab & Laruelle 1994). However, although our approach builds on constraint-based interval planning, there are significant differences. Among those are:

- The use of timelines to model and reason about concurrent activities
- No distinction between actions and fluents
- Greater expressiveness of domain constraints

*Authors in alphabetical order.

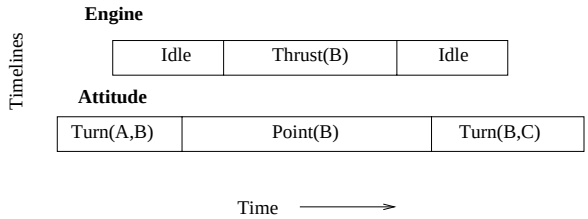


Figure 2: Plans as Parallel Timelines.

Humans find it natural to view the world in terms of interacting objects and their attributes. In planning, we are concerned with attributes whose states change over time. Such attributes are called *state ariables*. The history of states for a state variable over a period of time is called a *timeline*. Figure 2 shows `Engine` and `Attitude` state variables, and portions of the associated timelines for a spacecraft application (the attitude of a spacecraft is its orientation in space).

In classical planning (Fikes & Nilsson 1971; McAllester & Rosenblit 1991), and earlier interval planning, there is a dichotomy between fluents and actions. The former specify states, and the latter specify transitions between them. In terms of interval planning, this has resulted in intervals describing only actions, and fluent values being implicit. However, this distinction is not always clear, or even useful. For example, in a spacecraft domain, thrusting in a direction `P` can either be regarded as a state that implies pointing towards `P` or an action with pointing towards `P` as a precondition. Moreover, during execution, the persistence of fluent values over temporal intervals may be actively enforced by maintaining and verifying the value. For these and other reasons, we make no distinction between fluents and actions in this planning approach, and use the same construct to describe both fluents and actions.

From the point of view of execution, a state variable represents a single thread in the execution of a concurrent system. At any given time, each thread can be executing a single *procedure* P. A procedure P has n_P parameters ($n_P \geq 0$), each with a specified type. Each state variable is also typed, i.e., there is a mapping $Procs : \mathcal{S} \rightarrow 2^{\Pi}$, where $\mathcal{S}$ is the set of state variables and Π is the set of all possible procedures. Given a state variable σ, $Procs(\sigma)$ specifies the procedures that can possibly be executed on σ.

Thus, a timeline consists of a sequence of intervals, each of which involves a single procedure. We may think of the interval and its procedure as a structural unit, called a *token*, that has been placed on the timeline. Although each token resides on a definite timeline in the final plan, the appropriate timeline for a token may be undetermined for a while during planning. We refer to a token that is not on a timeline as a *oating* token.

A token describes a procedure invocation, the state variables on which it can occur, the parameter values of the procedure, and the time values defining the interval. To allow the specification of multiple values, e.g, to express a range of possible start times, variables are used to specify parameter, start and end time values for a token. As a result, a token T is a tuple $\langle v, P(\vec{x}_P), s, e\rangle$, where v is a variable denoting a state variable, P is the name of a procedure (satisfying $P \in Procs(v)$), the elements of $\vec{x}_P$ are variables that denote the parameters of the procedure (restricted to their types), and s and e are numeric variables indicating the start and end times respectively (satisfying $s \leq e$).

Each of the token variables, including the parameter variables, has a domain of values assigned to it. The variables may also participate in constraints that specify which value combinations are valid.

Domain Constraints

In a complex system, procedures cannot be invoked arbitrarily. A procedure call might work only after another procedure has completed, or it might need to be executed in parallel with a procedure on a different thread.

To specify such constraints, each ground token, $T = \langle v, P(\vec{x}_P), s, e\rangle$, has a configuration constraint $G_T(v, \vec{x}_P, s, e)$, which we call a *compatibility*. It determines the necessary correlation with other procedure invocations in a legal plan, i.e., which procedures must precede, follow, be co-temporal, etc. Since a given procedure invocation may be supported by different configurations, a compatibility is a disjunction of constraints. Therefore, we define $G_T(v, \vec{x}_P, s, e)$ in terms of pairwise constraints between tokens, organized into a disjunctive normal form:

$$G_T(v, \vec{x}_P, s, e) = \Gamma_1^T \vee \cdots \vee \Gamma_n^T$$

Each Γ_i^T is a conjunction of *subgoals* $\wedge_j \Gamma_{i,j}^T$ with the following form:

$$\Gamma_{i,j}^T = \exists T_j \gamma_{i,j}^T(v, \vec{x}_P, s, e, v_j, \vec{z}_{P_j}, s_j, e_j)$$

where T_j is a token $\langle v_j, P_j(\vec{z}_{P_j}), s_j, e_j\rangle$ and $\gamma_{i,j}^T$ is a constraint on the values of the variables of the two tokens involved.

In general $\gamma_{i,j}^T$ may take any form that appropriately specifies the relation between the two tokens. In practice, $\gamma_{i,j}^T$ is structured to limit its expressiveness and make planning and constraint propagation computationally efficient. In the RAX-PS framework, $\gamma_{i,j}^T$ is limited to conjunctions of:

- Equality (codesignation) constraints between parameter variables of different tokens.
- Simple temporal constraints on the start and end variables. These are specified in terms of metric versions of Allen's temporal algebra relations (Allen 1984); `before`, `after`, `meets`, `met-by`, etc. Each relation gives rise to a bound on the distance between two temporal variables. This bound can be expressed as a function of the start and end variables of T and T_j.
- Constraints on how the token T can be instantiated. These are represented as procedural constraints, which are an effective way to specify and enforce arbitrary constraints.

Subgoal constraints must guarantee that each state variable is always either executing a procedure or instantaneously switching between procedure invocations. This means that each Γ_i^T contains a *predecessor*, i.e., a requirement for a T_j on the same state variable as T, such that T `met_by` T_j. Similarly, each Γ_i^T must specify a *successor*.

The concept of subgoals generalizes the notion of preconditions and effects in classical planning. For example, ADD effects can be enforced by using `meets` subgoals while deleted preconditions correspond to `met_by` subgoals. Preconditions that are not affected by the action can be represented by `contained_by` subgoals.

Plan Database

Having laid out the representation of the planning domain, we can now turn our attention to what the planner represents and reasons about. In RAX-PS, this is a data structure called the *plan database*. At the most basic level, the plan database represents 1) a current *candidate plan*, which is essentially a set of timelines containing interrelated tokens, and 2) a current set of decisions that need to be made.

Formally, a candidate plan consists of the following:

- a horizon (h_s, h_e), which is a pair of temporal values satisfying $-\infty \leq h_s < h_e \leq \infty$
- a timeline $\mathcal{T}_\sigma = (T_{\sigma_1}, \ldots, T_{\sigma_k})$, for each state variable, with tokens $T_i = \langle v, P_{\sigma_i}(\vec{x}), s, e \rangle$, such that each $P_{\sigma_i} \in Procs(\sigma)$
- ordering constraints $\{O_1, \ldots, O_K\}$, enforcing $h_s \leq e(T_{\sigma_1}) \leq s(T_{\sigma_2)} \leq \cdots \leq e(T_{\sigma_{k-1}}) \leq s(T_{\sigma_k}) \leq h_e$ for each timeline $\mathcal{T}_\sigma$
- a set of constraints $\{C_1, C_2, \ldots, C_N\}$, each relating sets of variables from one or more tokens; includes temporal, equality and local procedural constraints

The constraints in a candidate plan give rise to a constraint network, consisting of the variables in the tokens and the constraints that link token variables in different ways. This network determines the set of all legal instantiations of the given tokens. As a result, any candidate plan that has an inconsistent underlying constraint network cannot be part of a valid plan. Limited plan consistency checking can therefore be done by *constraint propagation* (Mackworth & Freuder 1985), which is a method for eliminating values that can be proven not to appear in any solution to the constraint network.

In addition to a candidate plan, the plan database may also contain a set of decisions that need to be made. A decision corresponds to a *a* in a candidate plan, an aspect of the candidate that may prevent it from being a complete and valid plan. In this framework, there are four types of flaws: *uninstantiated ariables*, *oating tokens*, *open dis unctions of compatibilities*, and *unsatisfied compatibility subgoals*. Each flaw in the plan database gives rise to choices for how that flaw can be resolved. Resolving a flaw is a reasoning step that maps the given database to another database. Categorized by the types of flaws, the following is a list of the possible choices for resolving a flaw:

- Variable restriction flaws are resolved by selecting a non-empty subset of the variable domain and restrict the variable to that domain.
- Floating token flaws are resolved by selecting two adjacent tokens on a timeline and inserting the floating token between them.
- Open disjunction flaws are resolved by selecting one item in the disjunction and making it true.
- Unsatisfied subgoal flaws are resolved by either finding an existing token and using that to satisfy the subgoal, or by adding a new token to satisfy the subgoal.

It is important to note that it is not necessary to resolve all flaws in order to have a plan. In most cases, however, we require that each token satisfy the applicable compatibility specification, i.e, that the subgoals from at least one of the disjunctions are satisfied. In that case, we say that the token is *fully supported*.

Executable Plans

Based on the notions we have introduced here, we can now turn our attention to the semantics of a candidate plan, and the task of developing a formal definition of what a valid plan is. Traditionally, valid plans have been defined in abstract terms, based only on the candidate plan and the domain model. However, this approach is not realistic, as the validity of a plan in the real world is inherently tied to the mechanism that executes it. To address this, we start by discussing the basics of plan execution and then go on to derive a realistic definition of what constitutes a valid plan.

From the point of view of the executing agent (called the executive or EXEC) a plan is a concurrent program that is to be interpreted and executed in a dynamic system. Recall that the plan contains variables that specify how and under which circumstances procedures are to be instantiated. For variables that correspond to system values, such as the current time, the EXEC will sense actual system values, compare them with the values specified in the plan, and then determine which procedure should be executed next. If the EXEC fails to match sensed values with the values in the plan, the EXEC triggers a fault-protection response (e.g., put the system in a safe state and start taking recovery actions). The question of whether the EXEC succeeds in matching values and selecting a procedure invocation depends in part on how much reasoning the EXEC can perform for this purpose. That, in turn, depends both on how much reasoning the EXEC is capable of and how much time it has before the next invocation must be activated.

Consider a candidate plan; tokens may not be fully supported, and variables may be uninstantiated. In order to instantiate the candidate, each flaw must be resolved successfully. For an execution agent with suf-

ficient time and reasoning capabilities, such an underspecified plan might be a viable plan. In fact, the lack of commitment would allow the execution agent to choose the flaw resolutions that best fit the actual conditions during execution. The Remote Agent system took advantage of this by letting the EXEC map high-level tasks into low-level procedures, during execution. This freed the planner from generating low-level procedure calls, and allowed the executive to choose the low-level procedures that best fit the actual execution.

In general, executability depends on the execution agent in question. It depends primarily on two aspects; how flexible the candidate plan must be to cover possible system variations, and how restricted the candidate plan must be for the executive to identify whether it is executable. The latter is an important issue to consider, as making this determination can be as expensive as solving a planning problem.

To represent the abilities of a particular executive agent, we use a *plan identification function* f_I that identifies executable candidate plans, by mapping each possible candidate plan to one of the values of $\{T, F, ?\}$. The intent is that if a candidate $\mathcal{P}$ can be recognized as being executable, then $f_I(\mathcal{P}) = T$; if a candidate is recognized as not being executable, then $f_I(\mathcal{P}) = F$; and if executability cannot be determined, then $f_I(\mathcal{P}) =?$.

We permit a great deal of variation in how different executives respond to different candidate plans, but we do require that a plan identification function behaves consistently with respect to the two aspects mentioned above. For example, the function should not reject one candidate on the basis of being too restrictive and then accept a restriction of that candidate. This leads us to the following formalization of what constitutes a plan identification function:

Definition 1 *A* plan identification function f_I *for a gi en e ecution agent is a function that maps the set of candidate plans to the e tended truth alue set* $\{T, F, ?\}$*, such that for any candidate plan* $\mathcal{P}$ *and any candidate plan* $\mathcal{Q}$ *that e tends the candidate* $\mathcal{P}$*, e ha e*

- *if* $f_I(\mathcal{P}) = F$ *then* $f_I(\mathcal{Q}) = F$
- *if* $f_I(\mathcal{P}) = T$*, then* $f_I(\mathcal{Q}) \in \{T, F\}$
- *if a token in* $\mathcal{P}$ *is not supported, then* $f_I(\mathcal{P}) =?$

The last condition is not strictly necessary, as some executives are capable of solving planning problems, but in the interest of clarity, we will limit the execution agents to solving constraint satisfaction problems.

Using this notion of plan identification functions, we can now provide a realistic, formal definition of what constitutes a plan, namely:

Definition 2 *For a gi en e ecuti e, represented by a plan identification function* f_I*, a candidate plan* $\mathcal{P}$ *is a* plan *if and only if* $f_I(\mathcal{P}) = T$

Planning process

We can now turn our attention to the plan generation process itself. The input to the planning process is an initial candidate plan, which includes an initialization token for each timeline, a set of floating tokens, and a set of constraints on the tokens in question. Together, these elements give rise to an initial plan database. The goal of the planning process is then to extend the given initial candidate to a complete valid plan. From the point of view of traditional planning, the initial plan database specifies both the initial state and the goals. In fact, our approach permits a much more expressive specification of goals. For example, we can request a spacecraft to take a specified sequences of pictures in parallel with providing a certain level of thrust.

```
plan (P,D)  {
   if f(P) = T
     return P
   else if f(P) = F
     return fail
   else
     given a flaw d from the flaw database D,
     choose a resolution res(d) for the flaw
     let (P',D') = apply res(d) to (P,D)
     return plan(P',D')
}
```

Figure 3: The planning process. The plan database consists of the candidate plan `P` and the set of flaws `D`.

The planning process we define is a framework that can be instantiated with different methods for controlling the search, selecting flaws, propagating constraints, etc. The planning process is a recursive function that non-deterministically selects a resolution for a flaw in the current plan database. An outline of the process is shown in Figure 3.

This planning process is clearly sound, as any resulting plan satisfies the given plan identification function. The planning process is also complete in the sense that if there is a plan, then a plan can be found. Furthermore, if a given initial candidate plan can be extended to some valid plan $\mathcal{P}$ (satisfying f_I), then the planning process can find some other valid plan (satisfying f_I) that can be extended to $\mathcal{P}$. A still stronger completeness criterion, that any given plan can be found, does not hold in general. The reason is that a lenient identification function f_I may return T even though the planning process has not addressed all remaining flaws. This highlights the importance of identifying properties of soundness and completeness for new planning frameworks such as this one.

Theorem 1 *Suppose a domain model, a plan identification function* f_I*, and an initial plan* $\mathcal{P}_0$ *are gi en et* $\mathcal{P}_T$ *be a alid plan (i e ,* $f_I(\mathcal{P}_T) = T$*) that e tends* $\mathcal{P}_0$ *hen, the planning process can generate a alid plan* $\mathcal{P}'$ *that e tends* $\mathcal{P}_0$*, and can be e tended to* $\mathcal{P}_T$

Practice

RAX PS extends the theoretical framework into a well-engineered system. The system had to operate under stringent performance and resource requirements. For

Model size:	State variables	18
	Procedure types	42
Plan size:	Tokens	154
	Variables	288
	Constraints	232
Performance:	Search nodes	649
	Search efficiency	64%

Table 1: Plan size and performance of RAX PS

example, the Deep Space 1 flight processor was a 25 MHz radiation-hardened RAD 6000 PowerPC processor with 32 MB memory available for the LISP image of the full Remote Agent. This performance is at least an order of magnitude worse than that of current desktop computing technology. Moreover, only 45% peak use of the CPU was available for RAX, the rest being used for the real-time flight software. The following sections describe the engineering aspects of the RAX PS system. First we describe the planning engine, the workhorse on which all development was founded. Then we describe the mechanism for search control used to fine-tune the planner.

RAX PS planning engine

As follows from the previously discussed theory, producing a planner requires choosing a specific plan identification function f_I, a specific way to implement non-determinism and a flaw resolution strategy. In RAX PS we designed the planner in two steps. First we defined a basic planning engine, i.e., a general search procedure that would be theoretically complete. Then we designed a method to program the search engine and restrict the amount of search needed to find a solution. In this section we talk about the planning engine.

The first thing we need to clarify is what constitutes a desirable plan for the flight experiment. RAX plans are flexible only in the temporal dimension. More precisely, in a *temporally e ible plan*, all variables must be bound to a single value, except the temporal variables (i.e., token start and end times, s and e). It is easy to see that under these assumptions the only uninstantiated constraint sub-network in the plan is a *simple temporal net ork* (Dechter, Meiri, & Pearl 1991). This means that the planner can use arc consistency to determine whether the plan can be instantiated and that the executive can adjust the flexible plan to actual execution conditions by using very fast incremental propagation (Tsamardinos, Muscettola, & Morris 1998). All of this is translated into a plan identification function f_I defined as follows: *When applied to a candidate plan, f_I checks its arc consistency If the candidate is inconsistent, f_I returns F If the candidate is arc consistent, f_I returns one of t o alues T if the candidate is fully supported and all the non-temporal ariables are grounded, and ? in any other case*

To keep a balance between guaranteeing completeness and keeping the implementation as simple as possible, non-determinism was implemented as chronological backtracking. Also, the planner always returned the first plan found. Finally, the planning engine provided a default flaw selection strategy at any choice points of the backtrack search. This guaranteed that no underconstrained temporal variable flaw would ever be selected, while all other flaw selection and resolutions were made randomly.

```
(:subgoal
 (:master-match (Camera = Ready))
 (:slave-match  (Camera = Turning_On))
 (:method-priority ((:method :add)(:sort :asap))
                   ((:method :connect))
                   ((:method :defer)))
 (:priority 50))
```

Figure 4: Search control rules for unsatisfied subgoal

Search control

By itself, the basic planning engine could not generate the plans needed for the flight experiment. However, RAX PS included additional search control mechanisms that allowed very localized backtracking. This is reflected in the the performance figures in Table 1, where search efficiency is measured as the ratio between the minimum number of search nodes needed and the total number explored.

Achieving this kind of performance was not easy and required a significant engineering effort. We outline the principal aspects of this effort in the rest of the section.

Flaw agenda management RAX PS made use of a programmable search controller. Ideally, the "optimal" search controller is an oracle that can select the correct solution choice without backtracking. In practice this is not possible and the control strategy can only make flaw resolution decisions on the basis of the partial plan developed so far. The search controller of RAX PS allows programming an approximate oracle as a list of search control rules. This list provides a prioritization of the flaws in a database and sorting strategies for the non-deterministic choices for each flaw selection. Figure 4 gives an example of a search control rule.

The rule applies to an unsatisfied subgoal flaw of a $\langle \texttt{Camera}, \texttt{Ready}, s, e\rangle$ token that requires a $\langle \texttt{Camera}, \texttt{Turning_on}, s_k, e_k\rangle$ token. Note that in the DS1 model the `Camera` can reach a `Ready` state only immediately after the procedure `Turning_on` has been executed. Therefore, in this case, matching the token types in the subgoal is sufficient to uniquely identify it. When the priority value associated with the flaw is the minimum in the plan database, the planner will attempt to resolve the flaw by trying the resolution methods in order. In our case the planner will first try to `:add` a new token and try to insert it in the earliest possible timeline gap (using the standard sort method `:asap`). The last resolution method to try is to `:defer` the subgoal. When this happens, the plan database will automatically force start or end of the token to occur outside of the horizon h_s. In our case, the deferment method will only succeed if the `Ready` token is the first

token on the timeline.

Search control engineering The rule language for the search controller is designed to be extremely flexible. It permits the introduction of new sorting methods, if the standard methods prove to be ineffective. Also, it is possible to prune both on solution methods (e.g., only `:connect` to satisfy a subgoal) and on resolution alternatives (e.g., schedule a token as early as possible and fail if you cannot). Unfortunately, this meant that completeness could no longer be guaranteed. On the other hand it allowed for a finely tuned planner. Designing search control became a trade-off between scripting the planner's behavior and exploring the benefits of shallow backtracking when necessary. Here are some issues that needed to be addressed.

Interaction between model and heuristics: Ideally, it is desirable to keep domain model and search control methods completely separate. This is because constraints that describe the "physics" of the domain should only describe what is possible while search control should help in narrowing down what is desirable from what is possible. Moreover, declarative domain models are usually specified by domain experts (e.g., spacecraft systems engineers) not by problem solving experts (e.g., mission operators). Commingling structural domain information with problem solving methods can significantly complicate inspection and verification of the different modules of a planning system.

In our experience, however, such an ideal separation was difficult to achieve. Model specifications that were logically correct turned out to be very inefficient because they required the discovery of simple properties by extensive search (e.g., a token being the first of a sequence of tokens with the same procedure). The standard method used in RAX-PS was to define auxiliary token variables and use search control to enforce a specific value, which in turn would prune undesired alternatives through constraint propagation. Including the control information within the model caused a significant level of fragility in domain modeling, especially in the initial stages of the project when we still had a weak grasp on how to control the search.

High-level control languages: The control rules described above can be thought of as an "assembly language" for search control; and the DS1 experience confirmed that programming in a low-level language is painful and error prone. However, this assembly language provides us with a strong foundation on which to build higher level control languages which are well founded and better capture the control knowledge of mission operators. The declarative semantics of the domain model also opens up the possibility of automatically understanding dependencies that point to effective search control. The synthesized strategies can then be compiled into the low-level control rules. Work is currently in progress to explore methods to alleviate the burden of control search programming.

Conclusion

In this paper, we have presented an overview of the Remote Agent Experiment Planning/Scheduling system, both from theoretical and practical points of view. Research and development of autonomous planning systems, capable of solving real problems, continues among the many scientists in the field. The work we have presented here is just another step in this development, but it is a step that has taken autonomous planning to interplanetary space.

References

Allen, J. 1984. Towards a general theory of action and time. *Artificial Intelligence* 23(2):123–154.

Bernard, D.; Dorais, G.; Gamble, E.; Kanefsky, B.; Kurien, J.; Man, G. K.; Millar, W.; Muscettola, N.; Nayak, P.; Rajan, K.; Rouquette, N.; Smith, B.; Taylor, W.; and Tung, Y.-W. 1999. Spacecraft autonomy flight experience: The DS1 Remote Agent experiment. In *Proceedings of the AIAA onference 1999*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Ghallab, M., and Laruelle, H. 1994. Representation and control in IxTeT, a temporal planner. In *Proceedings of the Second International onference on Artificial Intelligence Planning Systems*.

Jónsson, A. K.; Morris, P. H.; Muscettola, N.; Rajan, K.; and Smith, B. 2000. Planning in interplanetary space: Theory and practice. In *Proceedings of the Fifth International onference on Artificial Intelligence Planning and Scheduling*.

Mackworth, A. K., and Freuder, E. C. 1985. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence* 25:65–74.

McAllester, D., and Rosenblit, D. 1991. Systematic nonlinear planning. In *Proceedings of the Ninth National onference on Artificial Intelligence*, 634–639.

Muscettola, N.; Nayak, P. P.; Pell, B.; and William, B. 1998. Remote Agent: To boldly go where no ai system has gone before. *Artificial Intelligence* 103(1-2):5–48.

Muscettola, N. 1994. HSTS: Integrated planning and scheduling. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling*. Morgan Kaufman. 169–212.

Smith, D. E.; Frank, J.; and Jónsson, A. K. 2000. Bridging the gap between planning and scheduling. *no ledge ngineering Re ie* 15(1).

Tsamardinos, I.; Muscettola, N.; and Morris, P. 1998. Fast transformation of temporal plans for efficient execution. In *Proceedings of the 1 th National onference on Artificial Intelligence (AAAI-9)*, 254–261.

COMMENT

Planning in Interplanetary Space: Theory and Practice

Troy P. LeBlanc
NASA Johnson Space Center
Houston, TX 77058
troy.p.leblanc1@jsc.nasa.gov

Introduction

Autonomous closed-loop planning/scheduling engines will definitely replace the role of the human mission planner for manned interplanetary space missions. It is likely that the roles of mission director and mission managers will, as well, be eliminated. This commentary is divided into responses to the theoretical development and practical application of the Remote Agent Experiment Planner/Scheduler (RAX-PS) developed for NASA's New Millennium Deep Space One spacecraft.

Comment on Theory

RAX-PS theoretical descriptions range from integrated planning and scheduling to planning databases to a planning oracle. These topics are well laid out to establish the necessary framework for a planner/scheduler application to reside on board the restrictive computer architecture of an unmanned spacecraft.

Planning versus Scheduling

There is no defined difference in RAX-PS between planning and scheduling functions. The belief that intermixing these methodologies is unavoidable with complex synchronization constraints seems to be a future growth limitation. The ability to define a phased mission profile through planning against an abstract resource allocation should allow greater mission scheduling autonomy during execution. Put simply, mission goals will for the foreseeable future of space flight be decided well in advance, and the production of an abstract planned mission profile to ensure achievable schedules should be considered an advantage.

Planning Databases

Reading about RAX-PS's planning databases provides some very exciting insight into the possibilities of autonomous scheduling. In the commenter's opinion, the use of reasoning steps to map given databases for conflict detection is very important to plan validation. Also, the inference that it is not necessary to resolve all flaws injects reality into the model.

In scheduling, the counterpart to conflict detection is conflict resolution, which requires evaluation or grading functions to be resident and available in a more persistent scheduling database. Of course, grading implies reaching the most optimized plan as an end goal; however, this should be driven by pre-execution definition of flight rules and a planned mission profile.

Oracles and Flight Rules

The application of flight rules to an oracle may, for the purposes of developing and operating an autonomous closed-loop planner/scheduler, be improper. The oracle's definitions for operational preferences within the RAX-PS search controller are equivalent to planning/scheduling groundrules for manned space flight. The oracle's operational preferences (i.e. groundrules), which should be driven by planned mission phases, should drive grading functions. Theoretically, the oracle should have all data concerning the mission profile, the planning/scheduling groundrules, and the grading functions to validly schedule a timeline.

Comment on Practice

The use of RAX-PS on the Deep Space One spacecraft definitely proves the viability of the software for unmanned space missions. The extension of a variant of RAX-PS to manned deep space missions brings into question knowledge capture, data archival, and extensibility.

Knowledge Capture

Here again, reading about capturing the domain expertise of mission operators to drive search control through development of a high-level language seems filled with possibilities. It is stated that it is plausible to define the semantics to "automatically understand dependencies that point to effective search controls", so it should also be plausible for a high-level control language to automatically update dependencies, such as consumables, in a planning database which change over a mission profile.

Data Archival during Flight

Although storage memory for ground-bound computer systems is no longer a problem, it seems RAX-PS did not have this luxury. Even onboard the International Space Station (ISS), the mission planners will be limited to 1000

automatically initiated activities in the plan for a typical day. However, for longer duration, deep space missions that may encounter limited communications for data downlink, autonomous closed-loop planners like RAX-PS will need to archive plans for executed mission phases. This data could also be utilized in grading the best new plans based on previous results.

Extensibility to Manned Missions

The use of an autonomous planning/scheduling capability onboard ISS has been discussed amongst space station engineers many times. The idea of allowing astronauts in low earth orbit to schedule their own day's work is not ideal to mission directors and mission managers concerned with less than perfect assembly operations. However, mission planners believe operations will, in the future, be more relaxed. This opens the possibility of having a software system define the crew's workday without ground control intervention.

For deep space manned vehicles that would also require assembly, RAX-PS has already shown a fundamental solution to incrementally develop a timeline against a model. However, the requirements to plan bringing new spacecraft systems on-line and then automatically schedule the system verification and autonomously include nominal operation of each additional system within daily timelines only hints at the complexity of the next generation RAX-PS.

Conclusion

The background of this commenter is manned low earth orbit mission planning and scheduling, and the RAX-PS, as it is designed, is eliminating that job. The framework of the software system definitely lends itself to future growth in manned deep space missions. The comment, though, is primarily concerned with the overall use of the planning database from oracles to operational preferences to data capture and storage. The commenter hopes that the issues raised in this paper will highlight pertinent questions to drive future development decisions of the first manned quality RAX-PS.

References

Jonsson, Ari K., Morris, Paul H., Muscettola, Nicola, Rajan, Kanna, Smith, Ben. 1999. Planning in Interplanetary Space: Theory and Practice

Questions?

Contact Troy LeBlanc (troy.p.leblanc1@jsc.nasa.gov)

First published by the authors under the same title in AIPS 2000, p. 254-263
Copyright (c) 2000, American Association for Artificial Intelligence. Reprinted with permission from AAAI

Challenges and Methods in Testing the Remote Agent Planner

Ben Smith
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
benjamin.smith@jpl.nasa.gov

Martin S. Feather
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
martin.s.feather@jpl.nasa.gov

Nicola Muscettola
NASA Ames Research Center
MS 269-2
Moffet Field, CA 94035
mus@ptolemy.arc.nasa.gov

Abstract

The Remote Agent Experiment (RAX) on the Deep Space 1 (DS1) mission was the first time that an artificially intelligent agent controlled a NASA spacecraft. One of the key components of the remote agent is an on-board planner. Since there was no opportunity for human intervention between plan generation and execution, extensive testing was required to ensure that the planner would not endanger the spacecraft by producing an incorrect plan, or by not producing any plan.

The testing process raised many challenging issues, several of which remain open. The planner and domain model are complex, with billions of possible inputs and outputs. How does one obtain adequate coverage with a reasonable number of test cases? How does one even measure coverage for a planner? How does one determine plan correctness?

As planning systems are fielded in mission-critical applications, it becomes increasingly important to address these issues. We describe the major issues encountered while testing the Remote Agent planner, how we addressed them, and what issues remain open.

Introduction

As planning systems are fielded in operational environments, especially mission-critical ones such as spacecraft commanding, validation of those systems becomes increasingly important. Verification and validation of mission-critical systems is an area of much research and practice, but little of that is applicable to planning systems.

Our experience in validating the Remote Agent planner for operations on board DS1 raised a number of key issues, some of which we have addressed and many of which remain open. The purpose of this paper is to share those experiences and methods with the planning community at large, and to highlight important areas for future research.

At the highest level there are two ways that a planner can fail. It can fail to generate a plan (*converge*) within stated time bounds, or it can generate an incorrect plan. We used empirical testing to detect these kinds of failures. We ran the planner on several inputs and used an automated test oracle to determine whether they satisfied the requirements as expressed in first order predicate logic. A second (trivial) oracle checked for convergence.

The key issue in empirical testing is obtaining adequate coverage (confidence) within the available testing resources. This requires a combination of strong test selection methods that maximize the coverage for a given number of cases, and strong automation methods that reduce the per-test cost.

The RAX test selection strategy required 289 cases to adequately exercise a narrow space of inputs similar to those we expected to see in operation. This was sufficient for the RA experiment, but broader coverage—with correspondingly more test cases— would be needed for operational profiles outside of the experiment scope. We developed a number of test automation tools, but it still required six work-weeks to run and analyze 289 cases. This high per-test cost was largely due to human bottlenecks. Better test selection strategies and more powerful automation methods are needed to permit broader coverage for a reasonable cost. This paper identifies several open issues in these areas, and suggests ways to address them.

The rest of this paper is organized as follows. We first describe the RAX planner and domain model. We then discuss the test case selection strategy and the open test selection issues. We then discuss the test automations employed for RAX, the demands for human involvement that limited their effectiveness, and suggest automations and process improvements that could mitigate these factors. We conclude with an evaluation of the overall effectiveness of the Remote Agent planner testing, and summarize the most important open issues for planner testing in general.

RAX Planner

The Remote Agent planner (Muscettola *et al.* 1997) is one of four components of the Remote Agent (Nayak *et al.* 1999). The other components are the Executive (EXEC), Mission Manager (MM), and Mode Identification and Reconfiguration (MIR).

When the Remote Agent is given a "start" command the EXEC puts the spacecraft in a special idle state, in which it can remain indefinitely without harming the

spacecraft, and requests a plan. The request consists of the desired plan start time and the current state of the spacecraft. The desired start time is the current time plus the amount of time allocated for generating a plan (as determined by a parameter, and typically between one and four hours).

The Mission Manager extracts goals from the *mission profile*, which contains all the goals for the experiment and spans several plan horizons. A special *waypoint* goal marks the end of each horizon. The MM extracts goals between the required start time and the next waypoint goal in the profile. These are combined with the initial state. The MM invokes the planner with this combined initial state and the requested plan start time.

The planner expands the initial state into a conflict-free plan using a heuristic chronological backtracking search. During the search the planner obtains additional inputs from two on-board software modules, the navigator (NAV) and the attitude control subsystem (ACS). These are also referred to as "plan experts." When the planner decides to decompose certain navigation goal into subgoals, it invokes a NAV function that returns the subgoals as a function of the goal parameters. The planner queries ACS for the duration and legality of turn activities as a function of the turn start time and end-points.

The fundamental plan unit is a *token*. These can represent goals, activities, spacecraft states, and resources. Each token has a start and end timepoint and zero or more arguments. The tokens exist on parallel *timelines*, which describe the temporal evolution of some state or resource, or the activities and goals related to a particular state. Some RAX timelines are attitude, camera mode, and power. The domain model defines the token types and the temporal and parameter constraints that must hold among them.

If the planner generates a plan the EXEC executes it. Under nominal conditions the plan is executed successfully and the EXEC requests a new plan. This plan starts at the end of the current plan, which is also the start of the next waypoint in the profile. If a fault occurs during execution, and the EXEC cannot recover from it, it terminates the plan and achieves an idle state. This removes the immediate threat of the fault. Depending on the failure, it may only be able to achieve a degraded idle state (e.g., the camera switch is stuck in the off position). It then requests a new plan that achieves the remaining goals from the achieved idle state. As with other requests, the required start time is the current time plus the time allowed for planning.

Domain Model. The domain model encodes the knowledge for commanding a subset of the DS1 mission known as "active cruise" that consists of firing the ion propulsion (IPS) engine continuously for long periods, punctuated every few days by optical navigation (opnav) images and communication activities.

The goals defined by the domain model are shown in Table 1. The initial state consists of an initial token for each of the timelines in the model. The legal start tokens for most timelines are fixed. Table 2 shows the non-fixed timelines and the set of legal start tokens for each one. Finally, the domain model defines 11 executable activities for commanding the IPS engine and MICAS camera, slewing (turning) the spacecraft, and injecting simulated faults. The latter allow RAX to demonstrate fault recovery capabilities, since actual faults were unlikely to occur during the experiment.

Goal Type	Arguments
waypoint	HZN,EXPT_START,EXPT_END
navigate	frequency (int), duration (int). slack (int)
Comm	none
power_estimate	amount (0-2500)
exec_activity	type, file, int, int, bool
sep_segment	vector (int), level (0-15)
max_thrust	duration (0-inf)
image_goal	target (int), exposures (0-20), exp. duration (0-15)

Table 1: Goals

Test Selection Strategy

The key test selection issue is achieving adequate coverage with a manageable number of cases. One selection strategy is to analyze the domain model to identify input values that would fully exercise the model according to some coverage metric. Although the validation and verification literature is full of coverage metrics for conventional systems, to our knowledge no such metrics exist for planner domain models.

Having neither a metric nor the time to devise one, we instead used a black-box selection approach that has been successful in several conventional systems. The idea is to characterize the inputs as an n-dimensional parameter space and use orthogonal arrays to select a manageable number of cases that exercises all pair-wise combinations of parameter values. These tests can be augmented as needed with selected higher-order combinations. Large input spaces can be tested tractably since the number of pair-wise cases grows only logarithmically in the number of parameters—proportional to $(v/2)\log_2 k$ for k parameters with v values each (Cohen *et al.* 1997).

One disadvantage of this all-pairs selection strategy is that each test case differs from the others and from the nominal baseline input in several parameter values. That often made it difficult to determine why a test case

state timeline	initial values
EXEC_ACTIVITY	0,1,2
ATTITUDE	Earth, image, thrust vector
MICAS_SWITCH	ready, off
MICAS_HEALTHY	true, false

Table 2: Variable Initial State Timelines

failed, especially when the planner failed to converge.

To address this problem we created a second test set in which each case differed in only one parameter value from the nominal baseline, which was known to produce a valid plan. This "all-values" test set exercised each parameter value at least once. If one of these cases failed, it was obviously due to the single changed parameter, and its similarity to the baseline case made it easier to identify the causal defect. The reduction in analysis cost comes at the expense of additional test cases. The all-values test set grows linearly in the number of parameter values: $1 + n(v - 1)$ for n parameters with v values each.

RAX Test Selection

We now discuss how the all-pairs and all-values test selection strategies were employed for RAX. The planner has the following inputs: a set of goals, which are specified in a mission profile and by the on-board navigator; an initial state; a plan start time; slew durations as provided by the ACS plan expert; and two planner parameters—a seed for the pseudo-random number generator that selects among non-deterministic choices in the search, and "exec latency" which controls the minimum duration of executable activities.

Each of these inputs is specified as a vector of one or more parameter values. The goals and initial states are specified by several parameters, and the other inputs are specified by a single parameter each. Several of the parameters, such as plan start time, have infinite or very large domains. It is clearly infeasible to test all of these values, so we selected a small subset that we expected to lie at key boundary points. This selection was *ad hoc* based on the intuition of a test engineer familiar with the domain model, or simply high, middle, and low values in the absence of any strong intuition. Table 3 shows the full list of parameters, the range of values each can take, and the values tested.

Parameters 6-11 specify the initial state, Parameters 14-18 specify the IPS thrusting and MICAS imaging goals requested by the onboard Navigator, and Parameter 20 specifies the duration of spacecraft slew (turns) activities in the plan as computed by the attitude control planning expert (APE). This Parameters 12, 13, and 19 specify the mission profile input. These generate mutations of the two baseline mission profiles that we expected to use in operations: a 12 hour confidence-building profile that contained a single optical navigation goal and no IPS thrusting goals, and a six day primary profile that contained all of the goal types in Table 1. The mutations were designed to cover possible changes to the least stable elements of the profiles. Since the profiles are finalized prior to operations, and we had control over their contents, focusing on mutations of these profiles seemed a reasonable strategy. As it turned out, the profile had to be changed radically at the last minute for operational reasons. We reduced the horizon from six days to two, deleted five goals and changed the parameters and absolute (but not relative) placement of others. The goal types and overall profile structure remained the same. Fortunately, no new bugs were exposed by the new profiles since there would have been little time to fix them. Testing a broader range of profiles would have mitigated that risk. Broader test strategies are discussed in the next section.

id	Parameter	Values Tested	Range
1	experiment start	3	integer
2	plan start	10	integer
3	profile	12h, 6day, 2day	same
4	random seed	3 seeds	integer
5	exec latency	1, 4, 10	0-10
6	micas switch	off, ready	same
7	micas healthy	true, false	same
8	micas healthy (prior plan)	true, false, n/a	same
9	attitude	SEP, Image, Earth	same
10	end last thrust	-2d, -1d, -6h	integer
11	end last window	-2d, -1d, 0	integer
12	window duration	1,2,3,4,6 hours	integer
13	window start	0, 1, 2, 4	integer
14	targets/window	2, 20	0-20
15	images/target	3, 4, 5	3-5
16	image duration	1, 8, 16	1-16
17	SEP goals	6 configurations	infinite
18	SEP thrust level	6, 12, 14	15
19	SPE	1500,2400,2500	2500
20	slew duration	30, 120, 300, 400, 600, 1200	30-1200

Table 3: Tested Parameters

RAX operational requirements imposed three constraints among the parameter values as shown in Table 4. The test generator considered these constraints to avoid generating impossible cases. Constraint set one enforces the operational requirement that plans generated from the 12 hour profile will never have SEP goals, will start at the horizon start, and will have one of the four RAX idle states as the initial state. The second and third constraints enforce the following requirement. The plan start time is always one of the horizon boundaries (horizon waypoint goals) except when the exec requests a replan after a plan failure. In that case the exec first achieves one of the four RAX idle states, which becomes the initial state for the replan. So if the plan start is not a horizon boundary, constraint set two restricts the initial state parameters to the four idle states. When the plan start is at the horizon boundary for the six-day plan, all initial states are possible. This situation is reflected by the third constraint set.

The all-pairs and all-values test cases were generated automatically from the parameters and constraints described above. The constraints were satisfied by generating one test set for each constraint set. The sizes of the resulting test sets are shown in Table 5. An additional 22 cases exercised the planner interfaces.

id	Parameter	Constraint Sets (req'd values)		
		1	2	3
2	plan start	0	$\neq$ 3 days	3 days
3	profile	12-hr	6-day	6-day
8	micas healthy (prior plan)	none	none	*
9	attitude	Earth	Earth	*
10	end prior thrust	0	0	*
11	end prior window	0	0	*
17	SEP goals	null goal	*	*
18	SEP thrust level	0	*	*

Table 4: Constraint Sets

	1	2	3	Total
all-pairs	24	61	41	126
all-values	23	51	45	119

Table 5: Test Set Sizes

Test Selection Challenges

The selected tests were ultimately successful in that the on-board planner exhibited no faults during the experiment, and the tests provided the DS1 flight managers with enough confidence to approve RAX for execution on DS1. However we still have no objective measure of the delivered reliability. Objective metrics are needed to evaluate new and existing test strategies. It seems likely that there were a number of coverage gaps, though again we have no way to measure that objectively. This section makes some informed guesses as to where those gaps might be and suggests some ways of addressing them.

Value selection was *ad hoc*. Many parameters had large or infinite domains, and so only a few of those could be tested. That selection was *ad hoc*, based primarily on the tester's intuition. This undoubtedly left coverage gaps. One way to close the gap is to select values more intelligently based on a coverage metric. The metric would partition the values into equivalence classes that would exercise the domain model in qualitatively different ways. This would ensure adequate coverage while minimizing the number of values per parameter, and therefore minimizing the number of test cases.

Broader goal coverage needed. RAX planner testing focused on mutations of the baseline profile. Bugs exercised only by other goal sets would not have been detected. For example, transitioning from the 6 day scenario to the 2 day scenario compressed the schedule and eliminated the slack time between activities. This led to increased backtracking which caused new convergence failures. Exercising the full goal space would eliminate this coverage gap. It is also necessary for future missions, which must be confident that any goal set (profile) they provide will produce a valid plan. The challenge is how to provide this coverage with a manageable number of test cases.

One possibility is to create parameters that could specify any mission profile and perform all-pairs testing on this space. This would require at least one parameter for the start time, end time, and arguments for up to k instances of each goal type. For $k = 3$ the RAX model would require 140 parameters. These would replace parameters 12-19 of Table 3. Testing 3 values for each parameter would require 175 cases, and 5 values would require 337. All-values testing would require 884 and 1700 cases respectively.

This indicates that all-pairs testing of the full goal space is feasible, and that all-values testing might be feasible with sufficient test resources. Mission profiles would need to be generated automatically from the parameter values since manual generation is infeasible. One issue is that parameter vectors specify unachievable or impossible goal sets that would never occur in practice. These cases have to be automatically identified and eliminated to avoid the high analysis cost of discriminating test cases that failed due to impossible goals from those that failed due to a defect. Determining whether an arbitrary goal set is illegal is at least as difficult as planning, but it should be possible to detect many classes of illegal goals with simpler algorithms (e.g., eliminate goals that are mutually exclusive with any one or two domain constraints).

Although all-pairs testing of this parameter space is feasible, it is subject to the same effectiveness issues as the narrower all-pairs testing. That is, there could be coverage gaps from ad hoc value selection, and from not testing higher-order parameter combinations. Coverage metrics would help answer that question.

Formal Coverage Metrics Needed. Formal coverage metrics are sorely needed for planner validation. Metrics based on analyses of the domain model can indicate which parameter values and goal combinations are likely to exercise the domain model in qualitatively different ways. Formal metrics can identify coverage gaps and inform cost-risk assessments (number of cases vs. coverage).

Formal coverage metrics, such as code coverage, have been developed for critical systems but to our knowledge no metrics have been developed for measuring coverage of a planner domain model. The most relevant metrics are those for verifying expert system rule bases. The idea is to backward chain through the rule base to identify inputs that would result in qualitatively different diagnoses (e.g., (O'Keefe & O'Leary 1993)). Planners have more complex search engines with correspondingly complex mappings, and a much richer input/output space. It is not immediately obvious how to invert that mapping in a way that produces a reasonable number of cases.

One possible metric would be to measure the number and strength of goal interactions exercised by the test cases. The idea is to analyze the domain model to determine how the goals interact, and only test goal combinations that yield qualitatively different conflicts. For example, if goals A and B used power, we would

Task	Effort
Update/debug cases, tools	3.0
Run cases and analyzers	0.1
Review analyzer output	1.5
File bug reports	0.5
Close bugs	0.5
Total	5.6

Table 6: Test Effort in Work Weeks by Task

test cases where power is oversubscribed by several A goals, by several B goals, and by a combination of both goals. The coverage could be adjusted to balance risk against number of cases. One could limit the coverage to interactions above a given strength threshold.

This metric would extend on prior work on detecting goal interactions in planners to improve up the planning search, such as STATIC (Etzioni 1993) and Alpine (Knoblock 1994). These methods are designed for STRIPS-like planning systems and would have to be extended to deal with metric time and aggregate resources, both of which are crucial for spacecraft applications. One of the authors (Smith) is currently pursuing research in this area.

Test Automation

Automation played a key role in testing the Remote Agent planner. It was used for generating tests, running tests, and checking test results for convergence and plan correctness. Even so, the demand for human involvement was high enough to limit the number of test cases to just under three hundred per six week test period, or an average of ten cases per work-day.

The biggest demand for human involvement was updating the test cases and infrastructure following changes to the planner inputs, such as the domain model and mission profile. The next largest effort was in analyzing the test results. The test effort by task is shown in Table 6. This section discusses the automations that we found effective, the human bottlenecks, and opportunities for further automation.

Test Automation Tools

We employed two test automation tools: a test harness, a plan correctness oracle, and a trivial plan convergence oracle (the case succeeds if and only if the planner creates a plan within the time limit). The test harness converted the parameters in each test case into planner inputs, ran the planner on them, and saved the output. The full test suite could be run automatically in about 16 hours. The plan correctness oracle (Feather & Smith. 1999) reads a plan into an assertions database and then verifies that the assertions satisfy requirements expressed in first order predicate logic (FOPL). This tool was critical since inspecting plans manually for correctness would have been prohibitively time consuming and error prone.

Analysis Costs

The two analysis tasks are determining whether a test case has failed, and why. The first task was performed by the automated test oracles. Once the oracles have identified the failed test cases, the next analysis task is to determine *why* they failed. For each failed test case, the analyst determines the apparent cause of the failure. Cases with similar causes are filed as a single bug report.

Analyzing the test cases took eight to ten work-days for a typical test cycle and were largely unautomated. To determine why a plan failed to converge the analyst looked for excessive backtracking in the search trace or compared it to traces from similar cases that converged. Plan correctness failures also required review, although it was somewhat simpler (2-3 days vs. 8-10) since the incorrect plan provided context and the oracle identified the offending plan elements.

Automated diagnosis could reduce these effort of determining why the planner failed to generate a plan. There has been some work in this area that could be applied or extended. Howe (Howe & Cohen 1995) performed statistical analyses of the planner trace to identify applications of repair operators to states that were strongly correlated with failures. Chien (Chien 1998) allowed the planner to generate a plan, when it was otherwise unable to, by ignoring problematic constraints. Analysts were able to diagnose the underlying problem more quickly in the context of the resulting plan.

Analysis costs could also be reduced by only running and analyzing tests that exercise those parts of the domain model that have changed since the last release. One would need to know which parts of the domain model each test was intended to exercise. This information is not currently provided by the all-pairs strategy, but could be provided by a coverage metric: a test is intended to exercise whatever parts of the model it covers. A differencing algorithm could then determine what parts of the model had changed, where the "parts" are defined by the coverage metric.

Impact of Model and Interface Changes

About half of the test effort in each cycle were the result of changes to the planner inputs and interfaces. The test harness and test cases then had to be updated to support the new inputs. Making these changes only required a day or two. The bulk of the effort was caused by undocumented interface changes which managed to creep into most of the software releases. Planner inputs that were correct before the change could be incorrect after it, resulting in cases that fail when they should have succeeded or vice versa. Some of these errors were obvious, and detected by dry runs with a few test cases. Others were more subtle and not detected until the analysis phase, at which point the cases had to be re-run and re-analyzed after fixing the harness.

Appropriate software engineering practices can help minimize interface changes. Automation can also help

reduce the impact of changes when they do occur. We present a few possibilities below. The first two were used successfully for RAX.

Private Parameters. To minimize the impact from token parameter changes, we created the notion of a *private* parameter in the domain specification language. These were used when parameters were added to propagate values needed by new domain constraints or heuristics, the most common reason for adding new parameters to the model. Private parameters do not appear in the initial state or profile. Their values are set automatically by propagation from other parameters. This reduced the number of impactful parameter changes from 30 to 10.

Special Test Interfaces. To reduce the impact of changes to the initial state tokens and the format of the initial state file, both of which changed frequently, we negotiated an alternative testing interface to the initial-state generating function in the EXEC code. The test harness constructed an initial state by sending appropriate inputs to those functions, which then created the initial state in the correct format with the correct tokens. The idea of negotiating stable testing interfaces applies to testing complex systems in general, and should ideally be considered during the design phase.

Automated Input Legality Checks. The effort of identifying unintended mission profile and initial state inputs could have been greatly reduced by automatically checking their legality. One could imagine automating these checks by using an abstraction of the domain model to determine whether a set of goals are achievable from the specified initial state.

Conclusions

The main requirements for the Remote Agent planner were to generate a plan within the time limit, and that the plan be correct. These requirements were verified by running the planner on several input cases and automatically checking the results for convergence and plan correctness. Correctness was measured against a set of requirements reviewed by system and subsystem engineers. The cases were selected according to an "all-pairs" selection strategy that exercised all pairs of input parameter values. The selected values were at key boundary points and extrema. They were selected informally, based on the tester's knowledge of the domain model.

The tests focused on mutations of the two baseline mission profiles (goal sets) we expected to use in operations. This was sufficient for the experiment, but may not scale to broader operational contexts. Formal planner coverage metrics are sorely needed to make the best use of available cases and objectively balance risk (coverage) against cost.

The number of manageable cases could be increased by reducing the demand for human involvement. Analysis costs were high because of the need to provide initial diagnoses for cases where the planner failed to generate a plan, and the need to review the plan checker's output. Changes to the planner interfaces, including changes to the model, also created an overhead for updating and debugging the test harness. We suggested a number of ways to mitigate these factors.

The Remote Agent was a real-world, mission-critical planning application. Our experience in validating the Remote Agent planner raised a number of key issues. We addressed several of these, but many issues remain open. As planning systems are increasingly fielded in critical applications the importance of resolving these issues grows as well. Hopefully the Remote Agent experience will spark new research in this important area.

Acknowledgments

This paper describes work performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract from the National Aeronautics and Space Administration, and by the NASA Ames Research Center. This work would not have been possible without the efforts of the rest of the Remote Agent Experiment team and the other two members of the test team, Todd Turco and Anita Govindjee.

References

Chien, S. 1998. Static and completion analysis for knowledge acquisition, validation and maintenance of planning knowledge bases. *International Journal of Human-Computer Studies* 48:499–519.

Cohen, D.; Dalal, S.; Fredman, M.; and Patton, G. 1997. The AETG system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering* 23(7):437–444.

Etzioni, O. 1993. Acquiring search control knowledge via static analysis. *Artificial Intelligence* 62:255–302.

Feather, M., and Smith., B. 1999. Automatic generation of test oracles: From pilot studies to applications. In *Proceedings of the Fourteenth International Conference on Automated Software Engineering (ASE-99)*, 63–72. Cocoa Beach, FL: IEEE Computer Society. Best Paper.

Howe, A. E., and Cohen, P. R. 1995. Understanding planner behavior. *Artificial Intelligence* 76(2):125–166.

Knoblock, C. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68(2).

Muscettola, N.; Smith, B.; Chien, C.; Fry, C.; Rajan, K.; Mohan, S.; Rabideau, G.; and Yan, D. 1997. On-board planning for the new millennium deep space one spacecraft. In *Proceedings of the 1997 IEEE Aerospace Conference*, volume 1, 303–318.

Nayak, P.; Bernard, D.; Dorais, G.; Gamble, E.; Kanefsky, B.; Kurien, J.; Millar, W.; Muscettola, N.; Rajan, K.; Rouquette, N.; Smith, B.; Taylor, W.; and Tung, Y. 1999. Validating the ds1 remote agent. In *International Symposium on Artificial Intelligence Robotics and Automation in Space (ISAIRAS-99)*.

O'Keefe, R., and O'Leary, D. 1993. Expert system verification and validation: a survey and tutorial. *AI Review* 7:3–42.

Commentary on the Paper: Challenges and Methods in Testing the Remote Agent Planner

G. Michael Tong
Code 583
NASA Goddard Space Flight Center
Greenbelt, MD 20771
Gordon.M.Tong@gsfc.nasa.gov

Introduction

The 2nd NASA Workshop on Planning & Scheduling for Space asked reviewers to provide a written commentary on submitted papers. This commentary discusses the following paper: Challenges and Methods in Testing the Remote Agent Planner (the "paper"), which described the approach used to test the software called the Remote Agent planner (the "planner"). The Remote Agent software was used to control the Deep Space 1 (DS1) spacecraft for two days. The Remote Agent planner is a step towards spacecraft autonomy where human intervention is not required.

General Observations

Here are several general observations. First, planning and scheduling systems are difficult to test. Second, flight software operates in a more restricted environment than the ground software. Third, simpler algorithms can be added to the Remote Agent planner for use on specific missions to facilitate testing.

Testing Planning and Scheduling Systems

Typically, planning and scheduling systems receive requests and planning data as inputs and generate a schedule of activities as an output. In evaluating the output, there is no single correct answer or result. Two or more schedules that have different scheduled activities and different activity start times can be deemed correct by the user. In contrast, for a command and telemetry system, there is only one correct bit pattern for a particular sequence of commands, and only one correct result in converting a raw 8-bit telemetry point to engineering units.

The person who tests the planning and scheduling system (the "tester") has the daunting task of evaluating the output data without knowing what the correct answer should be. The software programmer may be able to calculate the correct answer manually by examining the code and analyzing the system requirements; however, the tester does not do this except for simple cases. Even for the software programmer, the manual calculation of the correct answer becomes impractical for a large number of activities on the output schedule.

The Remote Agent planner uses an automated tool to evaluate the correctness of the output schedule. The tool uses an assertions database and first order predicate logic (FOPL). More information about the tool appears in other papers by the authors. However, the planner can generate output that appears to be correct and passes the automated tool checking, yet the planner still contains subtle bugs.

Flight Software Environment

The flight software environment has limitations. Typically, the flight hardware is ten or more years behind the ground hardware in system performance. The flight software and hardware is an embedded system that has a limited amount of main memory and disk storage. Most flight software testing occurs on a special testbed rather than on the actual spacecraft.

Intermittent problems can occur during flight system testing. Repeatable problems are much easier to find and debug than intermittent problems, which are difficult to reproduce. Embedded flight software may not capture good debugging information when an intermittent problem occurs. Software programmers may be unable to solve problems that occur only one time or that occur infrequently because of an interrupt timing condition. The "Test Harness" may not be in place when an intermittent problem occurs. Of course, these mysterious problems are worrisome.

The traditional approach to commanding the spacecraft involves the use of command loads. The science user generates a science plan that specifies a list of targets for the instrument on the spacecraft to observe. A schedule is produced that takes into account the time to take exposures of the target and the time to slew (turning) from one target to another target. A sequence of commands, called a command load, is generated from the schedule and uplinked to the spacecraft. The onboard computer executes the command load, which is the plan or schedule for the spacecraft. The Remote Agent performs these planning and scheduling functions onboard that are traditionally done by ground systems. It represents the trend of the migration of ground functions to the flight software environment.

Simpler Algorithms

The paper described the dilemma of trying to select test cases to cover the "paths through the search space." One possibility would be to simplify the software under test rather than trying to test complicated software. Simpler algorithms could be added to the Remote Agent planner. For example, fixed activity timelines similar to the Space Shuttle Crew Activity Plan can be used to make the testing problem manageable. This approach is suitable only for scheduling applications that employ pre-stored scenarios.

Mission-specific algorithms could be developed. Those algorithms would be designed to facilitate the testing. Perhaps inheritance could be employed to make this approach more general.

Another possibility is to formalize the simulation approach. NASA typically employs simulations to test complicated systems. The formalization would be an extension of the methods described in the said paper, such as "Multiple Variation Test Cases" and test automation tools.

Specific Comments

Specific comments include (1) the description of "tokens" is not clear, (2) the relationship between the use of formal coverage metrics and the testing of heuristics is not clear, and (3) the testing only describes one portion of the flight software.

The paper defines tokens as follows: "The fundamental execution units in the plan are tokens (activities). Tokens also track spacecraft states and resources." However, the names of the tokens in Tables 1 and 2 look like attributes/properties or states, not objects/activities. The initial state tokens in Table 2 do not appear to be compatible with a class member initialization list approach. The technical approach on the use of tokens may be innovative, but it needs more explanation.

The paper advocates the need for formal coverage metrics that measure coverage for a planner domain model. However, the use of heuristics contributes to the testing difficulties. Testing each heuristic individually is straightforward. On the other hand, the interactions among the executions of various heuristics produce a complicated behavior that may result in the generation of an incorrect plan. Sometimes, a heuristic-based algorithm may fail to find a solution. In other words, if the goal is too difficult, there may be no solution for a given situation. It is not clear how formal coverage metrics can help in these cases.

Flight software testing is already expensive. While the paper addresses only the testing of the planner portion of the Remote Agent software, the testing of the other flight software components is also challenging. For example, the paper mentions other functions such as generating a list of targets, calculating maneuvers, performing maneuvers, calculating slews (turning) from one target to another target, performing slews, taking images, and fault detection and recovery. The cost of testing these flight software functions is a factor.

Conclusion

The paper provides an account of the issues in testing a complicated planning system, the approaches and techniques that were used, the technical and conceptual problems that occurred, and possible improvements and research areas. The method for test case selection was discussed, as well as the automation of some parts of the testing process.

Testing an artificially intelligent agent requires grappling with the solution space. The testing techniques described in this paper may be applied to other artificial intelligence applications.

High-Stake Planning

Sven Koenig
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280, USA
skoenig@cc.gatech.edu

Abstract

Preference models determine which one of several plans to prefer. It is important that planners use the same preference models as rational human decision makers because planners should make the same decisions as their human users (as long as they are rational), otherwise the planners are not of much use. While artificial intelligence planning has made lots of progress in the areas of representations of planning tasks and planning methods, it has not yet paid sufficient attention to the preference models of human decision makers. Utility theory is a formal framework for modeling the preferences of human decision makers and making rational decisions in high-stake decision situations. This paper reports on first steps in the direction of building planners that fit the risk attitudes of human decision makers in high-stake planning domains better than current planners, by combining descriptive methods from utility theory with constructive methods from artificial intelligence planning and operations research, thus combining the strengths of the two decision-making disciplines and extending the applicability of planners from artificial intelligence.

Introduction

Artificial intelligence planning has developed knowledge-based planners. These planners can have advantages over methods from other decision-making disciplines because they exploit more of the structure of large-scale planning tasks. Planning methods from artificial intelligence, for example, represent search spaces implicitly (for example, with STRIPS rules) and exploit the resulting decomposability. Artificial intelligence planning has made lots of progress over the past couple of year in the areas of representations of planning tasks as well as planning methods. However, artificial intelligence planning has not yet paid sufficient attention to the planning objectives, which are still primitive. In deterministic domains, planners from artificial intelligence have traditionally been used with the objective to find any plan that achieves the goal. To make their preference model richer, planners then began to associate execution costs with plans and preferred plans that achieve the goal with minimal plan-execution cost, that is, minimal consumption of one limited resource such as time, energy, or money. In probabilistic domains, planners from artificial intelligence usually either minimize the average plan-execution cost or, if the goal cannot be achieved for sure, maximize the probability of goal achievement. However, these preference models are often too simplistic to model the preferences of human decision makers adequately. How to plan with more realistic preference models, however, is a topic that has been neglected in the literature on artificial intelligence planning. It is an important topic because the recommendations of planners should reflect the opinions of their users correctly (as long as they are rational – we are not interested in irrational decision making). After all, the planners make suggestions for how to act and should make the same suggestions that the users would have made themselves. Otherwise the planners would not be very helpful.

Our research program therefore investigates how to build efficient planners that fit the preference models of rational human decision makers better than current planners, by combining descriptive methods from utility theory with constructive methods from artificial intelligence planning and operations research, thus combining the strengths of the two decision-making disciplines and extending the applicability of planners from artificial intelligence. We are interested, for example, in planning with deadlines and other resource limits as well as planning with multiple attributes, such as energy, cost, time, probability of goal achievement, prestige, and so on. In this paper, we report on a very first step in the direction of building planners with more realistic preference models. We study how to plan in high-stake decision situations with one resource, taking the risk-attitude of decision makers into account. High-stake decision situations occur in domains in which huge wins or losses of money, equipment, or even human life are possible. Many NASA domains are high-stake domains, including planning for autonomous space craft (Pell *et al.* 1997), and many human decision makers prefer to avoid the huge losses that are possible in these domains. We are also interested in how the risk attitude changes the optimal plan, for example, its influence on how long to plan before starting to act and how frequently to sense. Util-

ity theory is a formal framework for modeling the preferences of human decision makers and making rational decisions in high-stake decision situations. It suggests that decision makers maximize their average utility in these domains, where the utility is a strictly monotonically decreasing but often nonlinear function of the plan-execution cost. However, it specifies only what optimal plans are but not how they can be obtained efficiently. The difficulty we need to overcome is how to combine ideas from utility theory, artificial intelligence, and operations research, which is nontrivial but allows us to exploit the structure of planning tasks to make planning with nonlinear utility functions efficient. In the past decades, artificial intelligence researchers have acquired a large body of knowledge about how to plan efficiently with the current preference models, by utilizing the structure of planning tasks. We will demonstrate that, in some cases, these insights can be used to plan efficiently with nonlinear utility functions. This promises to make planning with some nonlinear utility functions as fast as planning with standard planners from artificial intelligence or operations research and enables one to participate in performance improvements achieved by other researchers in the currently very active field of probabilistic planning, while extending the applicability of existing planners from artificial intelligence or operations research. It also promises to allow for an easy integration of risk attitudes into existing decision-support systems, automated planning systems, and agent architectures.

High-Stake Decision Situations

High-stake decision situations occur in domains in which huge wins or losses are possible. In high-stake decision situations, rational human decision makers usually do not minimize the average plan-execution cost or maximize the average reward because they take risk aspects into account. This is why many human decision makers buy insurance even though the insurance premium is usually much larger than the average loss from the insurance cause. Another example is the following simple decision situation with two alternatives, one of which has the larger average pay-off and the other one of which has the smaller variance. Consider the decision situation shown in Figure 1, where you can participate in one and only one of the following two lotteries at no charge. When human decision makers have to decide whether they would like to get 4,500,000 dollars for sure or get 10,000,000 dollars with fifty percent probability (and nothing otherwise), many human decision makers prefer the safe alternative although its average pay-off is clearly lower – they are risk-averse. (Similarly, there are decision makers that are risk-seeking.) It is important to realize that this is perfectly rational behavior. Risk-averse human decision makers are willing to accept a smaller mean of the pay-off for a decrease in variance because they fear for the worst case. They are trying to avoid catastrophes, and a small variance avoids pay-offs that are much smaller than average. If a planner chose the lottery with the larger average pay-off, then many human decision makers would be extremely unhappy half of the time. It is therefore important that planners reflect the opinions of rational human decision makers correctly. After all, planners make suggestions for how to act and should make the same suggestions that the human decision makers would have made themselves. (This is the reason why investment advisors ask for your risk attitude before making investment recommendations.) However, artificial intelligence planning has not studied how to determine plans that correctly reflect the risk attitudes of rational human decision makers in high-stake decision situations.

Preference Models

Bernoulli and Von Neumann/Morgerstern's utility theory (von Neumann & Morgenstern 1947; Bernoulli 1738) has investigated how rational human decision makers make decisions in high-stake decision situations. Utility theory can explain why they often do not minimize the average plan-execution cost. It suggests that it is rational to choose plans with maximal average utility, where the utility $u(c)$ is a strictly monotonically decreasing function of the plan-execution cost c. Human decision makers sometimes deviate from utility theory because, different from other theories, such as Kahneman and Tversky's prospect theory (Kahneman & Tversky 1979), utility theory does not model human inadequacies in decision making and thus is not able to explain all empirical findings about human decision making. This is not a problem for planners since planners are supposed to follow a theory of rational (normative) rather than empirical decision making. Maximizing average utility and minimizing average plan-execution cost result in the same decisions if either the domains are deterministic or the utility functions are linear. These assumptions, however, are often not satisfied. For example, nonlinear utility functions are necessary to account for the risk-averse attitudes of many rational human decision makers for the lottery example above. The lottery example can be explained as follows: Assume that a human decision maker is risk-averse and has the concave exponential utility function shown in Figure 2 and no other assets. This human decision maker associates utility (here: pleasure) 0.00 with a wealth of 0 dollars, utility 0.74 with a wealth of 4,500,000 dollars, and utility 0.95 with a wealth of 10,000,000 dollars. Then, the (average) utility of getting 4,500,000 dollars for sure is 0.74, whereas the average utility of getting 10,000,000 dollars with fifty percent probability is only 0.475. In this case, the safe alternative maximizes the average utility for this human decision maker, which explains why this human decision maker chooses the safe alternative over the one with the larger average pay-off. Other human decision makers can have other utility functions and thus arrive at different conclusions.

Choices	Probability	Pay-Off	Average Pay-Off	Utility	Average Utility
Choice 1	50 percent	10,000,000 dollars	5,000,000 dollars	0.95	0.475
	50 percent	0 dollars		0.00	
Choice 2	100 percent	4,500,000 dollars	4,500,000 dollars	0.74	0.740

Figure 1: Decision Situation

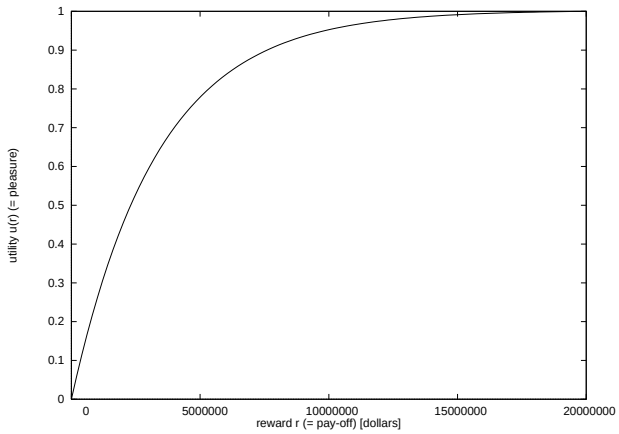


Figure 2: Risk-Averse Utility Function

Exponential Utility Functions

Utility theory specifies only what optimal plans are but not how they can be obtained efficiently, that is, other than by enumerating every trajectory of every possible plan. Operations research and control theory use dynamic programming methods to find plans with maximal average utility (Marcus *et al.* 1997; Whittle 1990) but these methods often do not exploit the structure of planning tasks completely. Artificial intelligence planning has developed knowledge-based planners that plan efficiently in large domains but usually either minimize the average plan-execution cost or, if the goal cannot be achieved for sure, maximize the probability of goal achievement. It might seem that one could simply replace the resource consumptions with their utilities and then continue to use planners from artificial intelligence. This is certainly true for one-stage decision situations, such as the lottery example. However, different from the lottery example, real-world planning tasks are usually much more complex and consequently involve much more complex decisions. For these multi-stage decision situations, it is impossible to simply replace the resource consumptions with their utilities and then continue to use planners from artificial intelligence. Assume, for example, that one incurs cost c_1 at the first time step and cost c_2 at the second time step, which is also the last one. Then, one obtains $u(c_1) + u(c_2)$ if one replaces the costs with their utilities whereas the correct utility is $u(c_1 + c_2)$. Instead, our multiplicative planning-task transformation (Koenig 1998) makes use of existing planners from artificial intelligence or operations research by transforming planning tasks with exponential utility functions to planning tasks that planners from artificial intelligence or operations research can solve. The transformed planning tasks can be solved by finding plans with maximal probability of goal achievement or minimal average plan-execution cost. The multiplicative planning-task transformation is such that optimal plans for the transformed planning task are also optimal for the original one, and good ("satisficing") plans for the transformed planning task are also satisficing for the original one. The multiplicative planning-task transformation works for convex and concave exponential utility functions. Convex exponential utility functions are of the form $u(c) = \gamma^{-c}$ for parameter $\gamma > 1$, and concave exponential utility functions are of the form $u(c) = -\gamma^{-c}$ for parameter γ with $0 < \gamma < 1$. These utility functions are expressive because the parameter γ can be used to trade-off between minimizing the best-case, average, and worst-case plan-execution cost (Watson & Buede 1987). As γ approaches infinity, the human decision makers become more risk-seeking and thus more interested in plans with small best-case plan-execution cost (under appropriate assumptions) (Koenig & Simmons 1994; Koenig 1998). As γ approaches one, the human decision makers become more interested in plans with small average plan-execution cost. Finally, as γ approaches zero, the human decision makers become more risk-averse and thus more interested in plans with small worst-case plan-execution cost. Thus, exponential utility functions can express a continuum of risk attitudes, that includes the utility functions of the lottery example above.

Crisis Management

Our current application area is managing environmental crisis situations such as oil spills (Desimone & Agosta 1994). The goal of planning in the oil-spill domain is to determine how to manage resources such as human teams, vessels, and equipment to contain and clean up oil spills, taking into account all costs incurred until they are cleaned up completely (expressed in dollars). Crisis management domains have several advantages: Efficient planning methods for them directly benefit the public. They are important high-stake planning domain for which many human decision makers are very risk averse since they prefer to avoid the huge losses that are possible in these domains. Thus, planners that minimize the average cost do not use the same preference model as many human decision makers and

arrive at different courses of action.

Sensor Planning

An interesting observation in crisis management situations is that human decision makers gather large amounts of information even if it is costly and thus might not be part of a plan that minimizes the average plan-execution cost. In the oil-spill domain, sensing operations include, for example, sending out helicopters to gather information about how oil spills drift. The knowledge that an oil spill drifts towards a nature preserve can be used to concentrate resources in the surrounding sea sectors to prevent the oil from reaching sensitive shores. Thus, sensing provides information that can reduce the plan-execution cost but comes at a cost itself. Sensor planning for risk-averse or risk-seeking human decision makers involves the same tradeoff as the lottery example above (Koenig & Liu 1999). For example, adding more sensing operations than is necessary to minimize the average plan-execution cost increases the mean of the plan-execution cost (because sensing is expensive) but also reduces its variance (because the information obtained can be used to avoid catastrophes). Consequently, we speculate that risk-averse human decision makers add more sensing operations than is necessary to minimize the average plan-execution cost.

Case Study

In the following, we provide a case study of how the risk attitude influences the sensing frequency to test our hypothesis that risk-averse human decision makers add more sensing operations than is necessary to minimize the average plan-execution cost. At the same time we demonstrate that our multiplicative planning-task transformation can be combined with existing planners that minimize the average plan-execution cost to yield planners that maximize average utility for exponential utility functions. We apply the multiplicative planning-task transformation to the sensor planner by Hansen (Hansen 1997) that combines methods from operations research (namely, policy iteration (Howard 1964)) and artificial intelligence (namely, the A* search method (Nilsson 1971)) to find plans with minimal average plan-execution cost. In (Koenig & Liu 1999), we present the resulting sensor planner and a proof of its correctness. Here, we present the results of a case study where we apply the resulting sensor planner to simple artificial robot-navigation tasks with actuator uncertainty. Notice that these robot-navigation tasks are not high-stake planning domains and *we do not suggest to apply our planning methods to planetary rover navigation in the way we do it here.* We use robot-navigation tasks merely as a test bed because they are much simpler to solve than the oil-spill domain (and we can thus run a much larger number of experiments in a reasonable amount of time), they allow us to visualize the planning results much more easily, and they have been studied before in the context of sensor planning (Hansen 1997) and can therefore be considered good test problems for new sensor planners. We consider robot-navigation tasks with the following properties. The robot has to navigate from a given start location to a given goal location in a known environment. Since motion is noisy, the robot can deviate from the nominal path but it can always opt to sense its current location. Sensing provides certainty about its current location but is costly (for example, consumes energy). We assume that there is a finite set of locations L. The robot knows that it starts at location $l_{start} \in L$ and its task is to navigate to location $l_{goal} \in L$ and be sure that it stops at exactly that location. There is a finite set M of movement actions, all of which can be executed at all locations. Motion uncertainty is modeled with conditional probability distributions. Executing movement action $m \in M$ results with cost $c(l, m) > 0$ and probability $p(l'|l, m)$ in location l'. The robot receives no feedback as to what its new location is (which makes the simplifying assumption that the cost of the executed actions cannot be observed directly) but there is one sensing action o that can be executed at all locations. Executing it incurs cost $c(l, o) > 0$ and reports the current location of the robot with certainty. We assume that it is possible to reach every location from every other location. Figure 3 (left) shows the gridworld that we use in our experiments, where the locations are squares. The start location is C1 and the goal location is J1. The robot can always sense its current location (O) or move north (N), east (E), south (S), or west (W) to an adjacent square. If the robot attempts to move in a certain direction (say, move east in square C1), then it either moves as intended (C2, with probability 0.6) or strays off by one square to the left (B2, with probability 0.2) or right (D2, with probability 0.2) due to actuator noise and not facing precisely in the right direction. The robot does not move when it bumps into the border of the gridworld. The movement cost ranges from 1.0 to 10.0. It is low for roads (white) and high for muddy terrain (darker colors). The sensing cost is always 0.2. Notice that even very risk-averse robots have to trade off between minimizing the worst-case and average plan-execution cost in our example domain even if they want to approximate plans with minimal worst-case plan-execution cost, a planning objective popular in robotics (Lozano-Perez, Mason, & Taylor 1984). It is not possible to minimize the worst-case plan-execution cost directly in our example domain because all plans cycle with some probability and thus have a worst-case plan-execution cost that is infinite. So, even very risk-averse robots need less risk-averse planning objectives than minimizing the worst-case plan-execution cost but more risk-averse planning objectives than minimizing the average plan-execution cost. Maximizing the average utility for an exponential utility function with γ sufficiently close to zero provides such a planning objective.

Figure 4 shows that the sensing frequency (that is,

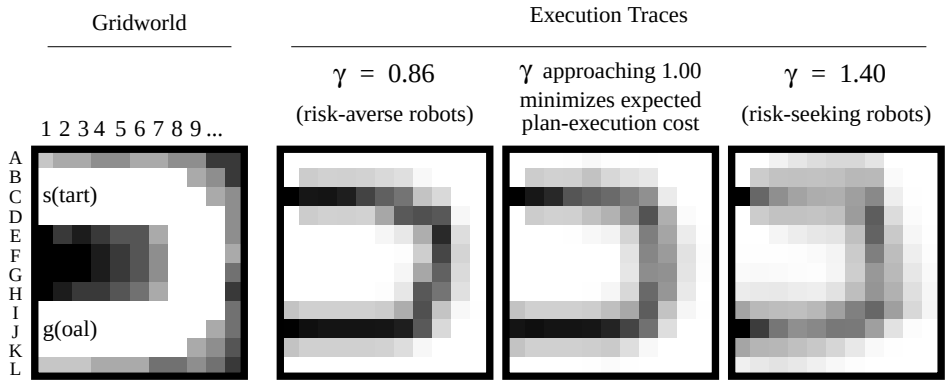


Figure 3: Gridworld and Execution Traces

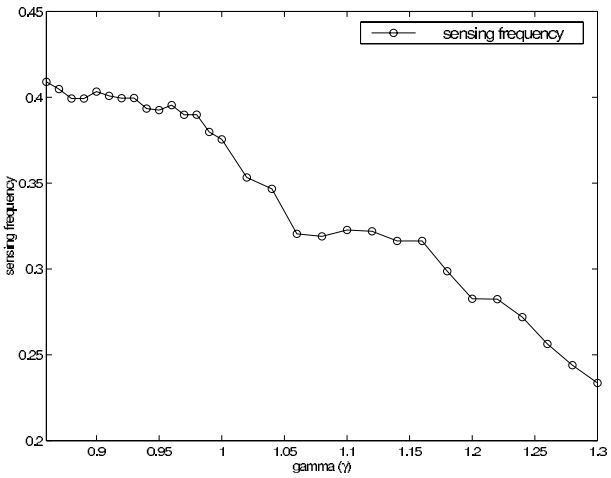


Figure 4: Sensing Frequency

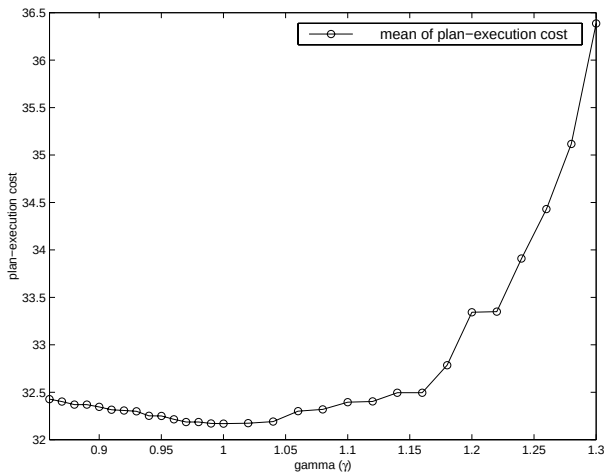


Figure 5: Mean of Plan-Execution Cost

the percentage of sensing actions among all executed actions) increases as the robots become more risk-averse and γ decreases. This is also illustrated in Figure 7, that shows optimal sensor plans for two different values of γ.[1] The sensor plans are depicted as gridworlds, each location of which is annotated with an action sequence. These action sequences are used as follows: After the robots have executed a sensing action, they look up the action sequence that corresponds to the sensed location, execute it, and repeat the process, until they sense that they are at the goal location. For example, for $\gamma = 0.86$, the action sequence of location B2 is SEO. Consequently, after the robot has sensed that it is at location B2, it first moves south (S), then moves east (E), and finally senses again (O). Locations whose action sequences are not used for getting from the start location (C1) to the goal location (J1) are left blank. Figure 3 (right) shows how often the robots visit each grid square during two million runs for three different values of γ. Darker colors indicate a larger number of visits. Thus, more risk-averse robots are more likely to stay on the road and close to the nominal path, which is possible due to the increased sensing frequency. That more risk-averse robots are more likely to stay on the road can be explained as follows: By staying on the road, the robots are likely able to avoid the large cost necessary for getting out of the mud, which risk-averse robots consider to be important. On the other hand, smaller sensing frequencies and attempts to cut the cor-

[1] There are some exceptions to this trend, for example, in the vicinity of the goal. This can be explained as follows: Risk-seeking robots assume that short action sequences that have a chance of reaching the goal location will indeed reach it. Thus, they execute these action sequences followed by a sensing action to confirm that they have reached the goal location. For example, for $\gamma = 1.40$, the action sequence of location I2 is SO. The robot hopes that it will drift to the goal location J1 as it moves south, although this is less likely than moving to location J2. More risk-averse robots are more cautious and execute longer action sequences. For example, for $\gamma = 1.86$, the action sequence of location I2 is SWO, which reaches the goal location with higher probability than the action sequence SO. This phenomenon and similar phenomena contribute to the small local minima in the graph of Figure 4.

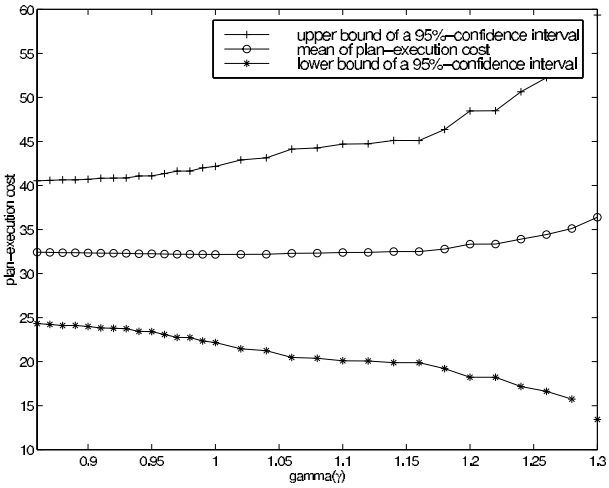


Figure 6: Mean with Confidence Interval

ners decrease the probability that the robots stay on the road but also decrease the plan-execution cost in the best case, which risk-seeking robots consider to be important. (In fact, the action sequences of the start location get longer and longer as the robots become more risk-seeking until the action sequences are able to move the robots to the goal location in the best case.) This explanation suggests that there is a mean-variance trade-off in our example domain. As pointed out earlier, mean-variance trade-offs are often used as crude but easy-to-understand explanations for trade-offs between minimizing the worst-case, average, and best-case plan-execution cost. For example, the graph in Figure 5 shows the mean of the plan-execution cost, and the difference of the upper and lower graphs in Figure 6 corresponds to four times the standard deviation of the plan-execution cost. The mean of the plan-execution cost increases but the variance decreases as the robots become more risk-averse and γ decreases from one to zero. The variance decreases because more risk-averse robots stay on the road and close to the nominal path. The mean-variance trade-off can be explained as follows: More risk-averse robots are willing to accept a larger mean of the plan-execution cost for a decrease in variance because they fear for the worst case. A small variance avoids a plan-execution cost that is much larger than average. Figure 6 illustrates this using the upper bound of a 95-percent-confidence interval (that is, mean plus twice the standard deviation) as an approximation of the worst-case plan-execution cost. The upper bound indeed decreases as the robots become more risk-averse since the decrease of the variance outweighs the increase of the mean. On the other hand, more risk-seeking robots are willing to accept a larger mean for an increase in variance since a larger variance promises a chance to realize a plan-execution cost that is much smaller than the average plan-execution cost. Figure 5 illustrates this using the lower bound of a 95-percent-confidence interval (that is, mean minus twice the standard deviation) as an approximation of the best-case plan-execution cost. The lower bound indeed decreases as the robots become more risk-seeking since the increase of the variance outweighs the increase of the mean. Our sensor planner is not only as easy to implement as the sensor planner that it extends but also almost as efficient. For $\gamma = 0.86$, our sensor planner expands 2,071 nodes and needs an average of 4.6 milliseconds per node expansion on a Sun Ultra 1 running Solaris 7. The original sensor planner by Hansen, that our sensor planner extends, corresponds to the case where γ approaches one. It needs 2,815 node expansions and 2.0 milliseconds per node expansion. For $\gamma = 1.40$, our sensor planner needs 5,808 node expansions and 5.2 milliseconds per node expansion. The number of node expansions depends on the sensing frequency. It increases as the sensing frequency of the optimal plans decreases. Our sensor planner has a slight run-time disadvantage per node expansion compared to the original sensor planner by Hansen because it has to calculate exponentials and logarithms, and its heuristic search method cannot calculate the heuristic values quite as efficiently as the original sensor planner.

Conclusions

We described a method for creating planners that find plans with maximal average utility for a given exponential utility function and thus reflect the risk attitude of rational human decision makers in high-stake decision situations better than plans with minimal average plan-execution cost. Our method generalizes the planning objectives of traditional planners from artificial intelligence that often either minimize the average or the worst-case plan-execution cost or, if the goal cannot be achieved for sure, maximize the probability of goal achievement. Our multiplicative planning-task transformation is a fast simple context-insensitive representation change that can be performed locally on various representations of planning tasks, including rule-based (STRIPS) representations (Fikes & Nilsson 1971) and (totally and partially observable) Markov decision process models. We applied the multiplicative planning-task transformation to an existing sensor planner. The resulting sensor planner is not only as easy to implement as the sensor planner that it extends but also almost as efficient. Our case study showed that the frequency of sensing depends on the trade-off between minimizing the best-case, average, and worst-case plan-execution cost. More risk-averse human decision makers tend to sense more frequently, and planners should reflect this behavior accurately. This research is beneficial for NASA since many NASA domains are high-stake planning domains. Human decisions makers often want to make high-level decisions themselves, for example, because they have background knowledge that planners do not have (or because of political considerations). However, there are disadvantages to having humans in the loop for every decision on-board of un-

$\gamma = 0.86$ (risk-averse robots)

	1	2	3	4	5	6	7	8	9	10	11
A								SSSO			
B		SEO	SEO	SEO	SEO	SEO	SO	SSO			
C	EO	EO	EO	EO	EO	EO	SEO	SO	SO	WSO	
D		NEO	NEO	NEO	EEEO	EEO	EO	ESO	SO	WSO	WWSO
E							EESO	ESO	SO	WSO	
F								SSSO	SO	WSO	
G								SSO	SO	WSO	
H						SSWO	SSWO	SO	WSO	WO	
I	SO	SWO	SWO	SWO	SWO	SWO	SWO	SWO	WO	WWO	WWWO
J	goal	WO	WO	WO	WO	WO	WO	WO			
K	NO	NWO	NWO	NWO	NWO	NWO	NWO	NWO	NWWO		
L							NNWO	NNWO			

$\gamma = 1.40$ (risk-seeking robots)

	1	2	3	4	5	6	7	8	9	10	11
A	SSEEEO				SSEO	SSEO	SSSO	SSSO			
B	SEEEEO			SEEO	SEEO	SEO	SSO	SSO	SSSSSSSWO		
C	EEEEEEESO			EEEESO	EEESO	EESO	SO	SO	SSSSSSWO		
D	EEEEEEESO			EEEESO	EEESO	EESO	ESO	SSSSSWO	SSSSSWO	WSSO	WWSSO
E	NNEEEEO	NEEEEEO	NEEEEO	NEEEO	NEEO	EESSSO	ESSSO	SSSSWO	SSSSWO	WSSO	WWSSO
F	SSSSO	SSSSO	SSSSO	SSSSWO	SSSSWWO	EESSO	ESSSO	SSSWO	SSSWO	WSSO	WWSO
G	SSSO	SSSO	SSSO	SSSWO	SSSWWO	SSSWWWO	SSSWWWWO	SSWO	SSWO	WSO	WWSO
H	SSO	SSO	SSWO	SSWWO	SSWWO	SSWWWO	SSWWWWO	SSWWWWWO	SWO	WWO	WWWO
I	SO	SO	SWO	SWWO	SWWWO	SWWWWO	SWWWWWO	SWWWWWWO	WO	WWO	WWWO
J	goal	WO	WWO	WWWO	WWWWO	WWWWWO	WWWWWWO	WWWWWWWO	WWWWWWWO	WWWWWWWWO	WWWO
K	NO	NO	NWO	WWWO	WWWWO	WWWWWO	WWWWWWO	WWWWWWWO	WWWWWWWO	WWWWWWWWO	WWWWWWWWWO
L	NNO	NNO	NNWO	NWWO	NWWWO	NWWWWO		NWWWWWWO			

Figure 7: Optimal Sensor Plans

manned spacecraft, including decisions in crisis situations, due to the resulting time delay. In these cases, it is important that artificial intelligence planners make similar decisions as rational human decision makers. The research that we reported here is only a first step in the direction of planning with more realistic preference models by combining descriptive methods from utility theory with constructive methods from artificial intelligence planning. Future work includes how to plan with multiple attributes (such as time, energy, and money) and resource limits.

Acknowledgments

This paper presents a high-level overview of work reported in more detail in (Koenig & Liu 1999).

References

Bernoulli, D. 1738. Specimen theoriae novae de mensura sortis. *Commentarii Academiae Scientiarum Imperialis Petropolitanae* 5. Translated by L. Sommer, *Econometrica*, 22: 23–36, 1954.

Desimone, R., and Agosta, J. 1994. Oil spill response for plannig in stochastic domains: The application of artificial intelligence planning technology. In *Proceedings of the Simulation Multiconference*.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Hansen, E. 1997. Markov decision processes with observation costs. Technical Report CMPSCI 97–01, Department of Computer Science, University of Massachusetts, Amherst (Massachusetts).

Howard, R. 1964. *Dynamic Programming and Markov Processes*. MIT Press, third edition.

Kahneman, D., and Tversky, A. 1979. Prospect theory: An analysis of decision under risk. *Econometrica* 47:263–291.

Koenig, S., and Liu, Y. 1999. Sensor planning with nonlinear utility functions. In *Proceedings of the European Conference on Planning*, (in press).

Koenig, S., and Simmons, R. 1994. How to make reactive planners risk-sensitive. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*, 293–298.

Koenig, S. 1998. Representation changes for planning with exponential utility functions. In *Proceedings of the Symposium on Abstraction, Reformulation, and Approximation*, 79–84.

Lozano-Perez, T.; Mason, M.; and Taylor, R. 1984. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research* 3(1):3–24.

Marcus, S.; Fernàndez-Gaucherand, E.; Hernàndez-Hernàndez, D.; Colaruppi, S.; and Fard, P. 1997. Risk-sensitive Markov decision processes. In et. al., C. B., ed., *Systems and Control in the Twenty-First Century*. Birkhauser. 263–279.

Nilsson, N. 1971. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill.

Pell, B.; Gat, E.; Kaesing, R.; Muscettola, N.; and Smith, B. 1997. Robust periodic planning and execution for autonomous spacecraft. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

von Neumann, J., and Morgenstern, O. 1947. *Theory of games and economic behavior*. Princeton University Press, second edition.

Watson, S., and Buede, D. 1987. *Decision Synthesis*. Cambridge University Press.

Whittle, P. 1990. *Risk-Sensitive Optimal Control*. Wiley.

Decision Analysis Meets High-Stake Planning
Commentary on Paper by Sven Koenig

Roger Arno

NASA Ames Research Center
MS 244-14
Moffett Field, CA 94035-1000
rarno@mail.arc.nasa.gov

Abstract

Thoughtfully structured, analytical processes should be part of NASA's (or any goal-oriented organization's) decision-making. Decision theory, incorporating schemes for characterizing outcomes, uncertainties, and risk preferences, can be very useful in structuring and formalizing logical decisions. This commentary on Koenig's *High-Stake Planning* paper gives one interpretation of the proper and possible use of structured decision techniques in NASA -- at least in those situations where unquantifiable political factors do not overwhelm the process.

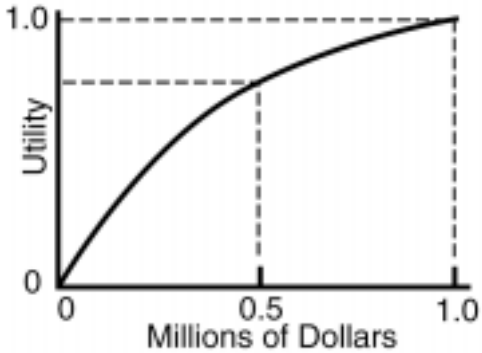
Perhaps the most interesting issue raised by Koenig's paper is whether it is useful to characterize or model NASA's (or any decision-maker's) risk preference in terms of "optimism" and "pessimism." Decision-makers can feel lucky or unlucky, but can such feelings be modeled in any useful or meaningful way? And more importantly, should unfounded hunches or irrational mood swings be included in business and technical decisions? The answer to both questions is very likely "no."

Expected Value Decisions

Decision analysts generally agree that choices can be made on the basis of the likelihood of an event (or "outcome") and the impact of that event. The foundation of most decision theory is that decision options can be compared and prioritized by calculating their "expected values," which are the products of their respective impact magnitudes (benefit or disbenefit of the outcome) and the probabilities of their occurrence. The reason for using "expected value" is that it works. It maximizes gain and minimizes loss over the long term. It is proven every day in casino operations, as well as in businesses that routinely make decisions with profit and loss potential.

In real-life situations, "expected values" frequently must be adjusted to reflect the perceived (subjective) worth of an outcome or the subjective perception of the probability of its occurrence. This kind of subjectivity is widely evident, as most decision makers shy away from extreme losses, or they disproportionately favor relatively sure, albeit smaller, gains. For example, a typical decision-maker might select an assured $200K over a 50/50 chance at $1M. Clearly, this approach would not produce the greatest profit over many opportunities, but this risk-averse behavior is real. It simply says that for this decision-maker, a sure $200K has a utility that is equal to or greater than an uncertain $1M. It does not say that any assured dollar amount is worth more than a 50/50 chance at a million. On the contrary, many people are willing to give up a dollar in hand for a five-million-to-one chance at $1M in the lottery.

It is important to notice that risk aversion is not dependent so much on the probability of winning or losing, as it is on the dollar amounts involved. NASA might be willing to spend thousands or millions of dollars on a very speculative research activity with uncertain payoffs, but just as a casino owner would not risk his entire business on a fair bet, NASA wants to avoid ventures that might jeopardize its credibility and future. This risk-averse nature (which could equally well be called loss-averse nature) can be reasonably well characterized for different decision-makers or organizations. It is frequently displayed as a plot of the subjective utility of dollar amounts against actual dollar values, as shown below.



Here in this typical risk-averse utility plot, it can be seen that a million dollars does not have twice the utility of half-a-million dollars. The utility curve for a casino owner might be straight. For a person in desperate

need of a million dollars, the curve would probably be bent the other way.

NASA and Risk

At the height of the space race, in the 1960s and 1970s, a national inferiority complex produced money and a willingness to take chances. Early US attempts to launch satellites resulted in frequent failure. The prevailing attitude was that the value of success far outweighed the risk and embarrassment of failure. In decision analysis parlance, the expected value of a successful launch was larger than the expected value of losing the nation's technical preeminence to a foreign power.

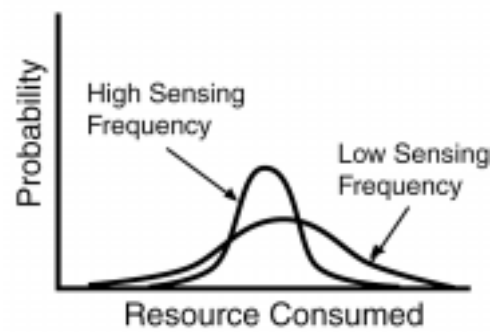
Today, with relatively tight budgets, ambitious projects, and decreased public anxiety about national pride, low cost and low risk are priorities. NASA's current risk-averse, safety first approach is largely derived from the high degree of success it enjoyed decades ago. NASA, through skill, luck, and ample funding, demonstrated impressive space exploration capabilities. The public and congress came to expect nothing less than excellence from the world's greatest space agency. In that context, failure became not only a contradiction to past performance, it was a tremendous embarrassment to NASA and the nation. The loss of the first teacher in space was devastating. The failure of space systems to do what previous space systems did seems inexcusable. The resulting risk averseness is partly an acknowledgement that failure has a cost that exceeds the missed science or money spent.

As the danger of world domination by other nations has, at least temporarily, subsided, decision criteria are more complex. At the same time NASA has proclaimed its willingness and ability to "do more for less," it is advocating the chancier "faster, better, cheaper" philosophy. In addition, lower budgets and ambitious programs have forced some risky international partnerships (e.g., the International Space Station). With no little green men on Mars and no jungles under the clouds of Venus, science goals are more esoteric. These crosscutting philosophical overtones make NASA decisions more political, opaque, obscure, and difficult to model. While there is some doubt that NASA decision processes can be modeled at all, the bigger question may be whether they should be modeled, or whether, perhaps, NASA decisions should be more methodical and structured, with decision criteria openly discussed.

The Rover Operations Example

The application of decision analysis to Koenig's rover is difficult due to a number of unspecified parameters (as of this writing). First, there is no specific value or utility metric for the rover completing its mission in a given time, or for partially completing its mission. Second, there is no limit indicated on the availability of the resource (presumably electrical energy, although it could be computer memory, or something else). Third, there are no penalties identified for getting lost or stuck in the mud. Nonetheless, we can apply decision theory by assuming that minimizing the resource consumption is the ultimate goal.

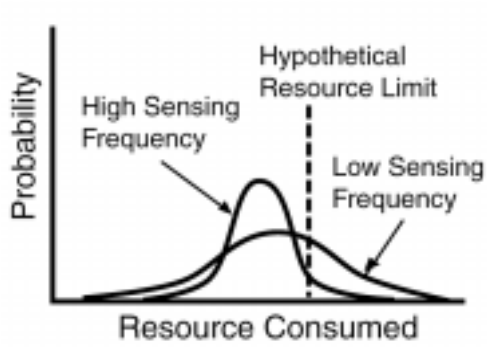
In this example, resource consumption is proportional to the distance the rover moves. Each location-sensing event provides useful navigation information, but it also consumes some of the resource. Less frequent sensing saves resource, but can cause the rover to stray off the path, resulting in more resource consumption. For each sensing frequency there is a statistical spread of possible resource consumption levels. The following figure shows typical consumption curves for two sensing frequencies.



It is easy enough to calculate the expected value (or "expected resource consumption") for each sensing frequency. On the average, high sensing frequency wins, but in a small percentage of trial runs, low sensing frequency produces even lower consumption levels.

Koenig's thesis is that real (e.g., NASA) decision makers are swayed by the variance in the resource consumption curve. He postulates that an "optimistic" decision-maker may well select the option with the highest average and highest possible resource consumption, because it also offers a small chance of a very low consumption. Conversely, he suggests a "pessimistic" decision-maker will select the sensing frequency with the lowest variance and lowest maximum consumption, in order to avoid the option with highest possible consumption. Realistically, without some inordinate advantage for low resource consumption, there is no rationale for selecting anything but the sensing frequency with the lowest expected resource consumption.

In a situation where the resource is limited, and the chance of failure has important negative consequences, the choice becomes clearer. The figure below arbitrarily sets a hypothetical upper limit on resource consumption. Clearly, the decision-maker who selects the lower sensing frequency is not only foolishly "optimistic," but accepting a significant probability of failure.



Indeed, some decision-makers feel lucky on some days, and unlucky on others. However, attempting to formally characterize and apply this behavior can be counterproductive. First of all, overlaying intuition and hunches on carefully constructed expected value calculations will ultimately result in reduced overall productivity. Second, attempting to overlay best estimates of risk and uncertainty with additional subjective feelings about the risk is really an admission that the original probability estimates are not believed and need to be adjusted. Placing subjective feelings on the best estimates of probabilities is a bit like a craps player saying, "I know the odds of throwing a seven are truly six out of 36, but today I feel like they are 18 out of 36." Baring fortune telling and supernatural intervention, decision making should be based on rational, honest, thorough assessments of probability and utility. The only time intuition is useful is when it stimulates further assessments of the variables used to calculate the expected values.

Problems with Decision Analysis in NASA

In most high-level, high-stake NASA decisions, political considerations (congressional support, fear of failure, threatened credibility, degraded image, budget threats, etc.) are dominant. Attempts to formalize and systematize the process would require that decision-makers identify and quantify personal and organization agendas that they prefer to keep hidden. For decisions that fall below political visibility, decision analysis can be fruitfully applied. However, there are still barriers that seem to escape the insight of the naïve decision analyst.

One reason that traditional decision analysis and utility theory cannot be applied directly to NASA decisions is purely mechanical. The curves plotted to show the nonlinear utility of money as a function of its numerical value do not readily translate to assessments of the worth of mission successes and failures. Imagine, for example, NASA mission option A, which collects 4 GB of data and is judged to have twice the value of mission option B, which collects 3 GB of data. There is an obvious nonlinearity of utility to data quantity, since the last gigabyte adds a disproportionately large amount of utility. The important underlying message is that the assessed mission option values already include the nonlinear subjectivity. If option valuation is done correctly and credibly, further adjustments are neither appropriate nor technically meaningful.

Extending the above example, assume that the probability of success for option A is 50%, and that for option B is 100%. The two options thus have identical expected values (i.e., 50% x 2.0 = 1.0 = 100% x 1.0). Adhering to decision analysis theory and practice, NASA should see the options as equivalent. Suppose, however, that a decision-maker refused to pick option A unless it had at least a 75% probability of success. The "expected value" decision method has either failed or needs some adjustment.

A likely explanation for the decision-maker's seemingly irrational behavior is that there are other factors not formally unaccounted for. These might include the negative impacts of mission failure. Formal decision analysis is of little value if important considerations are missing. If there are significant potential financial, science, personal or political losses, they must be incorporated, along with their respective probabilities of occurrence. The same is true for political or other payoffs that might come with option selection (e.g., money to a contractor in an important congressional district) or mission success (e.g., the favorable publicity of an elder astronaut hero in space).

Another possible reason for the decision-maker's inexplicable attitude is a subtle variation of the previously discussed problem. It may be that he/she has misrepresented the relative utility of the two options. Perhaps option A actually has a subjective value (or utility) of 1.25 or 1.33, or 1.5 (not 2.0) times that of option B. If that is the case, it would be silly to attempt plotting some new set of perceived values of the mission options against the original perceived subjective values. Piling subjectivity upon subjectivity is just an admission that the original subjective values were incorrect and not the ones that the decision-maker wants to use. The solution is to correct the 2.0 valuation or relative utility.

Another misused concept was addressed earlier. It is the notion that somehow risk preference is about the subjective reassessment of probabilities of outcomes. This line of thinking is to the effect that choices are based on the subjective assessment of the probabilities of success and failure. A decision-maker, it may be postulated, will avoid situations where the probability of winning is low or the probability of losing is high. However, lotteries and high-value business and personal decisions clearly show that risk averseness is determined more by the magnitude of gain and loss than by their associated likelihoods. Plotting utility curves of the subjective interpretation of probability against rational estimates of probability will not produce useful predictions of risk preference behavior. A different curve would have to be plotted for each and every relevant value of gain and loss.

Conclusion

Koenig's rover operations example is basically a straightforward exercise in engineering performance comparison. Since there is no apparent limit to the resource, and no penalty for its use, there is no reason to select any option other than the one that provides the lowest expected total consumption. The systematic construction and incorporation of intuition or feelings of luck would provide no advantage, and would most likely result in less than maximum productivity.

The characterization of risk preference and risk-aversion as "optimism" and "pessimism" may be useful in predicting the irrational behavior of decision makers, but there is no useful purpose in incorporating irrational predilections where the goal is to maximize gain and minimize loss. It is understandable that a decision-maker would be risk averse, but that trait can and should be reflected through the full and honest assessment of option probabilities and possible outcomes (both positive and negative) and the application of decision analysis methods.

Can Multiple Learning Agents Implement Responsiveness, Conflict Resolution, and Collaboration Between Autonomous Scheduling Systems and Human Operators?

○ **Keiki Takadama**
ATR HIP Research Labs.
2–2 Hikaridai, Seika–cho, Soraku–gun
Kyoto 619–0288 Japan
keiki@hip.atr.co.jp

Shinichi Nakasuka
Univ. of Tokyo
7–3–1 Bunkyo–ku
Tokyo 113–8656 Japan
nakasuka@space.t.u–tokyo.ac.jp

Katsunori Shimohara
ATR HIP Research Labs.
2–2 Hikaridai, Seika–cho, Soraku–gun
Kyoto 619–0288 Japan
katsu@hip.atr.co.jp

Abstract

This paper focuses on multiple learning agents in crew task scheduling problems and explores their capabilities of responsiveness, conflict resolution, and collaboration in interactions between autonomous scheduling systems and human operators. A careful investigation of these capabilities has revealed the following implications: (1) multiple learning agents provide responsiveness that enables them to quickly modify their schedules in cases of unexpected anomalies; (2) when human operators change the conditions of jobs in the schedule, conflicts are smoothly resolved through interactions among agents; (3) human operators can collaborate with agents by introducing preliminary or rough schedules into autonomous scheduling systems.

Keywords: multiple learning agents, crew tasks scheduling, responsiveness, conflict resolution, collaboration

Introduction

Recently, the importance of autonomy is recognized among mission technologists and scientists to address unpredictable and complex situations in space. Based on this understanding, several autonomous systems such as Hubble Telescope (Muscettola 93) have been successfully implemented in actual missions. This kind of system contributes to closed-loop control situations where experiments or other work can be completed without human operations. However, many systems still require human support especially those in scheduling domains. For example, an appropriate decision regarding a schedule change is needed when the system meets unexpected situations. In such a case, the relationship between autonomous scheduling systems and human operators is important for overcoming trouble.

However, how can we implement appropriate autonomous systems for the above case? We do not have a good answer to this question. To address this issue, we start by focusing on multiple learning agents in autonomous scheduling systems and explore their possibility through an evaluation of the following three capabilities: (1) responsiveness, (2) conflict resolution, and (3) collaboration, all of which are important aspects in an interaction between autonomous scheduling systems and human operators. In particular, the first capability is required in the case of anomalies to quickly provide new acceptable schedules and minimize the time loss. The second capability is needed to smoothly modify schedules according to mission changes or other schedule change requirements. Finally, the third capability is required to undertake the partially completed work of human operators and help them to reduce their burdens.

This paper is organized as follows. The next section describes a multiagent model, and then shows the crew task schedule on a space shuttle or station. Simulations and discussions are made in the following sections. Finally, our conclusions are given in the last section.

Organizational-Learning Oriented Classifier System

An Organizational-learning oriented Classifier System (OCS) (Takadama 99a) is one of the multiagent models that introduces the concept of organizational learning (OL) (Argyris 78; Cohen 95) studied in organization and management science.*1 Since OCS is based on Learning Classifier Systems (LCSs) (Goldberg 89), OCS has similar mechanisms to those of LCSs, but the big difference*2 between OCS and LCSs is that (1) OCS has a multiagent learning architecture, and (2) OCS addresses the division of work in multiple agents, both of which are difficult in conventional approaches.

Agents

OCS is composed of many LCSs as shown in Fig. 1. In OCS, agents (jobs in this paper) are implemented by their own LCSs which are extended to introduce four kinds of learning mechanisms reinterpreted from OL.*1 In order to solve problems that cannot be solved at an individual level, agents divide given problems by acquiring their own appropriate *functions* through interaction among agents. According to this approach, the *aim* of the agents is defined as finding appropriate functions. Since these functions are acquired through the change in each agent's rule sets and through the change in the strength *3 of rules, a *function* is defined as a rule set in OCS.

*Paper submitted to the 2nd NASA International Workshop on Planning and Scheduling for Space

*1 A detailed introduction or reinterpretation of the concepts of OL is discussed in (Takadama 99a).

*2 A detailed difference is described in (Takadama 99b).

*3 Strength is defined as the worth or weight of rules.

Architecture

As shown in Fig. 1, each agent in OCS has the same architecture that includes the following components.

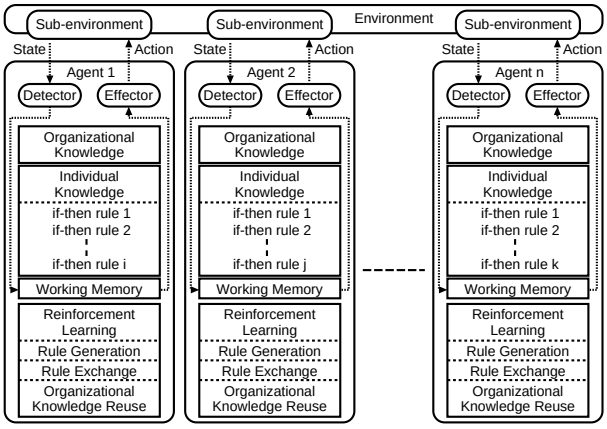


Figure 1: **OCS architecture**

< **Problem Solver** >

- **Detector** and **Effector** translate a part of an environment state into the internal state of an agent and derive actions based on the internal state (Russell 95), respectively.

< **Memory** >

- **Organizational knowledge memory** stores a set comprising each agent's rule set as organizational knowledge. In OCS, this knowledge is shared by all agents. Furthermore, organizational knowledge represents knowledge on the division of work because each agent's rule set derives the role of each agent.
- **Individual knowledge memory** stores a rule set (a set of *if-then* rules including a *strength* factor) as individual knowledge. In OCS, agents independently store different rules.
- **Working memory** stores the recognition results of sub-environmental states and also stores the internal state of an action of fired rules.

< **Mechanisms** >

- **Reinforcement learning, rule generation, rule exchange, and organizational knowledge reuse mechanisms** are reinterpreted from the four kinds of learning in OL [*4].

Learning in OCS

(1) Reinforcement learning mechanism: In OCS, the reinforcement learning (RL) mechanism enables agents to acquire their own appropriate actions required to solve given problems. In particular, RL supports the learning of the appropriate order of fired rules by changing the strength of the rules. Specifically, OCS employs a *profit sharing* method (Grefenstette 88) that reinforces a sequence of all rules when agents complete given tasks.

[*4] Details are described in (Takadama 99a).

(2) Rule generation mechanism: The rule generation mechanism in OCS creates new rules when none of the stored rules matches the current environmental state. In particular, when the number of rules is `MAX_RULE` (the maximum number of rules), the rule with the lowest strength is removed and a new rule is generated. In the process of rule generation, the condition (if) part of a rule is created to reflect the current situation, the action (then) part is determined at random, and the strength value of the rule is set to the initial value.

(3) Rule exchange mechanism: In OCS, agents exchange rules with other agents at a particular time interval (`RULE_EXCHANGE_STEP`[*5]) in order to acquire effective rules that cannot be acquired by agents themselves. In this mechanism, a particular number ((the number of rules)$\times$`GENERATION_GAP`[*6]) of rules with low strength values are replaced by rules with high strength values between two arbitrary agents. However, rules that have a higher strength value than a particular value (`BORDER_ST`) are not replaced to avoid unnecessary operations that increase communication costs. The strength value of replaced rules are reset to their initial values.

(4) Organizational knowledge reuse mechanism: Finally, agents in OCS store a set comprising each agent's rule set (individual knowledge) as knowledge on the division of work when they most effectively solve given problems,[*7] and agents reuse this knowledge when they solve other problems. For example, when n agents most effectively solve problems, a set comprising each agent's rule set is stored as {RS (1), RS (2), $\cdots$, RS (n)}, where RS(x) is the rule set for the x-th agent. In OCS, this set is called organizational knowledge[*8] and is updated through problem solving.

Supplemental setup

In addition to the above mechanisms, a particular number (`FIRST_RULE`) of rules in each agent is generated at random in advance, and the strength values of all rules are set to the same initial value.

Crew Task Scheduling

Problem Description

The crew task scheduling of a space shuttle or station is a job-shop scheduling problem where many jobs for

[*5] This step is defined later.

[*6] The ratio of operated rules.

[*7] Although it is difficult to generally define efficiency, agents, for example, recognize that they solve a given problem most effectively by measuring a "good solution" or a "small computational cost."

[*8] Note that agents cannot use both individual and organizational knowledge at the same time because the latter knowledge can be utilized by all agents as learned rules.

the crews must be scheduled under hard resource constraints. The goal of this problem is to find feasible schedules that minimize the total schedule execution (TSE) time of all jobs. We selected this domain because (1) this problem can be considered as a multiagent problem when one job is assumed to be one agent, and (2) a systemization of this problem is required to support schedulers at ground stations. In this task, there are several missions composed of jobs, and these jobs should be assigned while satisfying the following constraints to accomplish the missions.

1. **Power of space shuttle or station:** Each job requires a particular power (from 0% to 100%) in the experiments, but the summation of the power of all jobs at any time must not exceed 100%.
2. **Link to the ground station:** Some jobs require a link to the ground station, but only one job at a time can use it. Due to the orbit of the spacecraft, none of the jobs can use the link during certain times.
3. **Machine A:** Some jobs need to use machine A in the experiments, but only one at a time job can use it. Some examples of machines are computers, voice recorders, and so on.
4. **Machine B:** This is the same constraint as that for machine A.
5. **Execution order of jobs:** In a mission unit, jobs have an execution order, but some jobs in a mission may be only partially ordered, which means that some jobs may have the same order.[*9] In comparison with jobs, there is no order/priority among missions, and thus jobs in a certain mission are scheduled with jobs in other missions considering their respective execution orders.
6. **Crew assignment types:** The crew is divided into the following two types: Mission Specialist (MS) and Payload Specialist (PS). The former is mainly in charge of experiments, and the latter supports experiments!%In a unit of a job, one of the following crew assignment types is set: (a) Anybody, (b) PS only (but the concrete crew is not specified), (c) One specified PS with somebody, (d) One specified MS with somebody, and (e) Combination of PS and MS (but the concrete crews are not specified). These crew assignments are based on actual space shuttle missions.

In addition to the above six constraints, "the length" and "the required number of crew members" for each job are also decided beforehand.

Problem Setting

In the task, each job is designed as an agent in OCS, and each job learns to acquire an appropriate sequence of actions that minimizes the TSE time. Specifically, jobs can only observe local situations in the range of the length of their jobs[*10] except for the TSE time calculated from a location of an agent that is set at the latest time in the schedule.[*11] Based on this local observation, the rules of each job are designed to only consider local information including jobs' own primitive actions. Thus, only related (neighbor) agents can recognize the change made by another agent.

In a concrete problem-solving approch, all jobs are initially placed at random without considering overlaps or the six constraints described in the previous section. Due to this random placement, a schedule is not feasible at this time. After this initial placement, the jobs start to perform some primitive actions in order to reduce overlapping areas or to satisfy the constraints while minimizing the TSE time. When the value of the TSE time converges with a feasible schedule, all jobs evaluate their own sequences of actions according to the value of the TSE time. Then, the jobs restart from the initial placement to acquire more appropriate sequences of actions that find shorter times. In this cycle, one *step* is counted when all jobs perform one primitive action, and one *iteration* is counted when jobs restart from the initial placement.

Evaluation Criteria

The following two indexes are evaluated in the task, and the *performance* is defined as a criterion which considers the two indexes.

- Solution = $TSE\ time$
- Computational cost = $\sum_{i=1}^{n} step\ (i)$

The first index (*solution*) evaluates the execution time of a feasible schedule, and the second index (*computational cost*) calculates the accumulated steps. Especially in the latter equation, "*step* (i)" and "n", respectively, indicate the steps counted in i iterations and the number of iterations when the converged TSE time shows the same value in particular iterations.

Simulation

Experimental Design

A simulation investigates the abilities of multiple learning agents in the following three cases. All cases are tested with crew task schedules that involve 10 jobs in five missions.

- **Case 1: Responsiveness**
  This case addresses an anomaly situation where one crew member cannot perform experiments for a certain duration due to illness, and investigates the result after this anomaly using a comparison between

[*9] The order described here is most simple one. Other representations are required to handle partially order in general.

[*10] Examples of the local informations include information of an overlap, a power, a link, and so on.

[*11] The TSE time as global information is updated synchronously every after all agents perform one primitive action such as movements that satisfy power constraints.

the performance from the first placement of agents (jobs) and that from the current placement. When starting from the first placement, all agents are placed at their initial locations and then move their locations by considering the parts of the anomaly. When starting from the current placement, on the other hand, all agents move their locations from the current schedule that satisfies all constraints except for the parts of the anomaly. Specifically, only agents that do not satisfy constraints start by changing their locations in the schedule, and then other agents move their locations when their constraints are violated. In anomaly cases like that described above, new acceptable schedules must be obtained as quickly as possible. Therefore, both agents from the first and current placements utilize learned rules that are acquired before such situations in order to minimize the time loss.

- **Case 2: Conflict resolution**
  This case addresses a conflict situation caused by the change in the job's execution order from lower to higher, and investigates the result after such conflict using a comparison between the performance from the first placement of agents and that from the current placement. Since solutions may become worse due to rules that do not consider the conflict situations, agents from the first placement do not utilize learned rules while those from the current placement utilize learned rules acquired before the conflict situations.
- **Case 3: Collaboration**
  This case addresses a collaboration situation where human operators (the authors in this paper) introduce preliminary or rough schedules into agents, and investigates the result after such collaboration using a comparison between the performance of not using learned rules and that of using rules. Since a start placement is decided according to preliminary or rough schedules, both performance are calculated from the current placement of agents. Furthermore, agents have no learned rules in this case, and thus agents learn rules without collaboration and utilize them in the case of using learned rules.

In the above three cases, the learning mechanisms in agents process every iteration, which means the time when the TSE time converges. These mechanisms are turned off when the converged TSE time shows the same value in the particular iterations. Note that the TSE time is guaranteed to converge because in the end all agents are unable to find locations where the TSE time becomes small without overlapping as agents are placed at early times in the schedule.

Furthermore, in the case of not using the learned rules, agents start with randomly generated rules set and apply the learning procedure. In the case of using the learned rules, on the other hand, agents utilize the learned rules and apply the learning procedure. Note that the learned rules in the above three cases are not utilized on the same problem in the true sense but are utilized on a similar problem. The reasons for this are summarized as follows: (1) in the anomaly or conflict situations, the structure of the problem changes when an anomaly or a conflict occurs because the learned rules generated before anomaly or conflict situations do not consider such situations; (2) in the collaboration situation, on the other hand, the type of problem without collaboration is different from that with collaboration because the learned rules without considering collaboration do not always cover the situations of the collaboration.

Finally, since the learned rules are implemented by a set comprising each agent's rule set which includes only local information, these rules are assumed to be the organizational knowledge described in the section on OCS. In particular, considering the fact that organizational knowledge represents knowledge on the division of work, the learned rules can be applied to similar or other problems like those in this simulation.

Experimental Setup

Variables in OCS are set as follows: `FIRST_RULE` (the number of initial rules) is 25, `MAX_RULE` (the maximum number of rules) is 50, `RULE_EXCHANGE_STEP` (the interval steps for rule exchange operations) is 10, `GENERATION_GAP` (the percentage of operated rules) is 10%, and `BORDER_ST` (the lowest strength of the rule not for removal) is -50.0 [*12].

Experimental Results

Fig. 2 shows the solution (the TSE time) and the computational cost (the accumulated steps) in the crew task scheduling problem. In detail, Figs. 2 (a), (b) and (c) show the results of responsiveness, conflict resolution, and collaboration, respectively. In this figure, the left axis and the white box indicate the average of the "solution", and the right axis and the black box indicate the average of the "computational cost". Furthermore, all results are averaged from five different random seeds. The results suggest that (1) none of the solutions in the three figures change drastically even whether the agents start from the first or current placement or whether the agents utilize the learned rules or not; (2) all computational costs, on the contrary, become small when the agents start from the current location or utilize the learned rules in comparison with the computational costs of others.

Discussion

(1) Why Multiple Agents?

Figs. 2 (a) and (b) show that the computational costs become small while the solution is kept at the same level when the agents start from the current placement after anomaly or conflict situations. This is because

[*12] Note that the tendency in results does not drastically change according to the parameter setting.

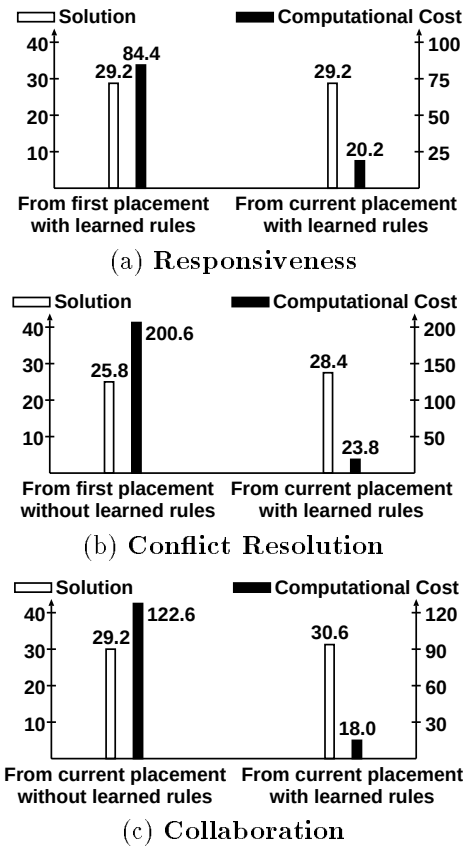


Figure 2: **Solutions and computational costs**

multiagent systems have the potential to be robust in dynamical environment changes caused by anomalies or conflicts. Specifically, multiple agents tend to explore a solution space according to independent decisions among agents, and this exploration contributes to a weakening or absorbance of an influence from anomalies or conflicts. This means that the multiagent environment frequently changes due to changes made by other agents, and thus agents simply address anomaly or conflict situations by considering them as one kind of environmental change. However, solutions never converge just by exploring a solution space. Therefore, agents in OCS decide their actions according to the local information as a main activity, but manage to converge by sharing only one global information (*i.e.*, the TSE time).*[13] Based on this implementation, agents explore a solution space until the TSE time converges. Thus, the solution level is maintained even if the agents start from the current placement, and this start contributes to a reduction in the computational costs by preventing from large modifications like those from the first placement.

Furthermore, agents in OCS have no way of knowing what the other agents are doing due to a multiagent approach. This seems that agents may not satisfy their constraints. However, agents find their appropriate places that satisfy their constraints, by checking whether their constraints are satisfied or not through local information.

(2) Why Learning Agents?

Figs. 2 (b) and (c) show that the computational costs become small while the solution is kept at the same level when the agents utilize the learned rules. This is because the learned rules acquired before conflicts or acquired without considering collaborations work as factors of both *reactiveness* and *deliberation*. In detail, the if-then part in the rules links input conditions and output actions, and thus it works as a kind of reactive planning (Agre 87) that works as a factor of reactiveness. The strength part in the rules, on the other hand, creates a chain among rules by changing the worth of rules,*[14] and thus it works as a kind of classical planning (McDermott 78) that works as a factor of deliberation. From these factors, it is quite important for the agents to acquire both indispensable if-then combinations and an appropriate rule chain in order to reduce the computational costs while keeping the solution at the same level.

(3) Effectiveness of Multiple Learning Agents

From the above discussions, we arrive at the conclusion that both *multiple* and *learning* are important aspects for agents to effectively address (a) responsiveness, (b) conflict resolution, and (c) collaboration, all of which are done through an interaction between autonomous scheduling systems and human operators. Although this result does not cover all practical cases, our previous research have found that multiple learning agents in OCS is also effective in other domains and large scale problems (Takadama 99a; Takadama 99b). Here, this section specifies the advantage of multiple learning agents in more detail by investigating the relationships among the (a), (b), and (c) cases as shown in Fig. 3.

- **Applicable range of multiple learning agents:** Cases (a), (b) and (c) compare different areas, respectively. For example, the responsiveness case compares the result from the first placement with that from the current placement, while the collaboration case compares the result of not using the learned rules with that of using the rules. These different focuses indicate the wide applicable range of multiple learning agents. In particular, unexpected situations often break predetermined schedules in space domains, and thus a wide applicable range has the potential to cope with such situations.

*[13] Other scheduling metrics that involve global calculations can be also handled by OCS like in the TSE time.

*[14] A rule with a high strength value is included in a rule chain, while a rule with a low strength value is excluded in the chain.

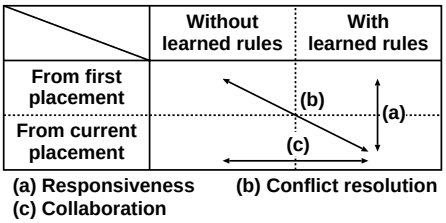


Figure 3: **Relationship among three cases**

- **Another axes of reactiveness and deliberation:** As described in the previous section, both reactiveness and deliberation are key factors to address in the cases (a), (b) and (c). However, this axis of reactiveness and deliberation is considered only from the viewpoint of the learning aspect. Here, we add another two axes of reactiveness and deliberation as follows: (1) a decision whether the agents start their locations from the current or the first placement, and (2) a decision whether the agents utilize the learned rules or not. These are because the agents from the current placement or the agents utilizing the learned rules contribute to finding feasible schedules quickly, while the agents from the first placement or the agents without utilizing the learned rules contribute to finding better schedules by exploring a wide range of search space. From this discussion, the three axes of reactiveness and deliberation are embedded in multiple learning agents. Considering the fact that these axes are tradeoff relationships and that they affect by each other, it is generally difficult to implement appropriate systems due to a lot of unpredictable factors. However, the results in this simulation suggests that agents staring from the current placement with the learned rules are robust in such tradeoffs.

(4) Possibility of Multiple Learning Agents

The above advantage of multiple learning agent approaches contributes not only to scheduling domains but also to multiple satellites or multiple space robots to address more complex and difficult tasks. In these cases, each satellite or robot is assumed as one agent. Furthermore, cooperation among many schedules can be implemented by assuming one schedule as one agent. This kind of cooperation is quite important in an international space station where a lot of schedules are performed asynchronously.

As another possibility, multiple learning agents have the potential to provide effective heuristic knowledge by investigating the trace of agent's movement when good schedules are found with small computational costs. An example of heuristic knowledge shows that jobs which require a long time must be placed first. Note that this kind of heuristic knowledge cannot be found from a global viewpoint but from a local one. This means that we must investigate what kind of rules each agent learn. As a result, the acquisition of heuristic knowledge provides the operators with explanations of the system's behavior.

Conclusion

This paper focused on multiple learning agents in crew task scheduling problems and explored their capabilities of responsiveness, conflict resolution, and collaboration in interactions between autonomous scheduling systems and human operators. The main results are summarized as follows: (1) multiple learning agents provide responsiveness that enables them to quickly modify their schedules in cases of unexpected anomalies; (2) when human operators change the conditions of jobs in the schedule, conflicts are smoothly resolved through interactions among agents; (3) human operators can collaborate with agents by introducing preliminary or rough schedules into autonomous scheduling systems.

Future research will include a further investigation of "understandability" for human operators. A comparison with benchmarks or conventional methods that involve the scheduling theory (Brucker 95) must be made. Furthermore, an evaluation in realistic scenarios is also needed in future work.

References

P.E. Agre and D. Chapman (1987). "Pengi: A implementation of a Theory of Activity," *The 6th National Conference on Artificial Intelligence (AAAI'87)*, pp. 268–272.

C. Argyris and D.A. Schön (1978). *Organizational Learning*, Addison-Wesley.

P. Brucker (1995). *Scheduling Algorithm*, Springer-Verlag.

M.D. Cohen and L.S. Sproull (1995). *Organizational Learning*, SAGE Publications.

D.E. Goldberg (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley.

J.J. Grefenstette (1988). "Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms," *Machine Learning*, Vol. 3. pp. 225–245.

D. McDermott (1978). "Planning and Action," *Cognitive Science*, Vol. 2, pp. 71–110.

N. Muscettola and S.F. Smith (1993). "Constraint-Directed Integration of Scheduling and Planning for Space-Based Observatory Management", *The 7th Annual Space Operations, Applications and Research Symposium (SOAR-93)*.

S.J. Russell and P. Norving (1995). *Artificial Intelligence: A Modern Approach*, Prentice-Hall International.

K. Takadama et al. (1999a). "Making Organizational Learning Operational: Implications from Learning Classifier System," *Computational and Mathematical Organization Theory (CMOT)*, Kluwer Academic Publishers, Vol. 5, No. 3, pp. 229–252.

K. Takadama et al. (1999b). "Can Multiagents Learn in Organization? ∼ Analyzing Organizational-Learning Oriented Classifier System ∼", *The 16th International Joint Conference on Artificial Intelligence (IJCAI'99) Workshop on Agents Learning About, From and With other Agents*.

Commentary on
"Can Multiple Learning Agents Implement Responsiveness, Conflict Resolution, and Collaboration Between Autonomous Scheduling Systems and Human Operators''

Reid Simmons
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
reids@cs.cmu.edu

The paper by Takadama, Nakasuka and Shimohara presents an architecture where multiple agents collaborate in solving scheduling problems. The agents learn scheduling rules and loosely collaborate by exchanging local information and (at times) learned rules. Experiments in the paper demonstrate that the learned rule sets help in solving similar problems, especially when changes occur, such as reordering of tasks or anomalies in the environment.

Scheduling is a difficult problem. Human schedulers need all the help they can get from automated systems. While early research focused on knowledge-intensive approaches [2, 4], researchers have obtained surprisingly good results with simpler, more local methods. In particular, the approach of starting with an infeasible (often random) plan and iteratively making changes in it using simple decisions has proven to be quite efficient and produces remarkably good results [5, 6, 8, 10]. Work in scheduling has a long history of applying other stochastic search methods to the scheduling problem, such as Genetic Algorithms (GA) [3, 9], simulated annealing [10] and randomization and restarting [1, 7], all with good success.

It is within this context that I view the current work. The process described in the paper of randomly generating rules and exchanging rules between agents strikes me as similar to a GA-type approach. The distinctions, though, are several. Foremost, in most GA-type approaches to scheduling, the individuals in a population are complete *schedules*. In this architecture, each agent (individual) is a single *job*. Thus, the agents must exchange sufficient information to enable them to avoid inconsistent schedules. Apparently, the authors limit this information to 1) the latest time of all jobs (which provides an upper bound on the total length of the schedule) and 2) overlaps with other jobs, in terms of requests for power, communication links, machines and crew.

Another distinction is how rules are created. As in GA, at set points, agents exchange their low-scoring rules with other agents and/or randomly generate new rules. However, in the architecture described in the Takadama paper, agents use reinforcement learning to learn rules that help decide how to schedule the jobs, given the available information from other agents and overall task constraints (such as ordering constraints).

It is surprising to me that this architecture performs as well as it does (although it would have been nice to see comparisons with other iterative-repair type scheduling algorithms). In particular, since agents have no way of knowing what the other agents may be doing, it would seem that this architecture could lead to thrashing. For instance, if two jobs over-subscribe for a given resource, it seems that there is nothing to prevent the two agents from changing their allocations in a way that happens to still be inconsistent. With more global information (or more synchronization between agents) this would not happen.

Similarly, it is not clear how the communication requirements between agents affect the efficiency of the scheduling task. In approaches where each agent is a complete schedule, the agents can solve their problems independently; Here, they must communicate. If there is a lot of potential for resource contention among tasks, there will be a lot of communication amongst agents. It would seem that at some point this communication would outweigh the multi-agent advantages of the approach. In such cases, a more reasonable decomposition might be to have each agent be a separate *resource*, and have each resource agent responsible for managing its own schedule by trading tasks with other resource agents.

Finally, it is not clear how this approach could handle other scheduling metrics, such as minimizing overall resource utilization, since that would involve global calculations. It would seem that, in such situations, either all the agents would need to collect, and process, this information, or else

one would need a separate agent that is responsible for maintaining the global state of the system.

The learning aspects of the paper seem strong. In particular, the combination of randomization (generation and exchange of rules) and task-directed learning of rules (using reinforcement learning) is intriguing. This seems to combine the best of genetic-mutation type approaches and more knowledge-based approaches. I think this is a generally useful technique, and could probably be fruitfully applied to other GA-type approaches. I would really like to see more rigorous experiments comparing this hybrid learning approach with the individual methods alone. My intuition says that the hybrid approach will show distinct advantages, but this really needs to be demonstrated.

In summary, I am intrigued by this approach. However, it must be viewed in the context of a very large body of work in scheduling that uses stochastic search and GA-type approaches. In particular, the current approach needs to be evaluated in realistic scenarios and must be compared to benchmarks. While I have doubts about the ability of the "one agent per job" approach to scale and perform well in more complex problems, I think that the hybrid learning approach could prove to be quite useful. I look forward to more detailed and rigorous studies by the authors.

References

[1] A. Cesta, A. Oddi and S.F. Smith. "An Iterative Sampling Procedure for Resource Constrained Project Scheduling with Time Windows". In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, Stockholm Sweden, August 1999.

[2] C. Cheng and S.F. Smith. "Applying Constraint Satisfaction Techniques to Job Shop Scheduling". *Annals of Operations Research, Special Volume on Scheduling: Theory and Practice*, 1997.

[3] H-L Fang, P.M.Ross and D.Corne, "A Promising Genetic Algorithm Approach to Job-Shop Scheduling, Rescheduling and Open-Shop Scheduling Problems". In *Proceedings of Fifth International Conference on Genetic Algorithms*, pp. 375-382, Morgan Kaufmann, 1993.

[4] M. Fox. "ISIS: A Retrospective". In *Intelligent Scheduling*, eds. Zweben and Fox, Morgan Kaufmann, 1994.

[5] S. Minton, M. Johnston, A. Philips, P. Laird. "Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems". *Artificial Intelligence* **58**:161-205, 1988.

[6] N. Muscettola, S.F. Smith, A. Cesta and D. D'Aloisi. "Coordinating Space Telescope Operations in an Integrated Planning and Scheduling Framework". *IEEE Control Systems*, **12(1)**, February 1992.

[7] A. Oddi and S.F. Smith. "Stochastic Procedures for Generating Feasible Schedules. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, Providence RI, July 1997.

[8] G. Rabideau, S. Chien, J. Willis, T. Mann. "Interactive, Repair-Based Planning and Scheduling for Shuttle Payload Operations". In *Proceedings of the 1999 Conference on Innovative Applications of Artificial Intelligence (IAAI)*, Orlando FL, July 1999.

[9] C.R.Reeves. "A Genetic Algorithm Approach to Stochastic Flowshop Sequencing". In *Proceedings IEE Colloquium on Genetic Algorithms for Control and Systems Engineering*, Digest No.1992/106, 1992.

[10] M. Zweben, B. Baun, E. Davis and M. Deale. "Scheduling and Rescheduling with Iterative Repair". In *Intelligent Scheduling*, eds. Zweben and Fox, Morgan Kaufmann, 1994.

Optimizing Resource Utilization in Planetary Rovers

Shlomo Zilberstein
Computer Science Department
University of Massachusetts
Amherst, MA 01003 U.S.A.
zilberstein@cs.umass.edu

Abdel-Illah Mouaddib
CRIL-IUT de Lens-Université d'Artois
Rue de l'université, S. P. 16
62307 Lens Cedex France
mouaddib@cril.univ-artois.fr

Abstract

Autonomous planetary rovers must operate under tight constraints over such resources as operation time, power, storage capacity, and communication bandwidth. To maximize scientific return, the rover is given multiple methods in which to accomplish each step of a plan. The different alternatives offer a tradeoff between resource consumption and the quality of the outcome. We show how to choose the best way to execute a task based on the availability of resources, the progress made with the task so far, and the remaining workload. Each task is controlled by a precompiled policy that factors the effect of the remaining plan using the notion of an opportunity cost.

Introduction

This paper is concerned with the design of a reactive meta-level controller that can optimize the operation of autonomous planetary rovers. Such rovers operate under tight resource constraints such as power, storage capacity, and communication bandwidth (Bresina *et al.* 1999; Washington *et al.* 1999). The time available to carry out experiments is limited as is the overall lifespan of the rover. The amount of power that is available (between recharges) is limited and must be carefully managed. Storage capacity to be used for raw data and processed data before transmission to a control center is also limited. Some of the resources are renewable: batteries could be recharged and storage space could be freed once the data is transmitted. Our focus in this paper is on optimal management of multiple resources between periods in which they can be renewed. The solution takes into account the high level of uncertainty regarding the consumption of resources by the rover's activities. For example, there is uncertainty about the amount of power and time required to bring the rover to a certain location, and there is uncertainty about the amount of storage that will be needed for a sequence of compressed images.

The combination of scarce resources and a high level of uncertainty present a complex meta-level control problem. The question is how to decide quickly during execution time which tasks should be executed and how to revise these decisions based on the actual progress being made, the availability of resources, and the remaining workload. Our approach to this problem is based on mapping each primitive activity (such as navigation, taking pictures, conducting experiments, or on-board data analysis) into a progressive processing task structure that specifies alternative ways to accomplish each aspect of the activity. For example, Figure 1 illustrates a possible task structure for taking a picture of a certain object. First the object must be located. Then, the rover may approach the object (either getting within an acceptable distance or an optimal distance for picture taking) and aim the camera. The picture can be taken at any one of several resolutions and then compressed at different levels of compression. The choices made along the execution of the task will affect the level of resource consumption as well as the quality of the outcome.

Progressive processing task structures make it possible for a system to trade off between resource consumption and quality of result (Mouaddib 1993; Mouaddib and Zilberstein 1997; 1998). The framework described in this paper is based on a similar approach we have developed for dynamic composition of information retrieval techniques (Zilberstein and Mouaddib 1999). The control of autonomous planetary rovers, however, has several different characteristics. First, we have multiple resources to monitor rather than one (time is the only resource monitored in previous work). Second, the reward structure depends on the ability to maximize scientific return with limited resources, but minimizing resource consumption in itself is not an explicit goal (minimizing response time is an explicit goal in the information retrieval application). Finally, unlike the information retrieval application, the set of tasks to be performed over a given time period is relatively stable (in previous work we considered a dynamic set of tasks with frequent updates). The specific characteristics of the rover control problem raise several fundamental challenges.

1. Handling *multiple* resources rather than execution time only.
2. Handling the *dependency* of quality and resource consumption on the intermediate quality or state.
3. Handling a *flexible task structure* in which some levels include several alternatives or optional steps.
4. Selecting the "best" set of methods in a dynamic en-

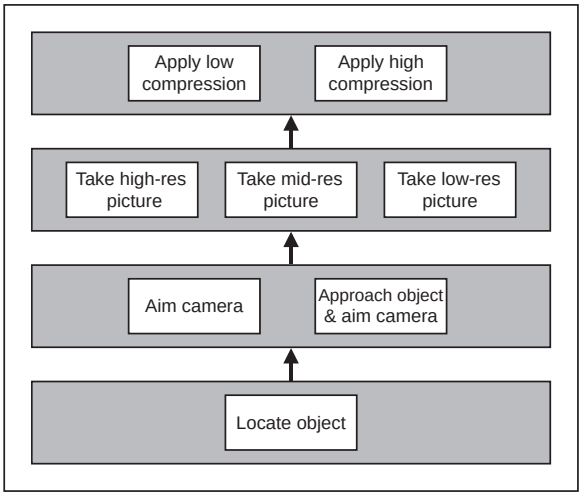


Figure 1: Illustration of a progressive processing task for taking a picture of an object

vironment taking into account the progress already made, the availability of resources, and the entire plan.

There are a number of complementary research efforts designed to develop planning and execution architectures for spacecraft systems (Blackmon *et al.* 1999; Bresina *et al.* 1999; Estlin *et al.* 1999; Muscettola *et al.* 1998) and, more generally, for real-time autonomous systems (Bonasso *et al.* 1996; Musliner *et al.* 1993). The work described in this paper is designed to complement these efforts by developing a reactive approach to managing multiple resources under a high level of uncertainty.

The ability to dynamically adjust computational effort based on the availability of computational resources has been studied extensively by the AI community since the mid 1980's. These efforts have led to the development of a variety of techniques such as *anytime algorithms* (Dean and Boddy 1988; Zilberstein and Russell 1996), *design-to-time* (Gravey and Lesser 1993), *flexible computation* (Horvitz 1988), *imprecise computation* (Liu *et al.* 1991), and *progressive reasoning* (Mouaddib 1993; Mouaddib and Zilberstein 1997). This work extends this flexibility to task structures that include both computational actions and physical actions. Section 2 gives a formal definition of the problem. We then solve the problem in two steps. In Section 3, we develop an optimal solution for a single task, ignoring the fact that additional tasks are waiting for execution. Section 4 shows how to handle multiple tasks by factoring the effect of the remaining plan using the notion of an opportunity cost. In section 5 we address some open problems and current work that examines the effectiveness this approach. We conclude with a summary and brief discussion of related work.

The control problem

This section defines an enhanced form of progressive processing task structures and the corresponding meta-level control problem. Each plan assigned to a rover is mapped into a set of task structures defined as follows.

Definition 1 *A* **plan**, $\mathcal{P}$, *is composed of a set of* activities, *a set of ordering constraints among activities, and an initial resource allocation* r_0.

The overall plan is generated off-line at the control center by a mixed-initiative planning process that is beyond the scope of this paper. We focus on the reactive, on-board scheduling process only. Resources are represented as vectors of discrete units measuring the availability of each resource. We assume initially that the plan is totally ordered and discuss the generalization of the technique to partially-ordered plans in Section 5.

The rover can perform a certain number of (parameterized) activities each of which has a predefined task structure associated with it. For simplicity of the discussion, we avoid the extra identifier indicating the type of activity when we consider the control of a single activity. Each activity is associated with a progressive processing unit.

Definition 2 *A* **progressive processing unit** *(PRU) is composed of a sequence of processing levels,* $(l_1, l_2, \ldots, l_L)$.

Definition 3 *Each processing level,* l_i, *is composed of a set of* p_i **alternative modules**, $\{m_i^1, m_i^2, \ldots, m_i^{p_i}\}$.

Each module can perform the logical function of level l_i, but it has different computational characteristics defined below.

Definition 4 *The* **module descriptor**, $P_i^j((q', \Delta r)|q)$, *of module* m_i^j *is the probability distribution of output quality and resource consumption for a* given *input quality.*

The module descriptor specifies the probability that module m_i^j consumes Δr resources (a discrete vector specifying the number of units of each resource used by the module) and produces a result of quality q' when the quality of the previously executed module is q. Module descriptors are similar to *conditional performance profiles* of anytime algorithms (Zilberstein and Russell 1996).

When the system completes a task, it receives a reward that depends on the quality of the output.

Definition 5 *Each PRU has a* **reward function**, $U(q)$, *that measures the immediate reward for performing the activity with overall quality* q.

Rewards are cumulative over different activities.

Given a plan $\mathcal{P}$, a library of task structures that specify a PRU for each activity in the plan, the module descriptors of all the components of these PRUs, and corresponding reward functions for each activity, we define the following control problem.

Definition 6 *The* **reactive control problem** *is the problem of selecting a set of alternative modules so as to maximize the expected utility over a complete plan.*

The meta-level control is "reactive" in the sense that we assume that the module selection mechanism is very fast, largely based on off-line analysis of the problem.

Optimal control of a single activity

We begin with the problem of meta-level control of a single progressive processing unit corresponding to a single activity. This problem can be formulated as a Markov decision process (MDP) with states representing the current state of the activity. The state includes the current level of the PRU, the quality produced so far, and the remaining resources. The rewards are defined by the utility of the solution. The possible actions are to *execute* one of the modules of the next processing level. The transition model is defined by the descriptor of the module selected for execution.

Note that in certain situation it might be beneficial to skip the execution of a particular level or the complete activity. To allow that, we introduce a *dummy* module in each level that consumes no resources and produces zero quality. This guarantees that at least one module is executable in each level regardless of the availability of resources.

The rest of this section gives a formal definition of the MDP and the reactive controller produced by solving it.

State transition model

The execution of a single progressive processing unit, u, can be seen as an MDP with a finite set of states $\mathcal{S} = \{[l_i, q, r] | l_i \in u\}$ where $0 \leq i \leq L$ indicates the last executed level, $0 \leq q \leq 1$ is the quality produced by the last executed module, and r is the remaining resources.

When the system is in state $[l_i, q, r]$, one module of the i-th level has been executed. (The first level is $i = 1$; $i = 0$ is used to indicate the fact that no level has been executed.) The states $[l_L, 0, r]$ represent termination with no useful result and remaining resources r.

The initial state of the MDP is $[l_0, 0, r]$, where r is the available resources. The initial state indicates that the system is ready to start executing a module of the first level of the PRU. The terminal states are all the states of the form $[l_L, q, r]$. In every nonterminal state, the possible actions are $\mathbf{E}^j_{i+1}$ (execute the j-th module of the next level). To complete the transition model, the probabilistic outcome of these actions are defined as follows.

To simplify the presentation, we assume that a module is executable only when there are enough resources to cover the worst-case possibility. This can be relaxed if we add a mechanism to abort an action once it requires more resources than available.

The outcome of each action, $\mathbf{E}^j_{i+1}$, is probabilistic. Resource consumption and quality uncertainties define the new state.

$$Pr([l_{i+1}, q', r - \Delta r] \mid [l_i, q, r], \mathbf{E}^j_{i+1}) = P^j_{i+1}((q', \Delta r)|q) \tag{1}$$

Rewards and the value function

Rewards are determined by the given reward function applied to the final outcome. Note that no rewards are associated with intermediate results, although this could be easily incorporated into this model.

We now define a value function (expected reward-to-go) over all states. The value of terminal states is defined as follows.

$$V([l_L, q, r]) = U(q) \tag{2}$$

The value of nonterminal states of the MDP is defined as follows.

$$V([l_i, q, r]) = \max_j \sum_{q', \Delta r} P^j_{i+1}((q', \Delta r)|q)\ V([l_{i+1}, q', r - \Delta r]) \tag{3}$$

This defines a finite-horizon MDP, or equivalently, a state-space search problem that can be represented by a decision tree or AND/OR graph. It can be solved using standard dynamic programming or using a search algorithm such as AO*

Theorem 1 *Given a progressive processing unit u, an initial resource allocation r_0, and a reward function $U(q)$, the optimal policy for the corresponding MDP provides an optimal strategy to control u.*

Proof: Because there is a one-to-one correspondence between the reactive control problem and the MDP (including the fact that the PRU transition model satisfies the Markov assumption), and because of the optimality of the resulting policy, we conclude that it provides an optimal reactive strategy to control the execution of the given progressive processing unit. □

We note that the number of states of the MDP is bounded by the product of the number of levels L, the maximum number of alternative modules per level $\max_i p_i$, the number of discrete quality levels, and the number of possible resource vectors. While resource measures could vary over a large range, the size of the control policy can be reduced by using a coarse unit. Therefore, unit choice introduces a tradeoff between the size of the policy and its effectiveness.

We have implemented the policy construction algorithm for a problem that involves only one resource (time). Figure 2 shows the results we got with a task structure composed of 3 levels, with 5 modules per level (all levels included a dummy 6th module that allows the controller to skip that level). The five unit sizes in this case represent multiples of 1, 10, 20, 40, and 80 of the original quality and time units. The dark bars show the time to construct the policy (logarithmic scale measured in milliseconds). The light bars show the relative reduction in the expected value of the initial state with respect to the optimal value. (The reduction in

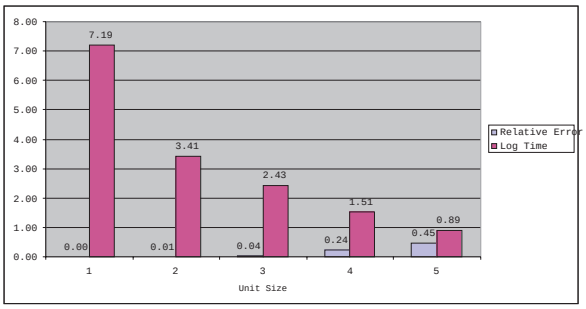


Figure 2: The effect of resource unit size on policy construction time and value

value is due to the fact that the process is modeled using coarse resolution and a compact policy.) It is interesting to note that it takes 15619 seconds to construct a precise policy with no error (left columns) while an approximate policy with a unit size of 20 (middle columns) takes only 0.27 seconds and the error is only 3.6%. These preliminary results are consistent with our intuition that the optimal policy can be approximated with a coarse resource unit and a compact policy.

Optimal Control of multiple PRUs using opportunity cost

Suppose now that we need to schedule the execution of a complete plan that includes $n+1$ activities. One approach is to construct an optimal schedule by generalizing the solution presented in the previous section. That is, one could construct a large MDP for the combined sequential decision problem including the entire set of $n+1$ PRUs. Each state must include an indicator of the activity (or PRU) number, i, leading to a general state represented as $[i, l, q, r]$.

This rather complex MDP is still a finite-horizon MDP with no loops. Moreover, the only possible transitions between different PRUs are from a terminal state of one PRU to an initial state of a succeeding PRU. Therefore, we can solve this MDP by computing an optimal policy for the *last* PRU for any level or resource availability r, then use the value of its initial states to compute an optimal policy for the previous PRU and so on.

Theorem 2 *Given a plan $\mathcal{P}$ represented as a sequence of progressive processing units, and a reward function $U_i(q)$ associated with each PRU, the optimal policy for the corresponding MDP provides an optimal strategy to control $\mathcal{P}$.*

This is an obvious generalization of Theorem 1. The complete proof, by induction on the number of PRUs, is omitted.

We now show how to measure the effect of the remaining n PRUs on the execution of the first one. This effect can be reformulated in a way that preserves optimality while suggesting an efficient approach to meta-level control that does not requires run-time construction of the policy.

Definition 7 *Let $V^*(i, r) = V([i, l_0, 0, r])$ denote the expected value of the optimal policy for the last $n-i+1$ PRUs.*

To compute the optimal policy for the i-th PRU, we can simply use the following modified reward function.

$$U'_i(q, r) = U_i(q) + V^*(i+1, r) \tag{4}$$

In other words, the reward for completing the i-th activity is the sum of the immediate reward and the reward-to-go for the remaining PRUs using the remaining resources. Therefore, the best policy for the first PRU can be calculated if we use the following reward function for final states:

$$U'_0(q, r) = U_0(q) + V^*(1, r) \tag{5}$$

Definition 8 *Let $OC(r, \Delta r) = V^*(1, r) - V^*(1, r - \Delta r)$ be the resource* **opportunity cost function**.

The opportunity cost measures the loss of expected value due to reduction of Δr in resource availability when starting to execute the last n PRUs.

Definition 9 *Let the* **OC-policy** *for the first PRU be the policy computed with the following reward function:*

$$U'_0(q, r) = U_0(q) - OC(r_0, r_0 - r)$$

The OC-policy is the policy computed by deducting from the actual reward for the first task the opportunity cost of the resources it consumed.

Theorem 3 *Controlling the first PRU using the OC-policy is optimal.*

Proof: From the definition of $OC(r, \Delta r)$ we get:

$$V^*(1, r_0 - \Delta r) = V^*(1, r_0) - OC(r_0, \Delta r) \tag{6}$$

To compute the optimal schedule we need to use the reward function defined in Equation 4 that can be rewritten as follows.

$$U'_0(q, r_0 - \Delta r) = U_0(q) + V^*(1, r_0) - OC(r_0, \Delta r) \tag{7}$$

Or, equivalently:

$$U'_0(q, r) = U_0(q) + V^*(1, r_0) - OC(r_0, r_0 - r) \tag{8}$$

But this reward function is the same as the one used to construct the OC-policy, except for the added constant $V^*(1, r_0)$. Because adding a constant to a reward function does not affect the policy, the conditions of Theorem 2 are met and the resulting policy is optimal. □

Theorem 3 suggests an optimal approach to scheduling an arbitrary set of $n+1$ activities by first using an OC-policy for the first PRU that takes into account the resource opportunity cost of the remaining n activities. Then the OC-policy for the second PRU is used taking into account the opportunity cost of the remaining $n-1$ activities and so on. To be able to implement this approach we need to have the control policies readily available. This issue is addressed in the following section.

Current work and open problems

In the previous section, we presented a solution to the control problem of multiple progressive processing units without accounting for its computational complexity. In particular, the opportunity cost must be computed and sometimes revised when the plan is modified during execution. Once the opportunity cost is revised, a new policy for the current PRU must be constructed. In principle, finding the exact opportunity cost requires the construction of an optimal policy for the entire plan. In this section we discuss current work aimed at reducing the complexity and enhancing the applicability of this framework.

Using precompiled control policies In order to implement an effective reactive controller for progressive processing, one should avoid reconstruction of control policies on-board or at the control center. Instead, we propose to:

1. use a fast approximation scheme to estimate the opportunity cost; and
2. use pre-compiled policies for different levels of opportunity cost.

We have examined several approaches to estimating the opportunity cost. Function approximation techniques seem to be suitable for learning the opportunity cost from samples of examples for which we can compute the exact cost off-line. In order to avoid computing a new policy (for a single PRU) each time the opportunity cost is revised, we can divide the space of opportunity cost into a small set of regions representing typical situations. For each region, an optimal policy would be computed off-line and stored in a library. At run-time, the system will first estimate the opportunity cost and then use the appropriate pre-compiled policy from the library. These policies remain valid as long as the overall task structure and the utility function are fixed.

Handling multiple resources As the number of resources grow, the size of each module descriptor and the overall policy also grow. This growth is exponential in the number of resources. Because we anticipate the number of resources in this application to remain small (two or three), the effect on computational complexity of policy construction is limited. Another source of complexity, however, is the approximation of the opportunity cost of multiple resources. In general, the OC function is not additive over resources. However, independence relationships between certain cost functions could simplify the approximation by allowing us to approximate separately each cost function.

More complex task structures The progressive processing task structures we have studied so far are rather limited. They are composed of a sequence of levels each with a set of alternative methods. We are examining several extensions including trees and directed acyclic graphs. Cycles in the task structure could be handled as well (making the control problem an infinite horizon MDP). Cycles could be used to represent multiple attempts to execute actions that fail.

Partially-ordered plans Another generalization of this work is to allow the components of a PRU or the overall plan to be partially ordered. The effect of this generalization on a single PRU is that the state of the MDP must include the "frontier" of the execution subgraph with several different modules being ready for execution. This is much more complex than the single point in sequence of levels we are currently using. This generalization will require some restrictions on how large the state space may grow (for example, by limiting the non-linearity of the plan to just a few choice points). Handling an overall plan that is partially ordered is not difficult, as long as the different activities remain independent.

Dependency among activities Right now we assume that different activities in a plan are independent, except for the fact that they share resources and contribute to the comprehensive value of the plan. Dependency among activities could be represented using additional state variables that capture the sources of dependency. Each action in this model will have possible stochastic effects on some state variables. Activities (or specific methods in the progressive processing task structure) could be conditioned on these state variables. The size of the state space grows exponentially with the number of additional state variables, making it hard to model highly dependent activities.

Conclusion

We present an approach to meta-level control of the activity of planetary rovers by mapping each activity into a progressive processing unit and formulating the control problem as an MDP. It is shown that an optimal policy for a plan composed of a sequence of activities can be constructed by controlling a single PRU at a time, taking into account the opportunity cost of the remaining tasks. To apply this model to control the operation of an autonomous rover, a fast approximation of the opportunity cost is needed. Finally, a highly reactive controller is described that uses a library of pre-compiled control policies to operate in a dynamic environment.

A less complex model of progressive processing that relies on heuristic scheduling has been studied in (Mouaddib and Zilberstein 1997). The task structure, however, is limited to a linear set of levels with one module per level and no quality uncertainty or quality dependency. The heuristic scheduler is fast, but it is not optimal. Heuristic scheduling of computational tasks has also been studied by Garvey and Lesser (1993) for the *design-to-time* problem-solving framework. The latter framework represents explicitly non-local interactions between sub-tasks.

The progressive processing framework relates to a large body of work within the systems community on *imprecise computation* (Liu *et al.* 1991). Each task

in that model is decomposed into a *mandatory* subtask and an *optional* subtask. A variety of scheduling algorithms have been developed for imprecise computation under different assumptions about the optional part. Our model allows for a richer representation of quality and resource uncertainty and quality dependency. Unlike imprecise computation, the schedule constructed by the MDP scheduler is a *conditional schedule*; the selection of modules is conditioned on the *actual* resource consumption and outcome of previous modules.

The application of dynamic programming to control interruptible anytime algorithms has been studied by Hansen and Zilberstein (1996). Optimal monitoring of progressive processing tasks using a corresponding MDP has been studied by Mouaddib and Zilberstein (1998) with respect to a simpler task structure and without the notion of quality uncertainty and quality dependency. The notion of *opportunity cost* is borrowed from economics. It has been used previously in meta-level reasoning by Russell and Wefald (1991). Horvitz (1997) uses a similar notion to develop a model of *continual computation* in which idle time is used to solve anticipated future problems.

The use of pre-compiled control policies to construct a highly reactive real-time system has been studied by several researchers. For example, Greenwald and Dean (1998) show how a real-time avionics control system can use a library of schedules that cover all possible situations. Each schedule is conditioned on the state of the flight operation.

Acknowledgments

We thank Richard Washington and John Bresina for introducing us to the domain of planetary rover control and for numerous helpful discussions. Andy Arnt implemented the policy construction algorithm. This work was supported in part by the National Science Foundation under grants IRI-9624992 and INT-9612092, by the GanymedeII Project of Plan Etat/Nord-Pas-De-Calais, and by IUT de Lens.

References

R.P Bonasso, D. Kortenkamp, D. Miller, and M. Slack. Experience with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical AI*, 1996.

T.T. Blackmon, L. Nguyen, C.F. Neveu, D. Smith, C. Anderson, and V. Gupta. Command generation for planetary rovers using virtual reality. *Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 1999.

J. Bresina, K. Golden, D. Smith, and R. Washington. Increased flexibility and robustness for Mars rovers. *Fifth International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, 1999.

T. Dean and M. Boddy. An analysis of time-dependent planning. *Seventh National Conference on Artificial Intelligence*, 49–54, 1988.

T. Estlin, A. Gray, T. Mann, G. Rabideau, R. Castano, S. Chien, and E. Mjolsness. An integrated system for multi-rover scientific exploration. *Sixteenth National Conference on Artificial Intelligence*, 541–548, 1999.

A. Garvey and V. Lesser. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1491–1502, 1993.

L. Greenwald and T. Dean. A conditional scheduling approach to designing real-time systems. *AI Planning Systems*, 1229–1234, 1998.

E.A. Hansen and S. Zilberstein. Monitoring the progress of anytime problem-solving. *Thirteenth National Conference on Artificial Intelligence*, 1229–1234, 1996.

E. Horvitz. Reasoning under varying and uncertain resource constraints. *Seventh National Conference on Artificial Intelligence*, 111–116, 1988.

E. Horvitz. Models of continual computation. *Fourteenth National Conference on Artificial Intelligence*, 286–293, 1997.

J. Liu, K. Lin, W. Shih, A. Yu, J. Chung, and W. Zao. Algorithms for scheduling imprecise computations. *IEEE Transactions on Computers*, 24(5):58–68, 1991.

A.I. Mouaddib. *Contribution au raisonnement progressif et temps réel dans un univers multi-agents.* PhD thesis, University of Nancy I, (in French), 1993.

A.I. Mouaddib and S. Zilberstein. Handling duration uncertainty in meta-level control of progressive reasoning. *Fifteenth International Joint Conference on Artificial Intelligence*, 1201–1206, 1997.

A.I. Mouaddib and S. Zilberstein. Optimal scheduling of dynamic progressive processing. *Thirteenth Biennial European Conference on Artificial Intelligence*, 449–503, 1998.

N. Muscettola, P.P. Nayak, B. Pell, and B.C. Williams. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2), 1998.

D. Musliner, E. Durfee, and K. Shin. Circa: A cooperative, intelligent, real-time control architecture. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6), 1993.

S. Russell and E. Wefald. *Do the Right Thing: Studies in Limited Rationality* MIT Press, 1991.

R. Washington, K. Golden, J. Bresina, D.E. Smith, C. Anderson, and T. Smith. Autonomous rovers for Mars exploration. *IEEE Aerospace Conference*, 1999.

S. Zilberstein and S. Russell. Optimal composition of real-time systems. *Artificial Intelligence* 82(1-2):181–213, 1996.

S. Zilberstein and A.I. Mouaddib. Reactive control of dynamic progressive processing. *Sixteenth International Joint Conference on Artificial Intelligence*, 1268–1273, 1999.

Commentry on:

Optimizing Resource Utilization in Planetary Rovers by Shlomo Zilberstein and Abdel-Illah Mouaddib

Abdullah Aljabri
Abdullah.S.Aljabri@jpl.nasa.gov

The reactive meta-level controller being developed by Zilberstein and Mouaddib is novel in borrowing techniques previously used on pure computational tasks to physical rover mission tasks. One of the methods key ideas is to apply cost/benefit metrics to every activity level to be executed on the path to accomplish a given mission goal. This is a potentially powerful method to assess alternate paths from which an optimum solution can be selected for overall mission effectiveness.

As noted by the authors the additional problems in applying this approach to space missions are:

- instead of a single metric for a given step there are multiple metrics that govern the cost/benefit of space mission activities,
- the determination of the metrics value is imprecise,
- activity levels may include several alternate steps and
- identifying the optimum solution in a dynamic complex system.

I recommend that the authors also consider two other important aspects of space missions:

- the time required by the method to both make predictions of the mission states and to decide what path to recommend, and
- the identified alternate steps are likely to be quite complex and invoke additional set of constraints and resource needs that must be satisfied before execution.

It is in the addressing of all the above problems that the potentials of their method can be reached. A prototyping approach using a segment of a rover mission scenario will both mature the method and showcase its potential benefits. This requires detailed models of the components of the system, the mission rules and constraints. Additionally a real-time simulation allowing closing the loop will reduce the uncertainties associated with the cost/benefit metrics.

From Rovers to Orbiters: Continuous Task-Distribution-Based Coordination

Anthony Barrett
Jet Propulsion Lab, M/S 126-347
California Institute of Technology
4800 Oak Grove Drive, Pasadena, CA 91109-8099
anthony.barrett@jpl.nasa.gov

Abstract

This paper compares and contrasts several coordination schemes for a system that continuously plans to control collections of rovers or spacecraft using collective mission goals instead of goals or command sequences for each spacecraft. A collection of self-commanding robotic systems would autonomously coordinate itself to satisfy high level science and engineering goals in a changing partially understood environment – making feasible the operation of tens or even a hundred spacecraft.

1. Introduction

While explicitly commanding a spacecraft via low level command sequences has worked spectacularly on previous NASA missions, there are limitations deriving from communications restrictions – scheduling time to communicate with a particular spacecraft involves competing with other projects due to the limited number of deep space network antennae. This implies that a spacecraft can spend a long time just waiting whenever a command sequence fails. This is one reason why the New Millennium program has an objective to migrate parts of mission control tasks onboard a spacecraft to reduce wait time by making spacecraft more robust [Muscettola *et al*. 1997]. The migrated software is called a "remote agent" and can be partitioned into 4 components:

- a mission manager to generate science goals,
- a planner/scheduler to turn goals into executable activities through reasoning about expected future situations,
- an executive/diagnostician to initiate and maintain activity execution while interpreting sensed events through reasoning about past and present situations, and
- a conventional reactive controller to interface with the spacecraft to implement an activity's execution.

In addition to needing remote planning and execution for isolated spacecraft, a trend toward multiple-spacecraft missions points to the need for remote distributed planning and execution. The past few years have seen missions with growing numbers of probes. Pathfinder has its rover (Sojourner), Cassini has its Huygens lander, and Cluster II is scheduled to launch in 2000 and has 4 spacecraft for multi-point magnetosphere plasma measurements. This trend is expected to continue to progressively larger fleets. For example, one proposed interferometer mission [Mettler& Milman 1996] would have 18 spacecraft flying in formation in order to detect earth-sized planets orbiting other stars. Another proposed mission involves 44 to 104 spacecraft in Earth orbit to measure global phenomena within the magnetosphere.

This paper compares and contrasts 3 ways to distribute a planner/scheduler amongst a population of spacecraft or rovers that have separate executive/diagnosticians and reactive controllers. The first places the planner/scheduler on a single platform that remotely commands the others. The second is more distributed in that it replicates a planner across the population to let each platform plan its own activities, but a single platform handles goal distribution. The last approach advertises all goals and lets each platform bid for a goal based on how well its local planner can satisfy the goal given local information. These approaches delineate a space of approaches where the platform that distributes tasks maintains progressively less information on the entire constellation.

This paper's sections subsequently describe 3 thought experiments for multi-platform missions that motivate 8 performance metrics for evaluating approaches toward continuous task-distribution-based coordination, compare and contrast 3 coordination methods, discuss related work, and finally conclude.

2. Multi-Platform Thought Experiments

In order to focus this discussion on distributed autonomy in space, consider different types of future multi-platform missions. There are 4 kinds of such missions depending on the reason for proposing multiple platforms:

- improved coverage when observing/exploring large areas (like the number of identical small satellites with scatterometers proposed for the Ocean Surface Wind Measurement Program (EOS-5));
- specialized probes with explicitly separate science objectives (like Cassini with Huygens and Pathfinder with Sojourner);

- multi-point in-situ sensing for observing large scale phenomena that are only detectable with multiple spatially separated in-situ sensors (like for observing global magnetospheric phenomena with spatially separated plasma sensors in the Magnetospheric Multi Scale or Cluster II missions); and
- building large synthetic aperture sensors with many small spatially separated sensors for imaging very remote targets (like Constellation-X, Terrestrial Planet Finder, and TechSat-21).

These reasons for having multiple platforms in a mission are not exclusive. For instance, the Air Force's TechSat-21 mission concept [Martin&Stallard 1999] involves a constellation of clusters of platforms. Each cluster forms a synthetic aperture for radar sensing, and the number of clusters depends on the desired global coverage.

2.1. Coordinating Task Distribution

In missions where each probe performs its task in isolation, the difference between an autonomous multi-platform mission and many autonomous single platform missions involves distributing tasks to the different platforms. While the task distribution for multiple autonomous single platform missions is determined on the ground, an autonomous multi-platform mission can distribute and redistribute tasks remotely. This feature improves both distribution quality and robustness by letting the spacecraft use local information to optimize the initial task distribution and to redistribute tasks when a spacecraft suffers an anomaly, unexpectedly finishes a task early, or detects an unanticipated science opportunity.

As an example of coordinated autonomous task distribution, consider multiple rovers surveying rocks in an area on Mars using MISUS [Estlin *et al.* 1999]. In this system a Mars lander manages a population of rovers by analyzing data from past observations, determining new observations, assigning observation goals to rovers, and collecting data as each rover moves from rock to rock and performs its experiments in isolation (fig. 1). This system autonomously maximizes science return while minimizing the execution time of the most heavily tasked rover.

While MISUS focused on a multi-rover scenario, much of the developed infrastructure applies to any multi-platform mission with a number of identical platforms that operate in isolation. This is includes most of the improved coverage and specialized probe classes of missions. As an example of this generalization, consider replacing the Mars lander and the rover population with a ground station and a constellation of RADARSATs like the one illustrated in figure 2. Currently the Canada Centre for Remote Sensing manages a single RADARSAT [CCRS 1998] in a sun-synchronous orbit. This satellite can observe any location around the poles on a daily basis and any location around the equator in 6 days or less. To decrease the equatorial delay time, consider replacing the single satellite with 6 equally spaced RADARSATs. The resultant constellation decreases the equatorial delay time to one day. A system like MISUS could manage this constellation since each of the 6 satellites operates independently of the others.

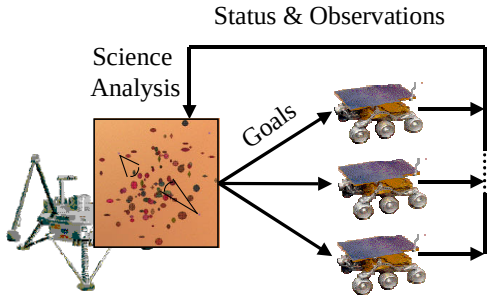


FIG 1: Coordinating multiple rovers with MISUS

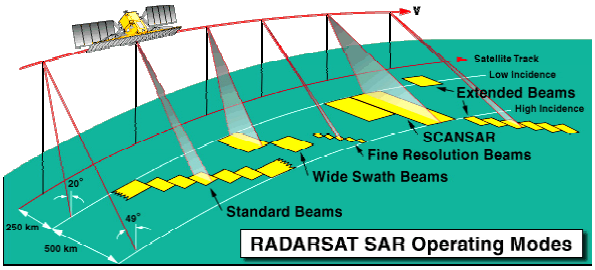


FIG 2: Operating modes for one of a constellation of radar satellites

2.2. Coordinating Task Execution

The multi-point in-situ sensing and large synthetic aperture missions differ operationally from the other 2 classes in that the separate spacecraft do not operate in isolation. For instance consider the Air Force's TechSat-21 mission concept (fig. 3). TechSat-21 involves a constellation of clusters of spacecraft. While each cluster functions in isolation, spacecraft in a cluster have tightly coordinated

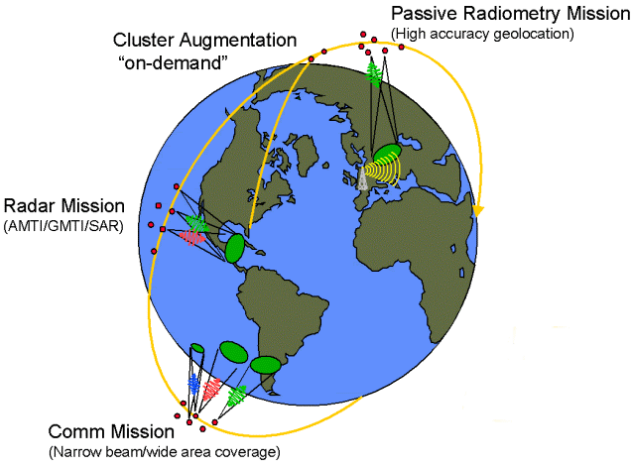


FIG 3: Operating modes for clusters in TechSat-21

activities. These activities include tight beam communications, synthetic aperture radar (SAR), and geolocation.

In many respects coordinated task execution is easier than coordinated task distribution. For the smaller missions designating a master spacecraft that commands the other (slave) spacecraft as though they were physically attached solves this problem, but bandwidth restrictions keep this approach from scale with either the number of slaves or the complexity of each slave. Resolving this scaling issue is outside this paper's focus.

2.3. Autonomy Architectures

In an earlier paper [Barrett 1999], I described 3 different autonomy architectures for a constellation of spacecraft involving leaders, followers, and slaves. Here I expand this taxonomy to also include contractors. The number of autonomy modules on a spacecraft determines which of the 4 classes it falls into:

- a *slave* has no modules and is tele-operated by the reactive control module of another nearby spacecraft;
- a *follower* has both an executive/diagnostician and a reactive controller (like many existing spacecraft);
- an *contractor* has a follower's components and a planner/scheduler to optimize local activities (like DS1's remote agent experiment); and
- a *leader* has all four components.

With these 4 classes, we can define a multi-platform mission's autonomy architecture by stating the class of each platform, and how the collection of platforms coordinate their activity. In terms of MISUS, the architecture consists of having the lander lead, and letting the rovers act as followers or contractors depending on the desired local autonomy.

Given a multi-platform mission, there are two sets of metrics for evaluating the acceptability of autonomy software. The first set motivates minimizing the amount of remote autonomy and has 4 metrics:

- the amount of explicit control an operator has over the constellation's activities,
- the feasible accuracy of modeling the constellation's activities on the ground,
- the autonomy software's testability, and
- the amount of needed onboard computing power.

While the first set of metrics tend to be maximized by limiting the amount of autonomy on a constellation, the second set of 4 evaluation metrics are maximized by increasing the amount or remote autonomy:

- the platforms' event response time,
- the required bandwidth between platforms and to Earth,
- the quality of the downlinked data, and
- the functional redundancy.

3. Coordinating Multiple Planners

In [Rabideau *et al*. 1999], others and I compared 3 methods for coordinating a population of rovers from a central lander in the MISUS scenario (figure 4). We used central planning to manage a population with a leader-follower architecture, where the leader generates plans that are subsequently executed by the followers. In order to assure each plan's correctness, the lander needs to acquire large amounts of state information on the rovers to appropriately determine if they can execute their plans.

Distributed planning reduces the amount of needed state information by using a goal distribution planner. This planner takes a subset of the rovers' collective state information with less precise models of the rovers, and it produces an abstract plan with enough detail to determine how to distribute the goal activities among the rovers. The lander then transmits goals to the appropriate rovers.

Another way to migrate planner/schedulers onto the

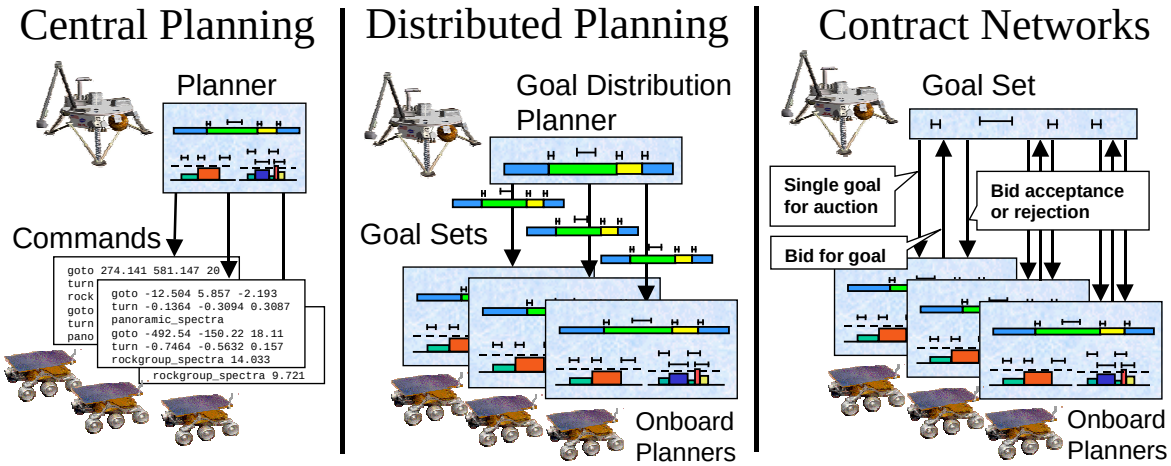


FIG 4: Approaches to task distribution based coordination

rovers uses a central auctioneer to distribute goals, and the rovers use onboard planners with local state information to determine appropriate bids for each goal as it arises. This approach is an instance of the *contract network protocol* [Smith 1980, Sandholm 1993] – a commonly used coordination paradigm within the distributed artificial intelligence community. Within a contract net protocol, a leader announces a task to a set of contractors, each contractor bids for it, and the leader awards the task to the contractor with the best bid.

3.1. Central Planning

The simplest way to extend single-platform spacecraft autonomy research to autonomous multi-platform missions involves using a master/slave approach where a single leader performs all autonomy reasoning. The slaves only transmit sensor values to the leader and forward control signals received from the leader's reactive controller to their appropriate local devices. In this way all spacecraft are treated as a single multi-platform spacecraft.

Both single and multiple platform autonomous spacecraft must respond to a (somewhat) dynamic, unpredictable environment. In terms of high-level, goal-oriented activity, a planner needs to modify spacecraft sequences to account for fortuitous events such as observations completing early and setbacks such as a failure to acquire a guide star for a science observation.

The need to rapidly respond to unexpected events motivates continuous planning, an approach where a planner continually updates a sequence in light of changing operating context. In such an operations mode, a planner would accept and respond to activity and state updates on a one to ten second time scale. CASPER [Chien *et al*. 1999] is an example of a continuous planner based on a heuristic iterative repair approach toward planning [Zweben *et al*. 1994, Fukunaga *et al*. 1997]. This approach takes a complete plan at some level of abstraction and manipulates its actions to repair detected flaws. Example flaws would involve an action being too abstract to execute or many simultaneous actions with conflicting resource needs.

Making a heuristic iterative repair planner continuous within a planner/scheduler module results in figure 5's algorithm. The first line assures that the PROJECTION variable always reflects how the state of a rover, or a spacecraft, should evolve as its plan executes, and the last line causes this execution by passing near-term activities to the executive/diagnostician.

The expected state evolution changes as a plan gets new goal activities and the perceived state diverges from expectations. This divergence is caused by unexpected exogenous events and activities having unexpected outcomes. Since a planning model can only approximate the reality experienced during execution, these unexpected state changes can always to happen.

At any moment the projection can detect flaws in a local plan, and lines 2 through 4 select and apply repair

Given: a PLAN with multiple activities
a PROJECTION of PLAN into the future

1. Revise PROJECTION using the currently perceived state and new goal activities from the mission manager.
2. Heuristically choose a plan flaw found in PROJECTION.
3. Heuristically choose a flaw repair method.
4. Use method to alter PLAN & PROJECTION.
5. Release relevant near-term activities in PLAN to the real-time system.
6. Go to 1.

FIG 5: Continuous planning using heuristic iterative repair

methods to fix these flaws. For instance, a satellite's observation activity can take an unexpectedly long time to complete. Depending on the delay, a subsequent observation may be impossible due to the target being too far behind the satellite when the observation starts. A repair method might fix the flaw by rescheduling the observation at a later time.

With respect to our evaluation metrics, using a continuous planner with a master/slave approach toward multi-platform coordination facilitates allowing a variable amount of remote autonomy. At one extreme the continuous planner is given low-level command sequences and can only apply a go-to-safe-mode repair method upon detecting a flaw. This extreme maximizes the first set of metrics. Another extreme reduces the first set of metrics while improving the second set. Here the planner is only given a set of abstract activities and uses local information and heuristics improve event response time and the quality of downlinked data. While functional redundancy and inter-platform bandwidth are unaffected by moving from one extreme to another, turning the slaves into followers increases redundancy and reduces bandwidth. Due to how easily this change can degrade the event response time, turning slaves into followers is an active research topic in the multi-agent research community [Tambe 1997].

3.2. Distributed Planning

Turning followers into contractors raises issues regarding how to coordinate multiple planners. In distributed planning, this coordination is achieved through using a continuous goal distribution planner on one platform, and this planner continuously manages the distribution of goals based on continuously updated partial information on the other platforms. For instance, the distribution planner might model rovers in a multi-rover scenario as points on a plane where each rover can travel in a straight line from one goal activity's observation target to another's.

With this abstract characterization, the distribution planning problem becomes a Multiple-Traveling Salesman

Problem (MTSP) [Johnson&McGeoch 1997] where the members of a sales team must collectively visit each of a set of cities and the maximum traveling time of the salesmen is minimized. While this is a NP-Complete problem, there are fast greedy approaches that find slightly sub-optimal solutions. By encoding one of these approaches into our distribution planner, the lander can both determine how to distribute the goal activities and provide a rough estimate on the order in which a rover should visit its targets to perform the goal activities.

With respect to our evaluation metrics, distributed planning facilitates variable autonomy both with the ground and across the platforms. Minimizing autonomy across platforms involves making the distribution planner use full information and generate low level action sequences for the other platforms, which can only execute their actions. This restriction turns distributed planning into the previously evaluated central planning approach.

Maximizing autonomy on the contractor platforms has the same effects as maximizing autonomy for the central planner, but also adds a reduction to inter-platform bandwidth needs. The lead platform no longer needs to maintain full state information, and each platform's planner can locally respond to events without informing the leader. Now a contractor can resolve a flaw by either quietly shuffling its local activities or reporting failure to the leader upon deleting a local activity. This quiet shuffle reduces bandwidth needs while failure reporting facilitates moving activities between platforms via the leader's continuously repairing its goal distribution plan.

3.3. Contract Networks

Minimizing the amount of continuously updated contractor information on the leader results in taking a contract network approach toward coordinating multiple planners. Here a leader advertises each goal and each contractor bids on the goals based on its local information. To respond to an unexpected event, a contractor will either quietly shuffle its activities or delete a local activity and report failure to the leader. Upon hearing of a failure, the leader can re-advertise the failed goal for auction. Notice that there is no need for continuously updated partial contractor information – the leader does not need to know anything about the contractors to auction a goal.

As shown in figure 4, using a contract net protocol to implement a greedy solution to the MTSP involves making the lander take goal activities and incrementally advertise them to all rovers. Upon receiving a task, a rover uses an onboard planner to try to fit a solution to the goal activity into its current schedule. Upon succeeding, a rover bids its total projected travel distance upon including the new observation. Rovers that fail to insert the task within a time limit do not participate in the auction. Upon receiving all bids, the lander awards the task to the rover with the smallest bid. By bidding the total distance the rovers minimize the maximum rover travel distance – an MTSP solution.

With respect to our evaluation metrics, letting an operator restrict the platforms that can bid for certain activities results in a system with variable autonomy. At one extreme the operator can specify a low level activity sequence for each platform, and at the other the leader gets a set of high level goals that can go to any platform.

As before, the first extreme scores best on the autonomy minimization metrics and the second scores best on the autonomy maximization metrics. While this approach has lower inter-platform bandwidth needs than the other approaches, it has more computational overhead and assumes a greedy approach toward optimization.

4. Related Work

While there is a large literature on cooperating robots, most focuses on behavioral approaches that do not explicitly reason about partitioning goals and planning courses of action. Three notable exceptions are GRAMMPS [Bumitt&Stentz 1998], MARS [Fischer *et al*. 1995], and RETSINA [Paolucci *et al*. 1999]. GRAMMPS is a system coordinating multiple mobile robots visiting locations in cluttered partially known environments. This system shares quite a bit similarity with our central goal allocation with distributed planning architecture for rovers. Both systems solve an MTSP problem to distribute targets, and both have low level planners on each mobile robot, but GRAMMPS focuses on path planning while learning a terrain instead of focussing on resources and exogenous events.

MARS on the other hand is a cooperative transportation scheduling system that shares many similarities with the contract net approach. Once again the differences involve a focus on multiple resources, exogenous events, and variable autonomy.

Finally RETSINA uses peer-to-peer coordination with an HTN planner for local planning. While the use of heuristic iterative repair here points to one difference between the approaches. The main difference involves RETSINA's not modeling known exogenous events and not providing default mechanisms for initially distributing goals and transferring goals to resolve execution failures.

5. Conclusions

This paper compared and contrasted 3 continuous task-distribution-based coordination schemes for commanding multiple platforms with collective goals instead of goals or command sequences for each platform: central planning, distributed planning, and contract networks. All schemes supported variable autonomy and were evaluated with respect to 8 different metrics. At the lowest autonomy setting, all schemes devolved into commanding the platforms with low level sequences, and at the highest autonomy setting the schemes differed primarily in terms

of needed onboard computing, inter-platform bandwidth, and redundancy. While central planning kept all computing on the leader, distributed planning spread the computing overhead across all platforms. The result was a decrease in inter-platform bandwidth needs and an increase in redundancy with an unchanging total computing overhead. Contract networks further improved the bandwidth needs and redundancy, but this scheme also increased the total computing overhead by letting each platform see and bid for each goal.

Reasoning about incremental autonomy for distributed planning and contract networks results in a realization that these approaches toward coordinating multiple planner/ schedulers can be combined. The resultant approach would used a goal distribution planner, but would only collect enough information to limit the number of platforms that participate in an auction. One avenue for future work involves building a coordination mechanism that spans the space between contract networks and distributed planning. Another future research avenue involves generating joint activities for multiple spacecraft/ rovers to collectively satisfy and would extend our approach to handle constellations of clusters of platforms (in TechSat-21). Finally, a third research direction involves making the rovers/orbiters compete for shared resources, like communications opportunities.

Acknowledgements

This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

[Barrett 1999] A. Barrett, "Autonomy Architectures for a Constellation of Spacecraft," International Symposium on Artificial Intelligence Robotics and Automation in Space (ISAIRAS), Noordwijk, The Netherlands, June 1999.

[Bumitt&Stentz 1998] B. L. Bumitt and A. Stentz. “GRAMMPS: A Generalized Mission Planner for Multiple Mobile Robots In Unstructured Environments,” In *Proceedings of ICRA-98*.

[CCRS 1998] Canadian Centre for Remote Sensing, http://www.ccrs.nrcan.gc.ca/ccrs/tekrd/radarsat/specs/radovere.html.

[Chien *et al*. 1999] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau, "Using Iterative Repair to Increase the Responsiveness of Planning and Scheduling for Autonomous Spacecraft," IJCAI99 Workshop on Scheduling and Planning meet Real-time Monitoring in a Dynamic and Uncertain World, Stockholm, Sweden, August 1999.

[Estlin *et al.* 1999] T. Estlin, T. Mann, A. Gray, G. Rabideau, R. Castano, S. Chien and E. Mjolsness, "An Integrated System for Multi-Rover Scientific Exploration," in *Proceedings of AAAI-99*.

[Fischer *et al*. 1995] K. Fischer, J. Müller, M. Pischel, and D. Schier, “A Model For Cooperative Transportation Scheduling,” in *Proceedings of the First International Conference on Multi-Agent Systems*. San Francisco, CA.

[Fukunaga *et al*. 1997] A. Fukunaga, G. Rabideau, S. Chien, D. Yan 1997. “Towards an Application Framework for Automated Planning and Scheduling,” In *Proceedings of the 1997 International Symposium on Artificial Intelligence, Robotics and Automation for Space*, Tokyo Japan.

[Johnson&McGeoch 1997] D. Johnson and L. McGeoch. “The Traveling Salesman Problem: A Case Study in Local Optimization,” in *Local Search in Combinatorial Optimization*, edited by E. H. L. Aarts and J. K. Lenstra, John Wiley and Sons, London, pp. 215-310.

[Martin&Stallard 1999] M. Martin and M. Stallard, “Distributed Satellite Missions and Technologies – The TechSat 21 Program,” AIAA 1999 Space Technology Conference, Albuquerque, NM, September 1999.

[Mettler&Milman 1996] E. Mettler and M. Milman. “Space Interferometer Constellation: Formation Maneuvering and Control Architecture,” In SPIE Denver ’96 Symposium.

[Muscettola *et al*. 1997] N. Muscettola, *et al*. “On-Board Planning for New Millennium Deep Space One Autonomy,” in Proceedings of IEEE Aerospace Conference 1997.

[Paolucci *et al*. 1999] M. Paolucci, D. Kalp, A. Pannu, O. Shehory, and K. Sycara, “A Planning Component for RETSINA Agents,” Lecture Notes in Artificial Intelligence, Intelligent Agents VI. M. Wooldridge and Y. Lesperance (Eds.), forthcoming.

[Rabideau *et al*. 1999] G. Rabideau, T. Estlin, S. Chien, A. Barrett, "A Comparison of Coordinated Planning Methods for Cooperating Rovers," AIAA 1999 Space Technology Conference, Albuquerque, NM, September 1999.

[Sandholm. 1993] T. Sandholm, “An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations,” In *Proceedings of AAAI-93*.

[Smith. 1980] G. Smith, “The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver,” in *IEEE Transactions on Computers*, 29(12).

[Tambe 1997] M. Tambe, “Towards Flexible Teamwork,” *Journal of Artificial Intelligence Research*, 7:83-124.

[Zweben *et al*. 1994] M. Zweben, B. Daun, E. Davis, and M. Deale, “Scheduling and Rescheduling with Iterative Repair,” in *Intelligent Scheduling*, Morgan Kaufman, San Francisco, 1994.

Commentary on"From Rovers to Orbiters: Continues task distribution based coordination"

Jonathan Gal-Edd
Jgaledd@pop500.gsfc.nasa.gov

Paper scope:

Expand 4 components of "remote agent" to a distributed/constellation environment. The remote agent module has 4 components: Mission manager, Planner/scheduler, Executive/diagnostician, implement activity on spacecraft. The paper looks at 3 modules, where in each module we progressively distribute more functionality. Module a: centralized planing ; distributed executive and implementation; Module b: Distribute planning; Module c: Goal setting by bidding.

Comment on range of planning options

The paper presents the range of planning options available to support rovers. It is a good high level overview of the available options, and which variables should be addressed in each option. Moving from centralized planning through distributed planning to goal distribution.

"contractor" class for the autonomy module

Previous papers discussing Autonomy Architecture had a slave, follower and leader classes. This papers adds the new class of contractors. This class and the use of using contract networks with goal settings is the "new" interesting point of the paper.

Comment on Planning

The main objective of the paper is planning. It takes two and half pages until we reach the planning discussion. And then the more complicated topics such as distributed planning and contact networks are discussed in one short section. As this audience is familiar with planning, I would skip the back ground and get have a more detailed discussion about the distributed planning and contract networks. We need to show how the MTSP theory is applicable to space application.

Performance metrics

The paper mentions 8 performance metrics. The metrics are: the amount of explicit control an operator has, feasible accuracy of modeling on the ground, software testability, onboard computing power, platform event response time, bandwidth, quality of downlink data, redundancy. These are important Metrics however, the use of these metrics to compare between various planning modules is not detailed. For example Contract networks section has no discussion about these variables.

No details in Discussion

The paper is high level, there fore it is difficult to find any thing "wrong" with it. Being high level the paper does not give specifics when and how to use each planning module.

Conclusion

The background of this commentator is operability and dealing with spacecraft autonomy. I am used to dealing with specific problems and solving them. This paper is high level and is not detailed enough to understand the implementation of the various planning options. In goal setting contract network option it does not show how this can be done in space.

Poster Session

Adapting Control Methods for Autonomous Exploration of Unknown Environments

Barbara Engelhardt, Steve Chien, Darren Mutz
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109-8099
{firstname.lastname}@jpl.nasa.gov

Abstract

Proposed missions to explore comets and moons will encounter environments that are hostile and unpredictable. Any successful explorer must be able to adapt to a wide range of possible operating conditions in order to survive. The traditional approach of constructing special-purpose control methods would require information about the environment, which is not available a priori for these missions. An alternate approach is to utilize general control with significant capability to adapt its behavior, a so called *adaptive problem-solving* methodology. Using adaptive problem-solving, a spacecraft can use reinforcement learning to adapt an environment-specific search strategy given the craft's general problem solver with a flexible control architecture.

Introduction

Because of light-time communication delays, exploration missions require an autonomous explorer that can adapt to handle possible environments. For autonomous planning systems, the high-level actions of the spacecraft must be planned with sufficient environmental information to ensure that the resulting plans are admissible. A spacecraft could easily be lost based on inappropriate behavior in a particular environment due to overly-generic control methods (Minton 1996).

On the other hand, developing and testing domain-specific control methods is extremely difficult, and requires support of a domain expert. Moreover, the domain expert must have knowledge about the environment in which the spacecraft is operating, which is not available before the spacecraft arrives at the location to explore. If experts are not available, the spacecraft must be able to automatically adapt a flexible control structure specific to the new environment.

Adaptive problem solving addresses these problems by enabling the development and maintenance of effective control strategies without extensive domain-specific knowledge. An adaptive problem solver is given: (1) a generic set of control strategies and (2) a flexible control architecture, and uses a statistical method to estimate the quality of each control strategy or generate a more appropriate strategy. Adaptive problem solving also provides hard statistical guarantees on the quality of the behavior for each adapted control method. Using adaptive problem solving techniques, spacecraft exploration in unknown environments becomes feasible.

In this paper, we describe how adaptive problem solving can be used to adapt the control methods of a spacecraft in-situ. The value of this method is empirically shown in the context of two spacecraft operations scheduling problems in a generic planning and scheduling environment. By adapting control strategies for each domain, the lifespan of the spacecraft is improved since the adaptive problem solver can increase chances of spacecraft survival and continue to update the control methods based on aging hardware or environmental changes.

Motivational Example

The comet lander will land on a surface of unknown density, with the goals of drilling into the comet 90% and imaging its surroundings 10% of the time allocated to accomplishing goals. Situations will force these percentages to be innapropriate. One scenario might be that the surface of the comet is much denser than expected, so the rate of drilling is decreased and the wear on the drill is increased. The lander might need to adjust its priorities to take more images instead of drilling. Another scenario might be that drilling caused a layer of dust on the surface to drift up, the dust might limit the visibility of the lander. Taking images might be ineffective, so the lander might want to delay its drilling activities until the dust settles, or take images before drilling.

Failure to adapt to these situations could cost the lander the mission, by depleting resources too rapidly, not accomplishing mission objectives, or wearing out equipment. Not all possible situations can be enumerated before the mis-

sion; instead an adaptive problem solver checks the current control strategy's performance in the given environment and responds to changes by adapting the control strategy, independent of the cause of the change. An adaptive problem solver would continually adapt the control strategy if it found the current strategy non-optimal.

Planning System

The planning and scheduling system used to evaluate the control strategies for each model is a version of the ASPEN (Automated Scheduling and Planning ENvironment) system (Fukunaga *et al.* 1997). ASPEN is a configurable, generic planning/scheduling application framework that can be tailored to specific domains to create feasible schedules.

ASPEN employs planning and scheduling techniques to automatically generate a necessary activity sequence to achieve the mission goals. This sequence is produced by utilizing an iterative repair algorithm (Zweben *et al.* 1994) which classifies conflicts and attacks them individually. Conflicts occur when a plan constraint has been violated where this constraint could be temporal or involve a resource, state or activity parameter. Conflicts are resolved by performing schedule modifications such as moving, adding, or deleting activities. The target of the repair modification is chosen by a heuristic method, and the point in the search where this choice is made is called a choice point. For each type of choice point, the user creates a set of heuristic methods to use with varying usage weights. The set of heuristic methods impacts the outcome of the schedule, and effectively controls the behavior of the spacecraft.

The control strategies for adaptive problem solving are represented as sets of weighted heuristics so that they may be robust enough to perform well over the entire problem distribution even when they are slightly suboptimal, as opposed to a single heuristic which may not be as flexible to environment or hardware changes.

The quality of a resulting schedule generated by ASPEN is measured by a set of preferences specified by the user. This set of preferences specifies the quality functions associated with certain parameters in the schedule.

Adapting Control Strategies

To adapt control strategies, we can search the neighborhood of a current strategy, and select higher-scoring strategies. Given a set of possible control strategies, the adaptive problem solver selects the top strategies based on collecting samples of spacecraft performance in the current environment by running ASPEN evaluating the resulting schedule. The top strategies are returned to the search algorithm, which produces a subsequent set of hypotheses based on previously selected hypotheses using algorithm-specific techniques. This cycle continues until a certain amount of time has passed or another stopping criterion of the specific search algorithm has been met (see figure 1).

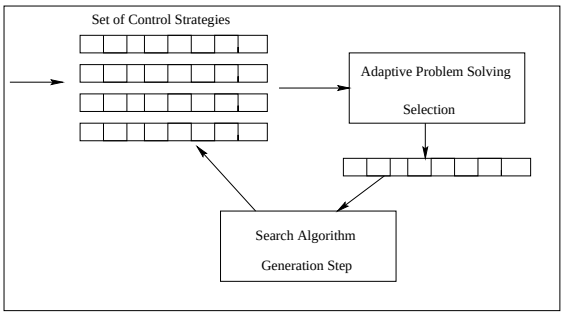


Figure 1: Hypothesis Generation Diagram

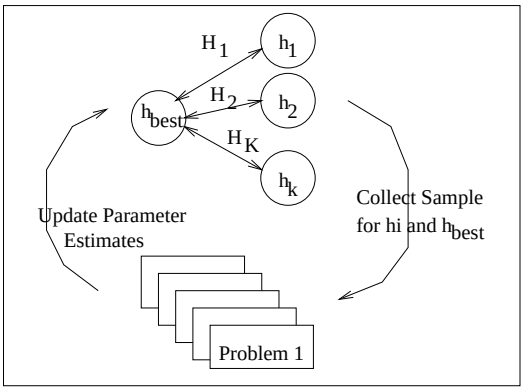


Figure 2: Adaptive Problem Solving Diagram

Adaptive Problem Solving

The adaptive problem solver attempts to select the top strategies from a set of strategies, supplied by the search algorithm, whose quality is a function of unknown parameters. It makes estimates of the parameters for strategy utility and sample cost in order to achieve a requested accuracy for a statistical decision requirement. The adaptive problem solver iteratively refines the utility and cost parameter estimates by acquiring training examples and reevaluating utility and cost (see figure 2).

The normal parametric model for reasoning about statistical error is used in this analysis, which assumes that the difference between the expected utility and estimated utility of a hypothesis can be accurately approximated by a normal distribution. This assumption is grounded in the Central Limit Theorem and is further discussed in (Chien *et al.* 1995). The analysis would change given a different parametric model, but the results should be analogous for conventional models.

Since parameter estimates are refined by random sampling, it is impossible to place perfect accuracy requirements on the selection algorithms. In practice, probabilistic

requirements, or *decision criteria*, on the relative accuracy of the parameter estimates (and subsequent strategy selection) are chosen as parameterized forms that allow a tradeoff between accuracy and cost.

Specifically, decision requirements take a set of hypotheses and a probabilistic error bound, and terminate when one of the hypotheses can be shown to have the greatest mean, evaluated through pair-wise comparisons, with a confidence higher than the given confidence level. The overall confidence for selection is a function of the confidence of each pair-wise comparison. Rational analysis can be used to allocate error to each pairwise comparison in such a way as to attempt to optimize the resource usage necessary to acquire a sufficient number of samples across the comparisons to achieve the decision requirement (Gratch & DeJong 1994).

In this analysis, the decision requirement that is used in the adaptive problem solver is the probably approximately correct (PAC) requirement. The choice of using PAC in this analysis is mostly based on its prevalence rather than specific attributes of the requirement. The expected loss decision requirement was evaluated and found to have minimal impact on the outcome.

PAC Requirement

In order to satisfy the PAC requirement, the hypothesis estimated to be the best must be within some user-specified constant ϵ distance of the true best hypothesis with probability $1-\delta$. The sum of the error from each pair-wise comparison is bounded by this probability. Let H_{sel} be the expected utility of the selected hypotheses and H_i be the expected utility for the remaining hypotheses. Let $\hat{H}$ be the estimate of the expected utility of a hypothesis. It is sufficient to bound the probability of error in selection for pair-wise comparisons with the following equation:

$$\sum_{i=1}^{k-1} Pr[\hat{H}_i < \hat{H}_{sel} - \epsilon | H_i > H_{sel} + \epsilon] \leq \delta \qquad (1)$$

Thus the problem of bounding the overall error reduces to bounding the error of each $k-1$ comparisons of the chosen best hypothesis to the rest of the hypotheses.

The normality assumption reduces equation 1 to a function of the parameter estimates, the number of examples n used to refine the estimates, the closeness parameter ϵ, and an unknown variance term σ^2. The two stopping criteria for selection are *dominance*, which is based on achieving a probability (δ) through sampling that h_i will perform better on a specific problem than h_j, and *indifference*, which is the probability that the difference between performances will fall within ϵ of 0. For the rest of this discussion, ϵ is ignored to simplify understanding. The equation for the probability of incorrect selection for a pair-wise comparison, α_i, is:

$$\alpha_i = \Phi\left(-(H_{sel} - H_i)\frac{\sqrt{n}}{\sqrt{\sigma^2_{sel,i}}}\right) \qquad (2)$$

We can use this relationship to determine how many training examples to allocate to each comparison, given the error bound on the probability of a mistake, an estimate of the difference in expected utility, and an estimate of the variance of each hypothesis:

$$n_{sel,i} = \frac{\sigma^2_{sel,i}}{(H_{sel} - H_i)^2}[\Phi^{-1}(\alpha_i)]^2 \qquad (3)$$

Rational Analysis

The hypothesis selection algorithm as presented does not take advantage of unequal distribution of error. By distributing error unequally across the pair-wise comparisons using the estimates of the sample cost and utility parameters, we can attempt to satisfy the requirements using the minimum possible cost. The general idea of rational analysis is to choose the error α_i for each comparison to minimize, subject to the given decision requirements:

$$\sum_{i=1}^{k-1} c_{sel,i} n_{sel,i}$$

The algorithm must only ensure that the *sum* of the errors remains less than the given bound. If one pair-wise comparison requires many more samples to achieve the same amount of accuracy as another comparison, then if the first is allowed to have more error and the second is allowed less, the overall cost of achieving those local requirements might be reduced. In practice, this method significantly reduces the number of samples necessary to achieve the requirement for certain domains, as shown in (Gratch & DeJong 1994).

Adapting Hypotheses

In order to adapt hypotheses, search algorithms are used to generate hypotheses in the neighborhood of the given hypotheses. At each level of search, an adaptive problem solving algorithm is used to evaluate the competing hypotheses with a given confidence bound.

Local Beam Search

One algorithm used to generate and search over hypotheses is local beam search (Russell & Norvig 1995). In a flexible planning and scheduling domain, each hypothesis, or combination of heuristics, can be represented as a vector of percentages where the percentages of heuristics associated with a certain type of choice point in ASPEN sum to 100% (see figure 3). A random heuristic is included for each plan problem. The basic algorithm is included below.

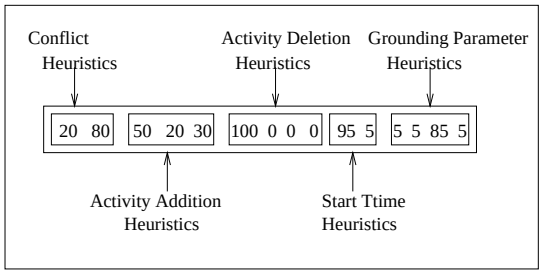


Figure 3: Hypothesis Vector Diagram

We chose a neighborhood of a vector to be defined as, for each subset of heuristics associated with a certain choice point, changing one of the usage percentages by a certain range, and scaling all of the other usage percentages equal amounts so that the sum is still 100% (see figure 3). Let l be the bound on the number of hypotheses the adaptive problem solver can evaluate.

Genetic Algorithm

Another algorithm that is used to generate hypotheses is a genetic algorithm (Goldberg 1989). Each hypothesis is represented as a vector of percentages, as in the local beam search. The three general operators (crossover, mutation, and reproduction) are used to generate the next set of hypotheses to search over, and ranking the hypotheses is done using adaptive problem solving. The crossover operator is not aware of the different subsets of heuristics, and may choose to split within one of those subsets. Mutation also works without knowledge of the constraint that subsets must sum to 100%, so each subset is scaled to 100 uniformly after the mutation operator is run. The basic algorithm is shown below.

Method Implementation

An adaptive control system of this type can be used in mission operations in multiple capacities. It can be used from the start to design spacecraft constraints and payload, by evaluating each of the potential designs against possible environments and comparing results. It can be used on the ground to perform mission planning and during flight to quickly develop new schedules based on changing domains or spacecraft deterioration. Environmental constraints for the spacecraft, such as the density or temperature of the surface for a lander, can be determined when they are available to the spacecraft. These constraints can be used to update the model of the environment, and adaptive problem solving can be used to efficiently determine the optimal planning heuristics for the current environment.

Empirical Evaluation

We claim that hypothesis adaptation can efficiently find a better set of hypotheses in a given domain. In this section we provide evidence that in practice, these methods can generate heuristic sets superior to those generated by model experts.

The test of real-world applicability is based on two domains related to planned space missions, using the ASPEN planning and scheduling system. The original set of hypotheses that is used is the set of heuristic combinations currently in use in these and related models. We hope this illustrates how this type of method can be useful in real-world domains, by improving on control strategies already in use or updating the strategies to handle domain shifts.

Evaluation

New Millennium EO-1 Domain – New Millennium Earth Observer 1 (EO-1) is an earth imaging satellite featuring an advanced multi-spectral imaging device. EO-1 mission operations consists of managing spacecraft operability constraints (power, thermal, pointing, buffers, consumables, telecommunications, etc.) and science goals (imaging of specific targets within particular observation parameters). The EO-1 domain models the operations of the EO-1 operations for a two-day horizon (Sherwood *et al.* 1998). It consists of 14 resources, 10 state variables and 38 different activity types. Each EO-1 problem instance includes a randomly generated, fixed profile that represents typical weather and instrument pattern. Each problem also includes 3 to 16 randomly placed instrument requests for observations and calibrations, and between 50 and 175 communications satellite passes.

The score for EO-1 includes preferences for more calibrations and observations, earlier start times for the observations, fewer solar array and aperture manipulations, lower maximum value over the entire horizon for the solar array, and higher levels of propellant.

Applying the quantile-quantile (Q-Q) test to the EO-1 hypotheses shows that they are very likely normal distributions. The Q-Q test compares the quantiles of the samples with a normal distribution, and departures in linearity of the resulting plot show how the samples differ from a normal distribution. Results of applying the Q-Q test to these two domains are shown in (Gratch & DeJong 1994).

Figures 4 and 5 show scores of the generated heuristic combinations over 35 cycles of the search algorithms. Although the curves for the scores of the two different search algorithms are different, the percentage of improvement for the high scoring hypothesis within each cycle is similar (128% for the linear search compared with 147% for the genetic algorithm). The percentage improvement for the mean score is somewhat greater, 161% for the genetic algorithm compared with 116% for the linear search. The

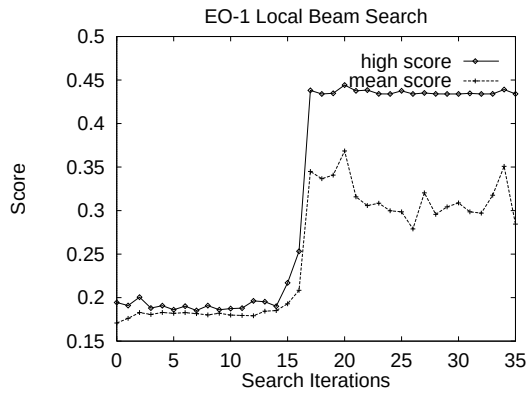


Figure 4: EO-1 model search iteration maximum and average scores for 35 iterations of the local beam search (beam = 2).

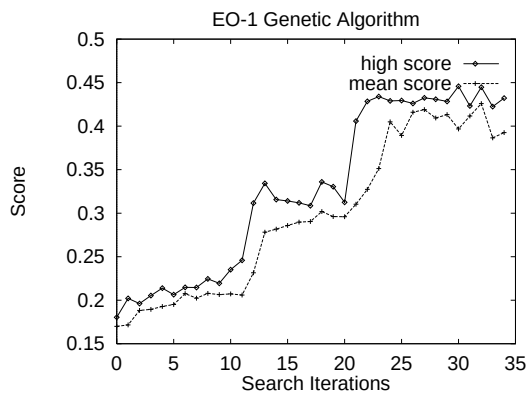


Figure 5: EO-1 model search iteration maximum and average scores for 35 iterations of the genetic algorithm search.

high scoring heuristic combinations are also somewhat different: the local search hypotheses use a significantly lower percentage of random heuristics than the genetic algorithm hypotheses, illustrating two different local maxima in the search space.

New Millennium Space Technologies Four Landed Operations Domain– The ST-4 domain models the landed operations of a spacecraft designed to land on a comet and return a sample to earth. This model has 6 shared resources, 6 state variables, and 22 activity types. Resources and states include battery level, bus power, communications, orbiter-in-view, drill location, drill state, oven states for a primary and backup oven state, camera state, and RAM. There are two activity groups that correspond to different types of experiments: mining and analyzing a sample, and taking a picture. Each ST-4 problem instance includes a randomly generated, fixed profile that represents communications visibility to the orbiting spacecraft. Each problem also includes between 1 and 11 mining activities and between 1 and 24 picture experiments at random start times.

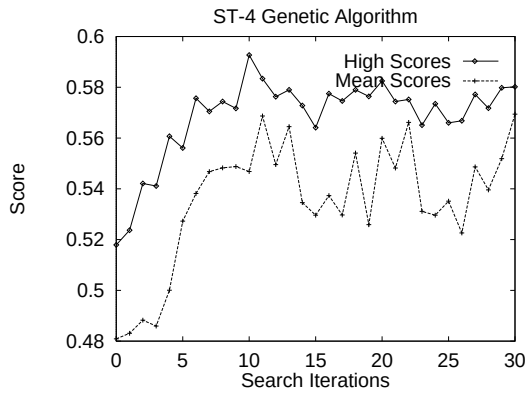


Figure 6: ST-4 model current iteration maximum and average scores for 30 genetic algorithm generations.

The preferences for ST-4 include more imaging, more mining, higher average battery power, fewer drill movements, and fewer uplinks.

Based on the Q-Q test, hypotheses from the ST-4 domain are likely to be normally distributed, and thus provides a good model for adaptive problem solving (Gratch & DeJong 1994). Graph 6 shows the mean and high scores of the generated heuristic combinations over 25 cycles of the search algorithms. The high score reaches a maximum improvement of 14%, and the mean score has a maximum improvement of 18%.

Related Work

Evaluating control strategies is a growing research topic. Horvitz originally described a method for evaluating algorithms based on a cost versus quality tradeoff (Horvitz 1988). Russell, Subramanian, and Parr used dynamic programming to rationally select among a set of control strategies by estimating utility (which includes cost) (Russell *et al.* 1993). The MULTI-TAC system considers all k-wise combinations of heuristics for solving a CSP in its evaluation which also avoids problems with local maxima, but at a large expense to the search (Minton 1996). Fink describes a method that sets time bounds for selection as opposed to parameter estimation accuracy, since sampling time is not large enough to attempt to minimize the number of samples (Fink 1998). Previous articles describing adaptive problem solving have developed general methods have been developed for transforming a standard problem solver into an adaptive one(Gratch & DeJong 1992; 1996), illustrated the application of adaptive problem solving to real world scheduling problems (Gratch & DeJong 1996), and showed how adaptive problem solving can be cast as a resource allocation problem (Gratch & DeJong 1994). We expand on these topics by evaluating different methods for generating hypotheses which can be used in

adaptive problem solving to efficiently estimate their utility and cost, considered separately.

Future Work

In the area of adaptive problem solving, additional work has been proposed for the stopping criteria, which can be resource bounded (specifically, time as a resource) instead of a relaxation of the ranking requirement, as in previous works on similar topics (Fink 1998). Different methods of combining heuristics could be applied to problems of this type. One method is composite strategies, from operations research, which involve logical decisions about the relative usage of heuristics as opposed to statistical methods. Another method is a portfolio approach, which combines heuristics in a method similar to a financial portfolio.

Current results do not indicate any direct benefit to using either local beam search or genetic algorithms over the alternative. In order to predict an effective search algorithm for each environment, it would be useful to generate a landscape of the utilities for the hypothesis space (Wolpert 1996), and choose the appropriate search algorithm for the environment. Previous work has been done in deterministic landscape generation (Wolpert 1996; Whitley 1995), but no practical work has been done in stochastic landscape generation, which is what this domain requires.

Conclusions

This paper outlines different methods for adapting control strategies using adaptive problem solving, with the goal of finding a control strategy or set of control strategies that performs well in the given planning and scheduling environment. The purpose is validated in all three planning and scheduling domains, by showing significant overall improvement in the generated plans. These results are significant in showing that autonomous spacecraft planning and scheduling is becoming a realistic option for missions to unknown environments.

ACKNOWLEDGMENTS

This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

Chien, S.; Gratch, J.; Burl, M. 1995. On the Efficient Allocation of Resources for Hypothesis Evaluation: A Statistical Approach. In *Proceedings of the IEEE Transactions on Pattern Analysis and Machine Intelligence 17(7)*. 652–665

Fink, E. 1998. How to Solve it Automatically: Selection among Problem-Solving Methods. In *Proceedings of the Fifth International Conference of AI Planning Systems*, 128–136.

Fukunaga, A.; Rabideau, G.; Chien, S.; and Yan, D. 1997. Towards an application framework for automated planning and scheduling. In *Proceedings of the 1997 International Symposium on Artficial Intelligence, Robotics and Automation for Space*.

Goldberg, D. 1989. *Genetic Algorithms: In Search, Optimization and Machine Learning*. Reading, Massachusetts: Addison–Wesley.

Gratch, J.; DeJong, G. 1996. A Decision-theoretic Approach to Solving the EBL Utility Problem. In *Artificial Intelligence* .

Gratch, J.; Chien, S.; DeJong, G. 1994. Improving Learning Performance Through Rational Resource Allocation. In *Proceedings of the Twelfth National Conference on Artificial Intelligence* , pp. 576-582.

Horvitz, E. 1988. Reasoning under Varying and Uncertain Resource Constraints. *Proceedings of the Seventh National Conference on Artificial Intelligence* , 111–116.

Minton, S. 1996. Automatically Configuring Constraint Satisfaction Programs: A Case Study. In *Constraints* 1:1(7-43).

Minton, S. 1988. Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence* 58:161–205.

Rabideau, G.; Chien, S.; Backes, G.; Chalfant, G.; Tso, K. 1999. A Step Towards an Autonomous Planetary Rover. *Space Technology and Applications International Forum*.

Russell, S.; Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ: Prentice Hall.

Russell, S.; Subramanian, D.; and Parr, R. 1993. Provably Bounded Optimal Agents. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*.

Sherwood, R.; Govindjee, A; Yan, D.; Rabideau, G.; Chien, S.; Fukinaga, A. 1998. Using ASPEN to automate EO-1 Activity Planning. In *Proceedings of the 1998 IEEE Aerospace Conference*.

Smith, S. 1994. OPIS: A Methodology and Architecture for Reactive Scheduling. In Zweben, M. and Fox, M., eds., *Intelligent Scheduling* . San Francisco: Morgan Kaufmann. 29-66.

Whitley, D.; Mathias, K.; Rana, S.l Dsubera, J. 1995. Building Better Test Functions In *Procedings of the International Conference on Genetic Algorithms*.

Wolpert, D.H.; Macready W.G. 1996. No Free Lunch Theorems for Search *SFI-TR-95-02-010*.

JPL 1999. *http://www.jpl.nasa.gov/missions/*.

Zweben, M.; Daun, B.; Davis, E.; and Deale, M. 1994. Scheduling and rescheduling with iterative repair. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling*. San Francisco, CA: Morgan Kaufmann. 241–256.

Automated Planning for the Antarctic Mapping Mission

Ben Smith
Jet Propulsion Laboratory
California Institute of Technology
benjamin.d.smith@jpl.nasa.gov

Barbara Engelhardt
Jet Propulsion Laboratory
California Institute of Technology
barbara.engelhardt@jpl.nasa.gov

Darren Mutz
Jet Propulsion Laboratory
California Institute of Technology
darren.mutz@jpl.nasa.gov

Abstract

The Second RadarSAT Antarctic Mapping Mission (AMM-2) is scheduled to begin in September of 2000. The Aspen planning system will automate the mission planning process and provide a fast-replanning capability for responding to anomalies during operations. AMM plans consist of several hundred SAR data swaths that must cover a ground region while obeying resource and operational constraints. This paper describes the planning problem, the system architecture, and the planning challenges involved.

Introduction

The Second RadarSAT Antarctic Mapping Mission (AMM-2) is scheduled for execution on September of 2000. This will be similar to the First RadarSAT Antarctic Mapping Mission (AMM-1) executed in 1997. The objective of AMM-1 was to acquire complete coverage of the continent, whereas the objective of AMM-2 is to acquire repeat-pass interferometry to measure ice surface velocity of the outer regions of the continent. The mission objective is to perform a synthetic aperture radar (SAR) mapping of the Antarctic over three consecutive 24-day repeat cycles. The SAR instrument has several "beams" each of which takes data in a rectangular swath. The incidence angle of each beam is separated by a few degrees and partially overlaps the swaths of adjacent beams. The location of the swaths at any given time is determined by the spacecraft orbit. The planning problem is to select a subset of the available swaths that fully cover the Antarctic and satisfy operational and resource constraints imposed by the RadarSAT Mission Management Office (MMO). The driving operational constraints are the limited on-board tape recorder (OBR) capacity and downlink opportunities which constrain the swath subsets that will fit on the OBR between downlinks.

The AMM-1 experience demonstrated that manually developing mission plans was laborious and error-prone. The plans took months to develop, and some constraint violations were not detected until the final MMO review. This required expensive and disruptive last-minute revisions. An automated planner could have quickly identified constraint violations, suggested repairs, and reduced the chance of errors, all of which would have significantly expedited the mission planning process.

Anomalies during AMM-1 operations caused several data takes to be lost. The missing data had to be rescheduled for later in the mission. The new plan had to be submitted within 36 of reacquisition, which meant replan options had to be identified within 8 hours and a final plan submitted within 8 to 72 hours. To manually turn around plans within these time constraints required a team of four people working from pre-generated contingency plan segments. The missed observations were placed into gaps in the original plan to minimize disruptions. More extensive changes were avoided to minimize the planning effort and the chance of introducing errors. Observations that could not be placed in this manner were simply dropped. Automated replanning during operations would allow faster turnaround with fewer people, and enable more extensive changes to the schedule in order to maximize science return.

The rest of this paper describes the automated planning system that is being constructed for AMM-2 based on the Aspen [1,4] planning environment. This system will develop baseline and contingency mission plans and will be used during operations to reschedule observations missed due to anomalies.

Automated Planning System

The core planning problem is to select a subset of the available swaths that will cover the Antarctic within the 30 day mission horizon while satisfying all of the OMM constraints. This requires a combination of constraint reasoning, for which planners are well suited, and geometric reasoning for which they are not. The planning system will operate in two modes: automated and mixed-initiative. In mixed-initiative mode the human user selects the swaths using a coverage analysis tool (SPA or STK) and has Aspen expand them into a detailed plan. The expansion primarily consists of deciding which downlink opportunities to use, tracking resource usage, and verifying adherence to operational constraints. Aspen then reports any constraint conflicts that it cannot resolve without modifying the

swaths (e.g., they oversubscribe the on-board recorder). The user modifies the swath selection accordingly and regenerates the plan. This continues until a conflict-free plan is generated. This rapid feedback allows the user to generate a plan much more rapidly than would be possible by hand. The mixed-initiative mode allows the user to use his/her scientific judgement in selecting swaths. In automated mode Aspen solves the problem automatically. The user provides a set of swath opportunities for each missed observation. Aspen selects at most one opportunity for each observation such that the resulting plan satisfies the operational constraints and recovers as many missed observations as possible. Aspen can be forced to use a specific swaths to recover a given observation by providing only one opportunity. Aspen solves the overall planning problem with a combination of forward dispatch and iterative repair [3,5]. A specialized set-covering algorithm (e.g., [2]) provides a solution to the swath selection problem which guides these two algorithms.

This mode will be used primarily in operations when new plans have to be turned around quickly following anomalies. Anomalies result in missing data takes which must be rescheduled. The user provides Aspen with a set of swaths for the missed observations. This same capability could also support a more aggressive mode where Aspen generates an initial mission plan from scratch, which the user then modifies to meet unarticulated scientific preferences. The user declares that each rectangular region of Antarctica to be imaged is a missed observation, and provides a set of opportunities for each one (this can be done automatically by SPA or STK). Aspen then selects swaths for these missed observations as it would in anomaly replanning.

System Architecture

The planning system takes as input a set of swath opportunities, downlink opportunities, and a partial plan. The swath opportunities may be fixed so that there is only one swath for each missed observation and Aspen has no swath selection decisions to make, or open so that Aspen must select among several opportunities for each missed observation. The partial plan can force swath-selection and downlink selection decisions, or leave them up to Aspen. This plan is in a spreadsheet format, which the mission planners are most comfortable with. The downlink opportunity file is provided by the MMO, and the user generates the swath opportunities file from a coverage analysis tool (SPA).

The downlink and partial plan files are converted into an Aspen plan file. Aspen expands the partial plan into a complete plan that satisfies the MMO constraints as encoded in the domain model. The model contains external dependency functions that ensure the data-take activities in the plan are consistent with the swaths in the swath opportunities file. The planning algorithm also consults this file to perform swath selection. If there is only one opportunity, the swath selection is trivially solved.

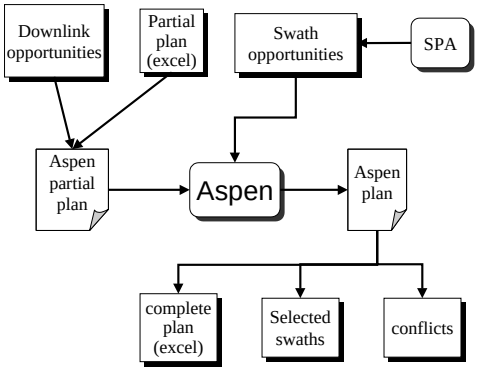


Figure 1: System Data Flow Architecture

Aspen generates a plan file and a list of conflicts that it was unable to resolve (e.g., because the swaths were fixed and caused conflicts). The plan file is converted into an Excel format that the mission planners prefer, and a list of swaths in SPA format (the swath request format required by the MMO). If the swath opportunities file was fixed, this is a pass-through operation; otherwise it is a down-selection of the original file. It will also generate a swath file in Satellite Tool Kit (STK) format. STK has more powerful coverage analysis capabilities than SPA.

Status

The constraint checking capabilities are completed and have been used to generate draft mission plans for submission to the Canadian Space Agency, and will be used to develop more detailed plans in the upcoming months. The planner takes a set of swaths and downlink opportunities as input, assigns the swaths to downlink opportunities, adds supporting activities for each swath and checks the resulting plan for constraint violations. These activities took weeks to perform manually during AMM-1 development, but only minutes with the automated planner. The operational replanning capabilities will be completed by Summer of 2000 for use in the September mission.

Acknowledgements

Thanks to John Crawford, the mission architect for AMM-1 and AMM-2, for championing this technology and sharing his knowledge of mission and operational constraints. This paper describes work performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract from the National Air and Space Administration.

References

1. Fukunaga, A.; Rabideau, G.; Chien, S.; and Yan, D. "Toward an Application Framework for Automated

Planning and Scheduling," in *Proceedings of the 1997 International Symposium on Artificial Intelligence, Robotics, and Automation for Space*, Tokyo, Japan (1997).

2. Garey and Johnson. Computers and Intractability: A guide to the theory of NP-completeness. W.H. Freeman and co., New York (1979)

3. Rabideau, Knight, Chien, Fukunaga, and Govindjee, "Iterative Repair Planning for Spacecraft Operations in the ASPEN System" in *Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation for Space (ISAIRAS)*, Noordjwijk, The Netherlands, June 1999.

4. Sherwood *et al.*, "Aspen User's Guide" Technical Document D-15482, Jet Propulsion Laboratory, California Institute of Technology. April, 1999. Also available at http://aspen.jpl.nasa.gov

5. Zweben, M.; Daun, B.; Davis, E.; and Deale M., "Scheduling and Rescheduling with Iterative Repair," in *Intelligent Scheduling*, Morgan Kaufman, San Francisco (1994).

A Call for Knowledge-based Planning

David E. Wilkins and Marie desJardins
Artificial Intelligence Center, SRI International
333 Ravenswood Ave., Menlo Park, CA 94025
{wilkins, marie}@ai.sri.com

Abstract

Much of the effort of the planning community is currently focused on improving the performance of disjunctive planners (DPs). We are interested in solving real-world planning problems and, to that end, argue for the use of domain knowledge in planning. Hierarchical task network (HTN) planners use more knowledge than DPs and have advantages such as scalability, expressiveness, continuous plan modification during execution, and the ability to interact with humans. We argue, however, that the field must develop methods capable of using even richer knowledge models than those used in HTNs (and therefore DPs) in order to make planning tools useful for complex problems. While we applaud the development of faster DP systems and their use for planning subproblems, it may not be best for the field to focus so many resources on techniques that solve only a narrow subset of the problems that are faced in real-world domains.

Introduction

Much of the effort of the planning community is currently focused on improving the performance of disjunctive planners (DPs). Kambhampati (1997) defines disjunctive planners as planners that retain the current planset without splitting its components into different search branches. This family of planners includes Graphplan (Blum & Furst 1995), SATPLAN (Kautz & Selman 1996), and their derivatives, all of which use STRIPS-style planning knowledge to derive and solve propositional representations of planning problems.

Most planners described in AAAI-99 and IJCAI-99 are DPs that could solve only simple, small toy problems from the standard corpus used for testing disjunctive planners. (By contrast, the planning applications in IAAI-99 used more expressive representations and methods.) The 1998 AIPS planning competition also focused considerable effort on simple STRIPS-encoded toy problems that were solvable (or nearly so) by DPs. Unfortunately, the difficulties of defining a common notation for more complex problems and knowledge led to the demise of a proposed HTN track. A recent survey of advances in AI planning (Weld 1999) focused entirely on DP methods, reflecting the current emphasis on this style of planning.

We are interested in solving real-world planning problems, and believe that doing so will require techniques that are more expressive and provide a wider range of capabilities than DPs. In particular, we argue for the use of *knowledge-based planning* (KBP)—that is, methods that use available domain knowledge to solve the planning problem, including interacting with humans when necessary to make use of their expertise. A planner may be more or less knowledge-based, depending on the range of knowledge it uses, and how effectively it uses it. We argue that KBP can solve problems that DPs in their current form cannot, because of the greater expressivity and more natural representations of KBP.[1]

We present some characteristics of real-world palnning problems that are not solvable by current DP approaches, and we argue that DPs are unlikely to extend to these problems. We summarize the features of an HTN planner to show the types of knowledge and capabilities that are exploited in an existing KBP system. However, current HTN planners are only a small step in the direction of the level of KBP that we envision. We next argue that achieving planful behavior in a complex, dynamic world will require the use of much more knowledge and richer knowledge models than those used in HTN planners. We discuss forms of knowledge that HTN planners do not use, and give some examples of problems that today's HTN planners are not able to solve. Finally, we draw some lessons from the history of the machine learning research community that are analogous to the current trends in the planning community.

Characteristics of Real-World Planning Problems

Real-world problems have been found by many researchers to require more expressive representations and capabilities than those provided by current AI planning systems. Chien *et al.* (1996) conclude from their experience with multiple NASA applications that "current plan representations are

[1]Our comments apply not only to DPs, but to any planner that cannot address hard, embedded problems, whether for computational reasons or a lack of expressiveness. We address our remarks to DPs as they are currently the most studied systems with that property.

impoverished." They discuss the requirements of an operational context in which users must interact with the system, and must be able to understand and modify the plans produced by the planner. Our experience with military planning applications supports these conclusions. In this section, we describe some of the specific capabilities that are needed to solve real-world problems: numerical reasoning, parallelism, context-dependent effects, interaction with users, execution monitoring, replanning, and scalability.

Reasoning with numbers is essential in every realistic domain that we have studied. Common needs for numbers are time, sharable resources having a specific capacity, continuous resources available in limited quantities, and goals of accumulation. In practice, DPs have difficulty handling problems involving reasoning about numbers.[2] In most existing non-HTN AI planners, the need for numerical reasoning is reduced by assuming that sharable resources have infinite capacity, and that continuous resources are unlimited (Srivastava & Kambhampati 1999).

Realistic domains may have dozens of (perhaps necessarily) parallel activities, as activities of various agents are coordinated. Parallelism can cause computational problems for DPs, and many systems produce only sequential plans.

Realistic domains often have numerous context-dependent effects, which can cause an exponential explosion in the number of STRIPS operators needed. This problem is being addressed to some extent in DPs. Extensions to the Graphplan algorithm to handle conditional effects are given in (Kambhampati, Parker, & Lambrecht 1997) and (Guéré & Alami 1999), but neither paper discusses the time or space complexity of the algorithms. Other approaches have also been tried, perhaps the most promising being factored expansion, in which an action with conditional effects is split into new actions called "components," one for each conditional effect. The cost is added complexity in the planning algorithms involving "tricky" extensions (Anderson, Smith, & Weld 1998).

Interacting with people is a critical aspect of real-world planning. Realistic problems are embedded in the world, and generally do not have precisely defined boundaries or evaluation functions. Thus, most interesting planning problems will be difficult or impossible to model fully. For example, criteria for plan evaluation often cannot be quantified, such as when the political consequences of a military or media action are crucial. In such cases, a human user must be able to guide the planner and evaluate the plans produced, allowing the planning system to take advantage of the user's expertise

In the real world, the goal of planning is not simply to build the plan, but to use it to control actions in the world. Therefore, realistic planning systems must support execution monitoring and continuous plan modification during execution. Because there is no dependency structure in DP plans, monitoring them is difficult. In addition, the DP approach is very brittle in the face of changing problem requirements, and any change in the environment may result in the planning system having to start from scratch.

Finally, realistic problems involve enormous search spaces, so scalability is essential. Vast strides have been made in the size of problems solved by DPs, such as solving instances with 10^{16} to 10^{19} configurations. However, these large DP problems are still representations of toy problems, such as a logistics problem with 9 packages, 5 trucks, 2 airplanes, and 15 locations (Kautz & Selman 1998). Simply increasing the number of locations to a realistic number will make even these toy problems unsolvable. In contrast, HTN planners can generate plans in domains with thousands of objects and hundreds or thousands of actions.

Using Knowledge in HTN Planners

Hierarchical task network (HTN) planning is one example of a KBP approach that is more knowledge-based than DPs. It has been known for some time that HTN formalisms are more expressive than the STRIPS formalism favored by many DPs, roughly analogous to the additional expressivity of context-free grammars over right-linear (regular) grammars (Erol, Hendler, & Nau 1994).

HTNs naturally model the world in the same way that human users do in many domains, using the same abstractions. This modeling approach enables users to control and understand the planning process and the resulting plans. (Note, however, that current HTN planners still leave much to be desired in terms of interactive planning.) In contrast, the DP approach models the planning problem as millions of conjunctive normal form expressions, making it difficult for users even to understand the planner's reasoning process, much less intervene to modify or guide it.

In this section, we describe some of the uses made of domain knowledge in a particular HTN planner, SIPE–2. These features may be candidates for extending non-HTN planners. Kautz and Selman (1998) identify three kinds of planning knowledge: knowledge about the domain, knowledge about good plans, and explicit search-control knowledge. In addition, HTN planners are also concerned with other types of knowledge, such as knowledge about interacting with the user, knowledge about a user's preferences, and knowledge about plan repair during execution. SIPE–2 has formalisms that allow encoding of these types of knowledge, in either HTN operators (also known as methods or schemas), advice, or other declarations.

It is sometimes argued that the knowledge used by HTN planners is "simply search-control knowledge," rather than part of the problem statement. However, if the goal is to solve realistic planning problems, then intelligent, principled search control that takes advantage of knowledge about the domain is precisely what is needed. This knowledge can often be naturally and efficiently captured in HTN operators, where much of the context is implicit and therefore need not be expressed or checked during each attempted application.

[2] While simple, finite arithmetic could be added to DPs, the combinatorics would generally explode. Another approach is given by Wolfman and Weld (1999), who describe a system that combines SATPLAN with an incremental Simplex algorithm for solving linear inequalities. A useful extension, but combinatorics allow solution of only toy problems.

HTN Capabilities: SIPE-2

SIPE–2 (Wilkins 1990) is a domain-independent HTN planner that uses more knowledge and has richer capabilities (such as numerical constraints and resource models) than the current DPs. Example applications include containing oil spills (Agosta & Wilkins 1996), planning air campaigns for the Air Force (Wilkins & Myers 1998; Lee & Wilkins 1996), and joint military operations planning (Wilkins & Desimone 1994). In the latter applications, the domain knowledge includes 100 to 200 operators, around 500 objects with 15 to 20 properties per object, and a few thousand initial predicate instances. Plans can include up to several hundred actions—several thousand if all abstraction levels are counted—usually having numerous parallel activities.

Expressiveness SIPE–2 provides a powerful approach to representing and reasoning about planning problems that makes it suitable for complex domains.

SIPE–2's HTN operators are encoded at multiple *abstraction levels*. The higher levels can model various solution methods. SIPE–2 operators can *dynamically generate a set of goals* at planning time, a capability that has been extensively used. For example, a defend goal can be generated for every currently known threat. *Situation-dependent effects* of actions are deduced by a causal theory. Such effects have proven their use in practice — without them, the number of operators can grow exponentially in complex domains.

SIPE–2 can *reason about numbers*, a capability crucial in nearly all of our applications. For example, a planning variable may be constrained to be a runway with a length greater than some number, sharable resources have a specific capacity, and continuous resources are available in limited quantities. In many application domains, it is necessary to *accumulate a certain quantity* of some resource, or achieve a certain level of effect, such as obtaining a sufficient length of boom to surround an oil spill. Such goals are not accomplished by a single action; rather, several actions contribute to the accumulation. Thus, SIPE–2 determines when a set of actions (that individually produce some amount of the resource in question) together achieve an accumulation goal.

Temporal reasoning is important in many domains. SIPE–2 has two different modes for reasoning about time. The most general allows specification of any of the thirteen Allen relations between any two nodes. The temporal constraints are solved separately from the other constraints by passing them to Tachyon (Arthur & Stillman 1992).

Finally, calls can be made to *arbitrary domain-specific* LISP *code*, for knowledge that cannot easily be modeled in the HTN formalism, and for sophisticated numerical calculations. Functions on planning variables may compute an instantiation (e.g., the duration of a flight) and procedural attachment on predicates may compute whether a condition is true.

User Interaction Because HTN plans and domain knowledge can be complex, a powerful graphical user interface (GUI) is essential. Without natural pictorial representations of the knowledge and plans, it would be nearly impossible for a human to understand them. SIPE–2 provides a GUI and a web server to aid in generating plans, viewing complex plans as graphs, and following and controlling the planning process.

SIPE–2 provides a flexible and powerful interactive planner. In our applications, this feature is critical because experienced human planners can guide the search effectively, and are reluctant to give control to an automated system. The user may interact with the planning process at many different levels of detail, and may direct the planner to solve certain parts of the problem automatically. Under interactive control, the user can control (among other things) when and how resources are allocated, which operators to select, which goal to expand next, how to instantiate planning variables, and how to resolve conflicts.

The user can also control or influence the plan development process using the Advisable Planner (Myers 1996). Users can direct the planning process by providing high-level guidance that influences the nature of the solutions generated. Advice consists of task-specific properties of both the desired solution and the problem-solving process to be used for a particular problem.

Constraints and Efficiency A sort hierarchy represents invariant object properties, describes the classes to which an object belongs, and allows for inheritance of properties. The sort hierarchy encodes large amounts of knowledge in our applications, and the planner can reason more efficiently about this knowledge because it knows the relationships cannot change as actions are performed.

SIPE–2 uses the least-commitment approach to variable binding. Constraints are placed on variables by domain knowledge in the operators (e.g., a particular truck must have a capacity greater than 100). Instantiations are not chosen until sufficient constraints accumulate to identify a unique acceptable value. Because uninstantiated variables increase computational complexity, domain-specific knowledge can be used to specify when early instantiation of variables can be done without adversely affecting solution quality (Myers & Wilkins 1998).

Predicates can be declared as functional in certain arguments, allowing a dramatic speedup, which has been documented experimentally (Myers & Wilkins 1998). Functional predicates are of particular importance to reasoning about locations in planning systems, and have proven valuable in nearly every application of SIPE–2.

The system relies on *plan critics* that check for and correct conflicts in plans, reducing computational costs during plan expansion. Examples of plan critics include the temporal reasoner and checking constraints on planning variables. These critics are applied by default after each *planning level*, i.e., an expansion of the whole plan to a greater level of detail, but the frequency can be increased or decreased as appropriate for the problem domain (Wilkins 1990).

Knowledge Beyond HTN

Despite the power of HTN planning systems, and their demonstrated ability to address real-world planning problems, they have limitations that make them inadequate for many problems of interest. In particular, HTN planners:

- require complete (except for anticipated incompleteness) and certain knowledge about the world
- model the effects of actions as deterministic, fully understood outcomes
- assume that the planner controls all agents that cause changes in the world state
- require significant effort in domain modeling and knowledge acquisition for complex problems
- cannot perform or incorporate complex or decision-theoretic evaluations of plan quality
- ignore the qualification problem
- use simplistic frame problem solutions that prevent drawing the most appropriate conclusions when contradictory (perceptual) information arrives (Pollock 1998)
- do not consider risks and utilities
- do not use knowledge and probabilities to handle uncertainty
- are brittle (may not work if the problem changes slightly).

These limitations are shared by the other major families of planners (DPs and causal-link planners), although some of them, such as handling uncertainty, are the subject of ongoing research (Boutilier, Dean, & Hanks 1999; Majercik & Littman 1998; Smith & Weld 1998; Weld, Anderson, & Smith 1999; Kushmerik, Hanks, & Weld 1994).

In addition to these limitations, planning systems that could solve interesting problems in a complex, dynamic world will need capabilities that represent a fundamental shift in how we think about planning problems. An ideal system would be able to behave like humans do in these sort of environments; in particular, it would have to:

- exhibit creativity, devising new actions that can solve a problem or shorten a plan
- use analogy to transfer solutions from other problems
- effectively interact with humans to use their knowledge in decisions
- behave intelligently in the face of conflicting or incomplete information.

We believe that these capabilities will require more knowledge, including background knowledge of other domains and of how the world works. Evaluation criteria other than correctness and plan length will have to be factored in explicitly. Also, interacting effectively with humans will be essential because we will never model every possible issue that might affect a planning decision.

Of course, not all interesting problems have these characteristics, and in any given case, it may be possible to formulate the problem in such a way as to remove the need for these capabilities. For example, in developing the Burton planner (not a DP planner), Williams and Nayak (1996) used a purely propositional representation. However, it seems unlikely that all interesting problems will be amenable to such an approach, and other NASA applications have required richer representations (Chien *et al.* 1996).

Knowledge in DP vs KBP

To impact realistic problems, we predict that DPs will have to incorporate the types of knowledge used by HTN planners, as well as knowledge to overcome the limitations of HTN approaches that we have discussed. It is encouraging that this is already starting to occur. For example, there is initial work on adding knowledge about temporal extent of actions to SATPLAN encodings (Smith & Weld 1999), and on encoding HTN method knowledge for satisfiability solvers (Mali & Kambhampati 1998).

However, while HTN planners can generally make effective use of additional knowledge, the same is not necessarily true of DPs. Additional knowledge encoded as axioms may increase the size of the problem with redundant axioms, and make the problem harder to solve. Initial experiments indicate that whether added knowledge helps or hurts may depend on the particular combination of knowledge, problem, and algorithm (Kautz & Selman 1998). For example, the "point of diminishing returns from the addition of axioms would be sooner reached for stochastic search than for systematic search" (Kautz & Selman 1998). Thus, the knowledge added to a DP may have to be carefully chosen for the problem being solved and the algorithm being used.

KBP approaches are rightly criticized for the expense of modeling a new domain. However, we conjecture that building computationally efficient encodings for DPs of complex planning domains is no easier than building HTN models. Many encoding issues are still under study, even for toy domains (Kautz & Selman 1999; Brafman 1999; Mali & Kambhampati 1998).

Lessons from Machine Learning

In every research community, there is an ongoing tension between well defined and more ambitious problems. On the one hand, if a field focuses on small, well understood problems, with well defined algorithmic properties and evaluation metrics, then a set of benchmark problems can be formulated to facilitate formal and empirical analysis and comparison of competing methods. On the other hand, many of the interesting challenges posed by realistic applications have broader implications and less well understood properties, and the problems are more difficult to define crisply and to evaluate.

Several years back, the machine learning community established a repository of benchmark problems to evaluate machine learning systems. Naturally, these problems all had commonalities: most used an attribute-vector representation; most consisted of "sets of instances" with no back-

ground knowledge. In practice, they could be used only to evaluate predictive accuracy on propositional, supervised learning algorithms. Despite these limitations, however, it became the de facto standard that papers submitted to the International Conference on Machine Learning (ICML) had to include an evaluation on these benchmark problems.

In some ways, this set of benchmarks, and the emphasis on evaluation, were good for the community: they forced researchers to think about metrics and about comparing their systems to other systems, and they provided a baseline of performance that researchers could test new ideas against. On the other hand, they tended to stifle research that didn't fit neatly into the problem space defined by the benchmark problems. Applications-oriented researchers reported that it was difficult or impossible to get their papers accepted to the leading ML conferences (Provost & Kohavi 1998). Meanwhile, more and more papers appeared showing minor tweaks and incremental improvements to existing algorithms (but they showed "statistically significant" improvements on the benchmark algorithms!)

As a result, there are now many well understood and effective methods for propositional supervised learning—and there has been much less progress in other areas of machine learning, such as incorporating background knowledge, feature engineering, relational learning, interactive learning techniques, visualization of learned knowledge, and complex evaluation criteria.

Most recently, there has been an explosion of interest in learning Bayes nets. Bayes net learning and inference techniques have appealing computational properties that are analogous to those of DP approaches: they efficiently capture certain types of problem structure, and significantly speed up certain types of inference over previous methods. However, like DP approaches, they use a propositional representation, and do not address many of the other challenges posed by realistic learning problems. As with DP approaches, the rush of enthusiasm over Bayes net techniques has threatened to overshadow the fact that despite their computationally attractive properties, they still solve only a small subproblem within the overall field of machine learning.

Similarly, in the planning community, there is a danger that by focusing too much attention and effort on DP methods and the problems they solve, we risk losing the ability to recognize other kinds of contributions and advances. In particular, if the benchmark of performance becomes solely how many blocks our planners can stack, and how fast they can do it, then it will become increasingly difficult to recognize and learn from research that performs well along other dimensions—or that addresses problems that DP systems overlook completely. As we discussed earlier, DP researchers within the planning community are starting to look towards extending their systems to incorporate richer forms of knowledge. This is a trend that we applaud, and that we hope will continue, but it is not enough to simply broaden the uses of DP systems: we need to be open to completely different approaches and paradigms as well.

While improving the speed of solving problems we know how to formulate precisely is a valuable research activity, so is continuing to investigate problems that we don't yet have a good handle on formulating or solving. Results may be more difficult to achieve or quantify for these the "hard problems," but that doesn't mean we shouldn't be working on them.

Conclusion

Ginsberg (1996) has pointed out that the SATPLAN approach is successful because it solves the "puzzle" part of a problem, and overlooks any commonsense reasoning aspects of the true problem. In Ginsberg's view, commonsense reasoning is the heuristic process by which we reduce extremely complex problems to NP-hard or simpler problems for which search is feasible. Which aspects of a problem to pay attention to, frame and context assumptions, and default strategies for organizing complex activities are all aspects of commonsense reasoning. As Ginsberg puts it, "It is Kautz and Selman who are solving the commonsense aspects of the problem; their 'planner' is solving the puzzle-mode kernel of the problem instead of the problem itself." Indeed, the problems solved by DP approaches are almost exclusively puzzle-style problems (or "real-world" problems that have been reformulated as puzzles).

We favor using DP to solve puzzle-style subproblems that can be represented as satisfiability problems and solved in acceptable time. DP may well be the appropriate method for those aspects of the overall problem. However, AI planners also need to provide support for the commonsense reasoning aspects of the problem so that plans can be used to guide planful behavior while embedded in a complex, dynamic environment. We have argued that incorporating knowledge into the planning process is the most promising way to provide these abilities.

Although DP methods are clearly useful approaches for solving certain subproblems, it is important for the field as a whole to continue to look at a wider range of problems. There is a danger of allowing the current popularity of DP approaches, and the associated evaluation techniques and "puzzle-style" problem suite, to overly influence the field, making it more difficult for advanced KBP methods to find an audience.

Acknowledgments This research was supported by Contract F30602-95-C-0235 with the Defense Advanced Research Projects Agency, under the supervision of Air Force Research Lab – Rome. Thanks to Dana Nau and Foster Provost for contributing their ideas to this paper.

References

Agosta, J. M., and Wilkins, D. E. 1996. Using SIPE-2 to plan emergency response to marine oil spills. *IEEE Expert* 11(6):6–8.

Anderson, C.; Smith, D. E.; and Weld, D. 1998. Conditional effects in Graphplan. In *Proc. of the 1998 International Conference on AI Planning Systems*, 44–53.

Arthur, R., and Stillman, J. 1992. Tachyon: A model and environment for temporal reasoning. Technical report, GE Corporate Research and Development Center.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1636–1642. Morgan Kaufmann.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11:1–94.

Brafman, R. I. 1999. Reachability, relevance, resolution and the planning as satisfiability approach. In *Proc. of the 1999 International Joint Conference on Artificial Intelligence*, 976–981.

Chien, S.; Hill-Jr, R.; Wang, X.; Estlin, T.; Fayyad, K.; and Mortensen, H. 1996. Why real-world planning is difficult: A tale of two applications. In Ghallab, M., and Milani, A., eds., *Advances in AI Planning*. IOS Press. 287–298.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. HTN planning: Complexity and expressivity. In *Proc. of the 1994 National Conference on Artificial Intelligence*, 1123–1128.

Ginsberg, M. L. 1996. Do computers need common sense? In *Conference on Knowledge Representation and Reasoning*.

Guéré, E., and Alami, R. 1999. A possibilistic planner that deals with non-determinism and contingency. In *Proc. of the 1999 International Joint Conference on Artificial Intelligence*, 996–1001.

Kambhampati, S.; Parker, E.; and Lambrecht, E. 1997. Understanding and extending Graphplan. In *European Conference on Planning*.

Kambhampati, S. 1997. Challenges in bridging plan synthesis paradigms. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*.

Kautz, H., and Selman, B. 1996. Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence*, 359–363. Wiley.

Kautz, H., and Selman, B. 1998. The role of domain-specific knowledge in the planning as satisfiability approach. In *Proc. of the 1998 International Conference on AI Planning Systems*, 181–189.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and graph-based planning. In *Proc. of the 1999 International Joint Conference on Artificial Intelligence*, 318–325.

Kushmerik, N.; Hanks, S.; and Weld, D. 1994. An algorithm for probabilistic least-commitment planning. In *Proc. of AAAI-94*, 1073–1078.

Lee, T. J., and Wilkins, D. E. 1996. Using SIPE-2 to integrate planning for military air campaigns. *IEEE Expert* 11(6):11–12.

Majercik, S., and Littman, M. 1998. Maxplan: A new approach to probabilistic planning. In *Proc. of the 1998 International Conference on AI Planning Systems*, 86–93.

Mali, A. D., and Kambhampati, S. 1998. Encoding HTN planning in propositional logic. In *Proc. of the 1998 International Conference on AI Planning Systems*, 190–198.

Myers, K. L., and Wilkins, D. E. 1998. Reasoning about locations in theory and practice. *Computational Intelligence* 14(2):151–187.

Myers, K. L. 1996. Strategic advice for hierarchical planners. In Aiello, L. C.; Doyle, J.; and Shapiro, S. C., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR '96)*. Morgan Kaufmann Publishers.

Pollock, J. L. 1998. Perceiving and reasoning about a changing world. *Computational Intelligence* 14(4):498–562.

Provost, F., and Kohavi, R. 1998. Guest editors' introduction: On applied research in machine learning. *Machine Learning* 30(2/3).

Smith, D., and Weld, D. 1998. Conformant Graphplan. In *Proceedings of the 15th National Conference on Artificial Intelligence*, 889–896. AAAI Press.

Smith, D. E., and Weld, D. S. 1999. Temporal planning with mutual exclusion reasoning. In *Proc. of the 1999 International Joint Conference on Artificial Intelligence*, 326–333.

Srivastava, B., and Kambhampati, S. 1999. Scaling up planning by teasing out resource scheduling. In *ECP-99*.

Weld, D. S.; Anderson, C. R.; and Smith, D. E. 1999. Extending Graphplan to handle uncertainty and sensing actions. In *Proceedings of the 15th National Conference on Artificial Intelligence*, 897–904. AAAI Press.

Weld, D. S. 1999. Recent advances in AI planning. *AI Magazine* 93–123.

Wilkins, D. E., and Desimone, R. V. 1994. Applying an AI planner to military operations planning. In Fox, M., and Zweben, M., eds., *Intelligent Scheduling*. Morgan Kaufmann Publishers Inc., San Francisco, CA. 685–709.

Wilkins, D. E., and Myers, K. L. 1998. A multiagent planning architecture. In *Proc. of the 1998 International Conference on AI Planning Systems*, 154–162.

Wilkins, D. E. 1990. Can AI planners solve practical problems? *Computational Intelligence* 6(4):232–246.

Williams, B. C., and Nayak, P. P. 1996. A model-based approach to reactive self-configuring systems. In *Proceedings of the 13th National Conference on Artificial Intelligence*. AAAI Press.

Wolfman, S. A., and Weld, D. S. 1999. The LPSAT engine and its application to resource planning. In *Proc. of the 1999 International Joint Conference on Artificial Intelligence*, 310–316.

Can Customers Schedule Their Own Payload Activities?

John Jaap
John.Jaap@msfc.nasa.gov
256-544-2226

Elizabeth Davis
Elizabeth.Davis@msfc.nasa.gov
256-544-2227

Mission Support Systems Group
Ground Systems Department
Flight Projects Directorate
Marshall Space Flight Center, AL 35812
National Aeronautics and Space Administration

Abstract

The term "customer" in the title refers to the payload developers; they are the *real users* of a space vehicle, after all. The answer to the question lies in the ability to design and deploy a system that allows multiple simultaneous users to schedule activities that require shared resources. In addition, the system must be designed so that it can easily be used by a community whose members, while being experts in their payloads, know little or nothing about scheduling. An effort is underway at Marshall Space Flight Center to demonstrate the feasibility of allowing users to schedule their own payloads. A web-based request-oriented scheduling engine and the infrastructure to support it are being investigated. This system will allow multiple users, each at a personal computer with a web browser, to formulate scheduling requests and submit the requests for immediate automatic scheduling.

Introduction

Scheduling payload operations has historically been done by a cadre of mission planners who act as agents for the payload developers. The members of this cadre are experts in the features, capabilities, limitations, and language of the scheduling engine. Different members of the cadre have different scientific and technical backgrounds; they are usually matched to the payload they represent. The members become experts in the payload that they represent – often with the help of extended input from the payload developers.

The process of collecting requirements is often protracted with multiple iterations. The payload developer submits payload requirements and descriptions in the requested format. A cadre member reviews the information and contacts the payload developer for a better understanding. The payload developer modifies his submission.

After the cadre has an adequate (in-depth) understanding of the requirements, the cadre members use the scheduler to produce a timeline. The payload developers review the timeline, make comments, and the cadre makes repairs to the timeline. Usually, several months elapse between the requirements submittal and the completion and publication of the timeline.

The scheduler currently used for the International Space Station payloads requires expertise to represent the payload requirements. The lack of several key features causes the cadre to spend a great deal of time describing a payload's actual scheduling requirements in the "vernacular" of the scheduler, and then to hand build the timeline with a manual timeline editor to get the desired results. The scheduler cannot represent or process optional sequences of operations (multiple scenarios), choice of constraints within a non-homogeneous group, soft (fuzzy) requirements, or resource lock-in across sequential tasks. Furthermore, the current scheduler only supports one cadre member at a time, so the cadre members must either take turns or funnel all scheduling through one member.

If a scheduling system existed that allowed payload developers to schedule their own payloads, a better timeline could be produced in less time with fewer cadre members supporting the system. The delay between describing and submitting a scheduling request and viewing the results would be reduced to minutes rather than months. The payload developers could then refine and resubmit their requirements until they got a satisfactory timeline. The cadre would only have to preload the scheduling engine with the envelopes and allocation constraints, define the station and payload equipment resource requirements, and post-check that the timeline is safe and meets programmatic constraints.

ROSE The Mission Support Systems Group at MSFC has embarked on an effort to design and demonstrate (and possibly deploy) a system that ameliorates many of the shortcomings of the current system by allowing the payload developers to schedule their own payload activities. It includes a graphics-based method for formulating scheduling requests and a centrally located, multi-user Request-Oriented Scheduling Engine (ROSE) working against one current set of resource availabilities and one current timeline. The system uses the World Wide Web to allow the user community (the payload developers – the customers) to readily access the system from a personal computer or workstation. The name ROSE is being applied to both the project as a whole and to the scheduling engine itself.

Web-Based Architecture

ROSE is a web-based application. The users access the system via the World Wide Web; no ROSE-specific software is installed on a user's computer and no data is stored on a user's computer. The user navigates via a web browser to the ROSE web site, logs on, and proceeds to formulate scheduling requests and submit them for scheduling. The user can also review the in-work timeline, delete tasks from

the timeline, and view ancillary information such as resource allocations and envelopes. An overview of the ROSE architecture is shown in Figure 1.

A web-based system is ideal for the payload developers. Maintenance is literally zero. Opening the web page with a browser will download and execute the latest version of the client without any effort on the user's part. The user can switch between client computers in his office, home, or even use a portable without installing ROSE-specific software or moving any files. Security is the only concern, and security is being integrated into the ROSE research and development effort so that an adequate solution can be deployed.

Implementation The user of ROSE initiates a "web session" by opening a web page on a remote web server. The web server is Microsoft's Internet Information Server (IIS) running on a Windows NT/2000 server. The web page downloads and runs a Java applet. Logon is handled by the Java applet; the web session is managed by IIS via Active Server Pages (ASP) and a global.asa file. All communication with the Java applet, including access to the database, is facilitated by a set of ASPs written in Visual Basic Script. Sessions are terminated when the user goes to another web page or exits the browser, when the server does not receive communication from the client for an extended period of time, or when the client or the server is restarted. The client automatically initiates a new session, without user intervention, when needed. The session manager, which handles locking of the data in the database, prevents an account from having more than one active session. This implementation does not require a continuous connection between the client and the server. It even allows the client computer to be disconnected from the network for an extended period without loss of cached data or edits.

Modeling – The Critical Element

Modeling in the context of activity scheduling has historically meant defining an activity's requirements for shared resources (power, crew, etc.), time-dependent constraints (when the vehicle is within view of a target), and sequencing of activities relative to each other (warm-up before data-take before shutdown). In the context of ROSE, modeling is also called "request formulation," because a model is the core of a scheduling request.

Modeling is the process of representing the requirements in a manner usable by the scheduling engine so that it can produce a correct and acceptable timeline. Modeling always requires an in-depth understanding of the payload and how the scheduling engine behaves. In the current operations concept, a payload developer provides an in-depth description of the payload to the group, called the scheduling cadre, which builds the models and eventually runs the scheduler. After several extended dialogs with the payload developer, the cadre builds the models for the payload, produces a timeline, and presents it to the payload developer for review. The payload developer reviews the timeline and requests changes; the cadre tweaks the models and (after sufficient iteration with the payload developer) produces an acceptable timeline.

If payload developers, who already have expertise in the payloads, are to formulate scheduling requests and submit them to a scheduling engine, they must have expertise in the scheduling engine – *this is critical*. Being an expert in the behavior of the scheduling engine means knowing how the engine will react to a given model, and how to build a model to achieve the desired results. ROSE will make the payload developers virtual experts in using the scheduling engine by making them experts in modeling. ROSE uses a request format that is a natural representation of the requirements without adding artificial constraints or constructs. We call this modeling methodology *high-fidelity* or *hi-fi* modeling – the model looks like the real world payload and the engine interprets the model as expected. ROSE provides immediate feedback (when a request is submitted, the resulting timeline is available immediately), thereby exposing users to the workings of the scheduling engine. After only a few submittals, the users will know what to expect from the engine when a request is submitted; i.e., they will become virtual experts.

In addition to supporting "hi-fi" models, ROSE provides a user-friendly interface that is neither laborious nor time consuming and which requires little or no training. The user interface of ROSE is the same as that of the Interim User's Requirements Collection (iURC) system currently being used to collect payload scheduling requirements for at least the

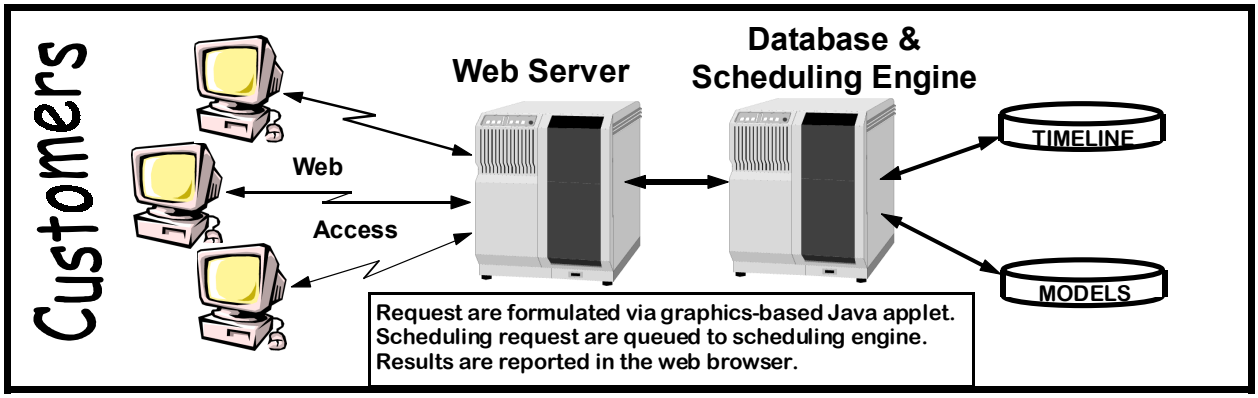


Figure 1. ROSE Architecture Overview

first four increments of the International Space Station. This interface is described in a previous paper (Jaap, Davis & Meyer, 1997) and in the on-line user's manual for iURC. The modeling process *itself* employs graphical methods to describe activities and sequences. The user is presented with

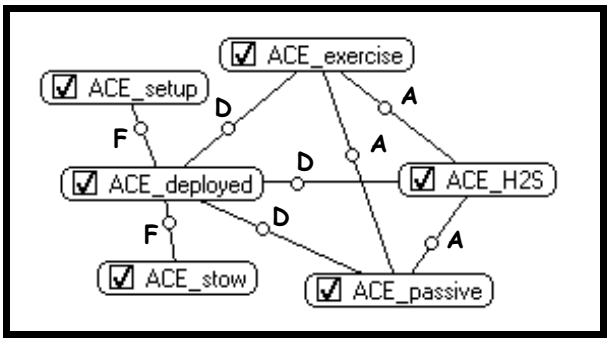


Figure 2. Typical Sequence

a canvas to which items are added and arranged in a hierarchy (for activities) or in a network (for sequences). Details of the requirements are entered via dialog boxes. The advantages of using a graphics method are best illustrated by example. In the typical International Space Station sequence shown in Figure 2, the temporal relationships marked with an **F** are "follows" relationships, those marked with a **D** are "during," and those marked with an ***A*** are "avoid." The sequence indicates that ACE_setup is followed by ACE_deployed, which is followed by ACE_stow; during ACE_deployed, ACE_H2S, ACE_passive, and ACE_exercise are done, but while avoiding one another. ACE_H2S, ACE_passive, and ACE_exercise are themselves sequences. This example is a simple sequence; International Space Station users frequently submit sequences with twenty or more tasks and many relationships, including some to station tasks like reboost and shuttle_docking. The graphics representation is understandable even when extended to large and complex sequences. The modeling methodology supports optional arrangements of tasks within the sequence (sequence scenarios), soft requirements, and choice of constraints within a group with lock-in across sequence members.

Implementation The Java applet provides both request formulation (modeling) and request submittal. Models are stored in Microsoft's SQL Server database hosted on a computer at the site of the web server. When the user selects a model or a portion thereof for editing, it is downloaded from the database via the web server (shown on left side of Figure 3). Edits to the models are posted to the database via the web server whenever the user selects a different activity or sequence, submits a scheduling request, or makes an explicit request to save the data. When posting updates, database stored procedures are used (shown on right side of Figure 3).

A model is pre-checked for errors before it is submitted to the scheduling engine; or a user can request a pre-check at any time. When a sequence is pre-checked, all the referenced sub-sequences and activities are also checked. Checking is done on the web server computer by an ISAPI (Information Server Application Programming Interface)

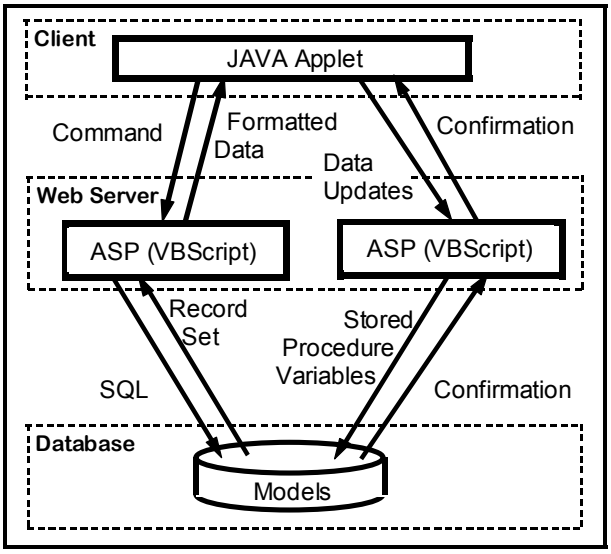


Figure 3. Reading and Writing

extension to IIS (shown in Figure 4). The ISAPI is written in C++. Since the ISAPI is multithreaded, one instance services multiple simultaneous users. A separate database connection is made for each pre-check request.

There is no explicit configuration control of the user's models. A user can always edit his models. However, when a model is added to the timeline (scheduled), an instance of

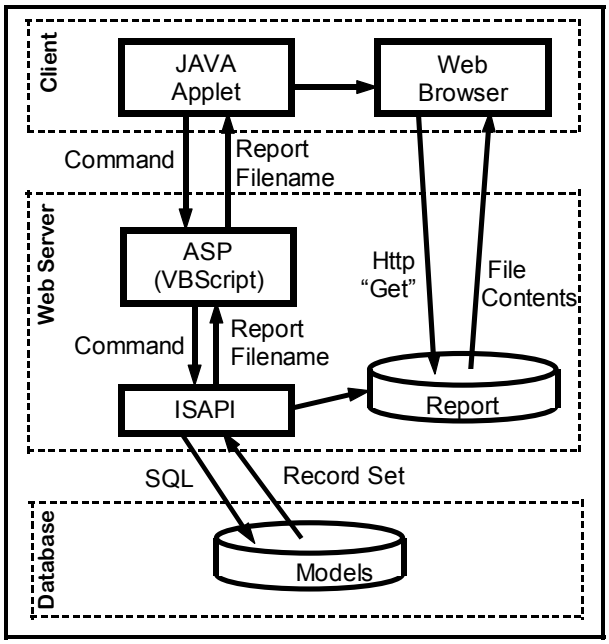


Figure 4. Checking a Model

that model is saved with the timeline and never edited. Thus, editing a model does not edit what is already scheduled, and configuration control of users' models is not required.

Scheduling

While the scheduling engine is the key element of the system, the user should not need to know how it works – the user needs only to hone his modeling skills. The scheduling engine must understand the "language" of modeling perfectly and should behave exactly as expected. Moreover, the scheduling engine must provide feedback (reports for successes and explanation for failures) to help the user improve his modeling skills. To be useful, the ROSE system as a whole must respond to scheduling requests from each of its multiple users in only a few minutes.

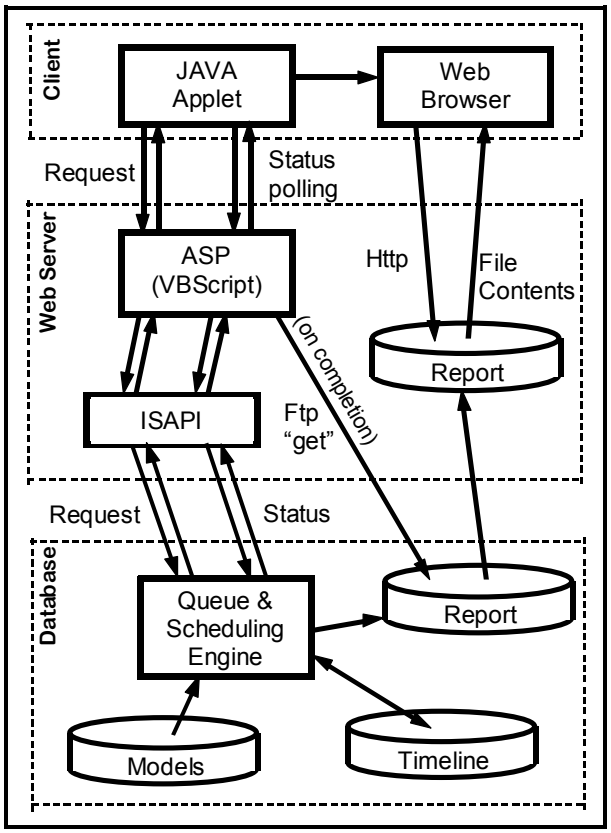


Figure 5. Scheduling Flow

In ROSE, each scheduling request initiates an attempt to schedule one performance of one sequence that may have embedded sub-sequences, repeated tasks, and multiple scenarios. The scheduling request is primarily the data in the model, but the user can override or add additional constraints such as a scheduling window, starting time frame, and scenario specification.

Due to the nature of ROSE, some characteristics of the scheduling engine are compulsory. It must handle the models. It must respond quickly. Everything it schedules must be "valid;" i.e., resources are never oversubscribed and constraints are never violated. Once something is in the timeline, only the user can remove it; the scheduling engine doesn't adjust one task to fit another task into the timeline. It must constrain each account so that its resource allocations are not exceeded. Furthermore, ROSE must not lose anyone's scheduled tasks if the scheduling engine or the computer crashes – scheduled tasks are committed to the database immediately after scheduling

Implementation The Java applet provides the user interface for submitting a sequence to the scheduling engine. When the schedule button is clicked, the model is saved to the database via the web server and is then pre-checked for errors. If the model has no errors, a dialog box permitting the user to make selected model overrides is presented. When the user completes the dialog, the request is sent to an ASP on the web server. The ASP passes the request to an ISAPI that forwards it to the scheduling queue. Only a single instance of the multi-threaded ISAPI runs on the web server; it funnels scheduling requests from all users into the head of a communications pipeline to the scheduling queue at the front end of the scheduling engine. After a request is queued, the applet regularly polls the scheduling engine, via an ASP and an ISAPI, to get the status of the request.

The scheduling engine is a Windows NT/2000 "service" with a control console for setup. The scheduling engine accesses the model, allocation, and timeline databases to determine when, in the timeline, to schedule the request. When the request is scheduled, a report is written on a local file in html format. If the request cannot be scheduled, an explanation is written on a local file in html format. The request is now satisfied, the entry is cleared from the queue and processing begins for another request. A summary of the results is returned to the applet when it next polls the status.

Upon receiving the notice of completion, the Java applet sends a directive to the user's web browser to display the report. For successful requests, the report web page contains a form that is preloaded with the request to delete what was just scheduled. Figure 5 shows the main information flow for a scheduling request.

When a request is scheduled, a copy of the model is stored with the timeline, thus allowing the user to continue editing the model without limitations.

Inspecting the Timeline

Implementation A web-page form provides the user interface to request a display of timeline segments. The user can specify the start and stop times of the report and that only the user's data should be included. The request is sent to an ASP that converts it to a SQL statement that is sent to the database. The resulting record set is reformatted into an html page by the ASP and returned to the web browser. Figure 6 shows the flow of information when inspecting the timeline.

By the time ROSE is deployed, inspection may be based on XML (extensible markup language), with the ASP returning XML data and an XML control in the web browser handling the display of data. Using XML will provide client-side control of what is displayed, thereby significantly increasing the responsiveness of timeline inspection and reducing the processing load on the web server and the database.

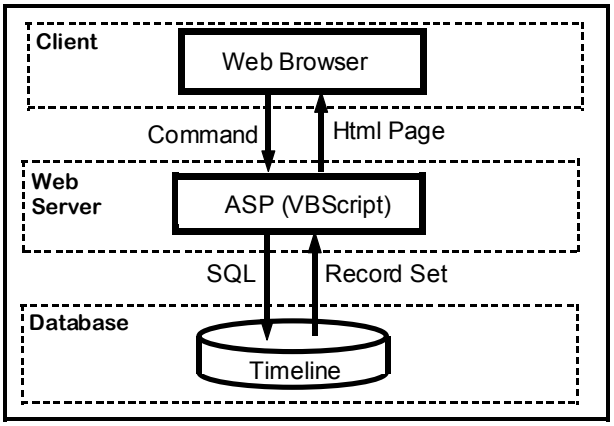


Figure 6. Inspecting the Timeline

Deleting from the Timeline

Implementation The timeline inspection web page provides the user interface for deleting tasks from the timeline. The user selects a sequence performance to delete and sends the request to an ASP on the web server. The ASP forwards the request to the same ISAPI that handles scheduling requests. The deletion request is put into the scheduling queue for processing by the scheduling engine; deletion requests are processed before scheduling requests. Figure 7 shows the flow of information when deleting from the timeline.

Manual Editing of a Timeline

The ROSE architecture does not support true manual editing of a timeline by remote users. A simulation of timeline editing can be accomplished by adding a command to reschedule (delete a specified sequence performance and schedule a performance of a sequence), with its supporting user interface. If the requested sequence could not be scheduled, the original sequence performance would be restored.

Other Applications

Replacement for Current Scheduler ROSE could be used as a replacement for the current scheduler with significant benefit. It would eliminate several of the shortcomings of the current scheduler. However, this approach does not make the payload developers experts in modeling because it does not provide the necessary immediate feedback. The scheduling expertise still exists only in the cadre, and they still have to become near-experts on the payloads they schedule. This shortcoming could be overcome by allowing the payload developers limited access to ROSE.

Standalone What-If Scheduler Payload developers could use ROSE to do "what-if" scheduling. If ROSE were also the scheduler used by the scheduling cadre, then payload developers could become modeling experts; and, by providing usable models, eliminate the requirement for the scheduling cadre to become experts on the payloads.

Job-Jar Scheduler ROSE could fill the need to have a scheduler that allows the crew of the International Space Station to schedule the "job-jar tasks." These are tasks that have been defined by payload developers, but not scheduled; instead, they have been put into a collection of tasks that the crew can do at their discretion. Currently, only tasks that do not utilize shared resources (other than crew) are candidates for the job jar because there is no way for the crew to know payload resources requirements and timeline availabilities. ROSE could easily fulfill the requirements of a job-jar scheduler.

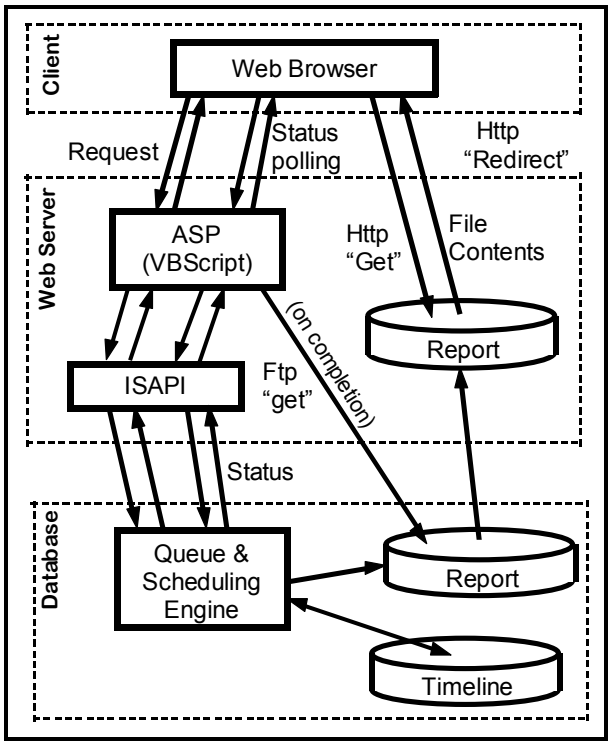


Figure 7. Deleting from Timeline

Summary

Can payload developers schedule their own payload activities? The answer is *YES.* A system is being designed and demonstrated that will provide all the necessary features to support this new approach to payload operations. The key is high-fidelity modeling and a scheduling engine that can schedule the models. The modeling challenge has been met, and the solution is operational in iURC. Feedback from the payload developers using iURC indicates that this modeling approach meets the objective of being able to represent even complex requirements in a straightforward, easy-to-use manner. The World Wide Web provides the needed remote access so that the payload developer community can access ROSE with ease. Standard web software languages and packages provide or enable most of needed features of ROSE.

Status of Research The ROSE project is a research and development effort to investigate and demonstrate a system that addresses all technical issues necessary to allow payload

developers to schedule their own payloads. To date, extensive work has been done on the critical element of modeling and it has been put into operation via iURC. The system architecture has been designed and demonstrated. A prototype of the scheduling engine has been developed, but it needs some rework to exactly match the modeling. Security issues are being investigated, but features such as firewalls are standard fare and will not be demonstrated. An end-to-end demonstration with a stubbed-out scheduling engine has been done.

Security Security of the ROSE architecture is a major concern. ROSE is a web-based system that directly affects spacecraft on-board operations. Therefore, security remediation has been integrated into the research from the beginning. Since ROSE may become a reality, it is not wise to reveal the tailored safeguards which are being developed. However, some of the standard safeguards that might be included are firewalls, perimeter security of the host computers, stringent password rules, challenge-response recognition of users, client computer certificates, address recognition of client computers, and secure socket layer communication.

Paradigm Shift Letting payload developers schedule their own payload activities is a paradigm shift for NASA. While it is clear that the ROSE approach will provide better customer satisfaction and that cost savings could be realized, a solution to the programmatic issues has not been embraced. How will the success of the mission be ensured? How will NASA ensure that the operations are safe, that they meet international and other agreements, that the timeline will be acceptable to all parties, and that scare resources will be shared equitably? A paper is being prepared which addresses these questions (Jaap & Muery, 2000).

References

Jaap, J.; Davis, E.; and Meyer, P. 1997. Using Common Graphics Paradigms Implemented in a Java Applet to Represent Complex Scheduling. In proceedings of International Workshop on Planning and Scheduling for Space Exploration and Science. 20-1—20-3. Oxnard, California.
http://payloads.msfc.nasa.gov/iURC/publications

Online User's Manual for the Interim Users' Requirements Collection System:
http://payloads.msfc.nasa.gov/iURC/help

Jaap, J.; and Muery, K. 2000. Putting ROSE To Work: A Proposed Application of a Request-Oriented Scheduling Engine for Space Station Operations. In proceeding of the sixth International Symposium on Space Mission Operations and Ground Data Systems (SpaceOps 2000). Toulouse France: Spacecraft Operations Oriented International Association (SpaceOps). Forthcoming.

Note: This paper is also available at
http://payloads.msfc.nasa.gov/ROSE/publications

First published by the authors under the same title in AIPS 2000, p. 283-290
Copyright (c) 2000, American Association for Artificial Intelligence. Reprinted with permission from AAAI

Dispatchable execution of schedules involving consumable resources *

R. J. Wallace, E. C. Freuder
Constraint Computation Center
University of New Hampshire, Durham, NH 03824
rjw,ecf@cs.unh.edu

Abstract

Earlier work on autonomous systems has demonstrated that schedules in the form of simple temporal networks, with intervals of values for possible event-times, can be made "dispatchable", i.e., executable incrementally in real time with guarantees against failure due to unfortunate event-time selections. In this work we show how dispatchability can be extended to networks that include constraints for consumable resources. We first determine conditions under which a component of the network composed of resource constraints associated with a single sequence of activities that use a resource ("bout") will support dispatchability. Then we show how to handle interactions between resource and temporal subnetworks to insure dispatchability and how to handle sequences of bouts interspersed with resource release. The results show that flexible handling of resource use can be safely extended to the execution layer for more effective deployment of consumable resources.

Introduction.

From both an intellectual and practical standpoint, the development of autonomous systems that can schedule their own operations is one of the most important areas of contemporary artificial intelligence. In this domain a *de facto* standard appears to have emerged, in which the overall task of plan creation and execution is apportioned to two distinct components, or "layers" of the system, a high-level Planner-Scheduler and a lower-level Executive. The Planner creates a plan together with an associated schedule of operations. This schedule is passed to the Executive, which carries it out by initiating execution of physical components of the system at designated times.

In the Remote Agent planning system developed at NASA-Ames and currently deployed for experimental testing in the Deep Space 1 spacecraft, the Executive is given certain leeway in selecting times for scheduled operations. This is necessary to adjust the schedule to the actual conditions of execution. For example, failure of a rocket engine to fire immediately could break a schedule that did not allow some slack in activity times. This is done by sending time bounds for each event to be scheduled and allowing the Executive to choose a specific time within each pair of bounds (Mus94) (MNP98).

Because the Executive acts in real time, the constraints on its operation are severe. In particular, during the instantiation of a schedule the Executive cannot afford to backtrack, i.e. it cannot reschedule earlier activities whenever its previous decisions have caused it to reach a point where there are no options (a 'dead end'), because these earlier activities may have already begun. For this reason, when actual plan execution begins there must be guarantees that a schedule derived from the time-envelopes is executable incrementally or "dispatchable". That is to say, regardless of the event times that are selected by the Executive, the result must be a viable schedule. In recent work it has been shown that consistent temporal constraint networks, which are a basic component of the Planner-Scheduler's output, can be made dispatchable (MMT98).

At present, flexibility of execution can only be provided with respect to temporal constraints. One would like to provide this flexibility for resource use as well, with similar guarantees. In the present work we extend the notion of dispatchability to networks that include constraints for *consumable* resources in addition to temporal constraints. An example of the kind of resource we are concerned with is the solid state recorder that is used in spacecraft to store data from recording devices prior to transmitting it to earth. In this case there is a series of activities involving data storage (resource use), punctuated at intervals by activities in which data is transmitted, thus freeing storage space (resource release). Here, the problem is to insure that resource capacity is not exceeded regardless of the start and end times that are chosen for these activities during plan execution.

As would be expected, establishing dispatchability for the resulting simple temporal plus consumable re-

*Carried out in association with NASA-Ames Research Center, Moffet Field, CA and supported in part by NSF Grant No. IRI-9504316. A version of this paper will appear in the AIPS-2000 Proceedings.

source network (STN-cRN) is less straightforward than the analagous problem for the STN alone. We handle this problem through a series of decomposition strategies. Specifically, we establish conditions for dispatchability that pertain to the cRN alone, then for effects of each component on the other (STN→RN and RN→STN interactions), and finally for successive 'bouts' of activities that use the resource, that are separated by instances of resource release.

In Section 2 we describe the STN-cRN. Section 3 discusses conditions for dispatchability involving the cRN. Section 4 discusses how to insure dispatchability for an STN-cRN network, despite possible interactions between the STN and RN. Section 5 extends the analysis to successive bouts of resource use. Section 6 gives conclusions.

A temporal-consumable-resource network

In the present Remote Agent scheduling system, the Executive receives an envelope of acceptable scheduling times in the form of an STN (Figure 1). The key feature of such networks is that each event is associated with a single interval. This insures that the network is tractable, since it can be transformed into a digraph and solved with shortest path algorithms (DMP91).

That the STN in Figure 1 is *not* dispatchable can be shown by a simple example. As in the original work by (MMT98), we assume that during execution an event x is selected from a pool of candidate events whose antecedents have already been instantiated, and that the current time is now within the interval bounded by the earliest and latest possible times for events in this candidate set. In addition, constraint propagation can take place after an event has been given a specific time of occurrence, and is restricted to adjacent nodes in the network. In the following example, instantiations are shown on the left and results of propagation on the right, in terms of the acceptable interval for events whose nodes are adjacent in the constraint graph.

a = 0	b = 4-9, c = 4-6
b = 5	d = 7 − 9
c = 6	e = 10 − 13
d = 7	f = 14 − 17
e = 13	g = 18 − 23
f = 17	g = ?

Here, legal assignments to d and e propagate to f and g, respectively, producing non-overlapping intervals for their domains, which causes execution to fail when the constraint specifying equality of the times for these latter events becomes active.

Figure 2 shows a dispatchable network derived from the STN of Figure 1. Inspection of the figure shows that an added explicit constraint between d and e prevents e from taking the value of 13 if d is given the value 7, as in our example. (For formal arguments that such a network is always dispatchable, the reader is referred to (MMT98).)

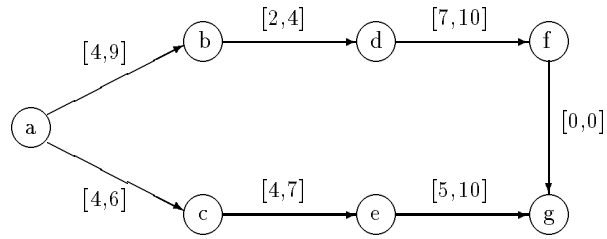


Figure 1: A consistent simple temporal network.

Resource constraints are incorporated into the data structure sent to the Executive via a separate subgraph with different characteristics (the cRN). In this case, intervals represent bounds on resource use for a given activity. For example, in Figure 3 each interval, [10,20], represents a range of possible use of a resource between 10 and 20 units. In addition, k-ary constraints between endpoints prevent the resource capacity from being exceeded. In the present example, the capacity is 30 resource units, and the sum of the upper bounds exceeds capacity by 10 units. Therefore, if activity x starts before y, and if the duration stipulated for the former activity results in its using more than 10 resource units, then the upper bound of y must be reduced by the excess amount to satisfy the constraint between x and y.

In the full data structure, cRN nodes are linked to STN nodes that correspond to the same activity (Figure 4). Resource use is assumed to be a nondecreasing function of time, and here we assume a linear relation, specifically, multiplication of the temporal bounds by a positive or negative quantity for resource use and release, respectively. Of importance is the fact that the mapping from STN event to associated resource use is bijective, i.e., one-to-one and onto, and monotonic. The linkage between STN and cRN is indicated by the dashed lines in Figure 4, each labeled with its constant of proportionality.

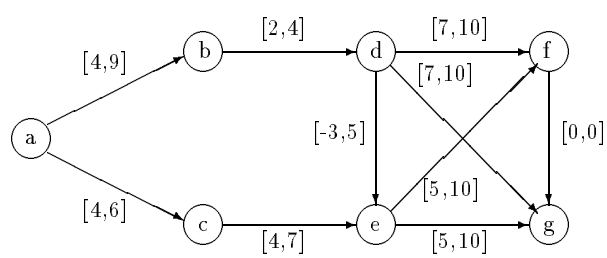


Figure 2: The temporal network of Figure 1 made dispatchable.

Both here and in what follows, we focus on a single consumable resource. If there is more than one such resource, then each is associated with a separate cRN,

and each member of the set of cRNs is connected to the STN as shown in Figure 4.

Before beginning the discussion of dispatchability of the composite network, it is important to note that the tractability of this network as a constraint satisfaction problem is not in question. This is because all constraints in both the temporal and consumable resource subnetworks are in the same tractability class, which (JCG95) refer to as Class 2 (constraints closed under binary operations that are associative, commutative, and idempotent).

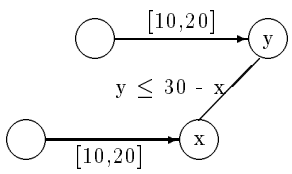


Figure 3: A consumable resource network.

Making the cRN support dispatchability.

If activities can either consume or release a given resource, then the entire sequence of such activities can be divided into 'bouts' of resource use separated by instances of release. In the next two sections we will confine our attention to a single bout of resource use and its associated cRN, and conditions for dispatchability will be specified within this context. In a later section we show that the conditions for dispatchability discovered for a single bout can be extended in a straightforward way to an entire sequence of activities.

To support dispatchability, a cRN must allow any sequence of instantiations to be made in the 'mother' STN without resource capacity being exceeded. Given the bijective mapping from STN event to resource use, this implies that any choice of value for an instance of resource use must allow some values to be chosen for all future (uninstantiated) variables. We refer to this loosely as "cRN dispatchability".

For a single bout, the simplest sufficient condition for cRN dispatchability is that the sum of the upper bounds on resource use be less than or equal to the initial resource capacity,

$$\sum_{i=1}^{k} ub_i \;\leq\; C_{init} \qquad (1)$$

Obviously, in this case the Executive does not need to process the cRN at all, since whatever values it selects from the STN, the resulting resource use will be within capacity.

Unfortunately, this simple condition (that we will call condition (i)) puts limitations on the range of choices that can be given to the Executive. This can be seen in Figure 3, where the sum of upper bounds (40) is well above the capacity (30). Nonetheless, the cRN in this figure is dispatchable, because for every value of resource usage chosen for activity X there is a usage value for activity Y within the designated bounds. Here, dispatchability obviously depends on the constraint between x and y. This suggests a weaker condition for dispatchability that at the same time allows more flexibility in the initial upper bounds on resource use by single activities.

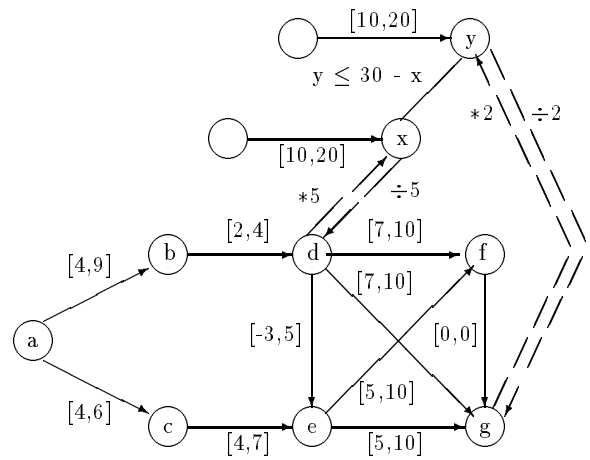


Figure 4: Combined temporal-and-consumable-resource network, in which intervals for duration (STN) and resource use (cRN) associated with the same activity are linked together. Such links are indicated by dashed lines; the linked intervals are those associated with arcs directed *toward* the nodes of origin and destination for the cross-links. For example, activity x in the cRN is associated with arc (b,d) in the STN.

For a set of k activities that use a given resource, this condition (termed condtion (ii)) can be stated as follows. For all subsets of $k-1$ activities, the difference between the initial resource capacity and the sum of the upper bounds of usage for these activities is greater or equal to the lower bound of usage for the remaining kth activity. Putting this in a form corresponding to the first inequation,

$$\sum_{i=1}^{k-1} ub_i \;+\; lb_k \;\leq\; C_{init} \qquad (2)$$

That the time complexity for determining whether this condition holds is no greater than that required for the first condition is shown by the following argument. There are O(k) subsets of size $k-1$, and these can be generated in sequence by swapping single activities in and out, and respective sums after the first can be generated by single additions and subtractions. This indicates that dispatchability in this sense can be determined for a set of k activities in O(k) time. (Note

also that this processing will be done during the planning phase, where time constraints are not as severe.)

Adjusting upper bounds when we find that the dispatchability condition is violated also appears to be easy, at least under some conditions. Thus, if the sum for one of the subsets is $\leq$ the capacity, then the upper bound that does *not* appear in the sum can be reduced to insure dispatchability. The amount it must be reduced is equal to a difference of differences. Suppose a is the element in question, i.e. the activity whose *lower* bound is in the sum that is $\leq$ the limit. And suppose that b is the element whose lower bound is in the sum that exceeds the limit by the greatest amount. Viz,

$$\sum_{i=1}^{k-2} ub_i \ + \ ub_b \ + \ lb_a \ \leq \ C_{init}$$

and

$$\sum_{i=1}^{k-2} ub_i \ + \ ub_a \ + \ lb_b \ > \ C_{init}$$

Then the upper bound of a can be reduced by the following amount to insure dispatchability for this set:

$$(u_a \ - \ l_a) \ - \ (u_b \ - \ l_b) \qquad (3)$$

Since $(u_a - l_a)$ must be greater than $(u_b - l_b)$, we know that this difference must be positive, and obviously it is $\leq$ the original difference between u_a and l_a. If there is no sum less than the limit, it is sufficient to choose a sum, reduce one or more upper bounds until the condition in equation (3) is met, and then use the above procedure.

This condition appears to be the weakest one possible that is still practical. Consider the next weakest condition,

$$\sum_{i=1}^{k-2} ub_i \ + \ lb_{k-1} \ + \ lb_k \ \leq \ C_{init} \qquad (4)$$

To insure dispatchability in this case, one must check $k(k-1)$ subsets of $k-2$ activities. In addition, instead of adding one constraint in the cRN, one must add k constraints to insure dispatchability. Obviously, the situation will be worse with still weaker conditions, and, although for sums of a few upper bounds, the number of subsets to test decreases, the number of constraints to add does not.

Handling Interactions between STN and cRN.

For purposes of schedule execution, the STN and cRN are combined into a single connected network (cf. Figure 4), so that changes in either component can affect the other. Therefore, to establish dispatchability in this network, we must consider interactions between these basic components. (Note that we are still considering a single bout of instances of resource use prior to release.) The basic problem is that propagation in one component that leads to domain restrictions can, in turn, lead to restrictions in the other component that can compromise the conditions for dispatchability. Specifically,

1. Reductions in cRN upper bounds may delete values in the STN that are necessary to insure dispatchability in the temporal subnetwork.

2. Increasing a lower bound of an STN interval may require an increase in the lower bound of the corresponding resource interval in the cRN, thus violating cRN dispatchability condition (ii).

In this section we describe procedures that can be followed during execution to avoid compromising dispatchability in these ways. Since these are different for the two kinds of interaction, each is described in turn.

For the cRN→STN interaction, the following observation is pertinent. If changes are made to the STN, the only part of the graph we have to worry about is between the point of change, which we will call the "critical point", and variables that are already instantiated (i.e. events that are already fixed). Dispatchability will still hold with respect to future domains by virtue of the original STN dispatchability. Now, when we detect that a resource constraint may be violated, if instead of lowering the upper bound of a future resource-interval in order to satisfy that constraint, we change the upper bound associated with the variable currently being instantiated, then we reduce the 'dangerous' region of the STN (variables with domains that might contain unsupported values) to NULL. Moreover, condition (ii) for cRN dispatchability insures that we will not have to reduce any upper bounds for resource use until we arrive at the penultimate member of the set of activities - regardless of the order in which these activities are fixed. In this case, reduction of the upper bound of the penultimate activity cannot compromise dispatchability, given the dispatchability of the original STN and the bijective character of the mapping of temporal onto the resource intervals.

To insure that the STN→cRN interaction does not compromise dispatchability, before selecting a temporal value for an event we must ascertain that this will not lead to an increase in any lower bound for the interval of a future activity that uses the resource. Given condition (ii), the possibility that increasing a lower bound for resource use will compromise dispatchability does not even arise until one reaches the last activity in the bout. This means that if a subset of the activities associated with use of a resource can be designated as "candidate-last activities", then we do not have to consider this problem unless one of these activities is affected. Alternatively, we can consider the set of "candidate-penultimate activities", and in this case we can coordinate the set of STN domains with the set of cRN domains that are relevant to the prior interaction.

Both interaction problems can be solved, therefore, if it can be guaranteed that when we encounter a situation where a change can compromise dispatchability by limiting future options, we can always choose a value that will not have this effect. Fulfilling the requirement that such values always exist is simplified by the following theorem.

Theorem 1. *The requirements, that the lower bound be present in the penultimate cRN domain, so that the final domain does not need to be adjusted, and that there will be a value in a temporal domain that does not necessitate increasing the lower bound of an adjacent domain in the candidate-penultimate set, each imply the other.*

Proof: Given the cRN guarantee, the bijective, monotonic mapping from STN to cRN implies that the original lower bound will be present in the corresponding STN domain and, therefore, that values were present in adjacent domains to support this value. Although the cRN guarantee involves a specific penultimate activity, it must hold for any activity that might become the penultimate one. It therefore pertains to the same set of activities as the STN guarantee. Conversely, the STN guarantee is that all lower bound values in the candidate-penultimate set can be supported, and this implies the cRN guarantee by virtue of the bijective, monotonic mapping.

Given this theorem, a demonstration of either guarantee is sufficient to solve the 'interaction problem'. We will show how to guarantee STN lower bounds.

First, we must determine which activities fall into the candidate-penultimate set This can be done as follows. First find the resource-activity in the current bout with the latest end-time. If this activity doesn't overlap with any other resource-activity, then it needn't be considered, and one can start with the next-latest activity. After locating the first activity to be considered, we must also find all other resource-activities whose time bounds overlap with the first. Together, these comprise the candidate-penultimate set.

Now, the only situation where the lowest value in a critical domain might necessarily be increased is one with, (i) a variable, or node, C that represents the end-time of a resource-activity in the candidate penultimate set and, (ii) an arc (constraint), AC to that node from a node other than the start-time, B. Moreover, there will only be a problem if constraint AC forces the end-time C to be greater than a given value, without putting similar constraints on the start-time. In this case, depending on the start-time chosen, the end-time and hence the interval-duration can be forced to take a value greater than the minimum. This can be avoided when the STN is made dispatchable by replacing the constraint between A and C with one between A and B, the start-time for the same activity. This can be done (given the triangle inequality) if $|AB| + |BC| = |AC|$ (MMT98). We will assume that this can be done during the planning stage, where there is more time for processing and even undoing results to meet this criterion. As a result of this manipulation, both the start- and end-times are subject to the same constraint, so the restriction on end-times that we must avoid cannot occur. If this is done for each such situation involving a candidate-penultimate activity, then this establishes the guarantee.

The following idea allows us to generalize these guarantees so that values need not be present to support lower bounds, but only a specified lowest value.

Definition 1. *We will refer to the accumulated difference between the original upper bounds for resource use, u and the actual usage r,*

$$\sum_{i=1}^{j} ub_i \; - \; \sum_{i=1}^{j} r_i,$$

as the (accumulated) credit that we may apply in the future when choosing values for resource use.

By "applying credit", we mean that one can allow for more than minimal usage, in effect increasing the lower bounds, as long as one does not exceed the credit. (In this case, of course, we must reduce the quantity of credit that is available by an amount equal to the increase in the lower bound.) An important special case is when the credit equals or exceeds the maximum excess use. Since we can calculate the latter quantity before execution by subtracting the capacity from the sum of upper bounds, we can compare this with the credit during execution. If at any point during a bout of resource use, the credit exceeds this quantity, then dispatchability cannot be compromised by any further choices of values for this bout.

More generally, the quantity of credit can be used to relax requirements on changing the upper bound for an activity in the candidate-penultimate set. In this case, we can select a value if the consequent decrease in the upper bound for the last activity is less than or equal to the credit.

With these procedures we can insure dispatchability in the combined STN-cRN network with only a very modest restriction on the 'free-wheeling' execution that was possible with the STN alone. That is, we must introduce a degree of look-ahead into the procedure in order to handle the cRN-STN interaction. Fortunately, condition (ii) insures that look-ahead will be fairly restricted. To handle cRN→STN interaction, at the time when a penultimate STN node is considered for instantiation, the Executive must check a node in the cRN adjacent to the associated cRN node. On the other hand, since the STN→cRN interaction is taken care of before execution, no look-ahead is required to effect the associated guarantee. Moreover, if we are able to

set up a schedule so that only a limited number of activities can ever become the penultimate activity, we can also reduce decision making during planning.

One other potential restriction on dispatchable execution with a simple STN must be mentioned. In the original description of dispatchable networks by (MMT98) the authors describe a procedure for deriving STNs with the minimum number of arcs consistent with dispatchability. In the present situation, although it is still possible to derive a "minimal network", this might not include all the arcs that represent activities associated with resource use. As a consequence, it would sometimes be more difficult to calculate resource use appropriately during schedule execution. In many cases, where activities associated with resource use are a small minority of all activities to be scheduled, the use of networks that are not completely 'minimalized' will probably have only minor effects on execution efficiency.

Activity Sequences with Resource Release.

The conditions described in previous sections pertain to schedule instantiation involving a single bout of resource usage, either before the first instance of resource release, or between such instances if these latter return the capacity to its initial value. Complications arise when the capacity is not restored to its original value. For one instance of release, dispatchability conditions associated with resource use can be expressed in terms of C_{new},

$$C_{new} = C_{init} - \max(0, \min(\sum_{R_{prev}}, C_{init}) - lb_p) \quad (5)$$

where lb_p is the lower bound for release and the sum on R_{prev} is a sum of upper bounds on resource use prior to release.

More generally, we have the following nested recurrence relation for the ith instance of resource release,

$$\begin{aligned} C_i = C_0 - \max_i(0, \max_{i-1}(0, \ldots \max_1(0, \min(\sum_{R_0}, C_0) - lb_{p_1}) \\ + \min(\sum_{R_1}, C_1) - lb_{p_2}) \ldots \\ + \min(\sum_{R_{i-1}}, C_{i-1}) - lb_{p_i}) \end{aligned} \quad (6)$$

This condition on dispatchability is conservative. However, during execution successive precise limits on capacity can be calculated when the values for usage and release are established. In this case, the formula is simply,

$$C_i = C_{i-1} - max(0, \sum_{R_{i-1}} - P_i) \quad (7)$$

The argument above does not consider situations in which resource use overlaps release. In such cases, a simple ordering by start times can be used to allocate such activities to bouts so that dispatchability conditions can be calculated correctly.

Summary and Conclusions.

In this paper we have shown how to extend the important property of dispatchability to the case where consumable resource constraints are involved. Moreover, we have shown that it is possible to allow the same kinds of flexibility with respect to feasible values that is possible with simple temporal networks, as demonstrated in the work of (MMT98).

Not surprisingly with a more complex network to process and with more conditions to test, the procedure during execution is more complicated than it was with STNs alone. However, it appears that by using the strategies outlined above, it is possible to minimize effects on efficiency of schedule execution.

The present discussion is pertinent to a large class of problems encountered in planning by autonomous systems such as spacecraft, of which the solid state recording problem is one example. In these systems, greater flexibility of resource use can make operations involving tasks such as data collection more effective, thus increasing the likelihood that overall mission goals are accomplished.

References

R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

P. Jeavons, D. Cohen, and M. Gyssens. A unifying framework for tractable constraints. In U. Montanari and F. Rossi, editors, *Principles and Practice of Constraint Programming - CP '95*, volume 976 of *Lecture Notes in Computer Science*, pages 276–291. Springer, Berlin, 1995.

N. Muscettola. HSTS: Integrating planning and scheduling. In M. Zweben and M. S. Fox, editors, *Intelligent Scheduling*, pages 169–212. Morgan Kaufmann, San Mateo, CA, 1994.

N. Muscettola, P. Morris, and I. Tsamardinos. Reformulating temporal plans for efficient execution. In *Proceedings KR-98*, 1998.

N. Muscettola, P. Pandurang Nayak, and B. Pell. Remote agent: to boldly go where no AI system has gone before. *Artificial Intelligence*, 103:5–47, 1998.

Features of an Onboard, Adaptive, Observation Plan Executive for NASA's Next Generation Space Telescope

Gary Welter
Computer Sciences Corporation, 7700 Hubble Drive, Lanham Seabrook MD 20706, gary.welter@gsfc.nasa.gov

Leslye Boyce, Lou Hallock, Keith Kalinowski, Ken Rehm, and Steve Tompkins
National Aeronautics and Space Administration, Goddard Space Flight Center, Greenbelt MD 20771

Glenn Cammarata
Space Systems Integration, Goddard Space Flight Center, Greenbelt MD 20771

Jim Legg
Raytheon, Goddard Space Flight Center, Greenbelt MD 20771

Vicki Balzano, Rodger Doxsey, John Isaacs, and Ray Kutina
Space Telescope Science Institute, 3700 San Martin Drive, Baltimore MD 21218

Abstract

Targeted for launch in 2008, NASA's Next Generation Space Telescope (NGST) will be a premiere research tool for extending mankind's understanding of the early universe. Largely because of its planned position at the relatively constraint-free second Earth/Sun Lagrange point (L2), the mission offers a unique opportunity for enhancements in science efficiency and spacecraft operability via the use of an onboard observation plan executive (OPE) that makes real-time plan and schedule adjustments based on immediate mission conditions. The goals for including such a system are to simplify ground system procedures for observatory scheduling and operations, enable more efficient use of observatory resources, and protect against the as-flown characteristics of the vehicle. The OPE's usual function will be the smooth, event-driven execution of a nominal observation plan provided by a ground-based scheduling system, but it will also have the authority and responsibility to autonomously inject standard housekeeping activities as they are needed, and to respond to various anomalous real-time conditions that may arise. The latter may take the form of suspending plan execution while taking corrective actions, deleting elements of the plan that are found to have become nonviable, or perhaps even inserting new elements into the plan as opportunities arise. This paper discusses issues and strategies pertinent to OPE operation. To provide a definite context within which the envisioned OPE will function, we include a proposal for a specific distribution of overall flight software functionality and a description of an interface protocol between the OPE and the other flight software applications. We also discuss both nominal OPE operations, as well as OPE responses to various anomalous conditions.

1. Introduction

The Next Generation Space Telescope (NGST), projected for launch in 2008, is a key component of NASA's origins program (cf. Stockman 1997). Its principal purpose is to enable studies of the cosmological "dark ages" at times and distances from just beyond those probed by the Hubble Space Telescope (HST) to near the "recombination" epoch studied by the Cosmic Background Explorer. Current plans are for NGST to be a large-aperture (~ 8 m) infrared observatory located near the second (i.e., anti-sunwards) Earth/Sun Lagrange point (L2). The technological challenges for creating the required hardware systems are substantial, particularly in areas such as assembly and figure control of the segmented primary mirror, cryogenic operation of complex space systems, and construction and rigidification of large, low-weight space systems.

Placement of the observatory in a halo orbit about L2 offers certain very appealing features for operations. In particular, it eliminates all of the moderate-frequency (i.e., near hourly) orbit-related constraints that plague missions in low Earth orbit, including frequent target occultation by the Earth, increased electronics interference during passage through the South Atlantic Anomaly, and thermal and power stresses related to regular passage through the Earth's shadow. Sun related constraints with year-long periodicity do still pertain. Additional advantages related to L2 placement for most of the proposed NGST mission designs include abundant and continuous solar power and continuous viewing of the Earth by the antenna system.

This paper presents a concept for an event-driven observation plan executive (OPE) that gives a moderate level of planning and scheduling autonomy to the observatory flight system. The goal is to achieve a balanced system that exploits the inherent near-term flexibility associated with the L2 location while at the same time relying on the ground segment's greater computational power to optimize the nominal moderate- and long-term science schedule. This contrasts with most current space observatories, including NASA's four Great Observatories (HST, the Compton Gamma Ray Observatory, the Chandra X-Ray Observatory, and the Space Infrared Telescope Facility), which are essentially purely time-based in their schedule execution design.

The presentation is divided into the following sections: (1) this introduction, (2) a review of the flight systems that constitute the OPE's operational context, (3) a definition of the structure of the observation plan provided by the ground segment as basic input to the OPE, (4) a description of the communications protocol between the OPE and the various spacecraft applications whose activities it coordinates, (5) a description of OPE operations under nominal conditions, and (6) a description of various important OPE responses to anomalous conditions.

2. Spacecraft Systems Overview

For this study, we assume that the NGST systems will be essentially those envisioned by the Goddard Space Flight Center (GSFC) yardstick version of the mission (cf. Stockman et al. 1997). We do so merely to provide a definite context for discussion; the details are not a strong driver for OPE design. Although many design decisions have not yet been finalized, we use present tense in this paper for smoother presentation.

NGST consists of three principal components: the spacecraft support module (SSM), the optical telescope assembly (OTA), and the integrated science instrument module (ISIM). Figure 1 shows these three components as envisioned for the GSFC yardstick. The large sunshield deployed from the SSM allows passive cooling of the OTA and ISIM to ~ 30 K. During normal operations, the attitude of the spacecraft is constrained so that the OTA and ISIM remain within the shadow of the sunshield.

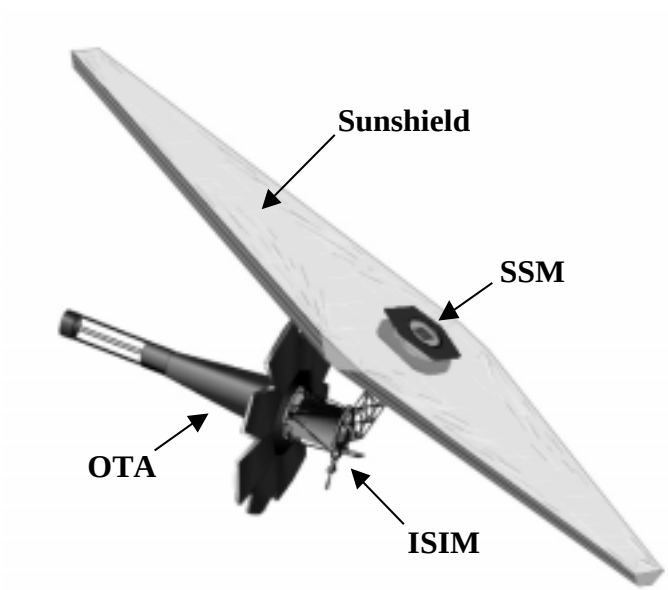


Figure 1 – Yardstick NGST configuration

2.1 – SSM Components

The following SSM systems are important from the OPE's perspective: health and safety control (HSC), communications, command and data handling, and attitude and orbit control (AOC). SSM thermal and electrical power systems also exist, but they are transparent to OPE operations.

HSC is of interest because it has ultimate responsibility for spacecraft health and safety and can therefore deactivate the OPE in emergencies. Communications is important as the conduit of ground-generated control information (observation plans and real-time commands). However, given the L2 mission location, the antenna can be designed to support communications at all valid spacecraft attitudes, which avoids observation schedule constraints associated with communication needs.

Command and data handling provides two services: (1) a command and data interface between the various spacecraft systems, and (2) a data recorder for storing mission observations. The recorder allows simultaneous recording and transmission of data and therefore usually imposes no schedule constraints on the OPE. However, because the data recorder has finite capacity, the OPE must verify that there is sufficient free space available before initiating the collection of new science data. This becomes an issue only in the infrequent event of problems in the ground system that delay ground collection of the recorded data.

AOC consists of hardware and software elements that monitor and control the spacecraft attitude and orbit. AOC hardware includes Sun sensors, star trackers, gyroscopes, reaction wheels, and gas thrusters. The star trackers, gyros, and reaction wheels together are sufficiently accurate to allow determination and control of the SSM attitude to ~ 1" (r.m.s.) during science observations, although significant levels of jitter are present during and just after large slews and momentum dumps. The reaction wheels are also used to absorb momentum resulting from the external torque imposed by solar radiation pressure on the sunshield; the reaction wheels can absorb the equivalent of ~ 24 hours' worth of torque at the worst case attitude. Gas thrusters are used to dump the excess momentum periodically. A separate set of thrusters is used for velocity adjustment roughly once every 23 days to maintain the orbit (cf. Lubow 2000).

2.2 – OTA Components

The OTA consists of the telescope mirrors and the hardware and software elements required to monitor and control them. There are five mirrors. The segmented primary mirror and the secondary mirror are open to free space, while the tertiary mirror, deformable mirror, and fast steering mirror are inside the ISIM box. Adjustments to the control actuators of the primary, secondary, and deformable mirrors are made periodically (say, monthly) to compensate for secular changes in their configuration. The OPE does not make decisions regarding when such adjustments are required; these decisions are based on ground analysis. The fast steering mirror is used to compensate for residual jitter beyond AOC control capability. The error signal used for fast steering mirror control is based on high-rate (~ 30 Hz) guide star observations acquired with one of the science instruments.

2.3 – ISIM Components

The ISIM consists of the three science instruments (SIs) and the hardware and software required to monitor and control them. (Strictly speaking, the software elements and much of the control hardware are located on the SSM side of the truss used to isolate the ISIM and OTA from thermal and mechanical disturbances, but this is a mere architectural "detail" from the perspective of the OPE.) The three SIs are the near infrared camera (NIRC), the near infrared spectrograph (NIRS), and the combined mid infrared camera/spectrograph (MIRC/S). Each of the SIs has a multi-configuration optical/mechanical system for transmission of the light and a focal plane array (FPA) detector system for measuring the incident light.

The NIRC serves double duty. Its 8k×8k FPA consists of 64 separate chips. During science observations, one of the chips is selected to serve as a fine guidance sensor and configured for readout at 30 Hz, with data accumulated from only a pre-selected 100×100 pixel region of the chip. The remaining 63 chips can be used for science data collection. The NIRC actually consists of four cameras, each with a separate optical system and FPA. A broadband filter can therefore be specified for use with the quadrant containing the chip to be used for fine guidance, while narrow band filters are used in the other three quadrants.

The NIRS and MIRC/S both have camera and spectrograph modes. The camera modes can be used either directly for science or as target location devices in support of spectrographic science. For the latter purpose, special target acquisition processing is used to identify desired targets and adjust either the configuration of the science instrument or the spacecraft attitude to place the target(s) in one or more selected apertures.

The ISIM flight software modules to which the OPE typically issues command directives are fine guidance support (FGS), to acquire and hold guide stars; NIRS and MIRC/S target acquisition support (TAS), to place targets in designated spectrograph apertures; and NIRC, NIRS, and MIRC/S science data collection (SDC), to collect and process data during science exposures. Each of these in turn makes use of two SI-specific utility modules: an SI configuration monitor and control (CMC) module, and an SI FPA data processing (FDP) module. The CMC modules are used for issuing configuration change commands (e.g., move focus relay mirror, change filter, activate test lamp, and activate FPA high voltage) and monitoring SI engineering telemetry to verify that the commands have executed properly. The FDP modules are used for applying basic data processing functions to the FPA data in support of the various high-level operations.

The duration of science exposures is limited by the level of cosmic ray background radiation at L2. The background flux is estimated to be such that ~ 10% of image pixels would be contaminated in exposures lasting 1000 s, which is currently taken as an approximate upper limit for exposure times (cf. Isaacs, Legg, and Tompkins 1999). Occasional solar flares or coronal mass ejections raise the background particle flux sufficiently to render observations useless, or even endanger the SI FPAs if detector high voltage remains active. Consequently, a radiation monitor is included that informs the OPE regarding the level of the background flux to allow it to either postpone observations or deactivate the SI FPA high voltage, as appropriate.

3. Observation Plan Structure

The ground-based planning and scheduling system constructs an observation plan for the OPE to execute on a regular (say, weekly) basis. The new plan is uplinked and appended to the end of the currently executing plan, so there is no break in execution. Each plan is composed of the following progressively smaller structures: visits, groups, activity sequences, and activities. These components have the following definitions and attributes.

Activity: The smallest logical unit uploaded from the ground for dissemination by the OPE. Some typical activities are *Slew*, *Acquire Guide Star*, *Acquire NIRS Target*, and *Acquire NIRC Exposure*. Each activity has a number of associated parameters, both generic and specific. Generic parameters include the ID of the target application (e.g., AOC for a *Slew* activity), and a flag indicating whether successful completion of the activity is required for the visit as a whole. Specific parameters are those required to define the activity, such as target attitude for a slew, as well as special conditions that must be met before the activity starts, such as sufficiently low spacecraft jitter for a guide star acquisition.

Activity sequence: A set of one or more activities to be executed sequentially, e.g., [*Slew*, *Acquire Guide Star*, *Acquire NIRC Exposure*].

Group: A set of one or more activity sequences to be executed in parallel. This allows, for example, a NIRS dark current calibration in parallel with a NIRC science exposure. All activity sequences within a group must be complete before the next group begins.

Visit: A logically complete set of one or more groups. Visits typically consist of a slew to a particular target attitude, followed by a number of science or engineering activities at that attitude. A visit may also have associated parameters, in particular a set of three time parameters: an earliest permitted start time, latest permitted start time, and latest permitted end time. These can be used, for example, to synchronize the visit with an anticipated astronomical event or with related observations by other observatories. Being logically complete, a visit is the natural unit to drop if one of its required components is found to be nonviable.

Auxiliary Visits: Although not expected to be used for the NGST mission, a natural extension of the OPE concept is the use of an onboard pool of auxiliary visits, provided by the flight operations team (FOT), that the OPE can use to fill gaps in the plan that may arise as a consequence of the OPE dropping one or more nonviable visits that precede a fairly tightly time constrained visit. We do not examine this extension here; see Welter, Legg, and Cammarata 1999 for further discussion.

4. OPE Communications Protocol

Four types of information exchange are of interest for the OPE: OPE control directives, activity validation dialog, activity execution dialog, and state change notifications. For the following discussion, a "directive" is a high-level instruction from some control entity (e.g., the OPE) to an application (e.g., AOC) to perform some function.

The OPE control directives are the mechanism by which a higher level authority (i.e., the FOT or HSC) controls the behavior of the OPE. In addition to the basic directive for appending a new observation plan to the current one, there are OPE control directives to suspend and restart the OPE's ability to issue directives, to cause it to suspend or terminate any outstanding directives that it has issued, and to cause it to smoothly shut down its own function.

An activity validation dialog takes the form of the OPE directing a flight software application to confirm that an activity is valid for execution. For each defined activity directive, the application has rules for ascertaining whether or not the activity is currently valid. For example, AOC is able to ascertain whether a slew target attitude violates Sun constraints. Each application responds to an activity validation directive with an indication of *valid* or *invalid*.

An activity execution dialog consists of the OPE directing an application to execute a specified activity. For safety, the application reapplies its validation procedure to confirm that the activity is valid. If it is valid, the application attempts to execute the activity. Upon completing the activity, the application issues a response message indicating that the activity *succeeded*, *failed*, or was *terminated prematurely* (e.g., by a subsequent OPE directive).

State change notifications are spontaneously generated messages issued by the various flight software applications that inform the OPE of important system state changes that are not the specific consequence of an earlier OPE execution directive. These include such notifications as *spacecraft jitter in low range*, *guide star lock lost*, *system momentum in high range*, and *external radiation in medium range*. This state information is used by the OPE to coordinate observation plan execution flow as well as possible insertion of appropriate response actions.

5. OPE Operations for Normal Conditions

The normal flow for OPE processing of observation plan visits is as follows. The NGST OPE does not review the plan as a whole immediately upon completion of uplink from the ground, rather it validates each visit only immediately before its execution. At that time, the OPE conducts a validation dialog with each of the flight software applications to which an activity in the visit is addressed. Any activity found to be invalid at that time is marked as such by the OPE and suppressed from the plan. If an activity found to be invalid was also marked as required for the visit as a whole (e.g., the slew to the visit target), the visit as a whole is rejected and the OPE moves on to the next visit.

If the visit has not been rejected, the OPE checks to determine whether current time is within the specified permitted start time window for the visit. If current time precedes the permitted window, the OPE delays further processing of the visit until the window is entered. If current time is within the window, the OPE proceeds to the next step of evaluation. If current time falls after the window, the OPE proceeds to the next visit.

After time window verification, the OPE checks on the need for insertion of any house-keeping tasks that (1) will be necessary before the end of the visit and (2) could interfere with the visit. The most significant such house-keeping task is the use of the AOC thrusters to dump angular momentum. Although the details for the NGST momentum management strategy have not yet been formalized, we have assumed the following variation as an example of OPE house-keeping insertion.

A constraint on nominal visit duration is imposed through the ground-based scheduling segment so that the momentum predicted to be accumulated during any visit will be less than the diameter of the permitted angular momentum sphere. With support from an AOC analysis utility, the OPE determines whether the expected momentum accumulation for the visit added to the current momentum load would cause violation of the permitted momentum sphere. If no violation is predicted, no momentum dump is inserted. If a violation is predicted, the analysis utility also computes the minimum dump necessary, actually the associated post-dump target momentum, such that the momentum load will just reach the permitted momentum sphere at the end of the visit. To allow for modest schedule deviations, the momentum sphere for this test is padded inwards slightly to a "yellow" limit, with the pad the equivalent of, say, two science exposures. The parameters for the dump are then packaged as an activity to be executed in parallel with (i.e., combined as a group with) the initial activity of the visit, typically a slew. To allow for the possibility that the initial activity is not compatible with a parallel dump, the OPE has a list of activities that are compatible with parallel momentum dumping. If the initial activity is not on the list, the OPE inserts the dump before the first planned activity of the visit.

The OPE then proceeds to issue the various activity directives for the activities of the visit. This process proceeds on an event-driven basis, thereby alleviating the ground-based scheduler of the need to model activity execution in detail or insert execution time pads into the observation plan. As activity sequences proceed, the next activity can only be started after the OPE has received an activity completed notification from the application executing the preceding activity. Furthermore, even if the prior activity has completed, the OPE delays directing the execution of any activity until required observatory state conditions specified for the activity have been realized. For example, an *Acquire Guide Star* activity can specify that the SSM jitter must be within a "green" range before acquisition can begin, which will have been reported via a

state change notification message by AOC. The OPE also requires completion of all parallel activity sequences within a currently active group before any activities in the next group are initiated.

6. OPE Responses to Anomalous Conditions

The OPE responds to various anomalous conditions. We present five examples pertaining to NGST operations: loss of guide star lock, violation of permitted momentum limits during visit execution, extreme background particle flux, visit end time violation, and attitude constraint violation.

To support the first three types of violations, each of the high-level ISIM applications (i.e., FGS, SI TAS, and SI SDC) support two special generic directives: *Suspend* and *Resume*. A *Suspend* directive causes the application to stop whatever process it is currently doing, reset itself to begin the process again at a natural starting point, issue a directive to the associated FDP utility to stop any support computations, and finally notify the OPE that the *Suspend* directive has succeeded. If an application is not currently active, it remains in the same state and issues the required notification of success.

For most applications, the natural starting point is simply the beginning of the activity, so the whole activity will be repeated. For FGS, the natural starting point is either (1) the beginning of the activity (i.e., to include a search for and verification of the candidate guide star) if the guide star has not yet been acquired, or (2) a simple reacquisition based on the known location of the guide star if the guide star has been acquired.

A *Resume* directive causes an application to proceed forward from its starting point if it is in a suspended state and issue a success or failure response notification message to the OPE. With the exception of FGS, all applications immediately respond to a *Resume* directive with a success notification. If FGS had not yet acquired a guide star, it too responds immediately with a success notification; a subsequent failure to acquire will be reported as a failure of the not yet completed *Acquire Guide Star* activity. However, if the guide star had already been successfully acquired, FGS postpones notifying the OPE regarding the outcome of the *Resume* directive until after it has succeeded or failed in its reacquisition attempt and then respond accordingly.

An anomalous event can result in activity failure. If a failed activity is marked as required for the visit as a whole, e.g., if FGS is unable to acquire a guide star, the OPE terminates the visit and moves immediately to the next visit. To accommodate this design, each application is required to autonomously perform any necessary clean-up tasks and place itself and associated hardware elements in one of a small number of easily identified standard states at the end of any activity.

6.1 Loss of Guide Star Lock

Upon being notified by FGS that a loss-of-lock has occurred, the OPE activates a script that causes *Suspend* directives to be sent to all ISIM applications. After all applications have reported success, the OPE issues a *Resume* directive to FGS. If FGS succeeds in acquiring the star, the OPE issues *Resume* directives to all of the remaining ISIM applications; otherwise, the OPE terminates the visit. To prevent a possible infinite loop of guide star loss-of-lock events followed by temporarily successful reacquisition attempts, the OPE keeps track of the number of loss-of-lock events that occur in each visit and terminates the visit if the number becomes excessive.

6.2 Violation of Momentum Limits

In addition to the yellow momentum limit, two other limits are defined: orange and red. The red limit is sufficiently close to actual reaction wheel saturation that the OPE must immediately respond to it. The orange limit, while still high, is sufficiently below the red limit that there is time to complete any exposures currently in progress, but no more. Given the logic for insertion of house-keeping momentum dumps previously described, it should only be possible for an orange or red limit violation to occur if some real-time anomaly has delayed completion of the visit.

Upon being notified by AOC that an orange momentum limit violation has occurred, the OPE activates a script that will suspend all ISIM applications after any currently outstanding activities have been completed. After all of the ISIM applications have been suspended, the OPE uses the AOC analysis utility previously described to determine parameters for a momentum dump that will reduce the momentum load sufficiently below the yellow limit so that the yellow limit will just be reached again at the predicted end time of the visit. Because the momentum dump will induce vehicle jitter, the script instructs the OPE to refrain from issuing an FGS *Resume* directive until after the "green" jitter range has been reachieved. Thereafter the script is the same as the end of that used for responding to a guide star loss-of-lock.

If the red limit is crossed, the OPE's response is the same except that it does not wait for the outstanding activities to run to completion before issuing the *Suspend* directives. To avoid interference between the red violation response script and the orange script that will have already been activated, the OPE deactivates the orange script when the red script starts.

6.3 Extreme Background Particle Flux

Three background particle flux ranges are of interest: low, medium, and high. Valid SI measurements can be made for low background flux. The SIs are safe but measurements are invalid for medium background flux. The SI FPAs are in danger of high-voltage arcing damage if the background flux is high and the FPA high voltage is on.

When informed of a low-to-medium or medium-to-high particle flux transition, the OPE activates a script that suspends all ISIM applications. If the flux is high, the OPE also issues high-voltage deactivation directives to the three CMC applications to deactivate FPA high voltage. When a high-to-medium transition occurs, the OPE issues

high-voltage activation directives and sets a timer to delay progress in the response script for a specified period. When a medium-to-low transition occurs, the OPE issues an FGS *Resume* directive, followed in due course by *Resume* directives to the remaining ISIM applications.

6.4 Combined Response Scripts

Given the similarity of the various responses just described, one can combine them into a single script with appropriate state condition checks to determine whether certain elements of the script (i.e., momentum dumping or FPA deactivation) should be used. The reception of any one of the anomaly notification messages appropriate to activation of the script would result in termination of any previous version of the script that may already be running and reactivation of the script from the beginning.

6.5 Visit End Time Violations

Delays in the completion of a visit resulting from anomalies such as those described in the previous subsections could result in the violation of the visit-specified latest permitted end time. If this occurs, the OPE terminates the current visit and proceeds to the next visit.

6.6 Attitude Constraint Violations

If the ground-based scheduler does not specify latest permitted end times for all visits, delays in visit completion could result in the spacecraft remaining so long at its current attitude that Sun constraints are violated. Upon detecting an attitude violation, AOC slews the vehicle to a local valid attitude and notifies the OPE. The OPE then terminates the current visit and proceeds to the next.

Summary

We have presented a proposal for an onboard observation plan executive (OPE) tailored for use on the Next Generation Space Telescope. The goal is to produce a system that provides flexible and efficient event-driven execution of observation plans generated by a ground-based planning and scheduling system, as well as robust response to real-time anomalies. The various sections of the paper have described the NGST flight systems that constitute the context within which the OPE will be operating, the structure of the observation plan provided by the ground segment as basic input to the OPE, the communications protocol between the OPE and the various spacecraft applications whose activities it coordinates, OPE operations under nominal conditions, and a set of example OPE responses to anomalous conditions.

Acknowledgements

Variations on the ideas reported here have been evolving over a number of years. We acknowledge, in particular, our colleagues on the NGST Software Operations Working Group, all of whom contributed to the evolving OPE concept. We have also made free use of the related ideas explored in Cammarata et al. 1998, Doxsey et al. 1998, Legg and Welter 1999, and Welter, Legg, and Cammarata, 1999.

Acronym List

AOC	attitude and orbit control
CMC	configuration monitor and control
FGS	fine guidance support
FDP	FPA data processing
FOT	flight operations team
FPA	focal plane array
GSFC	Goddard Space Flight Center
HSC	health and safety control
HST	Hubble Space Telescope
ISIM	integrated science instrument module
MIRC/S	mid infrared camera / spectrograph
NGST	Next Generation Space Telescope
NIRC	near infrared camera
NIRS	near infrared spectrograph
OPE	observation plan executive
OTA	optical telescope assembly
SDC	science data collection
SI	science instrument
SSM	spacecraft support module
TAS	target acquisition support

References

Cammarata, G., Hallock, L., Duran, S., Bruno, C., Myers, P., Welter, G., Stewart, K., & Legg, J., 1998. *Next Generation Space Telescope Adaptive Scheduler Prototype – Lessons Learned*, GSFC technical report.

Doxsey, R., Balzano, V., Henry, R., Isaacs, J., Johnson, C., & Kutina, R., 1998. *NGST Integrated Ground/Flight Software Operations Concept – Version 1*, Space Telescope Science Institute (STScI) technical report.

Isaacs, J., Legg, J., & Tompkins, S., 1999. *The NGST Communications Study: Science Data Volume Requirements and an Assessment of Communications Options*, STScI technical report.

Legg, J., & Welter, G., 1999. *Next Generation Study Space Telescope Adaptive Scheduling Study Report*, GSFC technical report.

Lubow, S., 2000. *Radiation Effects on the Orbit of NGST*, STScI technical report.

Stockman, H. (ed.), 1997. *Next Generation Space Telescope – Visiting a Time When Galaxies Were Young*, STScI technical report.

Welter, G., Legg, J., & Cammarata, G., 1999. *Adjustable Synchronization: A Proposal for Next Generation Space Telescope Operations*, Proceedings of the AAAI 1999 Spring Symposium "Agents with Adjustable Autonomy".

Reactive prioritization

Michael Freed
MS 262-4 NASA Ames Research Center
Moffett Field, CA 94035

mfreed@mail.arc.nasa.gov

Abstract

In many interesting task environments, agents must decide priority among competing tasks under considerable uncertainty. Moreover, which kinds of priority-relevant information are available for a decision will vary in different situations. An ideal priority determination process should use whatever information is available, even it becomes so just before a decision is required or after a task has been awarded priority and begun executing. In this paper, we identify several kinds of priority-relevant of information and describe a flexible priority computation method that uses whatever kinds are available in a given situation.

Introduction

In everyday environments such as the kitchen or automobile, people usually have many things they could reasonably be doing at a given moment. Some of these tasks are independent and can be pursued concurrently. Others interact in ways that demand a choice: which should be given priority and carried out immediately? Which should be deferred, interrupted, or aborted?

For artificial agents, one approach to making such decisions is to identify all tasks to be carried out and all the constraints on those tasks, then search for the best possible order. While this can produce optimal or nearly optimal action orderings, it is only practical in highly predictable environments where needed actions and relevant constraints are known in advance. Many everyday environments are not predictable in this sense. Unplanned actions may be needed to handle unexpected events – e.g. a car suddenly cuts in just ahead, the phone rings, an awful song comes on the radio. Similarly, the timing and specific actions needed to carry out a task may not may not be known until it is nearly time to carry them out. For example, an agent may know that it will have to turn left onto Elm Street, but not know which particular driving maneuvers will be needed to negotiate traffic or whether it will first have to stop at a red traffic signal.

An alternative approach, *reactive prioritization*, is to make rapid priority decisions just before committing to a course of action. Unlike the more deliberative approach in which priority decisions are made arbitrarily far in advance of execution, a reactive prioritization process makes such decisions in response to newly available information about, e.g., which tasks are eligible for execution at a given moment, whether they interact, and what timing constraints apply to each. While these kinds of information are sometimes available far in advance, they often remain uncertain until the last moments before a task becomes enabled.

Pervasive, priority-relevant uncertainty has at least two important implications. The first, as discussed, is that no maximally-informed priority decision for a task is possible until that task is eligible for execution. Thus, priority must be determined at execution-time, not (only) in the course of long-term planning. The second is that priority decisions will often have to be made in the absence of potentially significant information. For instance, not knowing about an impending task to answer the phone makes it impossible to choose between tasks on the basis of which better tolerates interruption. The best an agent can do is to use whatever information is available when a final (or otherwise consequential) priority decision is needed.

In designing reactive priority mechanisms, it is important to consider what types of information are likely to be available at decision-time and how each type should influence task priority. In this paper, we identify several kinds of information that may be available, each of which can be used as a heuristic basis for deciding priority. These heuristics are combined into a more general prioritization process that takes advantage of situations in which more than one kind of priority-relevant information is available. This paper extends work reported in Freed (1998) and has been implemented as part of a plan execution system used to simulate expert human operators doing complex tasks (Freed and Remington, 1997).

Heuristic Prioritization

Tasks need to be prioritized when they conflict. For example, when two tasks require looking at widely separated points in the visual field, prioritization is needed

to decide which gets to control the orientation of visual sensors. In some cases, not getting priority merely implies that a task will have to wait (or be interrupted if it is already ongoing). In others, the task cannot wait; deferring it means causing it to fail. Deferring a task imposes a (possibly zero) expected cost. More specifically, each task can be viewed as having a set of deadlines, each of which incurs a specific expected cost if missed; deferring a task increases the likelihood that one or more of its deadlines will be missed. For instance, the task of getting home from work might involve deadlines for arriving on time for a planned dinner, seeing a televised sports event from the beginning, and having time with the kids before they go to bed. The longer the task of driving home is deferred, the greater the risk of "deadline bust" to each associated deadline. *The job of an agent's priority mechanisms is to minimize the total expected cost of deadline busts across all tasks over an extended period.*

Designing a prioritization mechanism able to do this job in an uncertain task environment requires first identifying kinds of priority-relevant information that might be available at decision-time. In the worst case, the agent only knows the identity of tasks currently being considered for execution. In this situation, there is no choice but to select randomly between competing tasks. A somewhat more benign case occurs if there is information on one priority-relevant task attribute. These attributes include anything that affects the likelihood of missing a task deadline including: (1) the time remaining until a particular deadline bust, (2) the cost of missing a deadline, (3) task duration, and (4) the time cost resulting from task interruption. The role of each of these attributes in determining priority can be expressed as a set of heuristic decision rules as follows (cf. Firby, 1989):

All else being equal, …
- do the task with the nearer deadline first
- do the task with the most important deadlines first
- do the briefer task first
- do (continue) the ongoing task rather than switch

These rules make it possible to decide priority when only one kind of information is available. Ideally, a prioritization mechanism will make use of more information if more is available. To understand how such a mechanism should be realized, it is worth considering each of these information types in greater depth.

Urgency. Prioritization is only an issue because there may be undesirable consequences to deferring a task. The idea of a deadline, after which undesirable consequences will occur, is thus likely to be a central feature of any approach to prioritization. We generalize this idea slightly by treating deadline-pressure as one (very important) source of *urgency*. Others possible sources include value-decay-pressure in which the cost of waiting comes gradually rather than all at once (e.g. dinner slowly cools down and becomes less enjoyable to eat), and hazard-pressure where the likelihood of an undesirable occurrence remains constant for a given interval rather than approaching certainty during a given interval (e.g. lightbulb goes out). These forms of urgency differ in the shape of the expected cost curve. For a deadline, the curve remains relatively flat for a while and then quickly approaches maximum. A hazard function smoothly asymptotes to maximum following a curve of the form $1-b^t$ $\{0<b<1\}$. Value decay tends to increase linearly to maximum. However, in all forms of urgency, longer execution delay means greater expected cost.

Importance. Another crucial factor is the measure of how bad it would be to miss a given deadline. [1] This is especially relevant if meeting a deadline for one task precludes meeting a deadline associated with another. In such cases, the best thing to do is to make the more important deadline – i.e. avoid the effects of the bust with greater negative utility.

Duration. A task's duration affects its proper priority in two ways. First, doing one task requires deferring all competing tasks, thereby increasing the risk of deadline-busts for all these tasks' associated deadlines. Thus, as a task's duration increases, so does opportunity cost of carrying it out. The importance of this effect increases not only with increased task duration, but also with the number of tasks deferred. It is often desirable to execute a number of brief tasks before starting one of longer duration, even if the lengthier task is more important.

A second effect of duration is to reduce the amount of time available before a given task must be started (or resumed). For instance, a 2 minute duration task with a 5 minute completion deadline should be started within 3 minutes; doubling the duration but leaving the deadline constant would allow only one minute to start the task. This urgency-increase effect opposes the effect of increased opportunity cost since a longer duration task apparently needs to be started sooner.

The heuristic "all else being equal, do the briefer task first" applies because the opportunity cost effect is greater than the increased-urgency effect. If we assume that competing tasks have the same number of equally distant and equally costly deadlines, then the only basis for comparison is the amount of increased risk (a monotonically increasing function of deferral time) to

[1] Currently, all forms of urgency are treated analogously to deadline-pressure. Value-decay-pressure is handled by selecting some decay threshold to treat as a deadline-bust. Thus, one might assume that dinner will stay hot for 5 minutes; after this (arbitrary) deadline, it is assumed to have become "not hot" and thus a suffered cost. Hazard pressure is handled by selecting a probability threshold p and treating the time taken for an undesired event to have occurred with probability p as a deadline.

each. For example, assume there are five tasks to do, four with 1 minute durations and one with a 5 minute duration. Doing the latter task first imposes a combined 26 minutes of deferral on all tasks. Doing the task last imposes only 10 minutes. The next section discusses an approach to prioritization that accommodates both duration effects.

Interruption cost. Priority should be recomputed whenever changed circumstances indicate that a previous priority decision may have become obsolete. For example, when a task becomes enabled and thus eligible for execution, its priority should be assessed against any competing tasks including those ongoing. Similarly, situation changes that either lower the likely priority value of an ongoing task or increase the value of a previously deferred task should trigger reassessment of priority decisions. When reprioritization reveals that an ongoing task should have lower priority than a competitor, the task should be interrupted.

However, interruption imposes costs that should be considered when computing the priority of an ongoing task. For instance, interruption may require time-consuming transitional activities to avoid consequences of an overly-sudden interruption – e.g. pulling over to the side of the road to avoid crashing when interrupting a driving task. Other activities may be required to maintain task viability during the interruption interval and to correctly resume (Freed 1998). These activities require time and resources that may be useful for other tasks and thus impose an opportunity cost.

Interruption may impose other costs as well. Important preconditions may be undermined during the interruption, making the task impossible or more expensive. Facilitating conditions (opportunities) may lapse, raising task cost.

Priority mechanisms should take the expected cost of an interruption into account, suppressing interruptions unless the expected benefit exceeds the expected cost. When cost/benefit information is not available, the agent should assume a greater-than-zero interruption cost and inhibit interruption.

Robust Prioritization

The four heuristics listed above are useful for comparing candidate tasks, each on the basis of a single type of priority-relevant information. Ideally, an agent should take advantage when more information is available. One fairly simple case to consider is when the agent knows both the deadline proximity and duration of candidate tasks. Since deadline describes when the task should be completed, duration can be subtracted from this time value to determine when the task should be started. This is much more valuable measure of task urgency than deadline alone, since it makes it possible to determine how much time can be spent on alternative tasks before the risk of a deadline bust rises dramatically. Thus, as measured urgency (U = deadline-proximity – task-duration) approaches zero, priority should rise to task maximum.

The computation of urgency can be further refined by considering the effect of deferring a task in favor of some alternative. For example, if a task has a 5 minute completion deadline and lasts 3 minutes, then deferring it in favor of a competitor that also takes 3 minutes to carry out will cause a missed deadline. To avoid such scenarios, priority mechanisms should not only consider a task's duration, but also the duration of the strongest competing task. When the requisite information is available, priority should be determined on the basis of a tasks' *adjusted-urgency* (U' = U – competitor-duration).

A more complicated case is when information on both the time remaining until deadline (unadjusted urgency) and the cost of missing the deadline is available. What makes this complex is that the relevance of these two factor depends on another factor: how busy the agent is in the timeframe in which the tasks needs to be carried out. When the agent is not very busy and, thus, probably has time to do all intended tasks, the priority mechanism's job should be to make sure that a bad ordering decision doesn't lead to an unnecessarily missed deadline. In this case, urgency is a highly relevant determinant of priority while importance (the cost of a missed deadline) is largely irrelevant. In contrast, when the agent is so busy that there isn't time to do everything, missed deadlines are inevitable. In this case, the priority mechanism needs to make sure that important deadlines are met, even if that means allowing less important deadlines to bust. To handle both importance and urgency in a unified framework, priority mechanisms need to employ some measure of busyness.

To see how busyness might be usefully represented, it is worth formally characterizing the overall purpose of the prioritization process. A useful characterization, as indicated earlier, is that the process should try to minimize the long-term cumulative cost of missed deadlines. With respect to any given priority decision, the task whose deferral would add most to this total cost should be selected. A task's expected cost of deferral (ECD) is computed by summing the expected deferral cost for each of its associated deadlines. Thus,

$$\Pr iority(task) = ECD(task) = \sum_{d}^{deadlines} p(d) * Cost(d)$$

where *d* is a *task's* deadline, *p(d)* is the probability of missing *d* if *task* is deferred in favor of its current strongest competitor, and *C* is the cost of missing *d*. The identity and cost (importance) of the task's deadlines are assumed to be known. The probability of missing a given deadline

must be derived from how much time is available to do the task, U' (adjusted-urgency), and how great the task load will be (busyness) when the task, if deferred, again becomes eligible for execution. In particular, measured busyness must be a probability distribution B of minimum task delays – i.e. the probability that a deferred task will have to be further deferred for greater than or equal to a given time t. The value of p(d) is thus B(t) for given busyness distribution B with t = U'.

Consider a simple example. Task Q is eligible for execution. Its duration is 2 minutes. It has one associated deadline whose importance (the cost if the deadline is missed) is 7 and whose urgency (how much time is left to complete the task in order to avoid missing the deadline) is 5 minutes. Q is competing with other tasks for priority including the current strongest contender, R. R's duration is also 2 minutes. If R is selected for execution over Q, there will be (5-2-2) = 1 minute left in which to start Q after R is completed. This value is Q's adjusted-urgency. Given a busyness distribution B in which the value B(1 minute) = .3, .3 is the likelihood that, given all the other tasks that need doing, there will be at least a 1 minute delay before a given task can be started. The cost of deferring Q is thus .3*7 = 2.1. If this value is higher than that for R, Q becomes the strongest competitor in the priority competition.

Three problems remain to be solved to make this approach viable. The first and simplest one is to handle the possibility that needed information is not available – i.e. if a task's duration, deadline urgencies, or deadline importance values are unknown. The simplest approach is to factor out the influence of the missing information by assuming the same value as that of the task's strongest current competitor. For instance, if R in the example above has duration 2 minutes but Q's duration is unknown, assume a duration of 2 minutes. Similarly, if an unadjusted urgency value is missing, set it equal to the average of R's urgency values. If R's value is unknown, an arbitrary default value can be used instead.

The second problem is how to determine priority for an ongoing task. As mentioned earlier, interrupting an ongoing task may entail various costs such as the opportunity cost of time spent safely winding down (safing) the task and then, later, restoring any preconditions violated during the interruption interval. Our approach is to assume that the priority of an ongoing task is ECD + expected-interrupt-cost where the latter term is either known for the task or defaults to some nominal positive value. This has the effect of inhibiting interruptions in proportion to their undesirable consequences (Freed, 1998; cf. Gat, 1992).

The third problem is how to determine the shape of the minimum delay (busyness) distribution B. A few shape attributes are obvious. For example, p(0) = 1.0 – i.e. it is certain that a task will be delayed for at least 0 time. B is monotonically decreasing; thus, the more urgent the task, the greater likelihood of an unavoidable deadline bust. B asymptotes to 0 since, in principle, there is no limit to the amount of delay that might be required. Beyond this, B's distribution depends on various factors as described below:

Projection. In delaying a task for less important but more urgent alternatives, an agent risks losing the opportunity for timely execution because of emerging, higher priority tasks. To prevent this, the agent should take advantage of any predictions it can make about the set of future tasks a given current task will have to compete with if deferred. These future tasks can be divided into two categories: known and unknown. Known tasks are those that the agent already intends to execute when it gets a chance, including those whose enabling preconditions have not yet been satisfied. The unknowns include not-yet-specified subtasks of known tasks and tasks that do not yet exist (e.g. to handle a future phone call). Even the unknowns can be predicted to some degree. For example, an agent may know that this is a time of day when many phone calls should be expected, even if no specific answer-phone-call tasks currently exist. Similarly, it may be possible to project the most likely decomposition of a task into subtasks, even if the final decomposition decision has not been made.

The best possible delay estimate would, of course, be based on the most detailed and complete possible projection of future tasks. The described prioritization approach uses very coarse projections. Only a single known task, the strongest current competitor, is treated individually (used to compute adjusted-urgency). All other tasks including lower priority enabled tasks, non-enabled tasks, and unknown tasks, are considered in the aggregate via the busyness function. If instead, all known tasks were handled by explicit projection, the busyness function would only have to account for residual task-load – i.e. the "unknown" tasks.

Thus, the proper shape of this function depends on how projection is used in the prioritization process as a whole. Given the modest use of projection in our current approach, the function should incorporate any relevant, available information about future tasks. For example, if the total number or average duration of known tasks is relatively high, the distribution should be skewed towards relatively long delays. If an imminent surge of new (not yet existing) tasks is expected, expected delay should be proportionately greater.

Task attributes. Any delay distribution must make additional assumptions about the effect of task attributes – particularly duration and importance – on future priority decisions. Importance is significant because the more important a task is, the higher its priority will rise as the task deadline nears. This implies that importance should have two opposing effects on priority. Since the cost of

missing an important deadline is relatively great, importance increases expected-deferral-cost and, thus, priority. But because the likelihood of having to further delay an important task for even more important tasks is relatively low, the probability of a deadline bust is lower, making priority lower. The sum of these opposing effects should vary with urgency: as the deadline approaches, it is more crucial that the task be a strong competitor now rather than permit a delay that counts on it being a strong competitor later.

To the extent that short tasks have an advantage in gaining priority, a task's duration is also relevant for determining what kind of deferral distribution should be assumed. By raising a task's urgency in proportion to it's strongest competitor's duration, the prioritization process outlined above essentially penalizes long duration tasks – i.e. it has a strong short-duration bias. This bias should be taken into account in the expected-delay distribution by assuming relatively short delays for brief tasks.

Task duration affects B in other ways as well. First, longer tasks can often be interrupted for brief periods without significant effect. For example, a driver can look to the side of the road for a brief period without much risk, even though a longer look would significantly interfere with driving. An agent can take advantage a task's insensitivity to interruption to carry out brief tasks (Freed 1998). Similarly, lengthy tasks often have built in idle (slack) times during which brief tasks can be "fit in." For example, a driver stopped at a red light may have a brief time to use hands and eyes freely for non-driving tasks. The ability to execute brief tasks during idle intervals or within permissive interruption constraints means that short tasks can effectively be run concurrently with longer tasks they nominally conflict with. All of these effects imply that shorter tasks are less likely to be delayed for a long period.

Domain specificity. Another set of factors affecting the true distribution of task delays arises from characteristics of the task environment. This is especially the case for estimating the incidence of new tasks (e.g. rate of new phone calls, new customers at a bank, new aircraft for an air traffic controller,..). Environmental factors can also affect busyness less directly. For example, in conditions that make task failure likely, more time will have to ve spent on failure recovery. This essentially increases expected task duration and, thus, busyness.

An approach that makes it possible to address all of these factors is to assume a normal distribution of delays, using the complement of the cumulative distribution (a sigmoid) for B; each factor mentioned above affects the mean of the distribution. For instance, an increase in the expected incidence of new tasks increase the mean while a reduction in the average importance of known tasks reduces it. To account for the very significant effect of task duration D, we assume a default distributional mean equal to D. This captures the idea that a task is typically competing with a small number of tasks of about its own duration. Tasks that are significantly shorter will likely be executed earlier and with little effect on overall delay, while tasks that are much longer will be executed later or pseudo-concurrently.

Conclusion

In realistically uncertain worlds, an agent often lacks advance knowledge of what new tasks will arise and what specific actions will be needed to make progress at known tasks; thus, it cannot consider characteristics of these unspecified later actions when deciding order among earlier ones. Moreover, an agent often cannot be sure when an opportunity to execute its tasks will arise, and therefore cannot decide whether the task should come before or after some other, independently enabled task. With "when" and "what" uncertain, agents must instead reactively prioritize between currently eligible tasks based on whatever information is available.

This paper presents an approach based on the idea that the role of a prioritization process should be to minimize the cost of missed deadlines over a lengthy interval. The approach is designed to make best use of whatever priority-relevant is available at decision-time. If some useful piece of information is not available, the priority process should behave in a robust fashion, essentially falling back on simpler, more general decision heuristics. The greatest challenge has been to design an approach that flexibly decides whether to focus on meeting urgent deadlines or whether to insure that the most important deadlines are met.

The described approach captures a wide range of factors affecting priority decision-making under uncertainty. Furthermore, it is simple and computationally inexpensive enough to be realized in a practical agent architecture (Freed and Remington, 1997; Freed, 1998). However, the approach falls short of ideal in several of ways. First, contingent behaviors associated with a task such as dealing with failure, managing periodic behavior, coping with undesirable side-effects, are important contributors to task load but have yet been accounted for. Perhaps more importantly, the current approach makes too little use of projection The current approach can only make use of information on a single projected task (the one with the highest current priority), even though characteristics of other tasks may be known. Finally, the problem of setting the crucial mean-delay parameter is left almost entirely to domain-specific rules. While there may be no avoiding the need for such rules, the qualitative discussion of the effects of various factors does not provide a clear methodology for creating them. Future work should remedy these deficiencies.

References

Firby, R.J. 1989. Adaptive Execution in Complex Dynamic worlds. Ph.D. thesis, Yale University.

Freed, M. & Remington, R.W. 1997. Managing Decision Resources in Plan Execution. In Proceedings of the Fifteenth Joint Conference on Artificial Intelligence, Nagoya, Japan.

Freed, M. (1998b) Managing multiple tasks in complex, dynamic environments. In Proceedings of the 1998 National Conference on Artificial Intelligence. Madison, Wisconsin.

Gat, Erann. 1992. Integrating planning and reacting in heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of 1992 National Conference on Artificial Intelligence*.

Integrating Planning and Execution for Multiple Rover Operations

Tara Estlin, Gregg Rabideau, Darren Mutz, and **Steve Chien**
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109-8099
{firstname.lastname}@jpl.nasa.gov

Abstract

This paper describes a dynamic planning system for coordinating multiple rovers in collecting planetary surface data. A distributed planning system is shown to generate rover plans for achieving science goals, coordinate activities among rovers, monitor plan execution, and perform re-planning when necessary. Specifically, we describe how rover command generation can be automated to help relieve some of the burden on human operators. We describe the issues inherent in planning for a distributed set of rovers and discuss how these issues can be addressed in a dynamic and uncertain environment. Finally, we describe a prototype system for automatically generating low-level commands and monitoring their execution for a team of rovers with the overall goal of achieving a set of geology-related science requests.

Introduction

In the past few years, landmark events have recently taken place in the areas of space exploration and planetary rovers. The Mars Pathfinder mission was a major success, not only demonstrating the feasibility of sending rovers to other planets, but displaying the significance of such missions to the scientific community. Future missions are being planned to send additional robotic vehicles to Mars as well as to the outer planets and an asteroid (JPL 1999). In order to increase science return and enable certain types of science activities, future missions will require larger sets of rovers to gather the desired data. These rovers will need to behave in a coordinated fashion where each rover accomplishes a subset of the overall mission goals and shares any acquired information. In addition, it is desirable to have highly autonomous rovers that require little communication with scientists and engineers on earth to perform their tasks. An autonomous rover will be able to make decisions as how to best achieve science goals as well as being able to react to its environment and handle unforeseen events while achieving these goals.

An autonomous rover (or team of rovers) must respond in a timely fashion to a dynamic and unpredictable environment. Rover plans must often be modified in the case of fortuitous events such as science observations completing early and setbacks such as traverses taking longer than expected or hardware failures. We call this situation *continuous planning*, where a plan must be continually updated in light of changing operating context. In such an operations mode, a planner would be continuously updating the plan (e.g., every few seconds) based on sensor and other feedback, and then modifying the plan accordingly to accommodate any new data. Making an onboard planner capable of such timely responses has a number of benefits:

- The rover can be more responsive to unexpected changes in the environment. These changes could involve the status of executing activities, as well as updates to state (e.g., temperature, sun angle) and resource values (e.g., battery power).
- The planner can reduce reliance on predictive models since it will be updating its plans continually. Thus, inevitable modeling errors or uncertainties in the environment can be handled without causing plan failure and without explicitly representing all contingencies in the planning model.
- Rover fault-protection and execution layers need worry about controlling the rover over a shorter time horizon since the planner will replan within a shorter time span.

In a traditional cycle of plan, sense and act, planning is considered a batch process and the system operates on a relatively long-term planning horizon. For example, operations for Sojourner were planned on the ground on a daily basis (Mishkin *et al.* 1998). In this mode of operations, the rover state at the start of the planning horizon was pre-determined based on feedback from the previous day's operations. The science and engineering operations goals were then be considered, and a plan (i.e., command sequence) for achieving the goals would be generated. This plan was then uplinked to the rover for execution where it would be executed onboard the rover with minimal amounts of flexibility. If an unexpected event or failure occurred the rover would often be taken into a safe state by fault protection software. The rover would then wait in this state until the ground operations team could respond and determine a new plan. Correspondingly, if an unpredictable fortuitous

event occurred, the plan could not be modified to take advantage of the situation. This paper presents a continuous planning approach to rover operations, which is intended to achieve a higher level of responsiveness in situations where re-planning is required or beneficial.

Specifically we present work on using the CASPER (Continuous Activity Scheduling, Planning Execution and Replanning) (Chien *et al.* 1999) system to control a set of distributed rovers for planetary operations. Based on an input set of science goals and each rover's initial conditions, CASPER generates a sequence of activities that satisfies the goals while obeying each of the rover's resource constraints and operations rules. Plans are produced by using an "iterative repair" algorithm that classifies conflicts and resolves them individually by performing one or more plan modifications. Once a valid command sequence is generated, commands are relayed to the rover's low-level control software for execution. Execution updates are relayed back from this software where they are monitored by CASPER. As information is acquired regarding command status and actual resource utilization, the planner can update future-plan projections. From these updates, new conflicts and/or opportunities may arise, requiring the planner to replan in order to accommodate the unexpected events. Planning activities are distributed among the rovers where each rover is responsible for planning for its own activities. A central (non-dynamic) planner is responsible for dividing up the goals among the individual rovers in a fashion that minimizes the total time spent traversing by all rovers.

The planning system described in this paper has been integrated with a number of other software components to form a multi-rover execution architecture (Estlin *et al.* 1999b; 1999a). These components include: a machine-learning science analysis tool which analyzes planetary data and generates a set of goals for new science observations, a simulation environment that models multiple rover-science operations in a Mars-like terrain, a real-time multi-rover hardware and kinematics simulator, and control software from the NASA JPL Rocky 7 rover. This architecture is currently being tested on a geology-related science task where it must autonomously evaluate what science observations to perform, generate the necessary steps, and ensure execution of these steps is successful.

The remainder of this paper is organized in the following manner. We begin by characterizing the multiple-rovers application domain and describing our particular science scenario. Next, we present a multi-rover execution architecture which controls and coordinates operations for a team of rovers. We then focus on the planning aspects of this architecture; in particular we discuss our approach to distributed planning, our utilization of the CASPER continuous planning system, and our approach to plan optimization for this domain. The final sections discuss related and future work including system extensions and testing on real rovers.

Cooperating Rovers for Science

Utilizing multiple rovers on planetary science missions has many advantages:

- Multiple rovers can collect more data than a single rover. A team of rovers can cover a larger area in a shorter time where science gathering tasks are allocated over the team.
- Multiple rovers can perform tasks that otherwise would not be possible using a single rover. For instance, rovers landed at different locations can cover areas with impassable boundaries that would be unreachable by a single rover. Also, with several rovers, one rover can afford to take more risk and thus attempt tasks that usually might be avoided.
- More complicated cooperative tasks can be accomplished, such as taking a wide baseline stereo image (which requires two cameras separated by a certain distance).
- Multiple rovers can enhance mission success through increased system redundancy. If one rover fails, then its tasks could be quickly taken over by another rover, helping to ensure mission success.

In all cases, the rovers should behave in a coordinated fashion, dividing goals appropriately among the team and sharing acquired information. It is also desirable to have rovers behave in a dynamic fashion where plans can be adjusted when unexpected events or failures occur.

Coordinating multiple distributed agents for a mission to Mars or other planet introduces some interesting new challenges for the supporting technology. Issues arise concerning communication, control and individual on-board capabilities. Many of these design decisions are related, and all of them have an impact on any on-board technologies used for the mission. For example, the amount of communication available will determine how much plan data can be easily shared among rovers to perform necessary re-planning. It also affects how much each rover can coordinate with other rovers to perform tasks. The control scheme will determine which rovers execute what tasks. For instance, some rovers may be utilized only for science data gathering, while other may be used for planning and/or science analysis. Decisions on the on-board capabilities of each rover can also determine the independence of a rover. Planning, execution and plan monitoring can be performed onboard all rovers or on a select few which will provide these capabilities as a service to other rovers.

For the framework discussed in this paper, we have initially chosen the configuration of a team of three rovers where each rover has a planning and data analysis tool on-board. Each rover can thus plan for its assigned goals, collect the required data, and perform data analysis on-board which will direct its future goals. In addition, each rover can monitor its own plan execution and perform re-planning when necessary. Central planner and data-analysis modules are assumed to be

located on either a lander or one of the rovers, which are used to coordinate goals and science data.

Currently, we are evaluating our framework by testing its ability to build a model of the distribution of terrain rocks, classified according to composition as measured by a boresighted spectrometer. To perform testing for different planetary terrain models in a simulated environment, different rock fields (i.e., landscapes) are generated by using distributions over rock types, sizes and locations. Science goals consist of requests to take spectral measurements at certain locations or regions. These goals can be prioritized so that, if necessary, low priority goals can be preempted (e.g., due to resource constraints such as low battery power).

Science goals are divided among the three rovers. Each rover is identical and is assumed to have a spectrometer on-board as well as other resources including a drive motor, a solar panel that provides power for rover activities, and a battery that provides backup power when solar power is not available. The solar panel can also be used to recharge the battery. Collected science data is immediately transmitted to the lander where it is stored in memory. The lander has a limited amount of memory and can only receive transmissions from one rover at a time. The lander can also upload data (and simultaneously free up memory) to an orbiter whenever the orbiter is in communication contact.

Multi-Rover System Architecture

The distributed planning system described in this paper is part of a multi-rover execution architecture that coordinates multi-rover behavior and provides for autonomous rover operations (Estlin *et al.* 1999a). In particular, this architecture utilizes the MISUS system for autonomously generating and achieving planetary science goals (Estlin *et al.* 1999b).

The overall execution architecture is shown in Figure 1. The system is comprised of the following major components:

- **Planning**: A dynamic, distributed planning system that produces rover-operation plans to achieve input rover-science goals. Planning is divided between a central planner, which efficiently divides up science goals between rovers, and a distributed set of planners which plan for operations on an individual rover. Each rover planner provides execution monitoring and re-planning capabilities where plans are updated as necessary in reaction to unforeseen events.
- **Data Analysis**: A distributed machine-learning system which performs unsupervised clustering to model the distribution of rock types observed by the rovers. This model is used for prioritizing new targets for exploration by the rovers. This system is designed to direct rover sensing to continually improve the model of the scientific content of the planetary scene.
- **Rover Control Software**: Control software from the NASA JPL Rocky 7 rover that handles execution of low-level rover commands in the areas of navigation, vision and manipulation (Volpe *et al.* 1997). This software performs low-level monitoring and control of the rover's sub-systems.

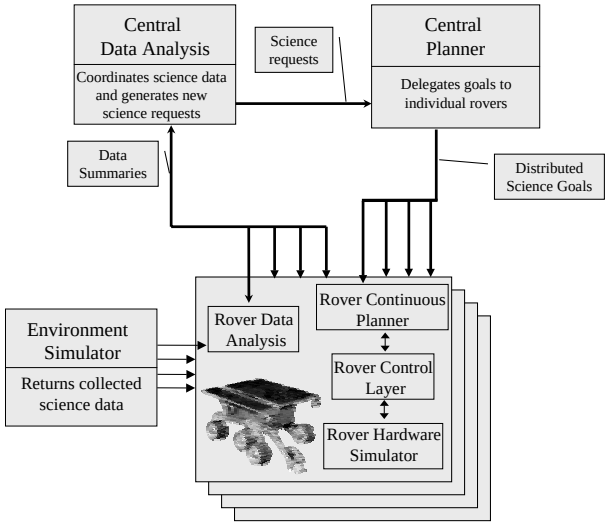


Figure 1: Multi-Rover Execution Architecture

- **Rover Hardware Simulator**: A multi-rover simulation environment that is used to simulate the rover terrain and rover hardware operations within that environment (Yen, Jain, & Balaram 1999). The simulator models rover kinematics and generates sensor feedback which is relayed back to the continuous planner for each rover.
- **Environment simulator**: A multiple rover simulator that models different geological environments and rover science activities within them. The simulator manages science data for each environment, tracks rover operations within a given terrain, and reflects readings by rover science instruments.

The overall system operates in a closed-loop fashion where the data analysis system can be seen to take the role of the scientist driving the exploration process. Spectra data are received by the on-board data analysis algorithms which broadcast information to the central analysis module. This module forms a global model of the data and generates a new set of observation goals that will further improve the accuracy of the model. These goals are passed to a central planner which assigns them to individual rovers in a fashion that will most efficiently serve the requests. Then each rover planner produces a set of actions for that rover which will achieve as many of its assigned goals as possible. These action sequences are executed using the rover low-level control software and a multi-rover simulation environment that relays action and state updates to each onboard planner, which can re-plan when unexpected events or failures occur. Action sequences are also executed within the environment simulator and any gathered data is sent back to the rover data analysis modules. This cycle continues until enough data is

gathered to produce distinct models for any observed rock types.

This architecture is currently being evaluated using the geological scenario previously described. The rest of this paper focuses on the planning aspects of this architecture. For more information on other components, please see (Estlin *et al.* 1999b; 1999a).

Planning for Multiple Rovers

To produce individual rover plans for a team of rovers, we have developed a distributed planning environment utilizing the CASPER continuous planning system (Chien *et al.* 1999). CASPER is an extended version of the ASPEN (Automated Scheduling and Planning Environment) system (Fukanaga *et al.* 1997), which has been developed to address dynamic planning and scheduling applications. Its components include:

- An expressive modeling language to allow the user to naturally define the application domain
- A constraint management system for representing and maintaining domain operability and resource constraints, as well as activity requirements
- A set of search strategies and repair heuristics
- A temporal reasoning system for expressing and maintaining temporal constraints
- A graphical interface for visualizing plans/schedules
- A real-time system which monitors plan execution and modifies the current plan based on activity, state and resource updates

CASPER employs techniques from planning and scheduling to automatically generate a rover-activity sequence which achieves the input goals. This sequence is produced by utilizing an iterative repair algorithm (Zweben *et al.* 1994) which attacks conflicts individually. Conflicts occur when a plan constraint has been violated where this constraint could be temporal or involve a resource, state or activity parameter. Conflicts are resolved by performing one or more schedule modifications such as moving, adding, or deleting activities. A rover that is at the incorrect location for a scheduled science activity is one type of conflict. Resolving this particular conflict involves adding a traverse command to send the rover to the designated site. Other conflicts may include having more than one rover communicating with the lander at the same time or having too many activities scheduled for one rover, which over-subscribe its power resources. Figure 2 shows an example plan in this domain displayed in the CASPER GUI.

Distributed Planning

To support missions with multiple rovers, we developed a distributed planning environment where it is assumed each rover has an on-board planner. This allows rovers to plan for themselves and/or for other rovers. If there is a slow communication link between rovers, or between a rover and the lander, it may be useful to have rovers

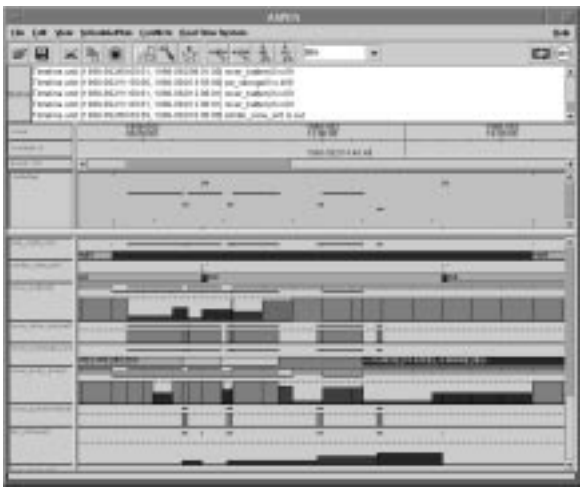


Figure 2: Example rover plan

construct their own plans and to re-plan dynamically when necessary. Also, by balancing the workload, distributed planning can be helpful when individual computing resources are limited.

The approach to distributed planning utilized in this environment is to include a CASPER continuous planner for each rover, in addition to one central, batch planner. The central planner develops an abstract plan for all rovers, while each rover planner develops a detailed, executable plan for its own activities. The central planner also acts as a router, taking a global set of goals and dividing it up among the rovers. For example, a science goal may request an image of a particular rock without concern for which rover acquires the image. The central planner could assign this goal to the rover that is closest to the rock in order to minimize the traversals of all rovers. This master/slave design is just one approach to distributed planning; we are also experimenting with several other forms of distributed planning (Rabideau *et al.* 1999).

Continuous Planning for each Rover

To achieve a high level of responsiveness for each on-board rover planner, we utilize a continuous planning approach. Rather than considering planning a batch process in which a planner is presented with goals and an initial state, each rover planner has a current goal set, a current state, a current plan, and state projections into the future for that plan. At any time an incremental update to the goals or current state may update the current plan. This update may be an unexpected event or simply time progressing forward. Each planner is then responsible for maintaining a plan consistent with the most current information. The current plan is the planner's estimation as to what it expects to happen in the world if things go as expected. However, since things rarely go exactly as expected, the planner stands ready to continually modify the plan. Iterative repair techniques, as mentioned above, enable

Initialize P to the null plan
Initialize G to the null set
Initialize S to the current state

Given a current plan P and a current goal set G

1. Update G to reflect new goals or goals that are no longer needed
2. Update S to the revised current state
3. Compute conflicts on (P,G,S)
4. Apply conflict resolution planning methods to P (within resource bounds)
5. Release relevant near-term activities in P to RTS for execution
6. Goto 1

Figure 3: CASPER continuous planning algorithm

incremental changes to the goals, initial state or plan and then iteratively resolve any conflicts that may arise.

The CASPER planning algorithm is shown in Figure 3. In this approach, the rover state is modeled by a set of plan timelines, which represent the current and expected state of the rover over time. At each loop iteration, the actual state of the rover drifts from the state expected by the timelines, reflecting changes in the world. As updates are relayed back from sensors and the rover control software, CASPER updates the timelines models with actual state values, resource values, start times and completion times for activities.

Each of these updates, when synchronized with the current plan, may introduce conflicts (Step 3). As explained previously, a conflict is when an action in the plan is inappropriate because its required state and/or resource values violate the plan constraints. Whenever such a conflict exists, CASPER notes the conflict and performs plan modifications to bring the plan back into sync with the current state and future-plan projections. Because this process is continuous, the plan rarely has the chance to get significantly out of sync. As a result the high-level actions of the system are more responsive to the actual rover state.

Plan Optimization

One of the dominating characteristics of the multi-rover application is rover traversals to designated waypoints. Decisions must be made not only to satisfy the requested goals, but also to provide more optimal (i.e., efficient) schedules. Both the central planner and the rover continuous planners can consider optimization goals during the repair process. As certain types of conflicts are resolved, heuristics are used to guide the search towards making decisions that will produce higher quality schedules. In other words, when several options are available for repairing a conflict, these options are ordered based on predictions on how favorable the resulting schedule will be.

For this application, we have implemented heuristics based on techniques for the Multiple-Traveling Salesmen Problem (MTSP), which is an extension of the well-known Traveling Salesman Problem (TSP) (Johnson & McGeoch 1997). For MTSP, at least one member of a sales team must visit each city such that total traveling time is minimized. Both the central and individual rover planners utilize the MTSP heuristics. These heuristics are used both to select which rover should be assigned a particular science activity and to select a temporal location for the science activity in a particular rover's plan. In previously reported results, they were shown to make a significant impact in reducing overall traversal distance and expected execution time (Rabideau, Estlin, & Chien 1999).

Related Work

While there has been a significant amount of work on cooperating robots, most of it focuses on behavioral approaches that do not explicitly reason about assigning goals and planning courses of action. One exception is GRAMMPS (Brummit & Stentz 1988), which coordinates multiple mobile robots visiting locations in cluttered, partially known environments. GRAMMPS also has a low-level planner on each robot and uses a similar approach to distribute targets. However GRAMMPS uses simulated annealing where we use a greedy approach, and GRAMMPS does not look at multiple resources or exogenous events.

Many cooperative robotic systems utilize reactive techniques (Mataric 1995; Parker 1999). These systems have been shown to exhibit low-level cooperative behavior in both known and "noisy" environments. However, they have not been shown useful for mission planning where a set of high-level science and engineering goals must be achieved in an efficient manner.

There are a number of approaches to robot autonomy which utilize AI planning and scheduling techniques, however, most of these are focused on controling single robots and don't directly address coordinating multiple robots. The Atlantis (Gat 1992) and 3T cite (Bonasso *et al.* 1997) architectures use different software layers to combine deliberation and reactivity. Another architecture for rover autonomy (Washington *et al.* 1999) integrates planning as a ground-based system, where a contingency planning approach is used to handle situations where a plan has been deemed likely to fail.

Future Work

We have several planned extensions to this work. First, we would like to extend the central (master) planner for this architecture to also utilize continuous planning techniques. Currently only the individual rover planners can perform re-planning. However, it would be beneficial to have this capability extended to the central planner which distributes science activities among the rovers. This extension would enable the central planner to re-assign goals when necessary or beneficial.

We plan to extend the distributed planning architecture to be more robust and able to handle rover failure situations. For instance, if a rover fails the planning system should recognize this failure (e.g., the rover has not responded for a certain amount of time), refrain from sending any new goals to that rover, and re-assign any current goals assigned to that rover.

We also want to handle more extensive communication between rovers. Currently, rovers share data through the central data-analysis module. We would like rovers to also share plan information which would enable us to experiment with different forms of distributed planning, such as team-based strategies (Tambe 1997) or market-based approaches (Sandholm 1993) to multi-agent coordination.

Last, we plan on testing the overall execution architecture in a more realistic setting using actual rovers as opposed to the hardware and environment simulators described previously. This testing will occur in the JPL Mars yard using rovers such as JPL's Rocky 7 and Rocky 8 (Volpe *et al.* 1997).

Conclusions

In this paper we have presented a distributed planning environment for coordinating multiple-rover activities. This environment utilizes continuous planning techniques to monitor plan execution for each rover and perform re-planning when necessary. Dynamic planning and re-planning techniques enable a team of rovers or act autonomously and be responsive to unexpected changes in the environment. This system is part of a multi-rover execution architecture which is currently being tested on its ability to autonomously classify a set of terrain rocks in a Mars-like environment.

Acknowledgments

This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

Bonasso, R.; Firby, R.; Gat, E.; Kortenkamp, D.; Miller, D.; and Slack, M. 1997. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence Research* 9(1).

Brummit, B., and Stentz, A. 1988. GRAMMPS: A generalized mission planner for multiple mobile robots in unstructured environments. In *Proceedings of the IEEE Conference on Robots and Automation.*

Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 1999. Integrated planning and execution for autonomous spacecraft. In *Proceedings of the 1999 IEEE Aerospace Conference.*

Estlin, T.; Yen, J.; Petras, R.; Mutz, D.; Castao, R.; Rabideau, G.; Steele, R.; Jain, A.; Chien, S.; Mjolsness, E.; Gray, A.; Mann, T.; Hayati, S.; and Das, H. 1999a. An integrated architecture for cooperating rovers. In *Proceedings of the 1999 International Symposium on Artficial Intelligence, Robotics and Automation for Space*, 255–262.

Estlin, T.; Gray, A.; Mann, T.; Rabideau, G.; Castano, R.; Chien, S.; and Mjolsness, E. 1999b. An integrated system for multi-rover scientific exploration. In *Proceedings of the Sixteenth National Conference on Ariticial Intelligence.*

Fukanaga, A.; Rabideau, G.; Chien, S.; and Yan, D. 1997. Towards an application framework for automated planning and scheduling. In *Proceedings of the 1997 International Symposium on Artficial Intelligence, Robotics and Automation for Space.*

Gat, E. 1992. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the Tenth National Conference on Artificial Intelligence.*

Johnson, D., and McGeoch, L. 1997. The traveling salesman problem: A case study in local optimization. In Aarts, E. H. L., and Lenstra, J. K., eds., *Local Search in Combinatorial Optimization.* London: John Wiley and Sons. 215–310.

JPL. 1999. *http://www.jpl.nasa.gov/missions/.*

Mataric, M. 1995. Designing and understanding adaptive group behavior. *Adaptive Behavior* 4(1):51–80.

Mishkin, A.; Morrison, J.; Nguyen, T.; Stone, H.; Cooper, B.; and Wilcox, B. 1998. Experiences with operations and autonomy of the mars pathfinder rover. In *Proceedings of the 1998 IEEE Aerospace Conference.*

Parker, L. E. 1999. Cooperative robotics for multi-target observation. *Intelligent Automation and Soft Computing* 5(1):5–19.

Rabideau, G.; Estlin, T.; Chien, S.; and Barrett, T. 1999. A comparison of coordinated planning methods for cooperating rovers. In *Proceedings of the AIAA 99 Space Technology Conference.*

Rabideau, G.; Estlin, T.; and Chien, S. 1999. Working together: Automatic generation of command sequences for multiple cooperating rovers. In *Proceedings of the 1999 IEEE Aerospace Conference.*

Sandholm, T. 1993. An implementation of the contract net protocol based on marginal cost calculations. In *Proceedings of the Eleventh National Conference on Artificial Intelligence.*

Tambe, M. 1997. Towards flexible teamwork. *Journal of Artificial Intelligence Research* 7:83–124.

Volpe, R.; Balaram, J.; Ohm, T.; and Ivlev, R. 1997. Rocky 7: A next generation mars rover prototype. *Journal of Advanced Robotics* 11(4).

Washington, R.; Golden, K.; Bresina, J.; Smith, D.; Anderson, C.; and Smith, T. 1999. Autonomous rovers for mars exploration. In *Proceedings of the 1999 IEEE Aerospace Conference.*

Yen, J.; Jain, A.; and Balaram, J. 1999. Roams: Rover analysis, modeling and simulation. In *Proceedings of the 1999 International Symposium on Artficial Intelligence, Robotics and Automation for Space*, 249–254.

Zweben, M.; Daun, B.; Davis, E.; and Deale, M. 1994. Scheduling and rescheduling with iterative repair. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling.* San Francisco, CA: Morgan Kaufmann. 241–256.

Interactive Autonomy for Space Applications

Amedeo Cesta
IP-CNR, Roma
cesta@ip.rm.cnr.it

Enrico Giunchiglia
DIST - Univ. di Genova
enrico@dist.unige.it

Paolo Traverso
ITC-IRST, Trento
leaf@irst.itc.it

Abstract

In this paper we describe JERRY, a system which supports the interactive design, planning, control and supervision of the operations of autonomous systems in a space environment. The requirement of Interactive-Autonomy in JERRY is achieved by a set of tightly integrated specialized sub-systems, which have been designed to perform effectively and efficiently their specific tasks, and, at the same time, to be open to the interaction among each other. This results in a system with a potential high level degree of autonomy, but which can still be controlled and guided through user interaction.

A System Supporting Interactive Autonomy

The increasing complexity of the services requested to robotic devices in space applications results in a need for more and more sophisticated and autonomous systems. A compelling requirement for space *autonomy* has led to the development of systems that perform automatically complex, time consuming and critical tasks without the need of human intervention (Muscettola *et al.* 1998).

The recent development of space autonomy for critical tasks results however in a set of novel problems, including the possibility of humans to control autonomous robotic devices (see for example (Dorais *et al.* 1998; Jerico 1998)) still guaranteeing a high level of safety of the operations that are perfomed interactively. This is due to quite a number of factors, such as the possibility of completely unexpected (and possibly dangerous) events that may require humans intervention. We call *Interactive-Autonomy* the ability of a system to provide a high level of autonomy still retaining the possibility for the user to monitor, control and override potentially autonomous operations in a safe way.

In this paper we describe JERRY, a system which supports the interactive design, planning, control and supervision of the operations of autonomous systems in a space environment. The requirement of Interactive-Autonomy in JERRY is achieved by a set of tightly integrated specialized sub-systems, which have been designed to perform safely, effectively and efficiently their specific tasks, and, at the same time, to be open to the interaction among each other. The user can directly operate each module step-by-step, and verify (at different levels of detail) the results of critical steps against safety requirements.

JERRY has been developed as part of an ongoing and more ambitious project funded by ASI, the Italian Space Agency. In this application, JERRY provides its functionality to different kinds of users which have to design, control and monitor a SPIDER Robot Arm performing quite complex tasks, e.g., the set up of several kinds of experiments in a space workcell. Even though the project is still running, a first prototype is already working and available for experimentation. In this scenario, e.g., the SPIDER arm is supposed to extract a tray from a shelf, fix it to one out of two tables and then automatically perform experiments moving objects contained in the tray.

In this paper, we first provide a global overview of JERRY by describing its high level architecture. We then describe the main features of each subsystem: the user interaction module, the planning module, and the execution module. Some conclusions end the paper.

JERRY's Architecture

The design of JERRY is based on three main components:

- **User-System Interaction Module.** It provides the user with the ability to inspect and direct every step of a system operation, via user requests of different services to specialized subsystems designed as "open systems".
- **Planning Module.** It provides a set of different planning services, including the generation of different kinds of plans of actions to achieve different kinds of high-level specifications of tasks to be performed (called goals), the validation of plans against requirements, their

step-by-step simulation.

- **Execution Module.** It provides a set of different execution services. It can compile a high level plan (provided by the planning module) into a program that is directly executable by a robotic device, execute it according to different modalities (e.g., either interactive or automatic), and monitor the execution. Further services include the step-by-step generation of actions, the verification that the executable program satisfies certain requirements and its step-by-step simulation.

The architecture of JERRY has been designed according to a main design choice: the User-System Interaction Module is central and can request different services from the other modules, in a "client-server style". The user can control the flow of JERRY 's operations and choose both the degree of automation and the level of interaction.

- **Degree of automation.** The user can decide to run the system within a wide range of options with different degrees of automation, from fully automatic to step-by-step interactive modes. The possibility is given to the user to decide to run the system in a fully automatic way: a goal is provided to the planning module that generates a plan to be executed by the execution module. failure). On the other hand, the user can decide to control interaction modality, the user may ask the Planning Module for a plan; the plan is inspected, validated and/or simulated; the first planning step is extracted from the plan and passed to the Execution Module that compiles it into an executable program; the program is inspected, possibly verified and simulated, and, finally, it is executed and monitored.

- **Level of interaction.** The user can access data and control the behavior of highly automatic systems by providing either high level specifications of what has to be achieved or detailed constraints on how the task should be performed. For instance, the user can request the planning module to generate automatically a high-level plan which achieves a high-level specified goal, or can direct the planner by imposing constraints on how to generate the plan. Analogously, the user can request the execution module to generate automatically the low level program corresponding to a plan, or can direct the execution module by imposing constraints on how the low-level robotic plan has to be generated.

The structure of the system is represented in Figure 1. In this figure, the planning and execution modules are visible in the top part, while the interaction module (with the sub-modules acting as interfaces with one of the other modules) is the

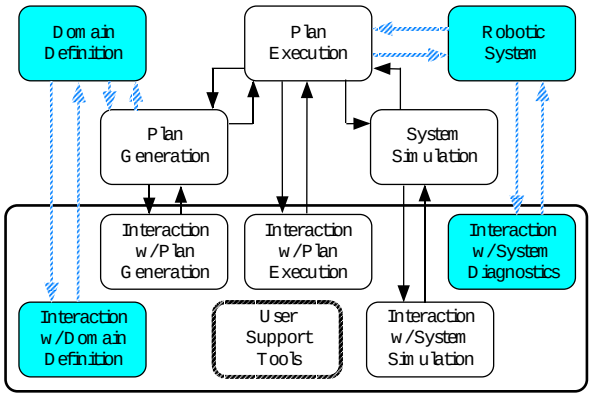


Figure 1: Structure of the System

"big box" at the bottom. The "Domain Definition" box represents a module that allows the user to specify the domain considered, and is currently part of the simulator. The "Robotic System" box represents the real robotic device. The solid arrows represent a flow of information, while the dotted arrows represent a still missing connection. For example, the dashed arrow between "Domain Definition" box and the interface, means that currently the user can specify a domain not through the interface, but only interacting directly with this module.

JERRY can work at two levels of interactions that are targeted to two typical users of space robotic devices: the "*programmer-level*" contains functionalities offered to the robotic system operator; the "*user-level*" deals with activities performed by on-ground scientists or payload operators. At the programmer-level, the user can program the behavior of the device using its typically low-level interface language, e.g. the language (called PDL2) currently used to control the SPIDER arm. A typical PDL2 instruction is "MOVE LINEAR TO `point-in-space`", where `point-in-space` is a triple of real values. This level of interaction is adequate for an experienced user. Nevertheless, programming complex tasks at this level may be very difficult for a user which has no experience with the programming language, e.g. PDL2. Moreover, low-level programs can be hard to maintain and re-use. For this reason, interaction at the user-level provides also non experts (e.g. scientists) with the ability to specify robotic tasks. Such users do not need any knowledge of the underlying physical structure of the robotic device (e.g. of the degrees of freedom of the arm) or of the physical scenario (e.g. of the exact position in space of the objects). A typical high-level instruction is "GET OBJECT `object-name`".

Operationally, the two interaction levels reflect two working modalities:

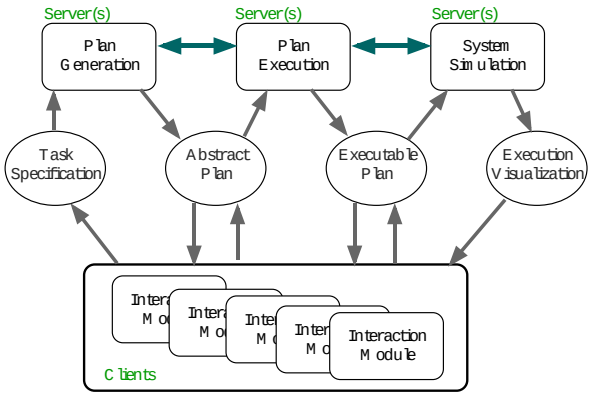


Figure 2: JERRY's Current Architecture

user-drives-system-supervises: in this modality an expert, knowledgeable of the underlying robotic device and mission interacts with the system by describing the mission in the robotic device interface language. The mission is encoded as a low level plan which is directly executable by the execution module.

system-drives-user-supervises: in this modality the user (even a non expert, e.g. a scientist) fixes the goal in a high level specification language. The high level specification cannot be executed directly. The system can generate automatically executable low level programs. This is achieved in two steps. First, the planning module generates a set of high level actions which have to be executed in different situations and which are guaranteed to achieve the goal. Then the execution module, for each high level action, generates a corresponding sequence of low-level actions in the robotic device interface language (e.g. PDL2). Independently from how the low-level plan is generated, the execution module is responsible for its execution, and for the monitoring of the behavior of the robotic system. At each step of the execution process, the user can be prompted for validating the high-level action to be executed, or, if required, the current low-level program.

The resulting architecture is highly modular and configurable: the system can be configured to work at different levels of automation (e.g. depending on the activity performed by the planning module) and the user has the possibility to flexibly access data manipulated at different levels of detail (e.g. data at the execution or at the planning level). The interface can be set to be used by users with different experience (programmers or scientists) and can also be adapted to different input devices (e.g., driven entirely from mouse or touchpad, entirely from keyboard, or, possibly, from custom input devices).

A first version of the demonstrator has been fully implemented, is available for inspection, and is currently under development to improve its general performance and to enrich the services offered to the user. This demonstrator (whose architecture is represented in Figure 2) is based on a client/server architecture in which a client interface service is able to continuously interact with the planning, execution and simulator modules. This has involved the development of specialized protocols that allow each interaction module to safely exchange data with the three servers through point-to-point communication. Current protocols are deliberatively designed to be very simple to minimize the overhead of communication between modules and to quickly arrive to a first integration.

Interaction Module

The role of software systems like JERRY is to allow different users to employ complex robotic devices while preserving the levels of responsibility that users have in their working contexts. Both the user-level and the programmer-level preserve the usual working activity, but offer a number of additional functionalities that allow the users to focus on strategic and decisional tasks and to delegate repetitive or very difficult tasks to the interactive planning software.

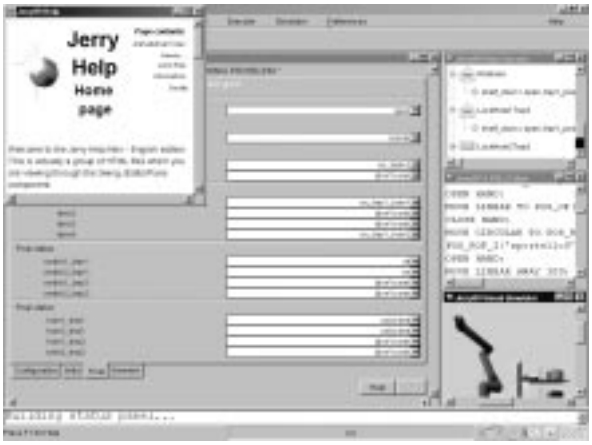


Figure 3: JERRY Interactive Module

The JERRY Interaction Module consists of a Graphical User Interface endowed with the following functionalities:

- Task oriented help.
- Problem specification targeted to the planner domain representation language.
- Inspection of high-level plans: a rather simple representation of the plan returned by the planner is shown and the possibly of inspecting the representation of single plan states is given.

- Inspection of plan compilation: the low-level code produced by the plan compilation and execution module is shown to the user.
- Robotic device simulator visualization.

The current look of the Interactive Module is shown in Figure 3. In the Figure we can see (*i*) the Help window (top-left) that is designed as a separate entity; (*ii*) the planning problem specification window (main window below the Help window); (*iii*) the plan current in execution (top-right); (*iv*) the PDL2 code corresponding to the action being executed (middle-right); and (*v*) the execution of the plan coming from the simulator (bottom-right). The size of the 4 windows corresponding to point from (*ii*) to (*v*) are interconnected and vary according to the user current focus of attention that is always contained in the main window.

According to the subdivision made between the "programmer-level" and the "user-level", the tasks allowed to each level have been defined. In the current implementation of the "user-level interaction" the users can: (*i*) get acquainted with an operating environment; (*ii*) define specific parameters of the scenario (e.g., decide the number of trays in an experiment); (*iii*) specify the goal he want to achieve and the constraints to satisfy in achieving it; (*iv*) ask the planning module to determine the set of actions (the plan) that achieves the goal; (*v*) display and comment the resulting plan; (*vi*) activate plan execution. Special attention has been dedicated to automatically checking the consistency of commands selected by the user and in offering explanation facilities for non-expert users. The "programmer-level interaction" offers: (*i*) the possibility of creating robot programs directly using the robot language, (*ii*) the choice of having the planning and execution mechanisms that work as background help of the programmer; (*iii*) the possibility of experimenting different operational situations offering a choice among alternative input modalities. The possibility of customizing the interaction modality is relevant for experimenting on-flight use of the programming ability. In is worth observing that being the Interaction Module configured as a client it is possible to serve multiple users at the same time each of them interacting with personalized functionalities.

This module is implemented in Java (compatible with JDK 1.2) and represent right now a quite effective platform for studying multiple interactions styles that refer to different ways of sharing task responsibility among users and the system.

Planning Module

The Planning Module provides the user with the ability of requesting services by means of high level specifications entered through the Interaction Module. It has been designed to be highly independent of the programming language that is directly executable by robotic devices (e.g. PDL2). The Planning Module works on data structures that encode a high level description of the possible situations in a given domain (called states), of the operations that a robotic device can execute (called actions), and of the requirements for tasks to be performed (called goals).

A state of the domain is represented symbolically, e.g. by expressions like `Y is on experiment tray Z`. Operations of the robot are represented as high-level actions, e.g. `Get object Y`, `put Y on experiment tray Z`. Plans are possibly conditional and iterative programs that specify the actions to be executed. An example of plan is the following.

```
Plan experiment-1 is:
Get object Y;
if this action succeeds,
then put Y on experiment tray Z,
otherwise get object Y1;
...
```

At this level of abstraction, the Planning Module provides three main functions: Validation, Simulation and Plan Generation.

The *validation function* allows the user to check properties of the scenario or of putative plans. For instance, the user can ask whether `Y is on experiment tray Z` in the current scenario, or whether `Y is on experiment tray Z` after executing plan `experiment-1`. This functions provides the user with a higher confidence that a task can be performed correctly in a given way. It can also be used to inspect and debug possible plans.

The *simulation function* allows the user to simulate the execution of a given plan. It shows the evolution of the states of the domain. For instance, the user can request a simulation of the plan `experiment-1`. The simulation is performed at the level of abstraction of the Planning Module, i.e., by showing high level actions and how they affect the state of the domain. Again this functionality can be used to gain a better understanding of how a plan can perform a given task and to debug plans.

The *plan generation function* is the core of the Planning Module. The user, through the Interaction Module, provides a goal to be achieved, i.e., a high level specification of the task to be performed. The goal is a high level description of what has to be achieved. It does not detail how the task should be performed. For instance, a simple goal can be `one of Y,Y1 on experiment tray Z`. The Planning Module generates automatically plans of actions (e.g. the plan `experiment-1`) whose aim is to achieve the task specified by the goal. The plan of actions is the output which can be passed, through the Interaction Module and possibly under control of

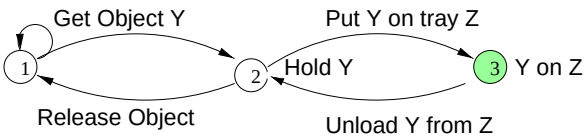


Figure 4: An example of FSM

the user, to the Execution Module.

This set of services has been designed to meet the requirement of *Interactive-Autonomy*. Indeed, a main characteristic of the planning module is that it is an open system, i.e. each of its operations (e.g. plan search) can be inspected, controlled and guided by the user. This fact opens up the possibility to provide a planning functionality which supports a "user-centered operation mode" for JERRY, in which the planner interacts flexibly with the user interface module. The user, beyond asking for a goal to be satisfied, can ask the planner for different services, e.g. show all the plans which satisfy a goal, select one of them, query the planner about the possible effects of the execution of plans, re-use existing plans, ask the planner to validate a user defined plan, inhibit some plans, query the planner about the current state of the execution in terms of high-level actions. This "user-centered" modality requires a design of the planning module which is different in philosophy wrt current state of the art planners. The planner is no longer the automatic generator of solutions, it becomes a system which exploits its automatic generation capabilities to support the user to find the right solution and is flexible enough to adjust its plan generation activity to different user requirements.

It is worth remarking that the functionalities of JERRY allow to implement an interesting set of mixed-initiative problem solving strategies that are somehow disjunct from other approaches devoted to inserting the user in the loop (Smith & Lassila 1994; Tate 1997). In our current framework the user is left free of choosing to either solve the problem himself or request the intervention of specific JERRY functionalities.

The Planning Module is developed on top of the MBP system (Model Based Planner) (Cimatti *et al.* 1997; Cimatti, Roveri, & Traverso 1998). MBP is an extension of the NuSMV Model Checker (Cimatti *et al.* 1999). It implements the "Planning as Model Checking" paradigm. The implementation relies heavily on existing work in the context of finite-state program verification and in particular on the work described in (Clarke, Grumberg, & Long 1994; Burch *et al.* 1992; McMillan 1993). The underlying idea is that domains are modeled as Finite State Machines (FMSs) and plans are generated by searching the states of FSMs. For instance, a FSM for a simple domain is depicted in Figure 4. We have three possible states (labeled as 1,2,3). Each state is labeled with the facts that hold in that state, e.g. `Hold Y` states that in state 2 the robot has at hand the object `Y`, `Y on Z` states that in state 3 the object `Y` is on the experiment tray `Z`. The actions `Get Object Y`, `Put Y on tray Z`, `Release Object`, `Unload Y from Z` are represented as transitions (arcs) between states.

Plan Compilation/Execution

The Plan Compilation/Execution module is responsible for transforming a high-level, user-oriented abstract plan into a sequence of low-level, machine-oriented execution plan. In more detail, the Plan Compilation/Execution module receives in input from the interface an arbitrarily long sequence of actions to be performed, and generates a sequence of actions (a "program") that the robot can directly execute. For example, in the case of a robotic arm, the program corresponding to a $move(o, l)$ ("move object o to location l") looks like the following sequence of instructions

```
move_near <pos_o> by <distance>;
open_hand;
move_linear <pos_o>;
close_hand;
move_near <pos_l> by <distance>;
move_linear <pos_l>;
open_hand;
```

where `<pos_o>` and `<pos_l>` are six tuples of real numbers specifying the positions of the object and of the location respectively, while `<distance>` is a real number specifying how far (along the vector specifying how to approach the position) the arm should be from the final position. Notice, the fundamental distinction between a "user-level" and a "programmer-level" plan: while the former is a sequence of symbolic actions, the latter necessarily involves some reasoning about the geometry of the scenario. As for the planning module, the execution module provides the possibility to *validate*, *simulate* and/or *generate* a PDL2 program.

Given that the positioning module has to deal with variables with infinite domains (typically intervals over the reals), the set of possible low-level plans cannot be directly encoded into FSMs. Because of this, we adopted a different mechanism in which the "validation function" rely on a user-defined LALR(1) grammar (see (Aho, Sethi, & Ullman 1986)) defining the whole set of *admissible* PDL2 plans. This grammar, though dependent on the particular scenario under consideration, can be inspected and/or modified by the user before each session. The grammar is then directly compiled into an executable code corresponding to a program accepting a plan only if it is admissible.

A "generation function" allows to associate to any high-level sequence of actions a corresponding sequence of low-level sequence of PDL2 instruc-

tions. In any case, the sequence of actions given to the execution module does not need to correspond to a complete plan. Instead, the user can (*i*) break a plan as given by the planning module into blocks of planning actions, (*ii*) require the compilation of all or some of the blocks, or (*iii*) ask for an execution program differing from the proposed one.

Finally the "simulation function" allows for the validation of either the generated or user-provided PDL2 program by a suitable call to the simulator. As obvious result, the movement of the SPIDER is displayed on a screen for validation by the user.

As for the planning module, the execution module is an open system in which the parameters affecting its behavior (e.g. the availability of a given low-level action) can be inspected, controlled and eventually modified by the user. For example, the user can inhibit the execution module from using a certain low-level action because it involves some dangerous or unavailable move for some joint. This will affect both the validation and the generation routines: no plan with the undesired instruction will be accepted and generated. As above, this fact opens up the possibility to provide a "user-centered operation mode" for JERRY, in which the execution module interacts flexibly with the user interface module.

A Java (compatible with JDK 1.2) implementation of the execution module has been realized, and is currently tested for improvements. The compilation from the grammar to the executable code has been realized via CUP (see (Hudson *et al.* 1998)).

Conclusions

This paper describes JERRY, a system for the automatic generation and execution of plans for robotic devices, and briefly reports about the case study of the SPIDER arm. The main feature of the system is the high-level of interaction that the user can decide to have with the system. This level of interaction is critical in the context of spatial missions, where (*i*) unforeseen emergencies can happen, and (*ii*) still the mission has to proceed, possibly under the humans' supervision.

JERRY has been designed to be a flexible, open architecture. Care has been taken in order to distinguish the domain-dependent from the domain-independent tasks in order to minimize the customization efforts. JERRY's architecture and underlying ideas have been tested and made operational for monitoring and controlling a SPIDER robotic arm operating in an indoor environment very close to the payload tutor experiment described in (Di Pippo *et al.* 1998).

Acknowledgments

We would like to thank Marco Daniele, Maria Grazia Maccione and Paola Riccucci for their work in JERRY. Thanks to Marco Schaerf for making the robot simulator available. This research is supported by ASI (Italian Space Agency) as part of the project "Un Sistema Intelligente per la Supervisione di Robot Autonomi nello Spazio". Special thanks to the other participants in the project for the many fruitful discussions and the technical help.

References

Aho, A. V.; Sethi, R.; and Ullman, J. D. 1986. *Compilers: Principles, Techniques, and Tools.* Reading, MA, USA: Addison-Wesley.

Burch, J. R.; Clarke, E. M.; McMillan, K. L.; Dill, D. L.; and Hwang, L. J. 1992. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation* 98(2):142–170.

Cimatti, A.; Giunchiglia, E.; Giunchiglia, F.; and Traverso, P. 1997. Planning via Model Checking: A Decision Procedure for AR. In *Lecture Notes in Computer Science*, volume 1348.

Cimatti, A.; Clarke, E.; Giunchiglia, F.; and Roveri, M. 1999. NuSMV: A new symbolic model verifier. *Lecture Notes in Computer Science* 1633.

Cimatti, A.; Roveri, M.; and Traverso, P. 1998. Automatic OBDD-based Generation of Universal Plans in Non-Deterministic Domains. In *Proceeding of the Fifteenth National Conference on Artificial Intelligence (AAAI-98).* Madison, Wisconsin: AAAI-Press. Also IRST-Technical Report 9801-10, Trento, Italy.

Clarke, E.; Grumberg, O.; and Long, D. 1994. Model Checking. In *Proceedings of the International Summer School on Deductive Program Design.*

Di Pippo, S.; Colombina, G.; Boumans, R.; and Putz, P. 1998. Future Potential Applications of Robotics for the International Space Station, Robotics and Autonomous Systems. *Robotics and Autonomous Systems* 23:37–43.

Dorais, G.; Bonasso, P.; Kortenkamp, D.; Pell, B.; and Schreckenghost, D. 1998. Adjustable Autonomy for Human-Centered Autonomous Systems on Mars. In *Proc. Mars Society Conference.*

Hudson, S. E.; Flannery, F.; Ananian, C. S.; Wang, D.; and Appel, A. W. 1998. CUP parser generator for Java.

1998. JERICO Project Description. ASI (Italian Space Agency) Internal Documentation.

McMillan, K. 1993. *Symbolic Model Checking.* Kluwer Academic Publ.

Muscettola, N.; Nayak, P.; Pell, B.; and Williams, B. 1998. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence* 103:5–48.

Smith, S. F., and Lassila, O. 1994. Toward the Development of Mixed-Initiative Scheduling Systems. In Burstein, M. H., ed., *Proceedings ARPA-Rome Laboratory Planning Initiative Workshop, Tucson (AZ)*, 145–154. Morgan Kaufmann.

Tate, A. 1997. Mixed-Initiative Interaction in O-Plan. In *Papers from 1997 AAAI Spring Symposium on Computational Models of Mixed-Initiative Interaction, Stanford (CA)*, 163–168.

A Prototype for Ground-based Automated Rover Command Generation

Rob Sherwood, Tara Estlin, Steve Chien, Gregg Rabideau,

Barbara Engelhardt, Andrew Mishkin, Brian Cooper

Jet Propulsion Laboratory
4800 Oak Grove Dr.
Pasadena, CA 91109
818-393-5378
rob.sherwood@jpl.nasa.gov

Abstract

This paper will discuss a proof-of-concept prototype for automatic generation of validated rover command sequences from high-level science and engineering activities. This prototype is based on ASPEN, the Automated Scheduling and Planning Environment. This AI-based planning and scheduling system will automatically generate a command sequence that will execute within resource constraints and satisfy flight rules. Commanding the rover to achieve mission goals requires significant knowledge of the rover design, access to the low-level rover command set, and an understanding of the performance metrics rating the desirability of alternative sequences. It also requires coordination with external events such as orbiter passes and day/night cycles. An automated planning and scheduling system encodes this knowledge and uses search and reasoning techniques to automatically generate low-level command sequences while respecting rover operability constraints, science and engineering preferences, and also adhering to hard temporal constraints. Enabling goal-driven commanding of planetary rovers by engineering and science personnel greatly reduces the requirements for highly skilled rover engineering personnel and Rover Science Team time. This in turn greatly reduces mission operations costs. In addition, goal-driven commanding permits a faster response to changes in rover state (e.g., faults) or science discoveries by removing the time consuming manual sequence validation process, allowing rapid what-if analyses, and thus reducing overall cycle times.

Introduction

Unlike more traditional deep space missions, surface roving missions must be operated in a reactive mode, with mission planners waiting for an end of day telemetry downlink--including critical image data--in order to plan the next day's worth of activities. Communications time delays over interplanetary distances preclude simple 'joysticking' of the rover. A consequence of this approach to operations is that the full cycle of telemetry receipt, science and engineering analysis, science plan generation, command sequence generation and validation, and uplink of sequence, must typically be performed in twelve hours or less. Yet current rover sequence generation is manual, with limited ability to automatically generate valid rover activity sequences from more general activities/goals input by science and engineering team members. Tools such as the Rover Control Workstation (RCW) and the Web Interface for Telescience (WITS) provide mechanisms for human operators to manually generate plans and command sequences. These tools even estimate some types of resource usage and identify certain flight rule violations. However, they do not provide any means to modify the plan in response to the constraints imposed by available resources or flight rules, except by continued manual editing of sequences. This current situation has two drawbacks. First, the operator-intensive construction and validation of sequences puts a tremendous workload on the rover engineering team. The manual process is error-prone, and can lead to operator fatigue over the many months of mission operations. Second, the hours that must be reserved for sequence generation and validation reduces the time available to the science team to identify science targets and formulate a plan for submission to the engineering team. This results in reduced science return. An automated planning tool would allow the science team and sequence team to work together to optimize the plan. Many different plan options can be explored. The faster turnaround of automated planning also permits shorter than once a day planning cycles.

The Rover Control Workstation (RCW) tool, used to operate the Sojourner rover during the Pathfinder mission, does provide visualization for vehicle traverse planning, a

command interface, constraint checking for individual commands, and some resource estimation (for sequence execution time and telemetry volume). However, this tool was never intended for automated goal-based planning of rover activities. To deal with these issues, there is a need for a new tool for automated goal-based planning of rover activities.

We are using AI planning/scheduling technology to automatically generate valid rover command sequences from activity sequences specified by the mission science and engineering team. This system will automatically generate a command sequence that will execute within resource constraints and satisfy flight rules. Commanding the rover to achieve mission goals requires significant knowledge of the rover design, access to the low-level rover command set, and an understanding of the performance metrics rating the desirability of alternative sequences. It also requires coordination with external events such as orbiter passes and day/night cycles. An automated planning and scheduling system encodes this knowledge and uses search and reasoning techniques to automatically generate low-level command sequences while respecting rover operability constraints, science and engineering preferences, and also adhering to hard temporal constraints. A ground-based interactive planner combines the power of automated reasoning and conflict resolution techniques with the insights of the PI to prioritize and re-prioritize mission goals.

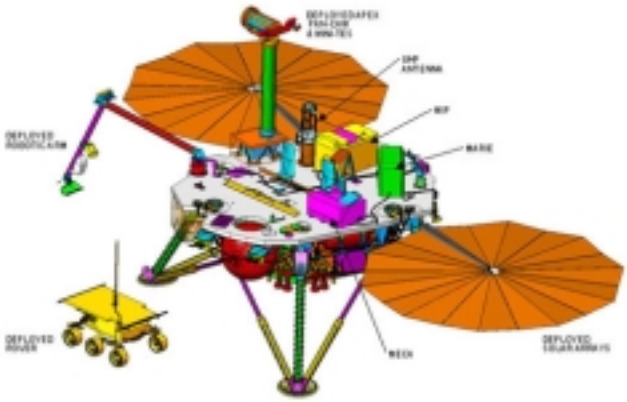


Figure 1 - Mars 2001 Lander

Mars Surveyor Lander

The Mars Surveyor 2001 Lander is scheduled for launch on April 10, 2001. It will land on Mars on Jan. 22, 2002, if launched on schedule. The 2001 Lander will carry an imager to take pictures of the surrounding terrain during its' rocket-assisted descent to the surface. The descent-imaging camera will provide images of the landing site for geologic analyses, and will aid planning for initial operations and traverses by the rover. The 2001 Lander will also be a platform for instruments and technology experiments designed to provide key insights to decisions regarding successful and cost-effective human missions to Mars. Hardware on the Lander will be used for an in-situ demonstration test of rocket propellant production using gases in the Martian atmosphere. Other equipment will characterize the Martian soil properties and surface radiation environment. Figure 1 contains a diagram of the lander and instruments. The Marie Curie rover will be deployed using a robotic-arm contained on the lander.

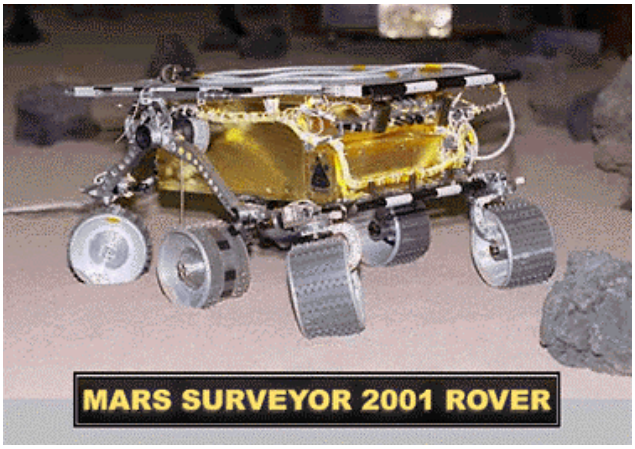


Figure 2 - Marie Curie Rover

The Marie Curie rover is very similar to the Mars Pathfinder Sojourner rover. (See Figure 2.) In fact, it is the same rover that was used in the Pathfinder test bed during the mission. (Mishkin et al., 1998; Mishkin 1998) Additional modifications were made to accommodate the robotic-arm-based deployment from the 2001 Lander. In addition, some minor engineering enhancements were added. A description of the rover components is contained in Table 1.

- 6-Wheeled robotic vehicle, rocker-bogie mobility chassis
- Mass: 10.5 kilograms
- Deployed volume: 65cm (l) by 48cm (w) by 30cm (h).
- Intel 80C85 CPU (~100Kips), 16K PROM, 64K rad hard RAM, 176K EEPROM, 512K RAM
- Forward Black & White stereo cameras, and rear B&W mono camera
- GaAs solar panel (16W peak)
- Primary (non-rechargeable) batteries
- UHF Radio Modem
- Laser stripers for hazard detection

Table 1 - Rover Description

ASPEN Planning System

Planning and scheduling technology offers considerable promise in automating rover operations. Planning and scheduling rover operations involves generating a sequence of low-level commands from a set of high-level science and engineering goals.

ASPEN (Fukanaga, et al., 1997; Rabideau, et al., 1999) is a re-configurable planning and scheduling software framework that includes the following set of software components:

- An expressive constraint modeling language to allow the user to define naturally the application domain
- A constraint management system for representing and maintaining spacecraft operability and resource constraints, as well as activity requirements
- A set of search strategies
- A temporal reasoning system for expressing and maintaining temporal constraints
- A graphical user interface for visualizing plans/schedules

In ASPEN, the main algorithm for automated planning and scheduling is based on a technique called *iterative repair* (Zweben et al., 1994). During iterative repair, the conflicts in the schedule are detected and addressed one at a time until no conflicts exist, or a user-defined time limit has been exceeded. A conflict is a violation of a reservation, parameter dependency or temporal constraint. Conflicts can be repaired by means of several predefined methods. The repair methods are: moving an activity, adding a new instance of an activity, deleting an activity, detailing an activity, abstracting an activity, making a reservation of an activity, canceling a reservation, connecting a temporal constraint, disconnecting a constraint, and changing a parameter value. The repair algorithm may use any of these methods in an attempt to resolve a conflict. How the algorithm works is largely dependent on the type of conflict being resolved.

Rover knowledge is encoded in ASPEN under seven core model classes: activities, parameters, parameter dependencies, temporal constraints, reservations, resources and state variables. An activity is an occurrence over a time interval that in some way affects the rover. It can represent anything from a high-level goal or request to a low-level event or command. Activities are the central structures in ASPEN, and also the most complicated. Together, these constructs can be used to define rover procedures, rules and constraints in order to allow manual or automatic generation of valid sequences of activities, also called plans or schedules.

Once the types of activities are defined, specific instances can be created from the types. Multiple activity instances created from the same type might have different parameter values, including the start time. Many camera-imaging activities, for example, can be created from the same type but with different image targets and at different start times. The sequence of activity instances is what defines the plan.

The flight rules and constraints are defined within the activities. The flight rules can be defined as temporal constraints, resource constraints, or system state constraints. Temporal constraints are defined between activities. An example would be that the rate sensor must warm up for two to three minutes before traversing (moving) the rover. In ASPEN, this would be modeled within the "move rover" activity as shown in Figure 3. The rate sensor_heat_up is another activity that is presumed to turn on a rate sensor heater.

Constraints can also be state or resource related. State constraints can either require a particular state or change to a particular state. Resource constraints can use a particular amount of a resource. Resources with a capacity of one are called atomic resources. ASPEN also uses non-depletable and depletable resources. Non-depletable resources are resources that can used by more than one activity at a time and do not need to be replenished. Each activity can use a different quantity of the resource. An example would be the rover solar array power. Depletable resources are similar to non-depletable except that their capacity is diminished after use. In some cases their capacity can be replenished (memory capacity) and in other cases it cannot (battery energy, i.e. non-rechargeable primary batteries). Resource and state constraints are defined within activities using the keyword "reservations." See Figure 3 for an example.

```
Activity move_rover {
 constraints =
   starts after end_of rate_sensor_heat_up by [2m,3m];
 reservations =
   solar_array_power use 35,
   rate_sensor_state change_to "on",
   target_state must_be "ready";
```

Figure 3 - ASPEN Modeling Language Example

The job of a planner/scheduler, whether manual or automated, is to accept high-level goals and generate a set of low-level activities that satisfy the goals and do not violate any of the rover flight rules or constraints. ASPEN provides a Graphical User Interface (GUI) for manual generation and/or manipulation of activity sequences. Figure 4 contains a screen dump of the GUI.

Model Description

The Marie Curie model was built to a level at which all flight rules and constraints could be implemented. The resources include the three cameras, Alpha Proton X-Ray Spectrometer (APXS), APXS deploy motor, drive motors, solar array, battery, RAM usage, and EEPROM usage.

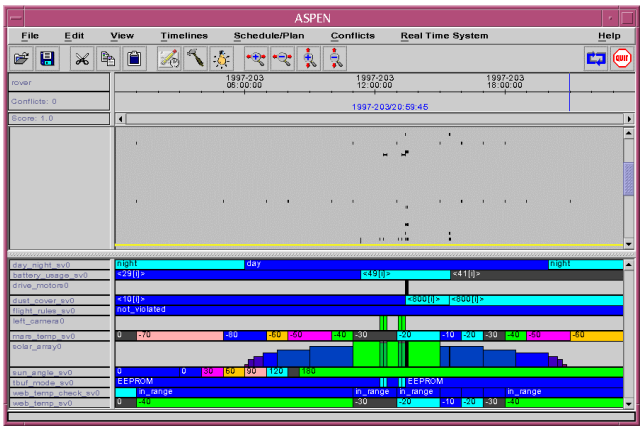


Figure 4 - ASPEN GUI

There are 27 different state variables used to track the status of various devices, modes, and parameters. Some of these parameters map directly onto rover internal parameters and others are related to the ASPEN specific model. We are not modeling all rover internal parameters because many are not useful for automating planning. We have defined 162 activities of which 63 decompose directly into low-level rover commands.

There are several constraints that affect overall operations of the Marie Curie rover. These include:

- Earth-Mars one-way communications time delay (5-20 minutes)
- Limited communications bandwidth (generally < 10 Mbits downlink per sol available to rover)
- Limited communications opportunities (1 command uplink, 2 telemetry downlinks per sol)

The power system is the single most important resource for the Marie Curie Rover. This system consists of a .22 square meter solar array and 9 LiSOCL batteries. The batteries on Marie Curie are primarily used during the night for APXS data collection. They are primary batteries and therefore modeled as non-renewable depletable resources. The solar array is the primary power source used during the day. The predicted available solar power profile throughout the Mars day must be input before planning begins. Using a daily model is required due to changing solar array power available as a result of degradation from dust accumulation and seasonal solar irradiation variability.

A typical Mars day might involve a subset of the following activities:

- Complete an APXS data collection that was carried out during the prior night
- Capture a rear image of the APXS site
- Traverse to an appropriate site and perform a series of soil mechanics experiments, including several subframe images of soil mounds and depressions created by running individual wheel motors
- Traverse to a designated rock or soil location
- Place the APXS sensor head
- Capture end-of-day operations images with its forward cameras
- Begin APXS data collection
- Shut down for the night

APXS data collection usually occurs overnight while the rover is shutdown. Each of these activities can be input into ASPEN as a goal for that Mars day planning horizon. The format of the input goals is RML or Rover Modeling Language. RML is an application of Extensible Markup Language (XML) designed specifically for rover operations. RCW will use RML for input and output. Using a common language between the various operations tools simplifies the interface between tools.

The exact position of the rover after a traverse activity is subject to dead reckoning error. The timing of traverse activities is also non-determinant. Because of the inherent problems of coordinating activities between the event-based rover and time-based lander, wait commands are used to synchronize activities. When the lander is imaging the rover after a traverse, a wait command is used to ensure the rover will remain stationary at its' destination until the lander completes imaging. Because the rover executes commands serially, this ensures that another command will not start execution before the previous command has completed. All rover traverse goals are generated using the RCW. (ASPEN is not designed to generate rover motion planning.) The RCW operator can fly a 3-D rover icon through the stereoscopic display of the Martian terrain. By inspecting the stereo scene, as well as placing the rover icon in various positions within the scene, the operator can assess the trafficability of the terrain. By placing the icon in the appropriate position and orientation directly over the stereo image of the actual rover on the surface, the rover's location and heading are automatically computed. This position information is output to ASPEN to set the rover end position state. The rover driver specifies the rover's destinations by designating a series of waypoints in the scene, generating waypoint traverse commands. The traverse commands are only a small fraction of most command sequences.

Rover data storage is a scarce resource that must be tracked within the ASPEN model. The largest consumer of data storage is the imaging data taking activity. This activity can fill the on-board data storage if a telemetry session with the lander is not available during the data collection. ASPEN will keep track of the data storage resource to ensure that all data is downlinked before the buffer is completely full.

Status

Initial work in 1998 consisted of a preliminary proof of concept demonstration in which we used automated planning and scheduling technology integrated with WITS to demonstrate automated commanding for the Rocky-7 rover from the WITS interface. In 2000, we are providing an in-depth validation of the automated command-generation concept. The ASPEN planning and scheduling system will be integrated with the WITS rover

commanding interface and the Rover Control Workstation. High-level requests will be received through the WITS interface and ASPEN will automatically generate validated rover-command sequences that satisfy these requests and provide those sequences to the Rover Control Workstation. A Java-based interface will be integrated with the WITS interface to enable the user to access planned activities and to observe resource and state constraints. As the enhanced WITS interface will be Java-based, users will be able to access this commanding capability from anywhere on the Internet. The computation intensive aspects of the commanding capability (such as the planner/scheduler, path planner, uncertainty estimation software, vision and image processing software, etc.) will reside on one or more rover workstations based in a central location.

The end-to-end data flow for this system is shown in Figure 5. The interaction between ASPEN and RCW is an iterative process. Both ASPEN and RCW will receive high-level goals. RCW will generate initial traverse commands for input into ASPEN. ASPEN will merge these with other goals to produce an intermediate level plan. The plan will be output to RCW. This process will continue until an acceptable plan is generated. Finally a time ordered list of commands will be output for sequence generation.

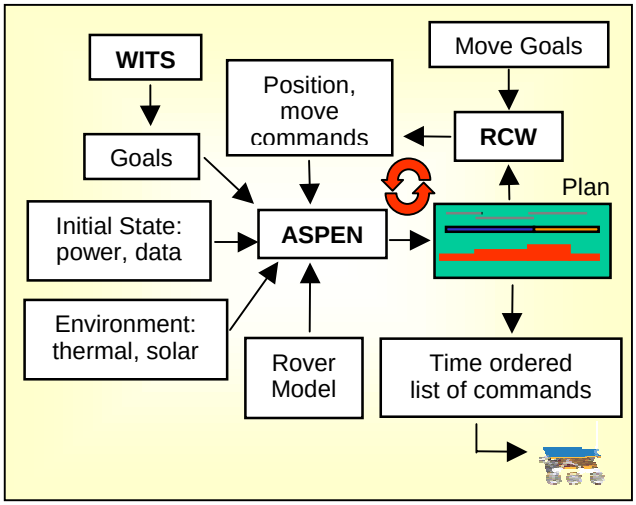


Figure 5 - End-to-End Automated Commanding System

The Marie Curie ASPEN model is nearly complete and ready for testing. Initial testing on a sample of 136 activities produced a conflict free plan in about 9 seconds. This testing was completed on a Sun Ultra-2 workstation. Relatively quick plan cycles will allow the Marie Curie Rover operations team to perform what-if analysis on different daily plans. In addition, the capability could be used to generate commands more frequently than once-per-day, if communications opportunities permit.

Our next level of testing will involve generating plans for two typical Sojourner rover days on Mars. These plans will be compared with the manually generated sequences that were run during the Sojourner mission. As a result of these tests, minor updates to the model may be required.

Once the model is validated, we will integrate ASPEN with RCW and WITS. Figure 6 shows a possible Marie Curie rover uplink operational data flow. The highlighted boxes show the planner that would be used at both the science planning and engineering planning level. The planner model would contain sufficient engineering information to ensure that the vast majority of science requests finally approved are feasible from an engineering standpoint. Eventually we would like to add performance metrics to the planner model to optimize the generated plans. This will enable automated "what-if" analysis to generate plans that maximize science and engineering value.

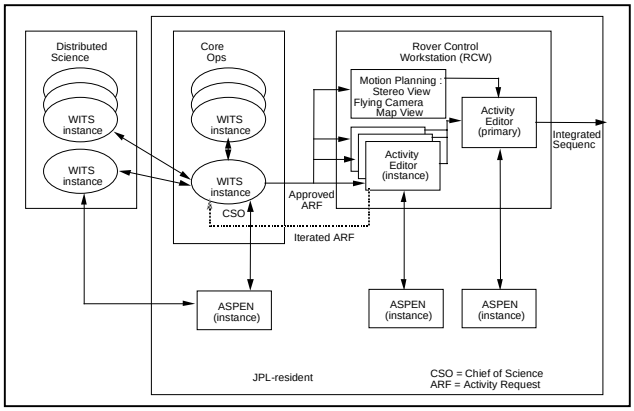


Figure 6 - Possible Rover Uplink Dataflow

Related Work

We are also developing a dynamic, onboard planning system for rover sequence generation. The CASPER (Continuous Activity Scheduling, Planning, Execution and Re-planning) system (Chien et al., 1999), is a dynamic extension to ASPEN, which can not only generate rover command sequences but can also dynamically modify those sequences in response to changing operating context. CASPER produces plans by utilizing an iterative repair algorithm, which classifies plan conflicts and resolves them by performing one or more plan modifications. If orbital or descent imagery is available, CASPER interacts with a tangent graph path planner to estimate traversal lengths and to determine intermediate waypoints that are needed to navigate around known obstacles.

Once a plan has been generated it is continuously updated during plan execution to correlate with sensor and other feedback from the environment. In this way, the planner is highly responsive to unexpected changes, such as an unexpected fortuitous event or equipment failure, and can quickly modify the plan as needed. For example, if the rover wheel slippage has caused the position estimate uncertainty to grow too large, the planner can immediately command the rover to stop and perform localization earlier than originally scheduled. Or, if a particular traversal has

used more battery power than expected, the planner may need to discard one of the remaining science goals. CASPER has been integrated with control software from the JPL Rocky 7 rover (Volpe et al., 2000) and is currently being tested on Rocky 7 in the JPL Mars Yard.

Conclusions

Current approaches to rover-sequence generation and validation are largely manual, resulting in an expensive, labor and knowledge intensive process. This is an inefficient use of scarce science-PI and key engineering staff resources. Automation as targeted by this system would automatically generate a constraint and flight rule checked time ordered list of commands and provides resource analysis options to enable users to perform more informative and fast trade-off analyses. Additionally, this technology would coordinate sequence development between science and engineering teams and would thus help speed up the consensus process.

Enabling goal-driven commanding of planetary rovers by engineering and science personnel greatly reduces the workforce requirements for highly skilled rover engineering personnel. The reduction in team size in turn greatly reduces mission operations costs. In addition, goal-driven commanding permits a faster response to changes in rover state (e.g., faults) or science discoveries by removing the time consuming manual sequence validation process, allowing rapid what-if analyses, and thus reducing overall cycle times.

Acknowledgement

The research described in this paper was funded by the TMOD Technology Program and performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

S. Chien, R. Knight, A. Stechert, R. Sherwood, G. Rabideau, "Integrated Planning and Execution for Autonomous Spacecraft", Proceedings of the IEEE Aerospace Conference (IAC), Aspen, CO, March 1999.

A. Fukunaga, G. Rabideau, S. Chien, D. Yan, "Toward an Application Framework for Automated Planning and Scheduling," Proceedings of the 1997 International Symposium on Artificial Intelligence, Robotics and Automation for Space, Tokyo, Japan, July 1997.

A. Mishkin, "Field Testing on Mars: Experience Operating the Pathfinder Microrover at Ares Vallis," presentation at Field Robotics: Theory and Practice workshop, May 16 1998, at the 1998 IEEE International Conference on Robotics and Automation, Leuven, Belgium.

A. Mishkin, J. Morrison, T. Nguyen, H. Stone, B. Cooper, B. Wilcox, "Experiences with Operations and Autonomy of the Mars Pathfinder Microrover," proceedings of the 1998 IEEE Aerospace Conference, March 21-28 1998, Snowmass at Aspen, Colorado.

G. Rabideau, R. Knight, S. Chien, A. Fukunaga, A. Govindjee, “Iterative Repair Planning for Spacecraft Operations in the ASPEN System,” International Symposium on Artificial Intelligence Robotics and Automation in Space (ISAIRAS), Noordwijk, The Netherlands, June 1999.

R. Volpe, T. Estlin, S. Laubach, C. Olson, and J. Balaram, "Enhanced Mars Rover Navigation Techniques" To appear in the Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA, April 2000.

Zweben, M., Daun, B., Davis, E., and Deale, M., “Scheduling and Rescheduling with Iterative Repair,” Intelligent Scheduling, Morgan Kaufmann, San Francisco, 1994, pp. 241-256.

Antenna Track Plan Generation Through AI Planning

Forest Fisher, Tara Estlin, Darren Mutz, and Steve Chien

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109-8099
{firstname.lastname}@jpl.nasa.gov

Abstract

This paper describes the application of Artificial Intelligence planning techniques to the problem of antenna track plan generation for a NASA Deep Space Communications Station. The described system enables an antenna communications station to automatically respond to a set of tracking goals by correctly configuring the appropriate hardware and software to provide the requested communication services. To perform this task, the Automated Scheduling and Planning Environment (ASPEN) has been applied to automatically produce antenna tracking plans that are tailored to support a set of input goals. In this paper, we describe the antenna automation problem, the ASPEN planning and scheduling system, how ASPEN is used to generate antenna track plans, the results of several technology demonstrations, and future work utilizing dynamic planning technology.

INTRODUCTION

The Deep Space Network (DSN) [4] was established in 1958 and since then it has evolved into the largest and most sensitive scientific telecommunications and radio navigation network in the world. The purpose of the DSN is to support unmanned interplanetary spacecraft missions and support radio and radar astronomy observations in the exploration of the solar system and the universe. The DSN currently consists of three deep-space communications facilities placed approximately 120 degrees apart around the world: at Goldstone, in California's Mojave Desert; near Madrid, Spain; and near Canberra, Australia. This strategic placement permits constant observation of spacecraft as the Earth rotates, and helps to make the DSN the largest and most sensitive scientific telecommunications system in the world. Each DSN complex operates four deep space stations -- one 70-meter antenna, two 34-meter antennas, and one 26-meter antenna. The functions of the DSN are to receive telemetry signals from spacecraft, transmit commands that control the spacecraft operating modes, generate the radio navigation data used to locate and guide the spacecraft to its destination, and acquire flight radio science, radio and radar astronomy, very long baseline interferometry, and geodynamics measurements.

This NASA Planning and Scheduling workshop paper is a shortened version of a IAAI-00 paper [7].

From its inception the DSN has been driven by the need to create increasingly more sensitive telecommunications devices and better techniques for navigation. The operation of the DSN communications complexes requires a high level of manual interaction with the devices in the communications link with the spacecraft. In more recent times NASA has added some new drivers to the development of the DSN: (1) reduce the cost of operating the DSN, (2) improve the operability, reliability, and maintainability of the DSN, and (3) prepare for a new era of space exploration with the New Millennium program: support small, intelligent spacecraft requiring very few mission operations personnel [10].

This paper addresses the problem of automated track plan generation for the DSN, i.e. automatically determining the necessary actions to set up a communications link between a deep space antenna and a spacecraft. Similar to many planning problems, track plan generation involves elements such as subgoaling to achieve preconditions and decomposing high-level (abstract) actions into more detailed sub-actions. However, unlike most classical planning problems, the problem of track generation is complicated by the need to reason about issues such as metric time, DSN resources and equipment states. To address this problem, we have applied the Automated Scheduling and Planning Environment (ASPEN) to generate antenna track plans on demand.

ASPEN [1,9] is a generic planning and scheduling system being developed at JPL that has been successfully applied to problems in both spacecraft commanding and maintenance scheduling and is now being adapted to generate antenna track plans. ASPEN utilizes techniques from Artificial Intelligence planning and scheduling to automatically generate the necessary antenna command sequence based on input goals. This sequence is produced by utilizing an "iterative repair" algorithm [9,11,14], which classifies conflicts and resolves them each individually by performing one or more plan modifications. This system has been adapted to input antenna tracking goals and automatically produce the required command sequence to set up the requested communications link.

This work is one element of a far-reaching effort to upgrade and automate DSN operations. The ASPEN Track Plan

Generator has been demonstrated in support of the Deep Space Terminal (DS-T), which is a prototype 34-meter deep space communications station intended to be capable of fully autonomous operations [5,6,7].

This rest of this paper is organized in the following manner. We begin by characterizing the current mode of operations for the DSN, and then describe the track plan generation problem. Next, we introduce the ASPEN planning and scheduling system and describe its modeling language and search algorithm(s). We then present an operations example of using this system for track plan generation and discuss several successful demonstrations that were performed with Mars Global Surveyor using a 34-meter antenna station in Goldstone, CA. Finally, we discuss some related work and describe current efforts to expand this system to incorporate a dynamic planning approach which will allow for closed-loop control and automatic error recovery when executing a DSN antenna track.

HOW THE DSN OPERATES

The DSN track process occurs daily for dozens of different NASA spacecraft and projects which use the DSN to capture spacecraft data. Though the process of sending signals from a spacecraft to Earth is conceptually simple, in reality there are many earthside challenges that must be addressed before a spacecraft's signal is acquired and successfully transformed into useful information. In the remainder of this section, we outline some of the steps involved in providing tracking services and in particular discuss the problem of track plan generation.

The first step in performing a DSN track is called network preparation. Here, a project sends a request for the DSN to track a spacecraft involving specific tracking services (e.g. downlink, uplink). The DSN responds to the request by attempting to schedule the necessary resources (i.e. an antenna and other shared equipment) needed for the track. Once an equipment schedule and other necessary information has been determined, the next step is the data capture process, which is performed by operations personnel at the deep space station. During this process, operators determine the correct steps to perform the following tasks: configure the equipment for the track, perform the actual establishment of the communications link, and then perform the actual track by issuing control commands to the various subsystems comprising the link.

Throughout the track the operators continually monitor the status of the link and handle exceptions (e.g. the receiver breaks lock with the spacecraft) as they occur. All of these actions are currently performed by human operators, who manually issue tens or hundreds of commands via a computer keyboard to the link subsystems. This paper discusses the application of the ASPEN planning system to automatically generate DSN track plans (i.e. the steps necessary to set up and perform the requested track) and dramatically reduce the need for many manual steps.

TRACK PLAN GENERATION: THE PROBLEM

Generating an antenna track plan involves taking a general service request (such as telemetry - the downlink of data from a spacecraft), an antenna knowledge-base (which provides the information on the requirements of antenna operation actions), and other project specific information (such as the spacecraft sequence of events), and then generating a partially-ordered sequence of commands. This command sequence will properly configure a communications link that enables the appropriate interaction with the spacecraft. To automate this task, the ASPEN planning and scheduling system has been applied to generate antenna operation procedures on demand.

ASPEN has been adapted to use high-level antenna track information to determine the appropriate steps, parameters on these steps and ordering constraints on these steps that will achieve the input track goals. In generating the antenna track plan, the planner uses information from several sources (see Figure 1):

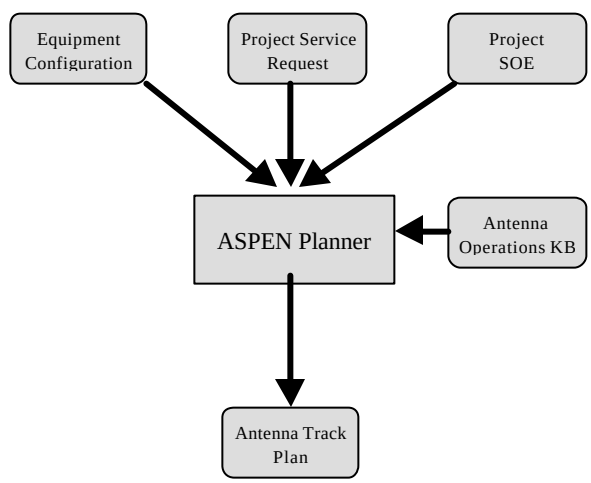


Figure 1 ASPEN Inputs and Outputs

Project Service Request - The service request specifies the DSN services (e.g. downlink, uplink) requested by the project and corresponds to the goals or purpose of the track.

Project SOE - The project sequence of events (SOE) details spacecraft events occurring during the track - including the timing of the beginning and ending of the track and spacecraft data transmission bit rate changes, modulation index changes, and carrier and subcarrier frequency changes.

Antenna Operations KB - The Antenna Operations Knowledge Base (KB) stores information on available antenna operations actions/commands. This KB dictates

```
Pre_track pre_track1{
  Start_time = 1998-213/13:32:26;
  End_time = 1998-213/13:47:26;
};

Track Track1{
  Start_time = 1998-213/13:47:26;
  End_time = 1998-213/16:40:00;
};

Post_track post_track1{
  Start_time = 1998-213/16:40:00;
  End_time = 1998-213/16:50:00;
};
```

Figure 3 Activity Instantiations

how actions can be combined to provide essential communication services. Specifically, this includes information such as action preconditions, postconditions, and command directives and also includes any other relevant information such as resource and state descriptions.

Equipment Configuration - This configuration details the types of equipment available and includes items such as the antenna, antenna controller, the receiver, etc.

The ASPEN Planning System

ASPEN is a reusable, configurable, generic planning/ scheduling application framework that can be tailored to specific domains to create conflict-free plans or schedules.

Due to space constraints details on the ASPEN planning system have been left out in favor of discussion on how ASPEN has been used in this specific application. For more details on ASPEN, see [1,9] and for details on ASPEN's use in DSN automation see [5,6,7,8].

TRACK PLAN GENERATION: AN EXAMPLE

Given a set of tracking requests, ASPEN can generate a conflict-free track plan within the order of seconds that will correctly set up the requested communications link. In order to begin the planning process, the tracking service request, the equipment configuration, and the project SOE are parsed and relevant information is placed in a initial setup file which lists the requested track goals and any relevant initial state information. For example, Figure 3 shows three activity instantiations that request that a "Pre_track", "Track" and "Post_track" activity be placed in the final plan at specific times.

ASPEN then decomposes these activities into the necessary steps that set up the antenna and subsystems (i.e. "Pre_track"), that perform the track (i.e. "Track"), and

```
Configure_equipment:

Start jsc_asn.prc(dss,sc,pass,&ret_status)
If (!ret_status) then
    Write("fatal error: cannot start pass")
    Goto fatal_err
Endif

Start ugc_hi.prc
If (!ret_status) then
    Write("fatal error: can't control UGC")
    Goto fatal_err
Endif

Start apc_hi.prc
If (!ret_status) then
    Write("fatal error: can't control APC")
    Goto fatal_err
Endif
.
.
.

Point_antenna:

Ret_status = exec("APC DCOS")
Start apc_track.prc(&ret_status)
If (!ret_status) then
    Write("fatal error: cannot point ant")
    Goto fatal_err
Endif
```

Figure 4 Antenna Control Script

that perform the necessary shutdown procedures once the track had ended (i.e. "Post_track"). Other initial state information is provided in a "Set_state_values" activity, which sets up the appropriate state variables. The information includes the spacecraft ID, antenna ID, the tracking goals, the carrier and sub-carrier frequency, the symbol rate, etc. ASPEN is also provided with the model files that hold the relevant activity, parameter, resource and state definitions, which were explained in the previous section.

Once the initial goals and state information are loaded, ASPEN utilizes its iterative repair algorithm to create a conflict-free track plan that provides the requested services. This final plan contains a large amount of information, including a list of grounded activities (where each activity has been assigned a start time and end time), and a list of constraints over those activities, including temporal, parameter, resource and state constraints. ASPEN also displays the final resource and state timelines which show the states of those entities over the course of the plan. The actual antenna control script that will be used to execute the track is output in a separate file which contains the command sequence necessary to set up,

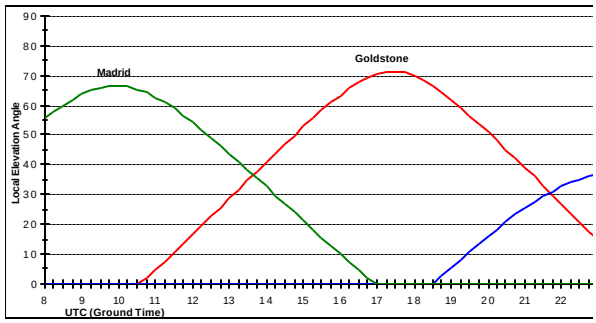
Figure 5 34m BWG Antennas at Goldstone

control and break down the link. In the model definition, a command (or set of commands) can be specified for each defined activity. These commands are then output in the correct sequence based on the final plan constraints. An example of this file format is shown in Figure 4. This control script is then sent to an antenna operator or execution agent where it will be used to perform the requested track.

DS-T DEMONSTRATIONS

The Deep Space Terminal (DS-T) [5,6,7] being developed at the NASA Jet Propulsion Laboratory is a prototype 34-meter deep space communications station intended to be capable of fully autonomous operations. When requested to perform a track, the DS-T station automatically performs a number of tasks (at appropriate times) required to execute the track. First, the Schedule Executive sets up the track schedule for execution and provides the means for automated rescheduling and/or manual schedule editing in the event of changes. The Configuration Engine is then responsible for retrieving all the necessary data needed for station operations. Next, the Script Generator (ASPEN) generates the necessary command sequence to perform the track. Finally, a Station Monitor and Control process executes the generated script and records relevant monitor data generated during the track.

The DS-T concept was validated through a number of demonstrations. The demonstrations began with the automation of partial tracks in April 1998, continued with 1-day unattended operations in May, and concluded with a 6-day autonomous *"lights-out"* demonstration in September 1998. Throughout these demonstrations ASPEN was used to automatically generate the necessary command sequences for a series of Mars Global Surveyor (MGS) downlink tracks using the equipment configuration at Deep Space Station 26 (DSS26), a 34-meter antenna located in Goldstone, CA. These command sequences were produced and executed in a fully autonomous fashion with no human intervention. During the September demonstration performed all Mars Global Surveyor coverage scheduled for the Goldstone antenna complex. This corresponded to roughly 13 hours of continuous track coverage per day.



Track Segments:

#1	11:15:00 – 14:00:23	3way/65	(Madrid)
#2	14:00:23 – 15:42:26	LOS	
#3	15:42:26 – 15:52:23	1way	
#4	15:42:26 – 15:52:23	3way/25	(Goldstone)
#5	15:52:23 – 20:03:34	1way	
#6	20:03:34 – 23:40:00	3way/34	(Canberra)

Figure 6 September 16, 1998 MGS Track

In Figure 5, we show a picture of the three 34-meter Beam Wave Guide (BWG) antennas at the Goldstone, CA facility. In the foreground is DSS-26, which was the station selected for prototyping the DS-T.

While the overall DS-T effort consisted of a large team and a project duration of approximately 1.5 years, the DS-T automation team consisted of three team members. Of this teams work, approximately one work year was spent on the script generation effort. This effort primarily consisted of knowledge acquisition and model development, while a small effort was made in the integration of the script generator. A key factor in the quick development was the ability to adapt a general purpose planning and scheduling system. As the domain of ground communication-station commanding shared many similarities to spacecraft commanding, ASPEN seemed like a logical choice. This was confirmed by the ease of knowledge base development and integration. Spacecraft commanding also consists of generating a sequence of commands, however it is predominately a resource-scheduling problem, whereas ground-station commanding is predominately a sequencing problem.

RESULTS

In order to provide qualitative results, we present statistical data from September 16, 1998, a representative day during our 6-day autonomous unattended demonstration, durring which we collected above 90% of the transmitted frames. This performance is on par with the operator-controlled stations, however required no support personnel (i.e. reduced operations cost).

In figure 6, the graph represents when MGS was in view of the ground stations at each of the three complexes (Madrid, Goldstone, and Canberra). DS-T, which is located at

Goldstone, tracked MGS through the five track segments indicated in the figure 6.

Before continuing with the analysis of the results, let us explain the different modes indicated in figure 6 for each of the different track segments. When a spacecraft is downlinking data it is said to be in 1way mode. When an uplink and a downlink are taking place simultaneously the spacecraft is said to be in 2way mode. If a station is communicating in 2way mode with a spacecraft, and another station is listening in on the downlink of the spacecraft, the second station is said to be in 3way with the 2way station. Because DS-T is not equipped for uplink , DS-T operates in either 1way or 3way mode. Because the downlink frequency is relative to the uplink frequency, it is critical to determine the station involved in the uplink when taking part in a 3way mode of operations. In this example, during segment 4 dss25 (deep space station) was in 2way and DS-T was in 3way with 25 (3way/25).

Track segment 2, which is labeled LOS, indicates that there was a scheduled loss of signal (LOS) so during this segment no frames were collected. During each of the other respective track segment DS-T collected 75%, 91%, 96%, 90%, 23% of the broadcasted frames. As shown by the graph, during segment 1 and 6 the elevation of the dish is low in the sky. Under these circumstances there is considerably more atmospheric interference which explains the lower percent of frame collection. On the other hand, if you look at segment 4 where there is a long segment with the spacecraft high in the sky the data collection is quite high. In segment 3 and 5 the values are a little lower due to the shortness of the segments. This is explained by the fact that some data is lost during a change in mode, as in the transition from LOS to 1way and 3way/25 to 1way.

As a component of the DS-T demonstrations, the SG performed flawlessly, producing dynamically instantiated control scripts based on the desired service goals for the communications pass as specified in the service request. The use of such technology resulted in a three primary benefits:

- Autonomous operations enabled by eliminating the need for hundreds of manual inputs in the form of control directives. Currently the task of creating the communications link is a manual and time-consuming process which requires operator input of approximately 700 control directives and the constant monitoring of several dozen displays to determine the exact execution status of the system.
- Reduced the level of expertise of an operator required to perform a communication track. Currently the complex process requires a high level of expertise from the operator, but through the development of the KB by a domain expert this expertise is captured with in the system itself.
- The KB provides a declarative representation of operation procedures. Through the capture of this expertise the KB documents the procedural steps of performing antenna communication services.

RELATED WORK

There are a number of existing systems built to solve real-world planning or scheduling problems [12,13,14]. The problem of track plan generation combines elements from both these fields and thus traditional planners and schedulers cannot be directly applied. First, many classical planning elements must be addressed in this application such as subgoaling to achieve activity preconditions (e.g. the antenna must be "on_point" to lock up the receiver) and decomposing higher-level (abstract) activities into more detailed sub-activities. In addition, many scheduling elements are presents such as handling metric time and temporal constraints, and representing and reasoning about resources (e.g. receiver, antenna controller) and states (e.g. antenna position, subcarrier frequency, etc.) over time.

One other system has been designed to generate antenna track plans, the Deep Space Network Antenna Operations Planner (DPLAN) [2]. DPLAN utilizes a combination of AI hierarchical-task network (HTN) and operator-based planning techniques. Unlike DPLAN, ASPEN has a temporal reasoning system for expressing and maintaining temporal constraints and also has the capability for representing and reasoning about different types of resources and states. ASPEN can utilize different search algorithms such as constructive and repair-based algorithms, where DPLAN uses a standard best-first based search. And, as described in the next section, ASPEN is currently being extended to perform dynamic planning for closed-loop error recovery, where DPLAN has only limited replanning capabilities.

FUTURE WORK: PROVIDING CLOSED-LOOP CONTROL THROUGH DYNAMIC PLANNING

Currently, we are working on modifying and extending the current ASPEN Track Plan Generator to provide a Closed Loop Execution and Recovery system (CLEaR) for DSN track automation [8]. CLEaR is a real-time planning system built as an extension to ASPEN [3]. The approach taken is to dynamically feed monitor data (sensor updates) back into the planning system as state updates. As these dynamic updates come in, the planning system verifies the validity of the current plan. If a violation is found in the plan, the system will perform local modification to construct a new valid plan. Through this continual planning approach, the plan is disrupted as little as possible and the system is much more responsive and reactive to changes in the real (dynamic) world.

This CLEaR effort is also being integrated with a Fault Detection, Isolation and Recovery (FDIR) system. FDIR is

an expert system providing monitor data analysis. As is often the case with large complex systems, monitor (sensor) data is often related in different ways that becomes difficult for a human to detect. The advantage of combining these two systems is that FDIR can first interpret the vast amount of data and summarize it into a set of meaningful values for a planning system to react to. We think of this union as intelligent analysis and intelligent response, much like a careful design and implementation; one without the other is of little use.

CONCLUSIONS

This paper has described an application of the ASPEN automated planning system for antenna track plan generation. ASPEN utilizes a knowledge base of information on tracking activity requirements and a combination of Artificial Intelligence planning and scheduling techniques to generate antenna track plans that will correctly setup a communications link with spacecraft. We also described several demonstrations that have been performed as part of the DS-T architecture where ASPEN was used to generate plans for downlink tracks with Mars Global Surveyor. Finally, we described a planned extension of this system, which will allow for closed-loop control, error recovery and fault detection using dynamic planning techniques.

ACKNOWLEDGEMENTS

This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract of the National Aeronautics and Space Administration. We thank members of the ASPEN scheduling team and members of the DS-T automation team for contributing to this work.

References

[1] S. Chien, D. Decoste, R. Doyle, and P. Stolorz, “Making an Impact: Artificial Intelligence at the Jet Propulsion Laboratory,” AI Magazine, 18(1), 103-122, 1997.

[2] S. Chien, R. Hill Jr., A. Govindjee, X. Wang, T. Estlin, A. Griesel, R. Lam and K. Fayyad, “A Hierarchical Architecture for Resource Allocation, Plan Execution, and Revision for Operation of a Network of Communication Antennas,” Proceedings of the 1997 IEEE Conference on Robotics and Automation, Albuquerque, NM, April 1997.

[3] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau, “Integrated Planning and Execution for Autonomous Spacecraft,” To appear in the Proceedings of the 1999 IEEE Aerospace Conference, Aspen, CO, March, 1999.

[4] Deep Space Network, Jet Propulsion Laboratory Publication 400-517, April 1994.

[5] F. Fisher, S. Chien, L. Paal, E. Law, N. Golshan, and M. Stockett, “An Automated Deep Space Communications Station,” Proceedings of the 1998 IEEE Aerospace Conference, Aspen, CO, March 1998.

[6] F. Fisher, D. Mutz, T. Estlin, L. Paal, and S. Chien, “The Past, Present and Future of Ground Station Automation with in the DSN,” To appear in the Proceedings of the 1999 IEEE Aerospace Conference, Aspen, CO, March 1999.

[7] F. Fisher, T. Estlin, D. Mutz, S. Chien, "Using Artificial Intelligence Planning to Generate Antenna Tracking Plans", Eleventh Annual Conference on Innovative Applications of Artificial Intelligence (IAAI-99), Orlando, Florida, July 1999.

[8] F. Fisher, R. Knight, B. Engelhardt, S. Chien, N. Alejandre, "A Planning Approach to Monitor and Control for Deep Space Communications", Proceedings of the IEEE Aerospace Conference (IAC), Big Sky, MT, March 2000.

[9] A. Fukanaga, G. Rabideau, S. Chien, and D. Yan, “Toward an Application Framework for Automated Planning and Scheduling,” Proceedings of the 1997 International Symposium of Artificial Intelligence, Robotics and Automation for Space, Tokyo, Japan, July 1997.

[10] R. W. Hill, Jr., S. A. Chien, K. V. Fayyad, C. Smyth, T. Santos, and R. Bevan, “Sequence of Events Driven Automation of the Deep Space Network,” Telecommunications and Data Acquisition 42-124, October-December 1995.

[11] S. Minton and M. Johnston, “Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems,” Artificial Intelligence, 58:161-205, 1988.

[12] A. Tate, B. Drabble and R Kirby, “O-Plan2: An Open Architecture for Command Planning and Control,” Intelligent Scheduling (Eds. M. Fox & M. Zweben), Morgan Kaufmann, 1994.

[13] D. Wilkins Practical Planning: Extending the Classical AI Planning Paradigm, Morgan Kaufmann, 1994.

[14] M. Zweben, B. Daun, E. Davis, and M. Deale, Scheduling and Rescheduling with Iterative Repair, in Intelligent Scheduling, Morgan Kaufmann, 1994.

Utility Estimation for Contingent Execution: Preliminary Report

Richard Washington* **and John L. Bresina**

NASA Ames Research Center, M/S 269-2
Moffett Field, CA 94035

{rwashington, jbresina}@arc.nasa.gov

Abstract

This paper presents a method for using expected utility distributions in the execution of flexible, contingent plans. A utility distribution maps the possible start times of an action to the expected utility of the plan suffix starting with that action. The contingent plan encodes a tree of possible courses of action and includes flexible temporal constraints and resource constraints. When execution reaches a branch point, the eligible option with the highest expected utility at that point in time is selected. The utility distributions make this selection sensitive to the runtime context, yet still efficient. Our approach uses predictions of action duration uncertainty as well as expectations of resource usage and availability to determine when an action can execute and with what probability. Execution windows and probabilities inevitably change as execution proceeds, but such changes do not invalidate the cached utility distributions; thus, dynamic updating of utility information is minimized.

Introduction

The work reported here is part of a research program to develop robust, autonomous planetary rovers (Washington, *et al.*, 1999). Traditionally, spacecraft have been controlled through a time-stamped sequence of commands (Mishkin, et al., 1998). The rigidity of this approach presents particular problems for rovers: since rovers interact with their environment in complex and unpredictable ways and since the environment is unknown or poorly modeled, the rover's actions are highly uncertain. We have developed a temporally flexible, contingent planning language, which enables the specification of rover actions that can adapt to the changing execution situation. The plan language is called the *Contingent Rover Language* or *CRL* (Bresina, *et al.*, 1999). CRL allows a rich specification of preconditions, maintenance conditions, and end conditions for actions. These conditions can include absolute and relative temporal constraints, resource constraints (*e.g.*, power), as well as constraints on the rover's state.

*NASA Ames contractor with Caelum Research Corp.

A contingent plan is a tree of possible courses of action; when execution reaches a branch point, the rover's on-board executive selects the eligible option with the highest expected utility. If all the actions were time-stamped, then it would suffice to precompute the expected utility for each contingent option, using classical decision theory. However, because the actions in a CRL plan can start within a flexible temporal interval, the expected utilities of the contingent options depend on the time that the branch point is reached during execution. Hence, a single utility measure is insufficient, and we need to compute a utility distribution that maps possible action start times to the expected *plan-suffix utility*, *i.e.*, the expected utility of executing the plan suffix starting with that action.

Expected plan-suffix utility depends on when actions can execute and with what probability. The time over which an action executes and the probabilities of success and failure are affected by all the constraints in the action's conditions (pre-, maintain, and end), as well as by the inherent uncertainty in action durations. As plan execution proceeds, the temporal windows for plan actions narrow, resource availability can change, and rover state can change in unpredictable ways. Such changes affect the execution time and success probabilities and, thus, the expected utilities. Note, however, that even though temporal changes can affect the probabilities of when future actions will start, the plan-suffix utility distributions of these actions do not have to be recomputed because they are conditioned on start time. Although the use of utility distributions does reduce utility recomputations, it does not eliminate them; *e.g.*, changes in resource availability can require dynamic utility updates.

In contrast to classical decision-theoretic frameworks, the uncertainty arises from an interaction of action conditions and execution time, which is uncertain because of variations in action durations. Modeling this with decision-theoretic tools would require covering the spaces of possible action times and available re-

sources. Thus, a decision-theoretic planning approach that *a priori* considers all possible decision points and pre-compiles an optimal policy is not practical.

In this paper, we present an approach for estimating the expected plan-suffix utility distribution in order to make runtime decisions regarding the best course of action to follow within a flexible, contingent plan. Our method takes into account the impact of temporal and resource constraints on possible execution trajectories and associated probabilities by using predictions of action duration uncertainty and expectations of resource usage and availability.

Plan-Suffix Utility Distributions

The utility of a plan depends on the time that each action starts, when it can execute, and its constraints. In CRL, an action may be constrained to execute within an interval of time, specified either in relative or absolute terms. In a plan with this type of temporal flexibility, the exact moment that a future action will execute cannot, in general, be predicted. We use a probability density function (PDF) to represent the probability of an action starting (or executing, or ending) at a particular time. The focus of this paper is on the ability to estimate the expected utility of a sequence of actions by propagating these PDFs from action to action. The propagation uses the temporal and resource conditions of the action to restrict the action's execution times.

The plan-suffix utility of an action is a mapping from times to values: $u(S,t)$ is the utility of starting execution of a plan suffix S at time t. The terminal case is $u(\{\},t) = 0$. For a plan $\{a, S\}$, denoting action a followed by the plan suffix S, there are two cases, depending on whether failure of a causes general plan failure or not. Let us denote $p_{success}(t'|a,t)$ as the probability of success of a at time t' given that it started at time t, $p_{failure}(t'|a,t)$ as the probability of a's failure at time t', and v_a as the fixed local reward for successfully executing action a. If the plan fails when a fails,

$$u(\{a,S\},t) = \int_{-\infty}^{\infty} p_{success}(t'|a,t) \cdot (v_a + u(S,t'))\ dt'$$

If the plan continues execution when a fails,

$$\begin{aligned} u(\{a,S\},t) \quad &= \textstyle\int_{-\infty}^{\infty} \quad [\, p_{success}(t'|a,t) \cdot (v_a + u(S,t')) + \\ &\qquad p_{failure}(t'|a,t) \cdot u(S,t')\,]\ dt' \end{aligned}$$

In the case of a branch point b with possible suffixes $\mathcal{S}_b = \{S_1, ...S_n\}$, the plan-suffix utility $u(\{b, \mathcal{S}_b\}, t)$ is a function of the utilities of each possible suffix:

$$u(\{b,\mathcal{S}_b\},t) = \sum_{S_i \in S_b} \int_{-\infty}^{\infty} p_{select}(S_i,t) \cdot u(S_i,t) dt$$

where $p_{select}(S_i,t)$ is the probability of suffix S_i being selected at time t (0 if the eligibility condition is unsatisfied). This is an average of the individual suffix utilities, weighted by the selection probabilities.

Given a planning language with a rich set of temporal, resource, and state conditions, the functions $p_{success}$ and $p_{failure}$ do not allow closed-form calculation of the plan-suffix utilities. We solve this by discretizing time into bins; the value assigned to a bin approximates the integral over a subinterval. Calculations of the integrals above become summations. The choice of bin size introduces a tradeoff between accuracy and computation cost, which we examine in the section Empirical Results.

Although the utility calculation is defined with respect to an infinite time window, the plan start time, action durations, and action conditions restrict the possible times for action execution and for transitions between actions. In this work we model only temporal and resource conditions; the time bounds we compute may be larger than the real temporal bounds because of the unmodeled conditions.

The basic approach is to propagate the temporal bounds forward in time throughout the plan, producing the temporal bounds for action execution. Those temporal bounds serve as the ranges over which the utility calculations are performed. Outside of these ranges, the plan fails. A failed plan receives the local utility of the actions that succeeded and zero utility for the remainder; failure could be penalized through a simple extension.

The temporal bounds are calculated forward in time because the current time provides the fixed point that restricts relative temporal bounds. The utilities, on the other hand, are calculated backward in time from the end(s) of the plan. The utility estimates are conditioned on the time of transitioning to an action; since they are not dependent on preceding action time PDFs, they remain valid as plan execution advances, barring changes in resource availability.

In the following sections, we describe the elements of an action and present the procedures for propagating temporal bounds and utilities in more detail.

Anatomy of an Action

In the Contingent Rover Language, each action instance includes the following information:

Start conditions. Conditions that must be true for the action to start execution.

Wait-for conditions. A subset of the start conditions for which execution can be delayed to wait for them to become true (by default, unsatisfied start conditions fail the action). Temporal start conditions are

treated as wait-for conditions, and may be absolute or relative to the previous action's end time.

Maintain conditions. Conditions that must be true throughout action execution. Failure of a maintain condition results in action failure.

End conditions. Conditions that must be true at the end of action execution. Temporal end conditions may be absolute or relative to action start time.

Duration. Action duration expressed as an expectation with a mean and standard deviation of a Gaussian distribution. Our approach would work equally well with other models of action duration.

Resource consumption. The amount of resources that the action will consume. It is expressed as an expectation with a fixed value, because we currently assume that resource consumption for a given action is a fixed quantity with no uncertainty.

Continue-on-failure flag. An indication of whether a failure of the action aborts the plan or allows execution to continue to the next action.

Resource conditions considered here are threshold conditions; *i.e.*, they ensure that enough of a given resource exists for the action to execute. The resource profile is an expectation of resource availability over time, represented by a set of temporal intervals with associated resource levels. A resource condition is checked against the availability profile to determine the intervals over which the condition is satisfied.

Temporal Interval Propagation

Each temporal aspect of an action is represented as a set of temporal intervals, and we distinguish the following temporal aspects of an action.

Transition time. The time that the execution of the previous action terminates. This is not the same as start time, since the action's preconditions may delay its execution. The transition-time intervals are the set of possible times that the previous action will transition to this action.

Start time. The time that the action's preconditions are met and it executes. The start-time intervals are the set of possible times that the action will start.

End time. The time at which the action terminates. We distinguish between successful termination and failure, due to condition violation, and determine a set of end-succeed intervals and end-fail intervals.

Execution proceeds according to the following steps:

1. If the current time is already past absolute start bounds, fail this action.

2. Execution waits until all wait-for and lower-bound temporal conditions are true (but if upper-bound temporal conditions are violated at any time, the action fails).

3. The start conditions are checked, and the action fails if any are not true.

4. The action begins execution. If any maintenance conditions fail during execution, the action fails. If the temporal upper bound is exceeded, the action fails.

5. The action ends execution. The end conditions are checked, and the action fails if any are not true.

6. Execution transitions to the next action.

As mentioned earlier, action failure either fails the plan or simply transitions to the next action, as specified within the plan (the continue-on-failure flag).

Temporal bounds and utilities are propagated to reflect the execution steps. We illustrate the temporal interval propagation by demonstrating how the various conditions affect an arbitrary transition-time PDF. The interval propagation is done simply through computations on the bounds, but since the utility computations propagate PDFs, the general case demonstrates the basics underlying both calculations.

Transition time

The possible transition times of the plan's first action is when plan execution starts; typically, this is a single time point (*e.g.*, the set time that the rover "wakes up"). For all other actions, the transition time PDF is determined from the previous action's end time PDFs, as follows. If the previous action's continue-on-failure flag is true, then the possible action transition times are the union of the possible end-succeed times and the end-fail times from the previous action. On the other hand, if the previous action's continue-on-failure flag is false, then the action's transition times are identical to the previous action's end-succeed times.

Start time

Given the possible transition times and a model of resource availability, we determine the set of temporal intervals that describes the possible action start times, along with a set of temporal intervals during which the action will fail before execution begins.

Consider an action with absolute time bounds $[lb_{abs}, ub_{abs}]$ (default $[0, \infty]$) and relative temporal bounds $[lb_{rel}, ub_{rel}]$ (default $[0, \infty]$)[1]. Consider also resource wait-for conditions R_{wait} and resource start conditions R_{start}. For a given resource availability profile, each resource condition r corresponds to a set of time intervals $I_{false}(r)$ for which the resource condition is not true. We define the set of *wait intervals*:

$$I^{wait} = [-\infty, lb_{abs}] \cup \bigcup_{r \in R_{wait}} I_{false}(r).$$

[1]In practice, a finite planning horizon bounds the absolute and relative time bounds; it also bounds the probability reallocation for unmodeled wait-for conditions.

We define the set of *fail intervals*:

$$I^{fail} = [ub_{abs}, \infty] \cup \bigcup_{r \in R_{start}} I_{false}(r).$$

The following rules partition the space of time; they are used to identify the possible start times and the possible fail times, given the conditions. In the rules, t is a given transition time.

1. If $t > ub_{abs}$, then the action fails at time t.

2. Else, if $t + lb_{rel} > ub_{abs}$, then the action fails at time ub_{abs}.

3. Else, if $t+lb_{rel}$ is within a wait interval I_i^{wait}, and the upper bound of the wait interval ub_{wait} is such that $ub_{wait} - t > ub_{rel}$ or $ub_{wait} > ub_{abs}$, then the action fails at time $min(t + ub_{rel}, ub_{abs})$.

4. Else, if $t + lb_{rel}$ is within a wait interval I_i^{wait}, and the upper bound of the wait interval ub_{wait} is such that $ub_{wait} - t \leq ub_{rel}$, then the action waits until time ub_{wait}. If ub_{wait} falls within a fail interval, then the action fails at time ub_{wait}. Otherwise the action starts at time ub_{wait}.

5. Else, if $t+lb_{rel}$ is not within a wait interval I_i^{wait}, and $t + lb_{rel}$ falls within a fail interval, then the action fails at time $t + lb_{rel}$.

6. Finally, if none of the preceding conditions hold, then the action starts at time $t + lb_{rel}$.

If all of the conditions could be accurately modeled, then a transition time would map to a single start time. However, as mentioned earlier, we currently model only temporal and resource conditions. The set of unmodeled conditions adds uncertainty about the time intervals over which the sets of conditions will be true. For start conditions, this adds a fixed probability of failure to every time point. For wait-for conditions, unsatisfied preconditions move probability mass later; to reflect this, we subtract a proportion α of the probability density at each time point and allocate it uniformly to each time later within the absolute bounds; after this, the rules above apply for the modeled conditions.

End time

Here we consider how end times are calculated for an action that has its start conditions true and has started execution. The successful end time of an action is determined by its start time, duration, maintenance conditions, and end conditions. Without maintenance or end conditions, the end time PDF is determined by convolving the start time and duration PDFs; for the bounds, each start time interval $[lb_{start}, ub_{start}]$ and duration interval $[lb_{dur}, ub_{dur}]$ yields an end time interval $[lb_{start} + lb_{dur}, ub_{start} + ub_{dur}]$.[2] All such intervals are unioned to yield the possible end times.

Maintenance conditions restrict the possible end times by defining valid execution time intervals; if execution exits a valid interval, the action fails. End conditions further restrict the successful times; if execution ends when an end condition is not true, the action will fail. The temporal end upper bounds are treated as maintenance conditions so that action execution is bounded.

An action will succeed only if the following four conditions are met:

1. It successfully begins execution.

2. Its start time falls within a valid execution interval. If not, the action will fail at that start time.

3. Its duration is such that its end time falls within the same valid execution interval. If not, the action will fail at the end of this execution interval.

4. The end time falls within a valid end interval. If not, the action fails at the end time.

Utility Propagation

Utility propagation follows the same basic rules as temporal interval propagation in terms of the effects of conditions, but it is calculated during a sweep back from the terminal actions of the plan tree. A terminal action in the plan tree has an empty plan suffix of utility 0. The plan-suffix utility is conditioned on the start time of the action: we calculate the utility of an action and its successors given a particular transition time. The plan-suffix utility composed with a PDF of possible transition times to this action yields the expected utility of the plan suffix starting with this action over the time distribution given by the PDF. Caching the utility conditioned on start times allows an efficient means of choosing the highest utility eligible contingent option.

An action's plan-suffix utility for a given transition time is computed as follows. First, the transition time is propagated to a discrete start time PDF according to the above rules for start time propagation. Second, the convolution of the start time PDF and duration PDF is computed to produce the PDFs for successful end times and failed end times according to the above rules for end time propagation. Third, the success end time PDF is composed with the local value and the plan-suffix utility of the next action to produce the plan-suffix utility for the given transition time. If the action's continue-on-failure flag is set, the failure end time PDF is also composed with the plan-suffix utility of the next action and added to the utility computed from the end time.

[2] To bound the duration interval, we truncate the normal distribution at ± 2 standard deviations and at 0 and then normalize the remaining distribution.

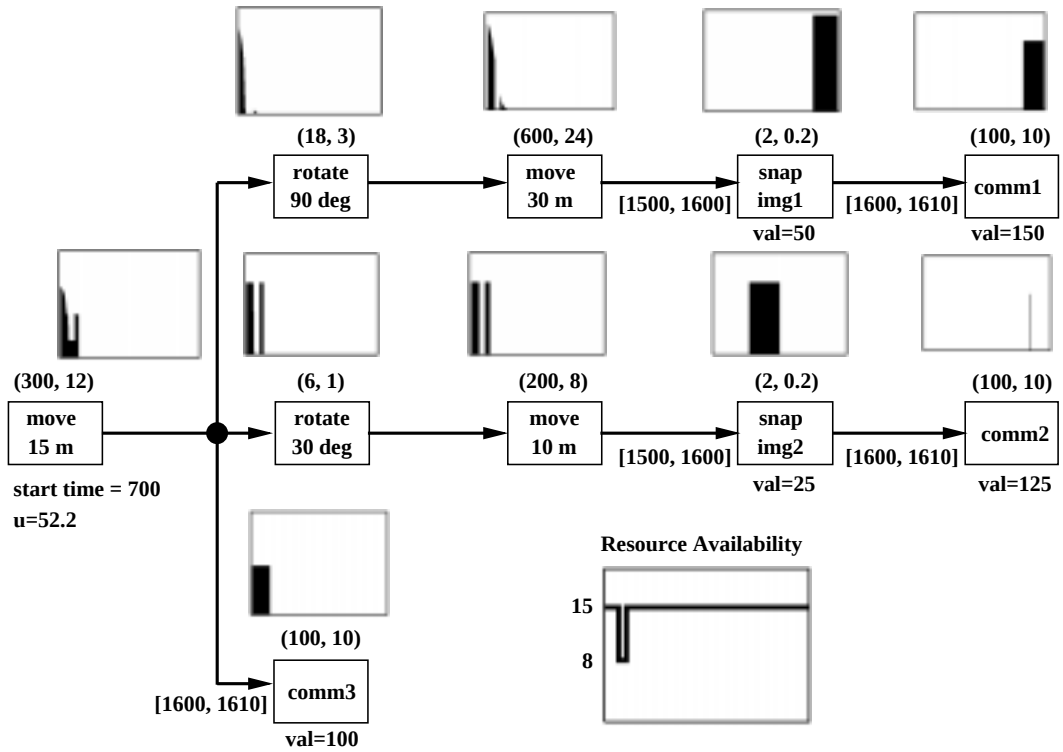


Figure 1: **Example contingent rover plan.** The (μ, σ) above an action indicate the duration's mean and standard deviation. Important start time constraints are shown in square brackets below the arrows. The local value, if nonzero, of an action is indicated below it; these values are assigned by the scientists. For a plan start time of 700, each action's plan-suffix utility distribution is plotted above it; all have x-range $[955, 1605]$ and y-range $[0, 210]$. The leftmost plot is the utility of the branch point. The utility of the entire plan is 52.2. The resource availability profile has x-range $[955, 1605]$ and a resource dip over $[1000, 1025]$.

Empirical Results

To demonstrate our approach, we use a small plan example, which is shown in Figure 1. The plan consists of an initial traversal and then a branch point with the following three contingent options: (*i*) travel toward a farther, but more important science target, capture its image, and communicate the image and telemetry, (*ii*) travel toward a nearer, but less important science target, snap its image, and communicate the image and telemetry, or (*iii*) communicate telemetry.

The communication must start within the interval $[1600, 1610]$. If communication does not happen, then all data is lost; hence, it has a high local value (higher for more important data). Thus, the primary determinant of which option has the highest expected utility is whether there is enough time to execute the communication action. The duration uncertainty of the actions affects the probabilities of successfully completing each of the contingent options and, hence, affects the expected utility. The time that plan execution starts also affects these probabilities and utilities. For the case shown, the start time (700) falls at a time when the first branch of the plan is likely to fail, as is clearly shown by the plan-suffix utility distributions in the figure. In addition, the power availability profile is such that it prevents motion over a small range of time; this is also reflected in the utility distributions.

The principal question that can be tested experimentally concerns the accuracy of the discrete propagation and the tradeoff of discrete bin size versus accuracy. To illustrate these experimentally, we use our example plan and a plan start time of 700 (as shown in Figure 1) and compare the utility of the entire plan when computed using bin sizes of 0.5, 1, 2, 5, 10, 20, 50, and 100; we also estimate the exact plan utility with a 100,000 trial Monte Carlo stochastic simulation. The results are shown in Figure 2. The results show increasing accuracy with decreasing bin size; the largest error is still less than 12%.

Concluding Remarks

In this paper, we presented *expected plan-suffix utility distributions*, described a method for estimating them within the context of flexible, contingent plans, and

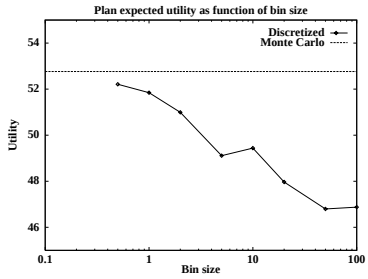


Figure 2: **Tradeoff of accuracy and bin size.** The reference line is the value reached through a Monte Carlo simulation. Note that the x-axis is log scale.

discussed their use for runtime decisions regarding the best course of action to take.

The approach presented in this paper attempts to minimize runtime recomputation of utility estimates. Narrowing the transition intervals of an action does not invalidate its utility distributions. Resource availability changes may affect the times over which an action's conditions are true and, thus, the probability distribution of successful execution. The plan-suffix utility of all actions before an affected action will need to be updated. Actions later than an affected action only need to be updated at newly enabled times.

In contrast to standard decision-theoretic frameworks (Pearl, 1988), uncertainty arises from an interaction of action conditions with an execution time of uncertain duration. Decision-theoretic tools would require covering the spaces of possible action times and available resources; thus, a decision-theoretic planning approach that considers all possible decision points and pre-compiles an optimal policy is not practical.

An earlier effort that propagated temporal PDFs over a plan is *Just-In-Case (JIC)* scheduling of Drummond, *et al.* (1994). The purpose was to calculate schedule break probabilities due to duration uncertainty. Unlike our rich set of action conditions, the only action constraint in the reported telescope scheduling application was a start time interval. JIC used the simplifying assumption that start time and duration PDFs were uniform distributions and that convolution produced a uniform distribution. Our discretized method is more statistically valid and could be used in JIC to increase the accuracy of its break predictions.

An alternative approach to utility estimation is to use Monte Carlo simulation on board, choosing durations and eligible options according to their estimated probabilities. The advantage of simulation is that it is not subject to discretization errors. On the other hand, a large number of samples may be necessary to yield a good estimate of plan utility; furthermore, the length of the calculation is data-dependent (*e.g.*, to reach a particular confidence level). We consider such an approach to be impractical for on-board use, given the computational limitations of a rover.

A number of issues are raised by this approach, and some remain for future work. The combination of plan-suffix utilities at branch points depends on the probability of choosing each sub-branch at each time. Given unmodeled conditions, this can only be estimated, but an interpretation of the conditions on each of the sub-branches can be performed to determine the expected probability of that sub-branch being eligible. If there are times for which more than one sub-branch is potentially eligible, then the resulting utility is some combination of the utility of each sub-branch at that time.

The use of discrete bins in calculating utility introduces error into the calculation; the probabilities and utilities of a precise time point are diffused over surrounding time points. As the chain of actions becomes longer, the inaccuracies grow. Smaller bin sizes minimize the error; however, the utility calculation is in the worst case $\mathcal{O}(n^3)$ for n bins. This tradeoff of accuracy versus computation time requires further study. Bin size could be scaled with the depth of the action in the plan, but this would require frequent recalculations as execution progressed through the plan.

Our approach can be extended by making more realistic modeling assumptions; *e.g.*, modeling uncertainty in resource consumption and modeling hardware failures. One possible next step is to introduce limited plan revision capabilities into the plan to handle cases where all possible plans are of low utility and are thus undesirable. Another extension would be to introduce additional sensing actions to disambiguate multiple eligible options with similar utility estimates.

References

A. H. Mishkin, J. C. Morrison, T. T. Nguyen, H. W. Stone, B. K. Cooper, and B. H. Wilcox. 1998. Experiences with Operations and Autonomy of the Mars Pathfinder Microrover. In *Proc. IEEE Aero. Conf.*.

J. Bresina, K. Golden, D.E. Smith, and R. Washington. Increased flexibility and robustness of Mars rovers. In *Fifth Intl. Symposium on Artificial Intelligence, Robotics and Automation in Space*. Noordwijk, Netherlands. 1999.

M. Drummond, J. Bresina, & K. Swanson. 1994. Just-in-case scheduling. In *Proc. 12th Natl. Conf. Art. Int.*

R. Washington, K. Golden, J. Bresina, D.E. Smith, C. Anderson, and T. Smith. 1999. Autonomous Rovers for Mars Exploration. In *Proc. IEEE Aerospace Conf.*.

J. Pearl. 1988. *Probabilistic Reasoning in Intelligent Systems*. Morgan-Kaufmann.

Mixed Initiative Scheduling for the Air Force Satellite Control Network

Adele E. Howe, and **L. Darrell Whitley** and **Laura Barbulescu** and **Jean-Paul Watson**
Computer Science Department, Colorado State University,
Fort Collins, CO 80523
{howe,whitley,laura,watsonj}@cs.colostate.edu

Abstract

Scheduling access to Air Force satellites is an over-constrained problem. Over 500 requests for access are received for each day's operation; after manual scheduling, over 100 conflicts typically remain in the schedule. We are developing a prototype scheduling system that will assist human schedulers in this problem. Automated scheduling promises to reduce the number of conflicts, but will never remove all of them. Thus, the system will allow the human schedulers to examine alternative schedules so that they can better negotiate changes with the users.

Introduction

The prevalent view of scheduling is that of automatically constructing the optimal or best solution to a single problem instance on a single objective. While this view works, more or less, for commonly studied scheduling problems such as static job-shop, it breaks down in applications that operate continuously, receive requests asynchronously, trade-off between multiple usually competing objectives or require judgment about priorities.

Our application, scheduling access to remote satellite tracking stations (RSTS), possesses all four characteristics. The tracking stations operate 24 hours each day with emergency requests arriving at any time, and the demand outpaces the available resources. This necessitates tough decisions about which accesses are most important and whether some accesses can be partially accommodated.

In a recently initiated project, we are developing a prototype system to assist human schedulers in constructing schedules for the RSTS application. Several key issues must be resolved. Which scheduling algorithms and heuristics are best suited to this over-constrained problem? How can automated scheduling best function for dynamic scheduling over variable intervals? What objective function(s) match the requirements of the application and of the algorithms for automating the process? How a mixed-initiative scheduling system can improve the efficiency of the process and solutions? Although all of these issues need to be resolved ultimately, in this paper, we briefly touch on the first question and focus primarily on the last issue by describing our methodology and design for a mixed-initiative system for RSTS scheduling.

RSTS Application

In the Air Force Satellite Control Network, Remote Tracking Stations provide the satellite-to-ground interface necessary for satellite command and control operations. Remote Tracking Stations are located at nine (9) sites with 16 antennas. The stations are placed so as to maximize area coverage and multiple support capability. Time on the various Remote Tracking Stations is a scheduled resource.

Scheduling is carried out by one of two Resource Control Complexes (RCCs), located at the Schriever Air Force Base (AFB) and the Onizuka AFB. Approximately 520 resource requests must be scheduled for each day of operations at Schriever.

Because the resources are scarce, requests may include backup time and resource specifications and may be subject to later negotiation. Thus, a client's request consists of a primary desired time window on a particular Remote Tracking Station, the amount of time required, the priority of the request and optionally, alternative time windows and resources. (Alternatives may also be determined automatically based on the availability of duplicate or similar resources and capabilities.)

Attempts to automate these tasks have either failed, or have yielded dubious results. One of the primary barriers to successful automation has been the need to keep the human schedulers in the loop; to create a feasible schedule, the human scheduler often must negotiate changes with the clients. However, the human schedulers are currently still doing the entire task manually; only schedule entry, storage and display have been computerized. It can take a human one to two years to learn to schedule Satellite Remote Tracking Station requests at an expert level.

Currently, a one day schedule is built manually by human experts over a one week period. On any given day, the schedulers manage seven sliding windows representing the next seven days of activity. During the course of a week, the schedule is gradually determined

and frozen, except for high priority, emergency requests which may arrive at any time and must be included.

Mixed-Initiative Scheduling in RSTS

Shortening the scheduling window of RSTS from one week down to two or three days would mean managing fewer active scheduling windows at any point in time. In addition, shortening the scheduling time is also desirable because high priority requests may arrive late, causing disruption to the overall schedule.

Automation can help reduce the scheduling window, but by itself, automation is not the complete solution. If two requests require exactly the same resource at the same time, and there is no alternative way to satisfy both requests, then there is no feasible solution. Even when there are alternative ways to satisfy a request, contention for resources covering global "hot spots" may still preclude satisfying all requests. Human intervention is generally required to adjust the objectives of the schedule, to arbitrate what requests to abandon or to negotiate alternatives off-line. Emergency or extremely high priority requests for satellite time may also arrive in the last 24 hours before a schedule is to be deployed. This creates a need for fast schedule repair. What the automated part can do best is to anticipate problems and to provide the human scheduler with promising alternative schedules allowing him/her to dictate trade- offs in objectives and request satisfaction.

Thus, the system must support fast schedule proposal, schedule visualization, rapid consideration of alternatives and on-line editing for new requests. We are developing a system that has a suite of algorithms and heuristics for building schedules integrated with a visualization tool for examining and manipulating proposed schedules.

Algorithms for RSTS Scheduling

Scheduling of remote tracking stations is a constraint satisfaction problem in which no conflict-free solution exists and yet some solution must be generated. Often, in scheduling applications, an attempt is made to turn this kind of constraint satisfaction problem into a combinatorial optimization problem: a weighting is used to indicate which constraints are more important– and then an optimization routine is run to minimize the weighted set of violated constraints.

Search and combinatorial optimization methods are not enough to support a fielded system for resource scheduling. The combinatorial optimization approach has several problems in real world applications. First, it may be difficult or impossible to impose a meaningful weighting. Tuning a linear set of weights so that the weighted evaluation function reflects the same kinds of (possibly nonlinear) relative evaluations made by a human expert can be extremely difficult. Second, all the information needed to resolve a scheduling conflict may not be available to the automated scheduling system. In this case, it is impossible to build an evaluation function that captures the scheduling abilities of the human expert. For example, clients requesting a resource tend to make their request the highest allowable priority, so that priority information is less accurate and useful than one might expect. Given two requests of equal priority, an automated system may give both requests 70 percent of the requested time in order to minimize an objective function. This may not at all be the kind of compromise that is appropriate.

Our approach to developing the automated portion of this system is to start by studying well known scheduling approaches as applied to this problem. We think it likely that several scheduling algorithms will ultimately be incorporated to support different trade-offs between quality of solution and time required to obtain it. We have started our study by extending high performing heuristics and algorithms from job-shop scheduling. In particular, at this point in our study, we have implemented slack-based (Smith & Cheng 1993) and texture-based heuristics (Beck *et al.* 1997) coupled with several types of search algorithm, e.g., LDS (Harvey & Ginsberg 1995) and HBSS (Bresina 1996).

Although the study is preliminary (we still have to implement other candidate algorithms and study a wider range of problems), the results are promising. The min-slack heuristic with greedy search can construct a preliminary schedule in approximately 2 minutes. Given 500 requests with 120 initial conflicts, we have been able to resolve about 80 percent of the conflicts using this combination. We have found that texture-based heuristics can further reduce the number of conflicts but at the cost of significantly more computation.

Interface Aiding Human RSTS Schedulers

When scheduling the Remote Tracking Stations, the human schedulers may negotiate changes with the individuals making the resource request. Such negotiation is beyond the abilities of any automated scheduler. Humans may also accumulate information that is very hard to quantify (or which should not be quantified). For example, if clients X and Y are in conflict over a resource, X may outrank Y and so usually win the conflict despite the fact that the requests are of the same priority. Human schedulers can anticipate and informally exploit this type of information.

Thus, instead of trying to fully automate the scheduling process, an alternative approach is to build an interactive system that *aids* the human scheduler by doing what a computer does best: tediously generating and trying out alternatives and managing information about the schedule. In practice, it takes a human expert about five hours to build a preliminary schedule for satellite scheduling, which our current algorithms can build in a few minutes.

Once an initial schedule is constructed, a user should then be able to interactively select, reject or fix parts of the schedule. When a scheduling decision is made

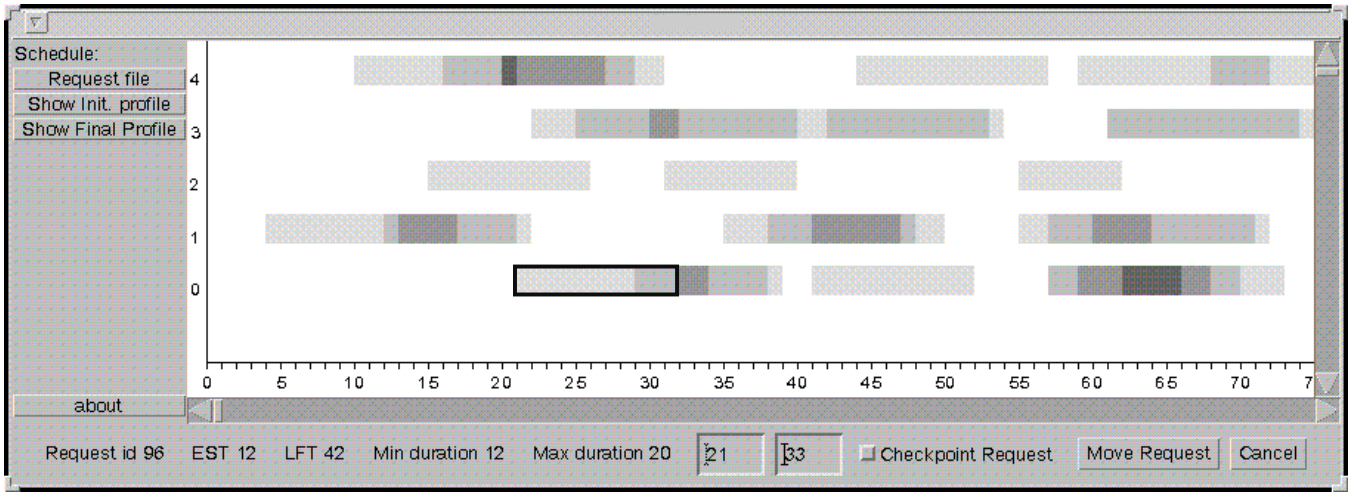


Figure 1: User can select a checkpoint in the schedule. A checkpoint indicates a key client request for which the user wishes to examine options when the system tries to schedule it.

that differs from the proposed schedule, the automatic scheduler can be re-run to find a new solution that optimizes around the fixed parts of the schedule. Since automated scheduling methods are to be used to generate initial schedules and to do rescheduling, very fast rescheduling methods may sometimes (but not always) be required.

Such an environment can do more than just provide an interface to the actual schedule and scheduling algorithms. It can also provide a data analysis tool to better understand the scheduling problems and statistic data about client requests and past schedules which may be useful in constructing new schedules.

It is common to find references to "interactive scheduling" in the literature, but there appears to be a limited number of ideas about how best to do this. There is general agreement that the user should interact with a schedule at an abstract, ideally graphical level that hides schedule implementation and optimization details to an appropriate degree (Smith & Lassila 1994) (Smith, Lassila, & Becker 1994) (Tate, Drabble, & Kirby 1994). Interactive Gantt Charts are a popular way to achieve this goal (Sadeh 1994). Generally, the user can make a change to the schedule; in response, the scheduler may be called upon to propagate the effects of the change and perhaps to optimize in response to the change (Yen & Pinedo 1994) (Deale *et al.* 1994). These ideas are incorporated in our prototype system.

Example of Scheduling Interface

To demonstrate the basics of our proposed system, we will step through an example showing how it will look to a user. In the example terminology, the "Clients" will be those requesting time on a Remote Tracking Station; the "Users" are the schedulers who assign satellite resources to clients. The figures that follow are mock-ups of proposed visualizations for the user.

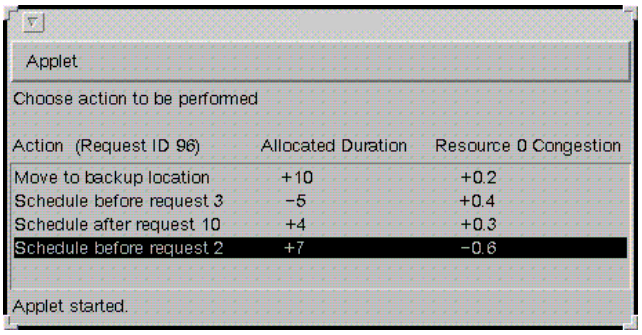


Figure 2: Display of suggested actions for scheduling a particular request. After a checkpoint has caused a stop, the user can determine what should be done with the request.

Figure 1 illustrates a hypothetical situation where the initial set of requests has been displayed, with darker levels of gray-scale indicating greater conflict (this would be in color online). The user can examine the set and then select a specific request to be "checkpointed". The dark rectangle outline highlights a specific client request. If this particular request appears to represent a significant bottleneck, the request can be checkpointed to allow the user to look at the partial schedule as it exists at the point when the system is trying to schedule this particular request.

Once the system has scheduled up until the checkpointed request, the user has some latitude in how to handle it. For example, Figure 2 illustrates suggested scheduling alternatives for a specific client request which has been highlighted and checkpointed as illustrated in the previous figure 1. The display shows several scheduling alternatives for this client request and data on how this impacts the schedule (i.e., the

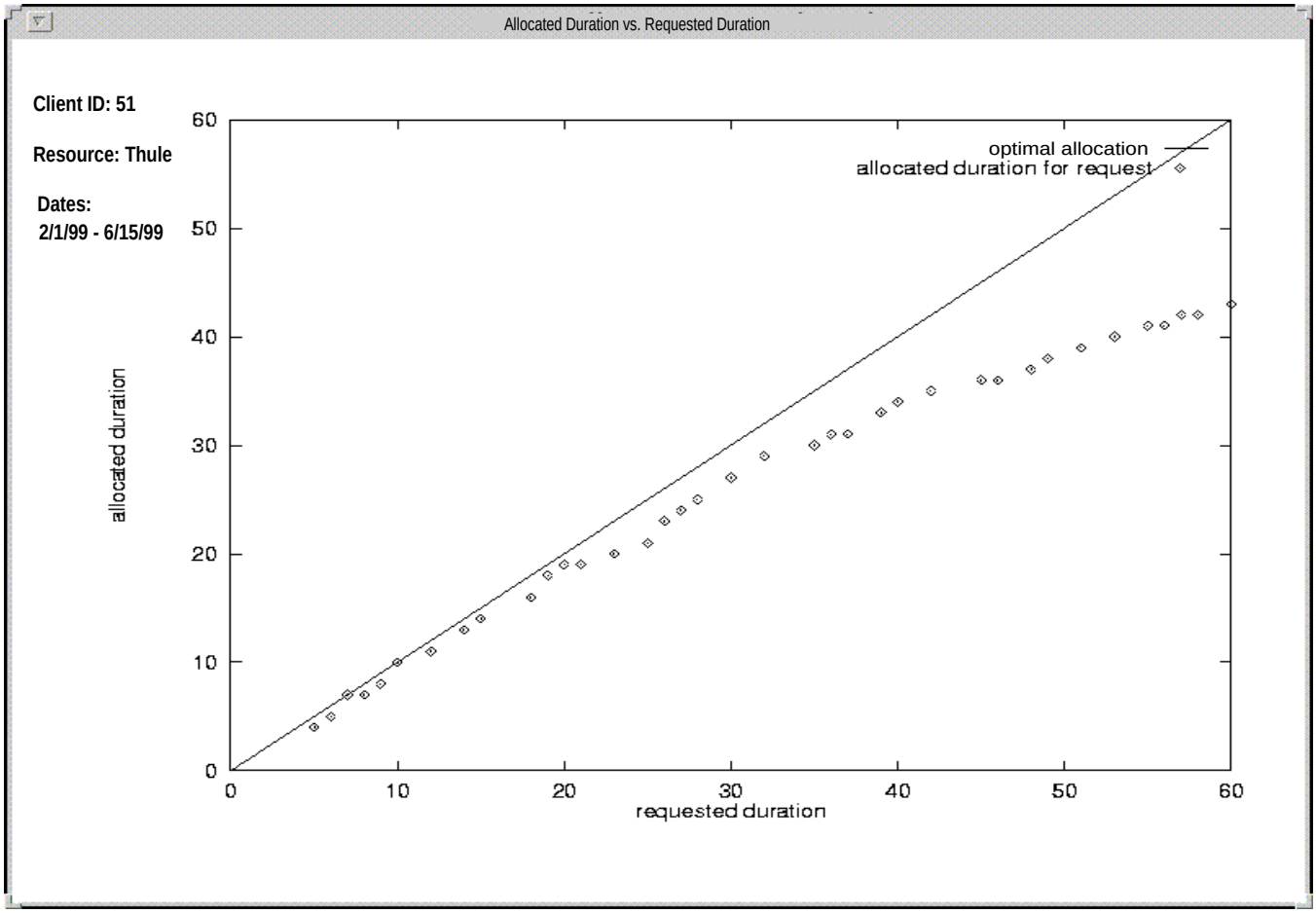


Figure 3: Graph for a single client mapping hypothetical previous resource requests by size on the x-axis against the actual time allocated on the y-axis. A scheduler could infer from this data that this client is more flexible about receiving less than the request time on a resource when the original request is larger.

change in evaluation expected for each action).

To expedite the decision, the user could decide to look at historical information about the client making the request. Figure 3 illustrates a hypothetical situation where historical data is available about a client requesting a resource. The graph shows the actual duration allocated as a function of the original resource request. We know from human schedulers that some clients are more flexible than others; flexibility may also be a function of the size of the resource requested (as shown in this graph) or a function of the type of resource requested (which is not shown here).

Finally, figure 4 illustrates the set of recent emergency requests. One might define emergency requests to be those client requests which arrive in the last 24 hours before a schedule becomes active, or any requests that arrive after the schedule is active. In this case, the idea is that such requests are not random. A possible scheduling strategy is to avoid scheduling tasks on resources and specific time slots that have a recent history of high emergency incidence. Human schedulers currently use a similar strategy for this application.

Project Status

At present, we have implemented a problem generator and a web based interface for viewing and manipulating schedules. The problem generator was necessary due to the sensitive nature of the application. We can automatically construct problems that are typical of the types received by the Air Force without compromising security. In addition, the generator allows us to exert experimental control during testing and evaluation. The generator (Howe *et al.* 2000) can be parameterized with the number of requests, disparity in their time window and duration sizes, and amount of clustering (contention at same times on the same resource). Based on the results of our evaluations, the best set of algorithms and heuristics will be included into the final system, allowing for particular algorithm/heuristic combinations to be selected. We recognize that users may wish to favor speed over optimality and thus iterative repair over optimization algorithms.

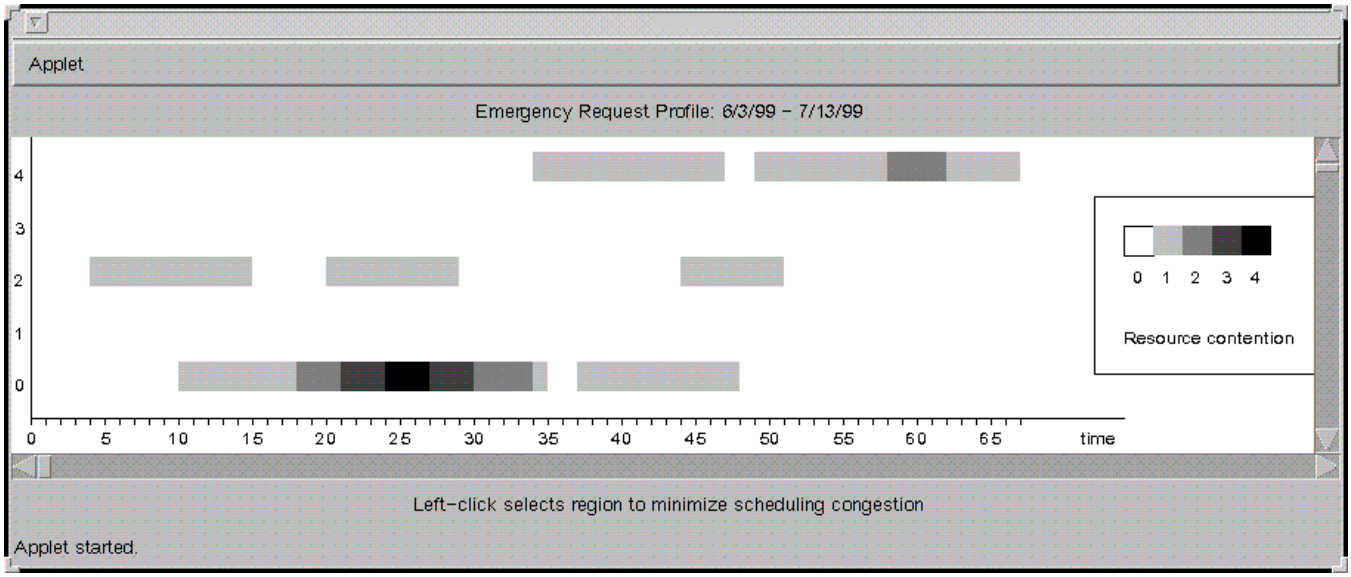


Figure 4: Profile of emergency requests submitted during the previous 40 days

Because of the importance of the interface for these types of systems and the need to solicit feedback from the users and experts for the applications, the interface has been implemented in Java and will be accessible via the World Wide Web. This will allow Air Force users all over the world to access our system and demo the interactive scheduling environment. At present, the system generates initial schedules using min-slack/greedy search and allows the user to view, add, delete or move requests within the proposed schedule.

Acknowledgments We would like to thank Alex Kilpatrick of AFOSR and Brian Bayless of Falcon Air Force Base for providing us with the information on the application. This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-97-1-0271. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

Beck, J.; Davenport, A.; Sitarski, E.; and Fox, M. 1997. Texture-based heuristics for scheduling revisited. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-97)*.

Bresina, J. L. 1996. Heuristic-biased stochastic sampling. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*.

Deale, M.; Yvanovich, M.; Schnitzuius, D.; Kautz, D.; Carpenter, M.; Zweben, M.; Davis, G.; and Daun, B. 1994. The space shuttle ground processing scheduling system. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling*. Morgan Kaufmann. 423–449.

Harvey, W. D., and Ginsberg, M. L. 1995. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*.

Howe, A. E.; Whitley, L. D.; Watson, J.-P.; and Barbulescu, L. 2000. A study of air force satellite scheduling. In *World Automation Congress 2000*. to appear.

Sadeh, N. 1994. Micro-opportunistic scheduling: The micro-boss factory scheduler. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling*. Morgan Kaufmann. 99–135.

Smith, S., and Cheng, C. 1993. Slack-based heuristics for constraint satisfaction problems. In *Proceedings AAAI-93*, 139–144.

Smith, S., and Lassila, O. 1994. Toward the development of flexible mixed-initiative scheduling tools. In *ARPA-Rome Laboratory Planning Initiative Workshop*.

Smith, S.; Lassila, O.; and Becker, M. 1994. Configurable, mixed-initiative systems for planning and scheduling. In Tate, A., ed., *Advanced Planning Technology: Technological Achievements of the ARPA/Rome Laboratory Planning Initiative*. 235–241.

Tate, A.; Drabble, B.; and Kirby, R. 1994. O-plan2: An open architecture for command, planning and control. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling*. Morgan Kaufmann. 213–239.

Yen, B., and Pinedo, M. 1994. On the design and development of scheduling systems. In *4th International Conf. of Computer Integrated Manufacturing and Automation Technology*.

Author Index

Aljabri, Abdullah, 169
Arno, Roger, 151
Balzano, Vicki, 205
Barbulescu, Laura, 247
Barrett, Anthony, 170
Berry, Pauline, 114
Bonasso, Pete, 25
Boddy, Mark, 42
Bornschlegl, Eric, 36, 122
Bowman, Alice, 17
Boyce, Leslye, 205
Bresina, John, 241
Brooks, Robert, 66
Cammarata, Glenn, 205
Cesta, Amedeo, 223
Chien, Steve, 11, 67, 92, 178, 217, 229, 235
Cooper, Brian, 229
Davis, Elizabeth, 193
Delhaise, Fabienne, 100
DesJardins, Marie, 187
Doxsey, Rodger, 205
Drabble, Brian, 114
Engelhardt, Barbara, 11, 178, 184, 229
Estlin, Tara, 217, 229, 235
Feather, Martin, 136
Fisher, Forest, 235
Frank, Jeremy, 76
Freed, Michael, 211
Freuder, Eugene, 199
Galal, Ken, 58
Gal-Edd, Jonathan, 176
Gat, Erann, 106
Giunchiglia, Enrico, 223
Glück, Peter, 127
Golden, Keith, 28
Guettier, Christophe, 36
Hallock, Lou, 205
Howe, Adele, 247
Isaacs, John, 205
Jaap, John, 193
Joesting, David, 52
Johnson, Patricia, 34
Jónsson, Ari, 128
Kalinowski, Keith, 205
Knight, Russell, 67, 92
Koenig, Sven, 144
Kramer, Laurence, 2
Krasner, Sanford, 82
Kutina, Ray, 205
Larsson, Gus, 52
LeBlanc, Troy, 134
Legg, Jim, 205
Littman, Michael, 60
Majercik, Stephen, 60
Merri, Mario, 73, 100
Mishkin, Andrew, 229
Morris, Paul, 128
Mouaddib, Abdel-Illah, 163
Muscettola, Nicola, 128, 136
Mutz, Darren, 178, 184, 217, 235
Nadel, Miriam, / 98
Nakasuka, Shinichi, 155
Niézette, Marc, 19
Norvig, Peter, 50
Palmer, Ian, 100
Patel, Keyur, 113
Peccia, Nestor, 100
Pemberton, Joseph, 84
Poncet, Jean-Clair, 36
Rabideau, Gregg, 11, 67, 92, 217, 229
Rajan, Kanna, 128
Rasmussen, Robert, 120
Rehm, Ken, 205
Sherwood, Rob, 67, 229
Shimohara, Katsunori, 155
Simmons, Reid, 161
Smith, Ben, 45, 128, 136, 184
Smith, David, 8
Stebbins, George, 45
Stechert, Andre, 67
Takadama, Keiki, 155
Tomkins, Steve, 205
Tong, Michael, 142
Traverso, Paolo, 223
Verfaillie, Gérard, 92, 122
Vidal, Thierry, 107
Wallace, Richard, 199
Washington, Richard, 241
Watson, Jean-Paul, 247
Welter, Gary, 205
Whitley, Darrell, 247
Wilkins, David, 187
Zilberstein, Shlomo, 163

Notes

Notes

Notes