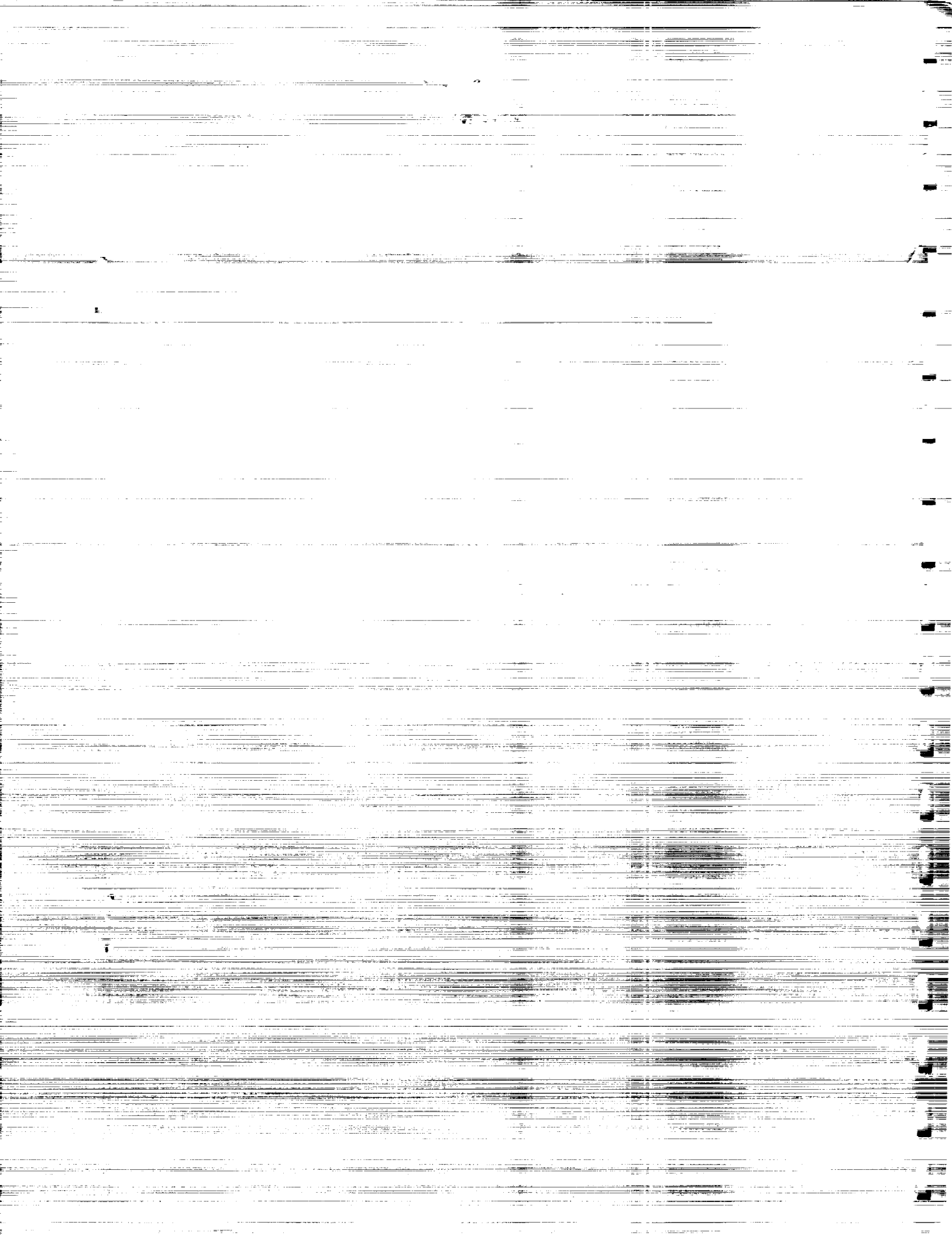
Working Notes from The 1992 AAAI Spring Symposium on Practical Approaches to Scheduling and Planning

Mark Drummond
Sterling Federal Systems
Mark Fox
University of Toronto
Austin Tate
University of Edinburgh
Monte Zweben
NASA Ames Research Center

NASA Ames Research Center
Artificial Intelligence Research Branch

Technical Report FIA-92-17

May, 1992



1992 Spring Symposium Series

Practical Approaches to Scheduling and Planning

Working Notes
(Distribution limited to symposium attendees)

March 25 – 27, 1992
Stanford University

Sponsored by the
American Association for Artificial Intelligence

PRIMARY
137226

Practical Approaches to Scheduling and Planning

Government and industry require practical approaches to a diverse set of complex scheduling and planning problems. While scheduling has been studied in isolation for many years, recent advances in artificial intelligence, control theory, and operations research indicate a renewed interest in this area. In addition, the scheduling problem is being defined more generally, and work is beginning to consider the closed-loop use of scheduling systems in operational contexts. This symposium will serve to bring together theorists and practitioners from diverse backgrounds, with the aim of disseminating recent results and fostering the development of a cross-discipline understanding.

The symposium will focus on issues involved in the construction and deployment of practical scheduling systems that can deal with resource and time limitations. To qualify as "practical", a system must be implemented and tested to some degree on non-trivial problems (ideally, on real-world problems). However, a system need not be fully deployed to qualify. Systems that schedule actions in terms of metric time constraints typically represent and reason about an external numeric clock or calendar, and can be contrasted with those systems that represent time purely symbolically.

Issues to be discussed at the symposium include, but are not strictly limited to, the following.

- Integrating planning and scheduling.
- Integrating symbolic goals and numerical utilities.
- Managing uncertainty.
- Incremental rescheduling.
- Managing limited computation time.
- Anytime scheduling and planning algorithms, systems.
- Dependency analysis and schedule reuse.
- Management of schedule and plan execution.
- Incorporation of techniques from discrete event control.

- Incorporation of techniques from operations research.
- Learning.
- Measures of schedule and plan quality.
- Search techniques.
- Methodology.
- Applications.

Program Committee

Mark Drummond
NASA Ames Research Center
MS: 269-2
Moffett Field, CA 94035 U.S.A.
Email: med@ptolemy.arc.nasa.gov

Mark Fox
Department of Industrial Engineering
University of Toronto
4 Taddle Creek Road
Toronto, Ontario M5S 1A4 Canada
Email: msf@phoenix.rose.utoronto.ca

Austin Tate
AI Applications Institute
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN U.K.
Email: A.Tate%ed@nsfnet-relay.ac.uk

Monte Zweben
NASA Ames Research Center
MS: 269-2
Moffett Field, CA 94035 U.S.A.
Email: zweben@ptolemy.arc.nasa.gov

Schedule
Practical Approaches to Scheduling and Planning
AAAI 1992 Spring Symposium Series
Stanford University
March 25 - 27 1992

Wednesday, March 25

9:00 - 10:30 am

Presentations: Applications
Spike: AI Scheduling for Hubble Space Telescope After 18 Months of Orbital Operations
Mark D. Johnston

Temporal Planning for Transportation Planning and Scheduling
Robert E. Frederking and Nicola Muscettola

A Simulated Annealing Approach to Schedule Optimization for the SES Facility
Mary Beth McMahon and Jack Dean

10:30 - 11:00 pm

Break

11:00 - 12:30 pm

Presentations: Methods
Decomposability and Scalability in Space-Based Observatory Scheduling
Nicola Muscettola and Stephen F. Smith

Managing Disjunction for Practical Temporal Reasoning
Mark Boddy, Bob Schrag, and Jim Carciofini

TOSCA: The Open Scheduling Architecture
Howard Beck

12:30 - 2:00 pm

Lunch

2:00 - 3:30 pm

Presentations: Applications
Scheduling of an Aircraft Fleet
Massimo Paltrinieri, Alberto Momigliano, and Franco Torquati

Adaptive Planning For Applications With Dynamic Objectives
Khosrow Hadavi, Wen-Ling Hsu, and Michael Pinedo

Global Planning of Several Plants
Sylvie Bescos

Space Shuttle Ground Processing
Eric Clanton

3:30 - 4:00 pm

Break

4:00 - 5:30 pm

Panel: Robustness and Schedule Quality
Donald Rosenthal
Monte Zweben
Austin Tate
Mark Drummond

6:00 - 7:00 pm

Reception, Oak Lounge, Tressider Union

Thursday, March 26

9:00 - 10:30 am

Presentations: Methods
The MICRO-BOSS Scheduling System: Current Status and Future Efforts
Norman M. Sadeh

Iterative Refinement Scheduling
Eric Biefeld

CABINS: Case-Based Interactive Scheduler
Kazuo Miyashita and Katia Sycara

10:30 - 11:00 pm

Break

11:00 - 12:30 pm

Panel: Learning and Scheduling
Steven Minton
Monte Zweben
Steve Chien
Stephen Smith

12:30 - 2:00 pm

Lunch

2:00 - 3:30 pm

Presentations: Applications
Scheduling Lessons Learned from the Autonomous Power System
Mark J. Ringer

Planning for the Semiconductor Manufacturer of the Future
Hugh E. Fargher and Richard A. Smith

Uncertainty Management by Relaxation of Conflicting Constraints in Production Scheduling
Jürgen Dorn, Wolfgang Slany, and Christian Stary

3:30 - 4:00 pm

Break

4:00 - 5:30 pm

Panel: Relevance of AI Planning Systems
Nicola Muscettola
Roberto Desimone
Austin Tate
Mark Drummond

7:30 - 10:00 pm

Public Forum, Kresge Auditorium

Friday, March 27

9:00 - 10:30 am

Presentations: Methods
Experiments with a Decision-Theoretic Scheduler
Othar Hansson, Gerhard Holt, and Andrew Mayer

Generating Effective Project Scheduling Heuristics by Abstraction and Reconstitution
Bhaskar Janakiraman and Armand Prieditis

Real-time Scheduling Using Minimin Search
Prasad Tadepalli and Varad Joshi

10:30 - 11:00 pm

Break

11:00 - 12:30 pm

Panel: Rescheduling
Mark Johnston
Stephen Smith
Katia Sycara
Mark Fox

Table of Contents
Practical Approaches to Scheduling and Planning
AAAI 1992 Spring Symposium Series
Stanford University
March 25 - 27 1992

Spike: AI Scheduling for Hubble Space Telescope After 18 Months of Orbital Operations

Mark D. Johnston
Space Telescope Science Institute
3700 San Martin Drive,
Baltimore, MD 21218 USA
johnston@stsci.edu

Abstract

This paper is a progress report on the Spike scheduling system, developed by the Space Telescope Science Institute for long-term scheduling of Hubble Space Telescope observations. Spike is an activity-based scheduler which exploits AI techniques for constraint representation and for scheduling search. The system has been in operational use since shortly after HST launch in April 1990. Spike has been adopted for several other satellite scheduling problems: of particular interest has been the demonstration that the Spike framework is sufficiently flexible to handle both long-term and short-term scheduling, on timescales of years down to minutes or less. We describe the recent progress made in scheduling search techniques, the lessons learned from early HST operations, and the application of Spike to other problem domains. We also describe plans for the future evolution of the system.

1 Introduction

Efficient utilization of expensive space-based observatories is an important goal for NASA and the astronomical community: the cost of facilities like Hubble Space Telescope (HST) is enormous, and the available observing time is much less than the demand from astronomers around the world. The Spike scheduling system was developed by the Space Telescope Science Institute starting in 1987 to help with this problem. The aim of Spike is to allocate observations to timescales of days to a week, observing all scheduling constraints, and maximizing preferences that help ensure that observations are made at optimal times. Spike has been in use operationally for HST since shortly after the observatory was launched in April 1990.

Although developed specifically for HST scheduling, Spike was carefully designed to provide a general framework for similar (activity-based) scheduling problems. In particular, the tasks to be scheduled are defined in the system in general terms, and no assumptions about the scheduling timescale were built in. The mechanisms for describing, combining, and propagating temporal and other constraints and preferences were designed to be general. The success of this approach has been demonstrated by the application of Spike to the scheduling of other satellite observatories: changes to the system are required only in the specific constraints that apply, and not in the framework itself.

In the following we first provide a brief description of the HST scheduling problem and of the Spike scheduling framework. We then discuss some of the experience gained with the system since the start of HST flight operations. This is followed by a description of the changes required to adapt Spike to other satellite scheduling problems. We conclude with some comments on the implementation of Spike, and on our plans for future work.

2 Overview of HST Scheduling

HST scheduling is a large problem: some 10,000 to 30,000 observations per year must be scheduled, each subject to a large number of operational and scientific constraints. Most of the operational constraints arise from the low earth orbital environment of the telescope. With an orbital period of about 96 minutes, potential targets are only visible for a portion of each orbit before they are occulted by the earth. There are constraints due to guide star availability, avoiding the earth's radiation belts, and stray light from the sun, moon, or bright earth. There are also constraints arising from thermal and power considerations, which tend to restrict the allowable attitude of the satellite at different times during the year. Scientific constraints are specified by astronomers when they define the exposures to accomplish their scientific goals. These frequently take the form of minimum exposure times, temporal relationships among exposures (before, after, grouped within some time span, separated by some minimum and/or maximum interval, etc.). Astronomers may also constrain the state of the telescope in other ways, e.g. by requiring exposures when HST is in earth shadow (to exclude scattered earthlight), by specifying the orientation of the telescope, or by configuring one of the six instruments in a particular mode. A recent change to the HST ground systems now permits the scheduling of two instruments for simultaneous operation: this is expected to significantly increase the amount of useful data taken by the telescope.

Because of the design of the telescope and ground system, nearly all HST activities must be scheduled in detail in advance. The detailed schedule specifies what commands will be executed by the onboard computers, and when communications contacts will be available for uplinking commands and downlinking data. Real-time interaction by observers is limited essentially to small pointing corrections to place targets accurately into the proper instrument aperture.

Scheduling HST has been divided into two processes: the first is long-term scheduling, which allocates observations to week-long time segments over a scheduling period of a year or more in duration. This is the responsibility of the Spike system. Individual weeks are then scheduled in detail by the Science Planning and Scheduling System (SPSS),

which orders observations within the week and generates a detailed command sequence for the HST control center at NASA Goddard Space Flight Center. Further details on HST scheduling may be found in [1,2].

3 Spike and HST Long-Term Scheduling

HST observing programs are received at STScI in machine-readable form over national and international computer networks. They are then translated by an expert system called Transformation [3] into a form suitable for scheduling. The Transformation system collects exposures into "scheduling units" which are collections of exposures to be executed contiguously. Transformation makes use of the Spike temporal constraint mechanism to collect and propagate temporal constraints: these are made path-consistent and saved in files along with the scheduling unit definitions. Spike takes the saved scheduling units and derives scheduling constraints and preferences for them, based on operational and scientific factors such as those described above. Spike then determines an allocation of scheduling units to weeks which satisfies all hard constraints and as many soft constraints as possible. Constraints from different sources are combined using a weight-of-evidence mechanism generalized to cover a continuous time domain, as described in detail elsewhere [4]. The result is a set of "suitability functions" which indicates goodness over time for each scheduling unit, and also indicates times when a scheduling unit cannot be scheduled due to violations of strict constraints. Most of the HST-specific scheduling details go into the definition of the suitability functions, which, for long-term scheduling, are defined at a high level of abstraction and relatively coarse time granularity. More details about Spike constraint representation and manipulation may be found in [5].

Spike treats schedule construction as a constrained optimization problem and uses a heuristic repair-based scheduling search technique. An initial guess schedule is constructed, which may have temporal or other constraint violations as well as resource overloads (in fact, given that HST observing time is intentionally oversubscribed by about 30%, it is known ahead of time that there is no feasible schedule that can accommodate all the requested observations). Repair heuristics are applied to the initial guess schedule until a preestablished level of effort has been expended. At that point observations are removed to eliminate remaining constraint violations, until a feasible schedule remains. There are several important measures of schedule quality employed, including the number of observations on the schedule, the total observing time scheduled, and the summed degree of preference of the scheduled observations. The heuristic repair method is fast, and typically many runs are made and the best schedule is adopted as a baseline. The Spike algorithm has desirable "anytime" characteristics: at any point in the processing after the initial guess has been constructed, a feasible schedule can be produced simply by removing any remaining activities with constraint violations, as described further below.

The repair heuristics used by Spike are based on a very successful neural network architecture developed for Spike [6,7] and later refined into a simple symbolic form [8] which has made the neural network obsolete. The Spike repair heuristics make highly effective use of conflict count information, i.e. the number of constraint violations on scheduled activities or on potential schedule times. Min-conflicts time selection is one such repair heuristic, in which activities are moved to times when the number of conflicts is minimized. Both theoretical analysis and numerical experiments have shown that min-conflicts can be very effective in repairing good initial guesses [9]. We have found that further improvement comes from the use of a max-conflicts activity selection heuristic, which selects activities for repair which have the largest number of conflicts on their current assigned time. Spike permits different constraints to have different conflict weights, which can be used to cause the repair of the most important constraints first; in practice, however, all constraints have so far been given the same weight. Both hillclimbing and backtracking repair procedures have been tried, but hillclimbing has been shown to be the most cost-effective on problems attempted to date.

The choice of a good initial guess is important for repair-based methods, and to this end we have conducted experiments on different combinations of variable and value selection heuristics to find the "best" combination. Over a thousand combinations of heuristics were tried by making multiple runs on sample scheduling problems. The adopted initial guess heuristic selects most-constrained activities to assign first, where the number of min-conflicts times is used as the measure of degree of constraint. Min-conflict times are assigned, with ties broken by maximum preference as derived from suitability functions.

Spike currently uses a rather simple technique to remove conflicting activities from an oversubscribed schedule: activities to be removed are selected based on lower priority, higher numbers of conflicts, and lower preference time assignments. If there remain gaps when all conflicting activities have been deleted, then a simple best-first pass through the unscheduled activities is used to fill them. This final phase of "schedule deconflicting" has been little studied and is an area which could benefit from further effort.

Spike provides support for rescheduling in several ways. Two worth mentioning in particular are task locking and conflict-cause analysis. Tasks or sets of tasks can be locked in place on the schedule, and will thereafter not be considered during search or repair (unless of course the user unlocks them). These tasks represent fixed points on the schedule. Conflict-cause analysis permits the user to force a task onto the schedule, then display what constraints are violated and by which other tasks. The conflicting tasks can be unassigned if desired, either individually or as a group, and returned to the pool of unscheduled tasks. This helps with the most common rescheduling case, where a specific activity must be placed on the schedule, thereby disrupting at least some tasks which are already scheduled. A limited study of minimal-change rescheduling has been conducted [10], but much more work remains to be done in this area.

Hillclimbing repair methods like the one used in Spike have much in common with simulated annealing techniques such as described by Zweben et al.[11]. One of the open research issues is which technique has an advantage on which types of problems.

4 The Experience of HST Operations

Shortly after HST was launched it was discovered that the telescope main mirror had been figured incorrectly, resulting in lower resolution than anticipated. This has not only limited the scientific usefulness of HST (although it still remains far superior to any ground-based telescope), it has also greatly disrupted the scheduling process. Observing plans made years in advance of launch have had to be revised, leading to a shortage of ready-to-schedule observing programs and thus reducing the efficiency with which schedules could be generated. This problem still affects ongoing operations, and as a result Spike has only once been used to generate a true long-term schedule. Instead, Spike is used routinely to identify observations to place in the schedule approximately two months into the future. As the characteristics of the telescope and instruments have become better understood, the pool of observing programs has been growing: the second round of open proposal selection will be completed in December 1991, and we anticipate that by the Spring of 1992 a sufficient pool will exist to permit long-range planning as originally expected. NASA is now planning a servicing mission to correct the HST optics in early 1994.

The most significant lesson learned since launch, however, is the impact of high levels of change on the planning and scheduling systems. Instead of the anticipated level of about 10% of proposals changing, the actual rate of change has been closer to 100%. While some of this change is clearly attributable to the discovery of HST's spherical aberration, many other factors have contributed as well: nearly every instrument on the telescope has demonstrated unexpected behavior in one form or another, and each has led to revisions in observing plans to compensate. The net effect is that change is the norm, not the exception, to the extent that stress has been high on the software systems and on the people who operate them. The problem stems from the fact that an observing program may consist of many hundreds of exposures, which can all be at different stages of the scheduling pipeline. If an observing program is changed, users must back up to the beginning of the process for that program, thus work done on the previous version is potentially wasted. Alternatively, a new observing program can be created to describe the changed portions of the original one, but then keeping track of active and obsolete portions of the original is required.

If there is any recommendation to be made to developers of future systems like those for HST, it is to build in the expectation of change from the outset [12]. Even though the initial cost will be higher, the operational costs will be significantly lower.

Spike and the other HST ground systems have been exercised several times on "targets of opportunity" --- programs to be scheduled and executed on an crash basis. Turnaround has been as short as a few days, which is well within the pre-launch expectations. One such target of opportunity program took the pictures of the dramatic storm on Saturn in December 1990, which were subsequently made into a time-lapse movie.

5 Hierarchical and Short-Term Scheduling

Spike has been adopted for scheduling three future astronomical satellite missions:

- the Extreme Ultraviolet Explorer (EUVE), an ultraviolet telescope built and operated by UC Berkeley and Goddard Space Flight Center,
- ASTRO-D, a joint US-Japan X-ray telescope, and
- XTE, the X-ray timing Explorer (MIT/GSFC) to study time-variability of X-ray sources.

The adaptation of Spike for these problems has led to the successful demonstration of the flexibility of the Spike scheduling framework. As indicated above, Spike was designed so that new tasks and constraints can be defined without changing the basic framework. For ASTRO-D and XTE, Spike is operated in a hierarchical manner, with long-term scheduling first allocating observations to weeks much as they are for the HST problem (and with similar types of long-term constraints and preferences). Then each week is scheduled in detail, subject to the detailed minute-by-minute constraints of low earth orbit operation. The major changes required to implement short-term scheduling were:

- a new type of task that can have variable duration depending on when it is scheduled, and which can be interrupted and resumed when targets are occulted by the earth or the satellite is in the radiation belts
- new classes of short-term scheduling constraints which more precisely model target occultation, star tracker occultation, ground station passes, entry into high radiation regions, maneuver and setup times between targets, etc.
- an interface between different hierarchical levels, by which a long-term schedule constrains times for short-term scheduling and conversely
- a post-processor which examines short-term schedules for opportunities to extend task durations and thus utilize any remaining small gaps in the schedule to increase efficiency

All of the general constraint combination and propagation mechanisms, and the schedule search techniques, apply directly to both long-term and short-term scheduling. Figure 1 illustrates the application of Spike to short-term scheduling for a sample of X-ray targets such as might be observed by ASTRO-D or XTE. Note that several observations are broken to fit around occultations and so are taken in multiple segments.

Most of the effort required to apply Spike to the new problems was limited to the specific domain modelling necessary, which typically involves computation related to the geometry of the satellite, sun, target, and earth. These problems can be expected to differ from one satellite to another, and it is not surprising that different models are required. Some of the modelling includes state constraints, although Spike does not perform explicit planning (see, e.g. [13]).

EUVE is unusual in that it makes long (2-3 day) observations, in contrast to HST, XTE, and ASTRO-D which typically make numerous short (15-40 minute) observations. As a consequence, EUVE is schedulable over year-long intervals without breaking the schedule into hierarchical levels. One of the more interesting results from a comparison of search algorithms for scheduling EUVE was that the Spike repair-based methods gained an extra *20 days* of observing

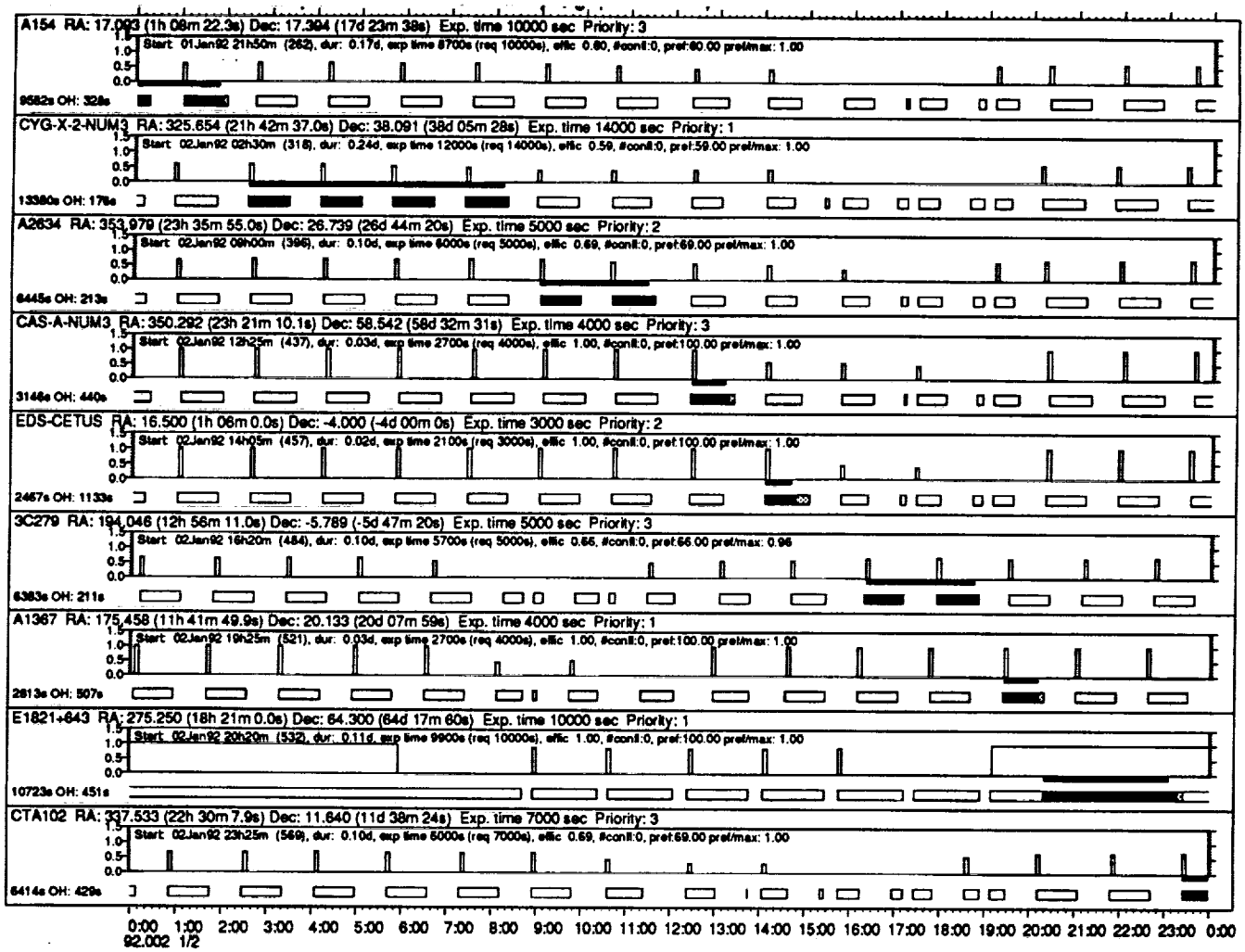


Figure 1: An example of Spike output on short-term scheduling of astronomical observations. Shown is a 24-hour portion of a 7-day schedule. The start-time suitability for each exposure is plotted as the upper graph, with interruptions due to target blockage by the earth and by satellite passage through high-radiation regions. The available exposure intervals are shown below as open bars, which are filled in to indicate the actual scheduled times. Some of the observations can be fit within one orbit; others must be interrupted and thus span several orbits.

time in a year, when compared to the best incremental scheduling approach.

6 Spike Implementation

The implementation of Spike started in early 1987 and was initially based on Texas Instruments Explorers as the hardware and software environment. The Spike graphical user interface was implemented in KEE CommonWindows (Intellicorps, Inc.), but the remainder of the system (about 40,000 lines of code) used only Common Lisp and the Flavors object system. At HST launch, STScI had a complement of 8 TI Explorers and microExplorers used for Spike operation, development and testing.

Since the initial development of Spike began there has been a great deal of evolution in Lisp hardware and software. A significant amount of effort has gone into modifying the system to keep current with these changes. In late 1991 we are in the process of moving from Explorers to Sun SparcStation IIs as the primary operations and development workstation. All of the Flavors code has been automatically converted to the Common Lisp Object System. The Lisp used on the SparcStation is Allegro Common Lisp from Franz Inc. Allegro CL supports a version of CommonWindows based on X-windows, and so the user interface continues operate on Unix platforms as it did on the Explorers. We are presently investigating the use of alternative window systems, and have prototyped the use of CLX, CLIM, and Motif for the user interface (the latter is based on the publicly available GINA/CLM). We expect to see a complete redesign of the user interface in the next year. Spike can also generate high-resolution Postscript versions of schedules and constraints; one example of this is shown in Figure 1.

Updating Spike for new Lisp language features has not been difficult. There are, however, plans to remove some features that were developed for Spike which have since become part of the language (such as a logical filename mechanism). At present there are no plans to convert any of the system to C or C++.

7 Future Directions

Several significant enhancements to Spike are planned over the next year. One of these, a rewrite of the graphical user interface, has already been mentioned above. Another enhancement deals with tracking the status of HST observing programs and exposures. All scheduled programs pass from the proposal entry system through Spike, while feedback on scheduling and execution status is received by Spike both from SPSS and from the HST data analysis pipeline. This provides information to Spike users which forms the basis for rescheduling decisions. We plan to integrate this data into a relational database, along with additional information from the HST optical disk data archive, which will provide a central source of information on the status of all HST observations.

We are also planning several systematic studies of the Spike scheduling search heuristics to see what further improvements can be made, either in performance or in quality of schedule. These will include the initial guess, repair, and deconflict strategies. We also plan to investigate whether the use of short-term scheduling on the HST observations can improve the quality of the long-term schedule sent to SPSS. There are, however, no plans to have Spike do the final short-term scheduling for HST, due to the extreme cost of integration with the existing telescope and instrument commanding software which generates the command sequences for the spacecraft.

8 Conclusions

The Spike system has performed as planned in the first 18 months of HST operations. The success of Spike helps demonstrate the utility of AI technology in NASA flight operations projects. The flexibility of Spike has been demonstrated by adapting it for several other missions, and by integrating long-term and short-term scheduling at different hierarchical levels of abstraction in the same constraint representation and scheduling search framework.

Acknowledgments: The author wishes to thank Dr. Glenn Miller and the present and former members of the Spike development team (Jeff Sponsler, Shon Vick, Tony Krueger, Dr. Mark Giuliano, and Dr. Michael Lucks) for their efforts which have led to the successful deployment of Spike. The patience and support of the Spike user group, led by Dr. Larry Petro, chief of the STScI Science Planning Branch, has been much appreciated. Stimulating discussions with Dr. Steve Minton, Monte Zweben, Don Rosenthal, and Hans-Martin Adorf are gratefully acknowledged. The EUVE project at UC Berkeley helped with the initial Unix port, and the ASTRO-D and XTE staff at MIT have continued to help push the system in new and fruitful directions. The Space Telescope Science Institute is operated by the Association of Universities for Research in Astronomy for NASA.

References

[1] Miller, G., Rosenthal, D., Cohen, W., and Johnston, M.D. 1987: "Expert System Tools for Hubble Space Telescope Observation Scheduling," in *Proc. 1987 Goddard Conf. on Space Applications of Artificial Intelligence*; reprinted in *Telematics and Informatics* 4, p. 301 (1987).

[2] Miller, G., Johnston, M.D., Vick, S., Sponsler, J., and Lindenmayer, K. 1988: "Knowledge Based Tools for Hubble Space Telescope Planning and Scheduling: Constraints and Strategies", in *Proc. 1988 Goddard Conf. on Space Applications of Artificial Intelligence*; reprinted in *Telematics and Informatics* 5, p. 197 (1988)

[3] Gerb, A. 1991: "Transformation Reborn: A New Generation Expert System for Planning HST Operations", in *Proc. 1991 Goddard Conf. on Space Applications of Artificial Intelligence*, ed. J.L. Rash, NASA Conf. Publ. 3110 (Greenbelt: NASA), pp. 45-58; to be reprinted in the Dec 1991 issue of *Telematics and Informatics*.

[4] Johnston, M.D. 1989: "Reasoning with Scheduling Constraints and Preferences," Spike Tech. Report 89-2, Jan. 1989.

[5] Johnston, M.D. 1990: "SPIKE: AI Scheduling for NASA's Hubble Space Telescope", M.D. Johnston, in *Proc. Sixth IEEE Conf. on Artificial Intelligence Applications* (Santa Barbara, March 5-9, 1990), (Los Alamitos, CA: IEEE Computer Society Press), pp. 184-190.

[6] Adorf, H.-M., and Johnston, M.D. 1990: "A Discrete Stochastic 'Neural Network' Algorithm for Constraint Satisfaction Problems", H.-M. Adorf and M.D. Johnston, in *Proc. Int. Joint Conf. on Neural Networks* (IJCNN 90), (San Diego, June 17-21 1990), (Piscataway, NJ: IEEE), Vol. III, pp. 917-924.

[7] Johnston, M.D., and Adorf, H.-M. 1991: "Scheduling with Neural Networks - The Case of Hubble Space Telescope", to appear in *Int. J. Computers and Operations Research*.

[8] Minton, S., Johnston, M.D., Philips, A., and Laird, P. 1990: "Solving Large-Scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Method", in *Proc. of the Eighth National Conf. on Artificial Intelligence* (Boston July 29-August 3, 1990), (Menlo Park, CA: AAAI Press), pp. 17-24.

[9] Minton, S., Johnston, M.D., Philips, A., and Laird, P. 1991: "Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling problems", submitted.

[10] Sponsler, J.L., and Johnston, M.D. 1990: "An Approach to Rescheduling Activities Based On Determination of Priority and Disruptivity", in *Proc. 1990 Goddard Conf. on Space Applications of Artificial Intelligence* (Greenbelt, Maryland, May 1-2, 1990), ed. J. L. Rash, NASA Conf. Pub. 3068 (Greenbelt: NASA), pp. 63-74, reprinted in *Telematics and Informatics*, 7, pp. 243-253.

[11] Zweben, M., Davis, E., Stock, T., Drascher, E., Deale, M., Gargan, R., and Daun, B. 1991: "An Empirical Study of Rescheduling Using Constraint-Based Simulated Annealing", submitted.

[12] Miller, G. and Johnston., M.D. 1991: "A Case Study of Hubble Space Telescope Proposal Processing, Planning and Long-Range Scheduling", in *Proc. AIAA Conf. Computing in Aerospace 8*, Oct 21-24, 1991, Baltimore.

[13] Muscettola, N., Smith, S., Cesta, A., and D'Aloisi, D. 1992: "Coordinating Space Telescope Operations in an Integrated Planning and Scheduling Architecture", to appear in *IEEE Control Systems Systems Magazine* 12 Feb. 1992.

Temporal Planning for Transportation Planning and Scheduling

Robert E. Frederking and **Nicola Muscettola**
The Robotics Institute
Carnegie Mellon University
5000 Forbes Av.
Pittsburgh, PA 15213

Introduction

Many problems in transportation can be represented as flow problems, and can be optimally solved using efficient linear programming techniques [BHM77] [BGAB83]. But in some cases this approach is seriously oversimplified. If the problem includes dependencies between different operations, planning is necessary. If the system parameters change dynamically, the assumptions on which flow models are based become false, as in the case when the capacity of transportation facilities can change during the interval being analyzed. Finally, if detailed schedules are to be produced, answers in terms of bulk quantities do not suffice.

These problems require approaches that combine capabilities traditionally associated with planning and with scheduling, and that do not require their parameters to remain constant. Historically, temporal planners [DFM88] [AK83] have dealt with combining general operators to achieve a set of goals over time but have poorly attended to issues related to the optimization of resource usage. On the other hand, schedulers [SOP+90] [Sad91] have been concerned with allocating times and resources to operations in fixed process plans, ignoring questions of goal-oriented problem solving. The HSTS temporal planning framework [MSCD91] is an attempt to combine the capabilities of the two approaches. HSTS has been previously used for planning and scheduling the observations for the Hubble Space Telescope. HSTS emphasizes the description of the problem domain as a dynamical system organized through the use of **state variables**, i.e. persistent properties of objects in the domain. It also allows the development of opportunistic planners, where constraint posting and temporal inferences are not restricted to predefined directions on the time horizon (as in simulation and temporal projection) but the focus of problem solving can concentrate on the most congested areas of the time line.

In this paper we describe preliminary work done in the CORTES project[FS90], applying HSTS to a transportation planning and scheduling domain. First, we describe in more detail the transportation problems that we are addressing. We then describe the fundamental characteristics of HSTS and we concentrate on the representation of multiple capacity resources. We continue with a more detailed description of the transportation planning problem that we have initially addressed in HSTS and of its solution. Finally we describe future directions for our research.

The transportation problem

We are interested in addressing large-scale, complex transportation planning and scheduling problems, such as are found in disaster relief operations or other large-scale, international responses to emergency situations. For example, the transportation aspects of military operational plans (or OPLANs) must be feasible, given the allocated transportation resources [Han88]. If not, they must be reworked, or have more resources allocated to them. OPLANs are very large, involving the movement of tens of thousands of individual units, which vary immensely in size and composition, from a single person or piece of cargo to an entire division. However, OPLANs do not explicitly represent justifications for precedence constraints due to the structure of the domain and are therefore difficult to modify or adapt to other situations. To concentrate on the representation of domain structure in a transportation schedule, we addressed the 'bare base' deployment scenario used at the Armed Forces Staff College (AFSC) to train joint planning officers. The goal is to turn a bare runway into a fully functioning air base. Documents are available from AFSC describing scenario assumptions and types of available units, including recommended sequencing, and some hints at the dependencies between units. This domain includes only 92 unit types, in 40 general categories. It is also simplified in that OPLANs generally involve much more than deploying a single air base, and more than one armed service.

Our analysis of the bare base domain revealed two facts:

- The domain requires the ability to represent and reason about aggregate capacity resources.
- This domain consists primarily of a moderate number (order of 10) dependency cycles, each centered around a different support function, such as air traffic control, aircraft refueling, personnel or cargo unloading, etc. The arrival of support units increases the possible arrival rate of additional support units.

We isolated one of the dependency cycles, the refueling capacity/throughput loop, as an initial 'atomic'

domain. A demand on the **base refueling capacity**, an aggregate resource, can be satisfied by bringing more **refueling units** to the base. The arrival of a unit permanently increases refueling capacity, which in turn affects the rate at which **planes** can arrive, since they use some amount of refueling capacity immediately after moving. This increases also the rate at which additional units can be brought in. These simplified refueling units have no support requirements, so when they are operational at the base they increase its capacity, without requiring any other units to be brought in.

The representation and solution of this problem is an important step toward a solution of the bare base scenario.

Representing plans in HSTS

Transportation problems require to be able to deal with dependencies involving state and resource capacity (e.g., a unit that requires a plane to move from A to B can be allocated space only on a plane that is also moving from A to B). This can be done by using the HSTS planning and scheduling framework [MSCD91]. The two main components of the framework are a **domain description language**, for modeling the structure and dynamics of the physical system at multiple levels of abstraction, and a **temporal data base**, for representing possible evolutions of the state of the system over time.

In this section we describe the basic primitives provided by HSTS and the extensions needed to represent aggregate resource capacity.

Representing state

An HSTS model is subdivided into **state variables**, each of which can assume one and only one value in any instant of time. A value has the form $R(x_1, x_2, \ldots, x_n)$. For example, a plane *?p* has a location, represented by state variable $Loc(?p)$, that can assume value $MOVE(?p, ?u, ?src, ?dst)$ representing the fact that *?p* is in transporting unit *?u* from location *?src* to location *?dst*. HSTS is interval based, i.e., if a value occurs on a state variable, it persists for a continuous non-zero time interval. A value can occur under conditions specified through a **duration specification** and a **compatibility specification**.

Figure 1 shows a hypothetical value descriptor. The **duration** is expressed as a range constraint, $[d, D]$, with d and D representing respectively a lower bound and an upper bound function. The rest of the descriptor specifies the **compatibilities** that have to be satisfied. A compatibility specification is an AND/OR graph connecting several elementary compatibilities. Each compatibility is composed of a temporal relation and the specification of a segment of behavior on a state variable. For example the compatibility $[met\_by\ \langle \nu, Loc(?p), AT(?src)\rangle]$ associated to $\langle Loc(?p), MOVE(?p, ?u, ?src, ?dst)\rangle$ in Figure 1 specifies that in every legal behavior, the value $MOVE$ must occur immediately after the value AT on $Loc(?p)$. The symbol ν is one of two different kind of segments of evolution of a state variable: ν, constraining a single

$\langle Loc(?p), MOVE(?p, ?u, ?src, ?dst)\rangle$
  duration: $[dur(?src, ?dst), Dur(?src, ?dst)]$
  compatibilities:
    AND ($[met\_by\ \langle \nu, Loc(?p), AT(?src)\rangle]$
          $[meets\ \langle \nu, Loc(?p), REFUEL(?dst)\rangle]$
          $[eql\ \langle \nu, Loc(?u), MOVE(?p, ?u, ?src, ?dst)\rangle])$

Figure 1: HSTS value descriptor

value, as in the example above, and σ, for sequence compatibilities. A sequence that can be substituted by an unspecified number of values occurring on the same state variable, all of which must satisfy a constraint associated with the sequence. We will see examples of sequences when we will discuss aggregate capacity state variables.

Behaviors can be constructed within the HSTS Temporal Behavior Data Base. The unit of description of temporal behavior is the **token**, a quadruple $\langle sv, type, st, et\rangle$, where sv is one of the state variables in the system model, $type$ is a subset of the state variable's possible values, and st and et are the token's start and end times respectively. Tokens represent an uninterrupted segment of evolution of a state variable. During the planning process a token can be refined by being split into any number of component tokens; however, a token that has been designated to represent the occurrence of a value cannot be further split. A token that can be split is referred to as a **plan constraint**; one that cannot be split is referred to as a **plan value**. The TDB also allows the representation of **token sequences** which implement the occurrence of a sequence specification. Tokens and token sequences are connected by a network of constraints: **temporal constraints**, relating the start and end times of each token, and **type constraints**, referring to the type of each token. Temporal and type constraints derive either from the expansion of compatibilities and durations extracted from the model of the system, from requirements directly imposed by the user and therefore constituting the problem to be solved, or from refinement decisions taken during the problem solving process where one of multiple alternatives needs to be explored.

Representing Aggregate Resource Capacity

At the base of the HSTS representation philosophy is the assumption that it is possible to identify each state variable into which a system model is decomposed and that each state variable can assume one of a handful of symbolic values. However, this basic mechanism of representation can become very cumbersome. For example, to reason on the allocation of available space on a plane to materials, we would have to subdivide space on the plane into "unit of space" state variables, with values 'free' or 'used', subdivide also the materials into units of space, and allocate capacity each unit of material space to a unit of plane space. Although this might be necessary for a detailed map of the allocation

of plane space, it is overly detailed for cases when we need only an aggregated characterization of the use of space.

HSTS can represent aggregated capacity as an **aggregate state variable**. The value of an aggregate state variable at a given time is a summary of the value of a corresponding set of atomic state variables at the same instant of time. In the transportation planning domain, the use of cargo or parking space or the generation or use of refueling capacity by a unit or plane at a base falls into this category.

A set of atomic state variables constitutes the conceptual base on which the aggregation is built. In our discussion, they are **atomic resources** that can be used by one and only one **operation** at a time. An operation OP_i is the value assumed by the *state* state variable of a **job** $?j$, $St(?j)$, while $?j$ is undergoing the specified operation. If $?j$ is not undergoing any operation, the value of $St(?j)$ is *IDLE*. An atomic resource $?r$ has a single atomic state variable, $St(?r)$, with possible values *OPER* (processing some operation) and *IDLE*.

The occurrence of OP_i and of *OPER* is regulated by the following bidirectional compatibility:

$$\langle \nu, St(?j), OP_i \rangle [eql\ \langle \nu, St(?r), OPER \rangle]$$

If the atomic resources in a pool $?r\_p$ are perfectly substitutable, they can be aggregated into a single aggregate state variable, the **aggregate processing capacity** of the pool, $Cap(?r\_p)$. At any instant of time, the aggregate state variable will assume a single value that will summarize the distribution of values over its component state variables at that time. $Cap(?r\_p)$ gives the number of resources in the pool that hold each of the values *OPER* and *IDLE*; its values are represented as follows:

$$\{\langle OPER, n_1 \rangle, \langle IDLE, n_2 \rangle\}$$

indicating that n_1 atomic resources in $?r\_p$ are in an *OPER* state and n_2 are in an *IDLE* state. The number of resources in $?r\_p$ at that instant of time is $n_1 + n_2$. In general, a value for an aggregate state variable is a list of such entries $\langle value, counter \rangle$.

Compatibility constraints on values of aggregate state variables specify one or more atomic values and, for each value, the number of atomic resources affected. For example, assuming that OP_i requires c_i atomic resources, we will have:

$$\langle St(?j), OP_i \rangle \rightarrow [eql\ \langle \sigma, Cap(?r\_p), \{\langle OPER, INC(+c_i) \rangle, \langle IDLE, INC(-c_i) \rangle\}]$$

This means that whenever OP_i occurs, a sequence of values must be found on $Cap(?r\_p)$, and the start and end times of the sequence must coincide with the start and end of OP_i, as indicated by the temporal relation *eql*. The type specification describes the local effect of the compatibility on each of the values in the sequence, i.e, the number of atomic resources that are *OPER* is incremented by $+c_i$, while the number of those that are *IDLE* is decremented by c_i.

At time τ, the actual value of an aggregate state variable can be computed once the set of constraints that contain τ is known. In the case of $Cap(?r\_p)$, if we

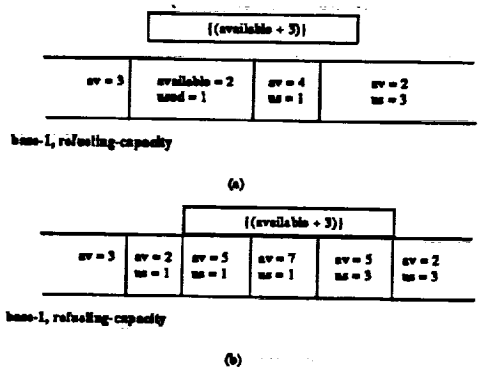


Figure 2: Posting a sequence constraint on an aggregate capacity state variable

suppose we have n_{opr} entries of type $\langle OPER, INC(c_i) \rangle$ and n_{idle} entries of type $\langle IDLE, INC(c_j) \rangle$, the value $\{\langle OPER, n_1 \rangle, \langle IDLE, n_2 \rangle\}$ at time τ satisfies the relations:

$$n_1 = \sum_{i=1}^{n_{opr}} c_i \qquad n_2 = \sum_{j=1}^{n_{idle}} c_j$$

where c_i and c_j can be both positive (creation) or negative (consumption).

During the planning process, the evolution of an aggregate state variable is represented in the temporal data base by a sequence of plan constraints determined by the imposition of a set of sequence constraints (Figure 2). Note that the temporal extension of each aggregate state variable's value is not fixed. This is an important difference from other scheduling systems, where the times must be fixed if the values of aggregate capacities are to be fixed [SOP+90] [Sad91].

Consistency of the state of a temporal data base can be checked by temporarily assuming that no more sequence constraints will be posted and, therefore, the plan constraints can be safely substituted with plan values that can be computed by applying constraints like those for n_1 and n_2, above. The data base will be inconsistent when an aggregate value contains a counter whose value is negative. Notice however that, in the case where the physical system allows the generation of capacity (as for aggregate processing capacity), partial inconsistency can be resolved without backtracking by posting additional compatibilities providing the missing capacity.

Planning within HSTS

The atomic domain was intended to demonstrate that the new extensions to HSTS for this type of domain (principally those for handling aggregate capacity) function correctly, and provide the necessary primitives to solve the fundamental problems that such a domain presents.

The atomic domain representation

The state variables in this domain are the refueling and throughput properties of three types of objects: **units**, **planes**, and **bases**.

Each unit has two associated state variables, its location *Loc* and its state *St*. A unit's *Loc* can have the values *AT* and *MOVE*. These correspond to the unit being stationed at some base (e.g., home or destination) or being in transit. A unit's *St* can have the values *NOT_OPER* or *OPER*. These indicate whether it is capable of providing refueling capacity. When *OPER*, it adds enough capacity to refuel one additional plane. Each plane has one state variable, *Loc*, which can have the values *IDLE*, *MOVE*, and *REFUEL*.[1]. A base has one aggregate state variable, its refueling capacity *R_C*, containing distributions of two values, *AVAIL* and *USED*, indicating the total amount of available and used refueling capacity at any time. The principal compatibilities describing this problem are:

- The *MOVE* of a unit is followed by it being *OPER* some non-zero amount of time later[2]:

  $\langle Loc(?u), MOVE(?p, ?u, ?src, ?b)\rangle \rightarrow$
  $[bf([\delta t, \delta t])\ \langle \nu, St(?u), OPER(?u, ?b)\rangle]$

- The *MOVE* of a unit is concurrent with the *MOVE* of a plane:

  $\langle Loc(?u), MOVE(?p, ?u, ?src, ?b)\rangle \rightarrow$
  $[eql\ \langle \nu, Loc(?p), MOVE(?p, ?u, ?src, ?b)\rangle]$

- The *MOVE* of a plane is immediately followed by *REFUEL*:

  $\langle Loc(?p), MOVE(?p, ?u, ?src, ?b)\rangle \rightarrow$
  $[meets\ \langle \nu, Loc(?p), REFUEL(?p, ?b)\rangle]$

- The unit increases *R_C*(?*b*) while it is *OPER*:

  $\langle St(?u), OPER(?u, ?b)\rangle \rightarrow$
  $[eql\ \langle \sigma, R\_C(?b), \{\langle AVAIL, INC(+1)\rangle\}\rangle]$

- The *REFUEL* of the plane creates a demand on *R_C*(?*b*):

  $\langle Loc(?p), REFUEL(?p, ?b)\rangle \rightarrow [eql\ \langle \sigma, R\_C(?b),$
  $\{\langle USED, INC(+1)\rangle, \langle AVAIL, INC(-1)\rangle\}\rangle]$

The atomic domain planner

The HSTS model of a domain describes domain constraints in terms of durations of and compatibilities between values of state variable tokens, as described previously. This creates an implicit space of legal sets of state variable value sequences, within which any partial (or complete) solutions to problems in this domain must lie.

However, in order to describe any specific partial solution, a particular set of legal choices must be made. Many such sets of choices will result in inconsistent sets of compatibilities, not corresponding to any possible system behaviors. Finding a consistent set of choices (i.e., planning) can still be very difficult. Within the HSTS least-commitment framework, the final solution is a representation of a range of behaviors that can be directly simulated, all guaranteed legal.

In the case of transportation planning and scheduling, one must select actual units to supply required support, and select actual ranges of arrival times for these units. This selection is ultimately based on the needs of some set of units whose operation at the destination directly fulfills external (top-level) goals. These top-level units require support of various kinds, and their support units in turn require support. Any of these units not already at the destination need to be transported there.

Thus, any unit that needs to be transported ultimately serves a top-level goal through some chain of dependencies. This means that the planner can work by finding 'operators' that satisfy goals, and then other 'operators' that provide these operators' preconditions and fix problems from their postconditions; except that, in HSTS, the planner is assigning values to certain time intervals of state variables, and using the compatibilities between these values and other values as 'preconditions' and 'postconditions'.

The planning goal is represented as a request for a large amount of *USED* refueling capacity during some future interval. The posting of the request creates an interval of time in which the *AVAIL* capacity is negative.

The planning process begins with an HSTS **fetch** for intervals where the base refueling capacity is below zero, locating the top-level problem. The planner then finds which types of values provide the type of capacity needed, and which state variables can have these values. It selects enough instances of these variables to satisfy the demand, creates the appropriate value tokens for them, and constrains these tokens to occur over the required interval. This solves the top-level problem.

Then, for each of these state variables, its token's compatibilities are **implemented**, that is, constraints between the token and other values are enforced, to guarantee that this is a legal behavior. Single-unit compatibilities are done first, followed by those that affect other units. This ordering is important in general, since local constraints may limit the choices available to more global ones. This corresponds to classical systems having process plans for individual jobs before scheduling their operations. Since dependencies between different units of the same type are expressed through aggregate variables, this ordering is equivalent to saying that compatibilities that do not affect aggregate variables are done first. The set of local compatibilities in this domain are simply the first three listed in the previous subsection.

Next, the compatibility for the effect of plane refueling on the aggregate capacity is implemented. This requires the choice of a particular time interval for plane refueling, relative to the intervals of different levels of aggregate capacity. This is done in one of three modes: planes are allocated times as late as possible, as early as possible, or at user-selected times. This variability demonstrates the complete flexibility of the order of decisions in 'simulated time' (time in the mod-

[1] For this abstract model, representing refueling state and location separately would have introduced irrelevant complications.

[2] This is necessary to prevent a degenerate problem, where each unit brought in adds enough capacity to handle its own plane, thus immediately allowing an arbitrary number of units to be brought in.

eled domain). This flexibility allows for **opportunistic** decision-making, where decisions are made in the most efficient order, not in any pre-determined temporal order. Other temporal planners generally cannot make decisions this flexibly when working at the most detailed level.

Finally, the effect of the unit becoming operational on the aggregate capacity is implemented. In both of these last two steps, some search may be needed. The interval initially chosen may not produce a legal configuration, due to the simple mechanism for picking an interval and a limitation of the current aggregate variable mechanism. Currently, once the contribution from a state variable to an aggregate variable is calculated, its relative position in the aggregate cannot be changed without backtracking. This violates the least-commitment principle, and leads to problems: some intervals on the aggregate variable have zero length. If our simple interval selection rule selects a zero-length interval for a non-zero length event, an inconsistency results. Currently the easiest way to handle this is to implement, detect the inconsistency, and backtrack. Very limited search is needed, since non-zero intervals are much more frequent. Fixing this failure of least-commitment is high on our research agenda.

When these steps have been carried out for the necessary number of variables, a complete and consistent behavior has been described that fulfills the top-level goals.

Conclusion

Temporal planning methodologies can be applied to solve transportation planning problems that are beyond the scope of traditional linear programming techniques. In our work we have addressed one such problem and identified a fundamental type of dependency among its entities. We have then demonstrated that problems involving this kind of dependency can be solved within the HSTS temporal planning and scheduling framework. To solve the full bare base deployment scenario, we need to extend our current problem solver to incorporate heuristic knowledge in order to select the most appropriate units and time intervals for values, and carry out local search if necessary.

In order to deal with real-world scale problems, it will be necessary to develop further problem aggregation and abstraction techniques. One promising direction concentrates on taking advantage of the temporal flexibility of the HSTS framework by combining least-commitment constraint posting methodologies with probabilistic estimates of resource usage [MS87]: the goal is to avoid spelling out unnecessary details whenever possible while insuring high quality possible executions of the temporal plan.

References

[AK83] J. Allen and J.A. Koomen. Planning using a temporal world model. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 741–747, 1983.

[BGAB83] L. Bodin, B. Golden, A. Assad, and M. Ball. Special issue on the routing and scheduling of vehicles and crews. *Computers and Operations Research*, 10(2), 1983.

[BHM77] S.P. Bradley, A.C. Hax, and T.L. Magnanti. *Applied Mathematical Programming.* Addison-Wesley Publishing Co., 1977.

[DFM88] T. Dean, R.J. Firby, and D. Miller. Hierarchical planning involving deadlines, travel time, and resources. *Computational Intelligence*, 4:381–398, 1988.

[FS90] M.S. Fox and K. Sycara. Overview of cortes: A constraint based approach to production planning, scheduling and control. In *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management.* ESPOM-90, 1990.

[Han88] S.H. Hanes, editor. *The Joint Staff Officer's Guide.* U.S. Government Printing Office, 1988. Publication 1, Armed Forces Staff College.

[MS87] N. Muscettola and S.F. Smith. A probabilistic framework for resource-constrained multi-agent planning. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 1063–1066. Morgan Kaufmann, 1987.

[MSCD91] N. Muscettola, S.F. Smith, A. Cesta, and D. D'Aloisi. Coordinating space telescope operations ina an integrated planning and scheduling architecture. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, pages 1369–1376, 1991.

[Sad91] N. Sadeh. *Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling.* PhD thesis, Schhol of Computer Science, Carnegie Mellon University, March 1991.

[SOP+90] S.F. Smith, P.S. Ow, J.Y. Potvin, N. Muscettola, and D. Matthys. An integrated framework for generating and revising factory schedules. *Journal of the Operational Research Society*, 41(6):539–552, 1990.

A Simulated Annealing Approach to Schedule Optimization for the SES Facility

Mary Beth McMahon and **Jack Dean**
Planning and Scheduling Technology Group
McDonnell Douglas Space Systems Co.
16055 Space Center Blvd
Houston, TX 77062

Introduction

The SES is a facility which houses the software and hardware for a variety of simulation systems. The simulators include the Autonomous Remote Manipulator, the Manned Maneuvering Unit, Orbiter/Space Station docking, and shuttle entry and landing. The SES simulators are used by various groups throughout NASA. For example, astronauts use the SES to practice maneuvers with the shuttle equipment; programmers use the SES to test flight software; and engineers use the SES for design and analysis studies.

Due to its high demand, the SES is busy twenty-four hours a day and seven days a week. Scheduling the facility is a problem that is constantly growing and changing with the addition of new equipment. Currently a number of small independent programs have been developed to help solve the problem, but the long-term answer lies in finding a flexible, integrated system that provides the user with the ability to create, optimize, and edit the schedule.

COMPASS is an interactive and highly flexible scheduling system. However, until recently COMPASS did not provide any optimization features. This paper describes the simulated annealing extension to COMPASS. It now allows the user to interleave schedule creation, revision and optimization. This practical approach was necessary in order to satisfy the operational requirements of the SES.

Statement of Problem

The SES facility is scheduled a week at a time. A work week consists of seven days, each of which is divided into six 4-hour "sessions." Each session has two sides, side-a and side-b. This allows two people to work in the facility at the same time. Each person requiring time at the facility makes a request telling what equipment is needed for the simulation. A request consists of the required simulators, the preferred days and sessions, and optionally, a preferred side. Each person may make one or more requests per week and may ask for multiple iterations of the same request. The SES scheduler satisfies their requests by creating a schedule based on priorities. The SES manager determines the priority of each request by the type of work being done and the number of repetitions requested. For example, mission related activities have a higher priroity than software development activities. And the fourth repetition of a request typically has a much lower priority than the first. Each week there are about 60 - 70 requests and 76 session slots to be filled. There are additional requests at the last minute for empty slots, as well as high priority requests coming through that may bump lower priority items.

There are a few guidelines by which the SES facility is scheduled. First, there is only one instance of each simulator; therefore the persons working on side-a and side-b must use mutually exclusive sets of equipment. Second, certain pieces of equipment reside only on certain sides; therefore side assignments must coincide with equipment requirements. Third, a person may state a preference for particular sessions, may state which sessions are acceptable, and may state which sessions are unacceptable. The schedule should try to accommodate the preferences, but can place the person in an acceptable session when the preferred sessions are not available. Under no circumstance should a person be placed in a session which has been marked as unacceptable. Fourth, a person can only work up to two sessions in one day, and if they do, the sessions should be consecutive so that a straight eight hour day is worked. Fifth, each person should have at least an eight hour break between non-consecutive scheduled sessions. Sixth, if a person works more than one third shift (session five or session six) then the third shift sessions worked should be on consecutive days or at least two days apart.

Each request has a primary and secondary requestor. The above rules must be satisfied in the event that either the primary or secondary requestor work the session.

There are two goals to consider when scheduling the SES facility. One is to produce a weekly schedule in which the largest number of requests are satisfied. The other is to fill the schedule with the highest priority items. These two goals must be satisfied simultaneously, but there are no rules defining the trade-offs be-

tween quantity and priority. It is left up to the scheduler to produce a schedule which, in his opinion, works the best. In fact, if so inclined, the scheduler may actually violate resource or timing constraints listed above when producing the schedule.

Currently, the requests are entered on a PC and then transferred to a Cyber computer where an optimization routine written in FORTRAN finds 10 candidate schedules. The SES manager then selects one of the 10 schedules and hand edits it. The editing usually consists of adding late assignments and moving assignments around for subjective reasons. This is done with paper and pencil to keep track of resource assignments. Finally the handwritten schedule is entered into a PC, using a drawing program, where it is printed out for distribution.

When providing an integrated solution for the SES problem, all phases of the scheduling process must be considered. First, the scheduling system must be able to accept and handle all of the constraints and preferences described by the requests. Second, the system must provide the SES manager with an initial feasible schedule which is at least as "good" as the initial schedules produced by the FORTRAN program. Third, the system must allow the schedule to be modified, even if it means overriding constraints. And fourth, the system must print the schedule in the prescribed SES format.

Background

An interactive scheduling system allows the user to impose subjective constraints such as the trade-off between the quantity of requests satsified and the priorities of the activities scheduled. A non-chronological system allows the user to place activities anywhere in the week, so that high priority items can be scattered throughout the week and low priority items can fill the leftover time slots. These two characateristics, along with the fact that COMPASS only produces feasible schedules, lay the ground work for solving the SES scheduling problem. The significance of these characteristics is described further.

An interactive scheduling system provides an environment where a mixed initiative is possible; that is, it lets the computer do what it does best (check constraints and calculate feasible intervals) and lets the human do what he/she does best (provide heuristic and subjective inputs into the schedule). Together the two can cooperatively produce a schedule which reflects both the hard constraints and subjective preferences. Subjective preferences may be controlled through input from the user. The input may reflect scheduling heuristics, such as the order in which to schedule the activities and whether to schedule as soon as possible or as late as possible. The input may reflect the desired look of the schedule, such as choosing where to place the activity from among the feasible intervals of time. Or the user may interactively direct the search, by specifying which items to freeze and which items to optimize.

In contrast, a fully automated system requires that all data be completely loaded before the system begins scheduling. All rules about scheduling preferences and optimization must be coded into the system before the scheduling process begins. The system then runs uninterrupted until it finds one solution (or many depending on the system) and then presents its findings as the final schedule. There is generally no effective way of editing the schedule once the solution is found. This method of scheduling is perfectly acceptable when the problem is bounded and the domain can be described completely. However, in a highly subjective scheduling domain, coding all of the rules (and exceptions-to-the rules) may become very laborious or even impossible.

A non-chronological scheduler allows the system to place activities anywhere on the timeline. The system has an omniscient view of time and can determine all the feasible intervals of time where the next activity may be placed. As each activity is placed on the schedule, constraints created by that activity must be propagated (either in the environment or directly to other activities). When new activities are placed on the schedule, they are constrained by the activities already on the schedule. A benefit of non-chronological scheduling is that high priority items may be placed on the schedule first and guaranteed that they be completed. Then the schedule may be filled with the lower priority items.

In contrast, a simulation-based scheduler starts at the beginning time of the schedule and as it progresses through time, it places activities on the schedule. When resources become available, the system has a choice about which item to place next on the schedule. Once the schedule reaches the ending time, the schedule can be evaluated and another pass may be made, perhaps making different choices about what to place at each decision point. Historically, simulation-based schedulers are very popular in the job shop arena as they naturally model the behavior of plant operations.

COMPASS, with its simulated annealing extension, searches only the feasible solution space. Some schedulers only search the feasible solution space, while others search both the feasible and infeasible solution space. It may be substantially easier to find good solutions if the scheduler is allowed to wander through the infeasible solution space. However, allowing infeasible solutions also greatly increases the size of the search space. There are far more infeasible solutions than feasible solutions. By prohibiting the search of the infeasible solution space, the scheduler has more time to spend evaluating feasible solutions. Deciding which solution space to search depends on the optimizing algorithm.

Approach

This section describes how the simulated annealing routine is used in conjunction with COMPASS.

Given a group of selected activities to optimize, the simulated annealing algorithm calls upon the COMPASS scheduling engine to unschedule then reschedule the selected activities in a different order. The activities are continuously rescheduled and the objective function is evaluated for each new schedule until a user specified time limit is up. When time is up COMPASS displays the schedule with the best score.

The user designates the focus of attention for the optimization by selecting a subset of activities. The user can select all of the activities, in which case the entire schedule is optimized with respect to the objective function. Or the user may select a subset of activities, in which case only part of the schedule is optimized. A benefit of this is that the user can selectively optimize parts of the schedule which need improvement, leaving the rest of the schedule intact.

The user may also specify the amount of time in which to run the simulated annealing algorithm. For simple schedules or small subsets of activities a small amount of time may be all that is necessary. COMPASS displays each new try as it is created. The user can actually sit and watch as the schedule is being modified. Once time is up, COMPASS redisplays the best schedule.

A scenario for using COMPASS and its simulated annealing extension is as follows. A first cut at the schedule can be created using the optimization function. The user can edit the schedule by unscheduling some activities or by forcing unscheduled high priority activities (overriding any constraints) onto the schedule. By evaluating the schedule, COMPASS will display all activities which now have conflicts. The user can unschedule the conflicting activities and reschedule them (using the optimization function or by placing each down interactively). The interaction between user placement and optimization continues until the final schedule is reached.

Implementation

Simulated annealing is an optimization technique which combines gradient descent with randomness to find global optima. The process used to control the optimization is analogous to the annealing of metal; hence the name simulated annealing. The annealing process is based on the laws of thermodynamics which state that atoms tend toward a minimum energy state. A metal is annealed by raising the metal to temperature over its melting point and then gradually cooling it. At high temperatures the atoms are in a high energy state, violently and randomly moving about. As the metal cools, lower and lower energy states become increasingly likely. By cooling the metal slowly, the lowest possible energy state, the global minimum, can be achieved.

Simulated annealing is used to find global minima in optimization problems in the following fashion. An initial solution to the optimization problem is found by some means. The search space of solutions becomes the state space of the simulated annealing algorithm. An objective function, for which a global minimum is to be found, is defined over the search solution space. The objective function corresponds to the energy function. For each iteration, a random change is made to the state to obtain a new state. If this new state has a lower energy than the previous state, the new state is kept. If the new state has a higher energy than the old state, it is kept with a probability that varies with the simulated temperature. Continuing the analogy to the annealing of metal, this probability is proportional to the exponential of -c/kT, where c is the change in the energy level, k is constant analogous to the Boltzmann's constant for physical systems, and T is the simulated temperature. At very high temperatures, most changes in state are accepted, and the result approaches a random walk through the solution space. At very low temperatures, the probability of accepting a change that increases the total energy vanishes, and the random walk is limited to changes which decrease the total energy. This results in a gradient descent to the local minima. To achieve the global minimum, the temperature is started off very high and gradually reduced. For each local minimum there is a temperature which will allow the random walk to escape the local minimum, but not the global minimum.

In order to apply this algorithm to the SES optimization problem, the following have to be defined: (1) the state space of searched solutions, (2) the energy or objective function to be minimized, (3) the method for calculating the initial solution, (4) the method for randomly changing from one state to the next, and (5) the temperature decay algorithm.

The solution space consists of all feasible schedules. A feasible schedule is one that satisfies all the constraints. The constraints that are applicable to the SES scheduling problem are the resource availability constraints, the temporal constraints, and the rules discussed in the Statement of Problem section of this paper.

The objective function is the negative of the sum of the values of the scheduled activities. (The negative is used so that minimization of the objective function indicates improving schedules.) The value for each activity is derived from the priority input field of the schedule request. The priority is an integer between 1 and 22 inclusive, with 1 being the highest priority (most important). The value for the activity is set to 23 minus the request priority. Thus increasing value means increasing importance of the task.

An initial solution is found using a first fit decreasing algorithm. The activities to be scheduled are sorted into decreasing value order. The sorted activities are then scheduled using a front loading, or first

fit, scheduling algorithm.

Once a feasible schedule is found, a new random schedule is calculated in the following fashion. First, the probability that a scheduled task should be removed is calculated, based upon the current simulated temperature. The probability of removal is calculated using an equation of the form of the Boltzmann equation described above. Thus the probability of removal is higher at high temperatures than it is at low temperatures. This has the effect of allowing larger state changes at high temperatures and minor changes at low temperatures.

Next, each activity is examined in decreasing value order. If the activity is already scheduled, and a randomly generated number is less than the probability of removal, the activity is removed from the schedule. If the examined activity is previously unscheduled, and a randomly generated number is less than a constant probability of placement, the activity is placed on a list of activities to be scheduled.

Once the entire activity list is examined, with some of the activities randomly selected and unscheduled, the list of activities to be scheduled is examined. For each activity, the program first tries to schedule the activity in one of the preferred sessions. If that fails (the activity is not scheduled), the program attempts to schedule the activity in any one of the acceptable sessions.

Once the new schedule has been created, the energy value for this new schedule is calculated. If the new energy value is lower than the current energy, the new schedule is kept since it reflects an improved schedule. If the new energy is greater than the current energy (reflecting a poorer schedule), the probability of accepting this schedule is calculated using the Boltzmann equation described above.

Finally, the temperature decay used is a simple inverse linear function. The simulated temperature is set according to the equation $T = T0 / (1 + t)$, where T is the current temperature, T0 is the initial temperature, and t is the simulated time.

Conclusions

The simulated annealing algorithm has been successfully implemented and integrated into the COMPASS architecture. This new addition allows the user to automatically, as well as interactively, create schedules. This combination of automatic and interactive capabilities provides the user with greater functionality and control over the development of the schedule. The user can define the level of interaction/automation necessary in order to produce the best schedule.

The user selects the activities that are to be optimized. The user may optimize the whole schedule by selecting all of the activities or part of the schedule by selecting a subset of the activities. (In the SES problem the objective function is based on the priorities of the activities, so it is feasible to apply it to subsets of activities as well as the entire set.) The previously scheduled activities that have not been selected remain frozen on the schedule. This is especially beneficial in rescheduling once the initial schedule is underway and an event occurs which requires parts of the schedule to be reworked.

The SES scheduling problem requires an integrated system which will create an initial feasible schedule, allow the user to alter or optimize parts of the schedule, and will print out the schedule in the desired format. COMPASS now provides all of these capabilities in one cohesive package. The user can schedule both interactivly and automatically. The user can override any constraints by forcing an activity onto the schedule at a specific time. The user can validate the schedule using existing evaluation functionalities. And the user can print out reports in the desired format using PostScript.

References

[1] Fox, Barry R., *Mixed Initiative Scheduling*, AAAI - Spring Symposium on AI in Scheduling, Stanford, CA, March 1989.

[2] Fox, Barry R., *Non-Chronological Scheduling*, Proceedings AI, Simulation and Planning in High Autonomy Systems, University of Arizona, March 1990, IEEE Computer Society Press.

[3] Lund, Chet, *Expert System for Scheduling Simulation Lab Sessions*, Proceedings of the 1991 CLIPS Conference, pp 784-791.

[4] Wasserman, Philip D., Neural Computing Theory and Practice, pp 80-83.

54-63
N93-18663
18923
P-5

Decomposability and Scalability in Space-Based Observatory Scheduling

Nicola Muscettola and Stephen F. Smith
The Robotics Institute
Carnegie Mellon University
5000 Forbes Av.
Pittsburgh, PA 15213

Introduction

The generation of executable schedules for space-based observatories is a challenging class of problems for the planning and scheduling community. Existing and planned space-based observatories vary in structure and nature, from very complex and general purpose, like the Hubble Space Telescope (HST), to small and targeted to a specific scientific program, like the Submillimeter Wave Astronomy Satellite (SWAS). However the fact that they share several classes of operating constraints (periodic loss of target visibility, limited onboard resources, like battery charge and data storage, etc.) suggests the possibility of a common approach. The complexity of the problem stems from two sources. First, they display the difficulty of classical scheduling problems: optimization of objectives relating to overall system performance (e.g., maximizing return of science data), while satisfying all constraints imposed by the observation programs (e.g., precedence and temporal separation among observations) and by the limitations on the availability of capacity (e.g., observations requiring different targets cannot be executed simultaneously). Second, a safe mission operation requires the detailed description of all the transitions and intermediate states that support the achievement of observing goals and are consistent with an accurate description of the dynamics of the observatory; this constitutes a classical planning problem.

Another characteristic of the problem is its large scale. The size of the pool of observations to be performed on a yearly horizon can typically range from thousands to even tens of thousands, and, for large observatories, the dynamics of system operations involves several tens of interacting system components. To effectively deal with problems of this size, it is essential to employ problem and model decomposition techniques. In certain cases, this requires the ability to represent and exploit the available static structure of the problem (e.g., interacting system components); in other cases, where an explicit structure is not immediately evident (e.g., interaction among large numbers of temporal and capacity constraints), the problem solver should be able to dynamically focus on different parts of the problem, exploiting the structure that emerges during the problem solving process itself.

In this paper, we discuss issues of problem and model decomposition within the HSTS scheduling framework. HSTS was developed and originally applied in the context of the HST scheduling problem, motivated by the limitations of the current solution and, more generally, the insufficiency of classical planning and scheduling approaches in this problem context. We first summarize the salient architectural characteristics of HSTS and their relationship to previous scheduling and AI planning research. Then, we describe some key problem decomposition techniques supported by HSTS and underlying our integrated planning and scheduling approach, and discuss the leverage they provide in solving space-based observatory scheduling problems.

Planning and scheduling for space-based observatories

The management of the scientific operations of the Hubble Space Telescope is a formidable task; its solution is the unique concern of an entire organization, the Space Telescope Science Institute (STScI). The work of several hundred people is supported by several software tools, organized in the Science Operations Ground System (SOGS). At the heart of SOGS is a FORTRAN-based software scheduling system, SPSS, originally envisioned as a tool which would take astronomer viewing programs for a yearly period as input and produce executable spacecraft instructions as output. SPSS has had a somewhat checkered history [Wal89], due in part to the complexity of the scheduling problem and in part to the difficulty of developing a solution via traditional software engineering practices and conventional programming languages. To confront the computational problems of SPSS, STScI has developed a separate, knowledge-based tool for long term scheduling called SPIKE [Joh90]. SPIKE accepts programs approved for execution in the current year and partitions observations into weekly time buckets, each of which can then be treated as a smaller, more tractable, short term scheduling problem. Detailed weekly schedules are generated through the efforts of a sizable group of operations astronomers, who interactively utilize SPSS to place observations on the time line.

In the HSTS project we have addressed the short term problem in the HST domain, efficiently generating detailed schedules that account for the major telescope's operational constraints and domain optimization objectives. The basic assumption is to treat resource allocation (scheduling) and auxiliary task ex-

pansion (planning) as complementary aspects of a more general process of constructing behaviors of a dynamical system [Mus90].

Two basic mechanisms provide the basis of the HSTS approach:

1. a *domain description language* for modeling the structure and dynamics of the physical system at multiple levels of abstraction.
2. a *temporal data base* for representing possible evolutions of the state of the system over time (i.e. schedules).

The natural approach to problem solving in HSTS is an iterative posting of constraints extracted either from the external goals or from the description of the system dynamics; consistency is tested through constraint propagation. For more details, see [MSCD91].

Three key characteristics distinguish the HSTS framework from other approaches:

1. the explicit decomposition of the state of the modeled system into a finite set of "state variables" evolving over continuous time. This enables the development of scheduling algorithms that exploit problem decomposability and provides the necessary structure for optimizing resource utilization.
2. the flexibility along both temporal and state value dimensions that is permitted by the temporal data base (e.g., the time of occurrence of each event does not need to be fixed but can float according to the temporal constraints imposed on the event by the process of goal expansion). This flexibility contributes directly to scheduling efficiency, since overcommitment (and hence the greater possibility of the subsequent need to backtrack) can be avoided.
3. the flexibility of the constraint posting paradigm to accommodate a range of problem solving strategies (e.g., forward simulation, back chaining, etc.). This allows the incorporation of algorithms that opportunistically exploit problem structure to consistently direct problem solving toward the most critical tradeoffs that need to be made.

The importance of integrating these three features within a single framework can be appreciated by considering the limitations of other approaches that address them separately or partially.

Planning research has focused on the problem of "compiling" activity networks that bring about desired goal states from more basic representations of the effects of actions in the world. In contrast to HSTS, however, the modeling assumptions of most approaches [FHN72, Wil88] do not support explicit representation of temporal constraints depending on continuous time (e.g., task duration, temporal separation between events), and representation of the world state is not structured into state variables. More recent planning frameworks have only partially addressed these issues [Lan88, DFM88, Ver83]. Furthermore, in most cases, these frameworks have placed fairly rigid constraints on the manner in which solutions are developed (e.g., strict reliance on top down goal refinement with forward simulation[DFM88]), preventing an adequate consideration of efficient resource allocation over time, an issue of fundamental importance in the space-based observatory scheduling domain.

The monitoring of state variables over continuous time has always been at the core of scheduling research [Bak74]. Operations research has produced optimal solutions for very simple scheduling problems [Gra81, BS90] or has focused on the definition of dispatch priority rules [PI77] for more realistic problems. More recent research in constraint-based scheduling [SOM+90, Sad91], has demonstrated the advantages of dynamically focusing decision-making on the most critical decisions first. HSTS differs from other scheduling approaches in its temporal flexibility and in its ability to dynamically expand auxiliary goals and activities.

Issues in Integrating Planning and Scheduling

We now highlight some aspects of our approach that support the development of solutions for large scale scheduling problems in complex dynamical domains and, in particular, their relevance to space-based observatory domains.

Use of Abstraction

The use of abstract models has long been exploited as a device for managing the combinatorics of planning and scheduling. In HSTS, where models are expressed in terms of the interacting state variables of different components of the physical system and its operating environment, an abstract model is one which summarizes system dynamics in terms of more aggregate structural components or selectively simplifies the represented system dynamics through omission of one or more component state variables. Given the structure of space-based observatory scheduling problems, the use of an abstract model provides a natural basis for isolating overall optimization concerns, and thus providing global guidance in the development of detailed, executable schedules. In the case of the HST, a two-level model has proved sufficient. At the abstract level, telescope dynamics is summarized in terms of a single state variable, indicating, at any point in time, whether the telescope (as a whole) is taking a picture, undergoing reconfiguration, or sitting idle. The duration constraints associated with reconfiguration at this level are temporal estimates of the time required by the complex of actual reconfiguration activities implied by the detailed model (e.g., instrument warmup and cooldown, data communication, telescope repointing). Execution of an observation at the abstract level requires only satisfaction of this abstract reconfiguration constraint, target visibility (a non-controllable state variable accessible to both the abstract and detailed models), and any user specified temporal constraints. Thus, the description at the abstract level looks much like a classically formulated scheduling problem: a set of user requests that must be sequenced on a single resource subject to specified constraints and allocation objectives.

Planning relative to a full detailed level is necessary to ensure the viability of any sequencing decisions

made at the abstract level and to generate and coordinate required supporting system activities. The degree of coupling between reasoning at different levels depends in large part on the accuracy of the abstraction. In the case of HST, decision-making at abstract levels is tightly coupled; each time a new observation is inserted into the sequence at the abstract level, control passes to the detailed level and supporting detailed system behavior segments necessary to achieve this new goal are developed. Given the imprecision in the abstract model, goals posted for detailed planning cannot be rigidly constrained; instead preferences are specified (e.g., "execute as soon as possible after obs1"). The results of detailed planning at each step are propagated upward to provide more precise constraints for subsequent abstract level decision-making.

Model Decomposability and Incremental Scaling

Large problems are naturally approached by decomposing them into smaller sub-problems, solving the sub-problems separately and then assemble the sub-solutions. We can judge how the problem solving framework supports modularity and scalability by two criteria:

- the degree by which heuristics dealing with each sub-problem need to be modified when adding sub-problem assembly heuristics to the problem solver;
- the degree of increase of the computational effort needed to solve the problem versus the one needed to solve the component sub-problems

To test the scalability of the HSTS framework, we conducted experiments with three models of the HST operating environment of increasing complexity and realism, respectively denoted as SMALL, MEDIUM and LARGE model. All models share a representation of the telescope at the abstract level as a single state variable; they differ with respect to the number of components modeled at the detailed level. The SMALL model contains a state variable for the visibility of each of the celestial objects of interest with respect to the orbiting telescope, a state variable for the pointing state of the telescope, and three state variables for the state of an instrument, the Wide Field Planetary Camera (WFPC). The MEDIUM model adds two state variables for an additional instrument, the Faint Object Spectrograph (FOS), while the LARGE model includes eight additional state variables accounting for data communication. The LARGE model is representative of the major operating constraints of the domain. Figure 1 shows the relations among the various models.

The problem solver for the SMALL domain contains heuristics to deal with the interactions among the different components of the WFPC (e.g., when a WFPC detector is being turned on, make sure that the other WFPC detector is kept off), with the pointing of the HST (e.g., select a target visibility window to point the telescope), and with the interaction among WFPC state and target pointing (e.g., observe while the telescope is pointing at the proper target). The heuristics

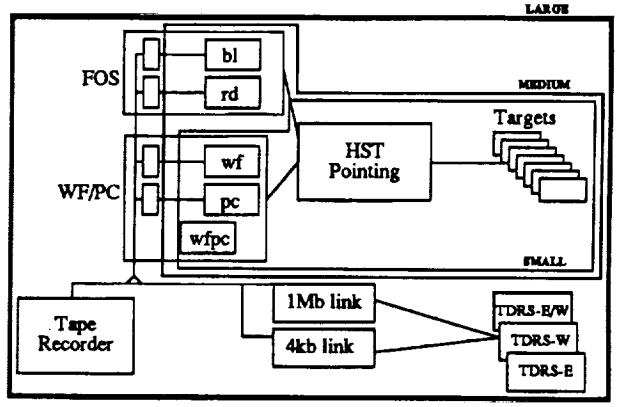


Figure 1: The SMALL, MEDIUM and LARGE HST models.

added for the MEDIUM domain deal with the interactions within the FOS, between FOS and HST pointing state, and between FOS and WFPC. Since the nature of the new interactions is very similar to those of the SMALL model, the additional heuristics are obtained by simply extending the domain of applicability of the SMALL's heuristics. Finally, for the LARGE model we have the heuristics used in the MEDIUM domain, with no change, plus heuristics that address data communication and interaction among instrument states and data communication (e.g., do not schedule an observation on an instrument if data from the previous observation has not yet been read out of its data buffer). The previous discussion supports the scalability with regard to the structure of the problem solvers.

To verify scalability with respect to the degree of computational effort, we run a test problem in the SMALL, MEDIUM and LARGE domain; the test consists of a set of 50 observation programs, each containing a single observation with no user-imposed time constraints. The experiments were run on a TI Explorer II+ with 16 Mbytes of RAM memory.

Table 1 supports the claim of scalability with respect to the required computational effort. The measure of the size of the model (number of state variables) excludes target and communication satellite visibilities since these can be considered as given data. The number of tokens indicates the total number of distinct state variable values that constitute the schedule. The temporal separation constraints are distance constraints that relate two time points on different state variables; their number gives an indication of the amount of synchronization needed to coordinate the evolution of the state variables in the schedule.

Notice that since the heuristics that guide the planning search exploit the modularity of the model and the locality of interactions, the average CPU time (excluding garbage collection) spent implementing each required compatibility constraint (corresponding to an atomic temporal relation among tokens) remains relatively stable. In particular, given the high similarity of the nature of the constraints between the SMALL and the MEDIUM models, this time is identical in the two

Model	SMALL	MEDIUM	LARGE
State Variables	4	6	13
Tokens	587	604	843
Time Points	588	605	716
Temporal Constraints	1296	1328	1474
CPU Time / Observation	11.62	12.25	21.74
CPU Time / Compatibility	0.29	0.29	0.33
Total CPU time	9:41.00	10:11.50	18:07.00
Total Elapsed Time	1:08:36.00	1:13:16.00	2:34:07.00
Schedule Horizon	41:37:20.00	54:25:46.00	52:44:41.00

Table 1: Performance results. The times are reported in hours, minutes, seconds and fraction of seconds

cases. The total elapsed time spent generating an executable schedule for the 50 observations is an acceptable fraction of the real time horizon covered by the schedules; this indicates the practicality of the framework in the actual HST operating environment.

Exploiting Opportunism to Generate Good Solutions

In the experiment just described, a simple dispatch-based strategy was used as a basis for overall sequence development: simulating forward in time at the abstract level, the candidate observation estimated to incur the minimum amount of wait time (due to HST reconfiguration and target visibility constraints) was repeatedly selected and added to the current sequence. This heuristic strategy, termed "nearest neighbor with look-ahead" (NNLA), attends directly to the global objective of maximizing the time spent collecting science data. However, maximization of science viewing time is not the only global allocation objective.

One critical tradeoff that must be made in space-based observatory scheduling is between maximizing the time spent collecting science data and satisfying absolute temporal constraints associated with specific user requests. The scheduling problem is typically over-subscribed; i.e., it will generally not be possible to accommodate all user requests in the current short term horizon and some must necessarily be rejected. Those requests whose user-imposed time windows fall inside the current scheduling horizon become lost opportunities if rejected. Those without such execution constraints may be reattempted in subsequent scheduling episodes.

As indicated above, the first objective (minimizing telescope dead time) is amenable to treatment within a forward simulation search framework. However, a forward simulation provides a fairly awkward framework for treating the second objective (minimizing rejection of absolutely constrained goals). A goal's execution window may be gone by the time it is judged to be the minimum dead time choice. Look-ahead search (i.e. evaluation of possible "next sequences" and potential rejections) can provide some protection against unnecessary goal rejection but the general effectiveness of this approach is limited by combinatorics. A second sequencing strategy of comparable computational complexity that directly attends to the objective of minimizing rejection of absolutely constrained goals

Sequencing Strategy	Pctg. Constrained Goals Scheduled	Pctg. Telescope Utilization
NNLA	72	21.59
MCF	93	17.20
MCF/NNLA	93	20.54

Table 2: Comparative Performance of NNLA, MCF and MCF/NNLA

is "most temporally constrained first" (MCF). Under this scheme, the sequence is built by repeatedly selecting and inserting the candidate goal that currently has the tightest execution bounds. This strategy requires movement away from simulation-based sequence building, since the temporal constraints associated with selected goals will lead to the creation of availability "holes" over the scheduling horizon. Adopting a sequence insertion heuristic that seeks to minimize dead-time can provide some secondary attention to this objective, but effectiveness here depends coincidently on the specific characteristics and distribution over the horizon of the initially placed goals. As is the case with the simulation-based NNLA strategy, one objective is emphasized at the expense of the other. This second MCF sequencing strategy, incidentally, is quite close to the algorithm currently employed in the operational system at STScI.

Both NNLA and MCF manage combinatorics by making specific problem decomposition assumptions and localizing search according to these decomposition perspectives. NNLA assumes an event based decomposition (considering only the immediate future) while MCF assumes that the problem is decomposable by degree of temporal constrainedness. Previous research in constraint-based scheduling[SOM+90] has indicated the leverage of dynamic problem decomposition selective use of local scheduling perspectives. In the case of NNLA and MCF, one aspect of current problem structure that provides a basis for selection at any point during sequence development is the current variance in the number of feasible start times remaining for individual unscheduled goals. If the variance is high, indicating that some remaining goals are much more constrained than others, then MCF can be used to emphasize placement of tightly constrained goals. If the variance is low, indicating similar temporal flexibility for all remaining unscheduled goals, then emphasis can switch to minimizing dead time within current availability "holes" using NNLA.

To test this multi-perspective approach, a set of short-term (i.e. daily) scheduling problems where solved with each base sequencing strategy and the composite strategy just described (referred to as MCF/NNLA). The results are given in Table 2 and confirm our expectations as to the limitations of both NNLA and MCF. We can also see that use of the opportunistic MCF/NNLA strategy produces schedules that more effectively balance the two competing objectives. Further details of the experimental design and the strategies tested may be found in [SP92].

These results should be viewed as demonstrative and we are not advocating MCF/NNLA as a final solution. We can profitably exploit other aspects of the current problem structure and employ other decomposition perspectives. For example, the distribution of goals over the horizon implied by imposed temporal constraints has proved to be a crucial guideline in other scheduling contexts [SOM+90, Sad91], and we are currently investigating the use of previously developed techniques for estimating resource contention [MS87, Mus92]. There are also additional scheduling criteria and preferences (e.g., priorities) in space-based observatory domains that are currently not accounted for.

Conclusions

In this paper, we have considered the solution of a specific class of complex scheduling problems that require a synthesis of resource allocation and goal expansion processes. These problem characteristics motivated the design of the HSTS framework, which we briefly outlined and contrasted with other scheduling and AI planning approaches. To illustrate the adequacy of the framework, we then examined its use in solving the HST short-term scheduling problem. We identified three key ingredients to the development of an effective, practical solution: flexible integration of decision-making at different levels of abstraction, use of domain structure to decompose the planning problem and facilitate incremental solution development/scaling, and opportunistic use of emergent problem structure to effectively balance conflicting scheduling objectives. The HSTS representation, temporal data base, and constraint-posting framework provide direct support for these mechanisms.

References

[Bak74] K.R. Baker. *Introduction to Sequencing and Scheduling.* John Wiley and Sons, New York, 1974.

[BS90] K. R. Baker and G.D. Scudder. Sequencing with earliness and tardiness penalties: a review. *Operations Research*, 38(1):22–36, January-February 1990.

[DFM88] T. Dean, R.J. Firby, and D. Miller. Hierarchical planning involving deadlines, travel time, and resources. *Computational Intelligence*, 4:381–398, 1988.

[FHN72] R.E. Fikes, P.E. Hart, and N.J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[Gra81] S.C. Graves. A review of production scheduling. *Operations Research*, 29(4):646–675, July-August 1981.

[Joh90] M.D. Johnston. Spike: Ai scheduling for nasa's hubble space telescope. In *Proceedings of the 6th IEEE Conference on Artificial Intelligence Applications*, pages 184–190, 1990.

[Lan88] A. Lansky. Localized event-based reasoning for multiagent domains. *Computational Intelligence*, 4:319–340, 1988.

[MS87] N. Muscettola and S.F. Smith. A probabilistic framework for resource-constrained multi-agent planning. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 1063–1066. Morgan Kaufmann, 1987.

[MSCD92] N. Muscettola, S.F. Smith, A. Cesta, and D. D'Aloisi. Coordinating space telescope operations in an integrated planning and scheduling architecture. *IEEE Control Systems Magazine*, 12(1), February 1992.

[Mus90] N. Muscettola. Planning the behavior of dynamical systems. Technical Report CMU-RI-TR-90-10, The Robotics Institute, Carnegie Mellon University, 1990.

[Mus92] N. Muscettola. Scheduling by iterative partition of bottleneck conflicts. Technical report, The Robotics Institute, Carnegie Mellon University, 1992.

[PI77] S.S. Panwalker and W. Iskander. A survey of scheduling rules. *Operations Research*, 25:45–61, 1977.

[Sad91] N. Sadeh. *Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling.* PhD thesis, School of Computer Science, Carnegie Mellon University, March 1991.

[SOM+90] S.F. Smith, J.Y. Ow, P.S. Potvin, N. Muscettola, , and D. Matthys. An integrated framework for generating and revising factory schedules. *Journal of the Operational Research Society*, 41(6):539–552, 1990.

[SP92] S.F Smith and D.K. Pathak. Balancing antagonistic time and resource utilization constraints in over-subscribed scheduling problems. In *Proceedings 8th IEEE Conference on AI Applications*, March 1992.

[Ver83] S. Vere. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5, 1983.

[Wal89] M. Waldrop. Will the hubble space telescope compute ? *Science*, 243:1437–1439, March 1989.

[Wil88] D.E. Wilkins. *Practical Planning*, volume 4. Morgan Kaufmann, 1988.

Extended Abstract: Managing Disjunction for Practical Temporal Reasoning

Mark Boddy† Bob Schrag Jim Carciofini
{schrag, boddy,carciofi}@src.honeywell.com
Honeywell Systems and Research Center, MN65-2100
3660 Technology Drive
Minneapolis, MN 55418

Abstract

One of the problems that must be dealt with in either a formal or implemented temporal reasoning system is the ambiguity arising from uncertain information. Lack of precise information about when events happen leads to uncertainty regarding the effects of those events. Incomplete information and nonmonotonic inference lead to situations where there is more than one set of possible inferences, even when there is no temporal uncertainty at all. In an implemented system, this ambiguity is a computational problem as well as a semantic one.

In this paper, we discuss some of the sources of this ambiguity, which we will treat as explicit *disjunction*, in the sense that ambiguous information can be interpreted as defining a set of possible inferences. We describe the application of three techniques for managing disjunction in an implementation of Dean's *Time Map Manager*. Briefly, the disjunction is either: removed by limiting the expressive power of the system, explicitly represented, one disjunct at a time, or approximated by a weaker form of representation that subsumes the disjunction. We use a combination of these methods to implement an expressive and efficient temporal reasoning engine that performs sound inference in accordance with a well-defined formal semantics.

1 Introduction

One of the problems that must be dealt with in either a formal or implemented temporal reasoning system is the disjunction arising from uncertain information. Lack of precise information about when events happen leads to uncertainty regarding the effects of those events, and thus to uncertainty in what propositions are true at some point in time. Incomplete information regarding what propositions are true when, and nonmonotonic inference (e.g., the persistence assumption or qualified causal projection) lead to situations where there is more than one set of possible inferences, even when there is no temporal uncertainty at all [6]. In a formal system, this ambiguity is noted and in some way dealt with, either by changing the semantics to exclude it (e.g. by assigning a preference relation to the possible models of a given theory), or simply by acknowledging it (i.e. couching conclusions in terms of the *set* of possible models).

In an implemented system, this ambiguity is a computational problem as well as a semantic one. In this paper, we discuss some of the sources of this ambiguity, which we will treat as explicit *disjunction*, in the sense that ambiguous information can be interpreted as defining a set of possible inferences. We describe how these sources of disjunction are dealt with in our current implementation of Dean's *Time Map Manager* [5; 2]. Briefly, we take one of three approaches:

1. The disjunction is removed by limiting the expressive power of the system.
2. The disjunction is explicitly treated, but the system considers only a single disjunct at a time.
3. The disjunction is approximated by a weaker form of representation that subsumes the disjunction.

The semantics that we are attempting to capture in our implementation are defined in [1], which provides a precise formal semantics for the current version of the TMM.

In the rest of this paper, we briefly discuss the ontology and semantics of the TMM, provide some specific examples of the kinds of disjunction that arise, and discuss the costs and benefits of various ways of handling these types of disjunction.

2 The TMM

Dean's *Time Map Manager* [5; 2] is an implemented temporal reasoning system, intended as a foundation for building planning and scheduling systems. The TMMincludes capabilities for reasoning about partially-ordered events, persistence and clipping, and two simple forms of causal reasoning: projection and temporal implication (sometimes called "overlap chaining" in previous work). The version of the system described in [5; 2] that was distributed from Brown (hereinafter referred to as "α-TMM") implements forward persistence only, and does not implement temporal implication.

Besides these limitations, the inference performed by α-TMM is not sound for partially-ordered time points [3], and so has no well-defined semantics. For partial orders, the inference done by the system is interpreted as quantification over total orders consistent with a given partial order: a formula of the form holds(t, P) is interpreted to mean that the proposition P holds at the time point t in all possible total orders. The sense in which the original system is unsound is that it will sometimes infer holds(t, P) when there were total orders in which P does not hold at t. As Dean and Boddy show in the same paper, reasoning about what is true in the total orders consistent with a given partial order is an NP-complete problem.

We have addressed these difficulties by implementing a sound but incomplete decision procedure that approximates quantification over time points (*i.e.*, if the system infers holds(t, P), the proposition P does in fact hold at the

†This work is supported by DARPA and the Air Force Rome Laboratory under Rome Laboratory contract F30602-90-C-0102.

time point t in every total order, but sometimes this property will be true and the system will not infer holds(t, P) [4]. We have made other extensions, including generalizing persistence to run backward as well as forward (in order to handle cases like Kautz's "parking lot problem." [7]), and implementing *temporal implication*: reasoning in which the truth of some set of facts at a point can be used to conclude that some other fact is true at the same point. We have retained from the old system the concepts of *persistence clipping* and *causal projection* (referred to hereinafter as simply "projection").[1] The new TMM implementation we will refer to as "β-TMM."

As far as we know, β-TMM is the first implementation of sound-and-incomplete temporal reasoning as described in [4]. The process of implementing this decision procedure has made clear precisely how the resulting system is incomplete; this point will be addressed in Section ??.

2.1 Ontology and Inference

In this section we present a simplified version of the TMM representations that is sufficient for this discussion. A *domain theory* in the language includes a *time map* and a *causal theory*. The time map consists of a set of *time points* $\mathcal{T}$ and a set of formulas. Time map formulas include the following:

- *Temporal relations* between time points, denoted by the binary infix predicates $<$, $\leq$, $=$, $\geq$, and $>$, and the predicate distance(t1, t2, *bounds*), where t1, t2 $\in \mathcal{T}$ and *bounds* = [r1 r2] where r1, r2 $\in \Re$ are the bounds of a closed interval. We represent temporal relations in the time map as *constraints*.
- *Temporal formulas*, holds(t1, t2, P), where t1, t2 $\in \mathcal{T}$ and P $\in \mathcal{P}$, the set of *propositions*. The period between t1 and t2 is called the "observation interval" (throughout which the proposition must necessarily hold.) We use the abbreviation holds(t, P) when this interval is a point. We represent temporal formulas on the time map using *time tokens*.
- *Persistence assumptions*, persists_f(t1, P) and persists_b(t2, P), where t1, t2, and P appear in some temporal formula as above. We associate persistence assumptions with time tokens on the time map.

The causal theory for a TMM theory includes *causal rules*, intended to encode the physics of a domain in a simple way, of the following kinds.

- *Projection rules*, project((and $\langle P_1, \ldots, P_k \rangle$), E, R). The propositions $P_1, \ldots, P_k$ are *antecedents*; E is a "trigger" proposition; R a forward-persistent "result" proposition. When the antecedent propositions are believed to hold throughout the trigger, the result is believed starting at a specified time after the trigger.
- *Temporal implication rules*, (and $\langle P_1, \ldots, P_k \rangle$) $\Rightarrow_t$ R. At any point for which the propositions of the antecedent conjunction are all believed to hold, the result proposition R is believed to hold.

[1] Details of extensions planned and accomplished can be obtained by request from Bob Schrag, at the address at the beginning of this paper.

The TMM implements an epistemic semantics, in the sense that a proposition may be known (or believed) to hold at a point, or known not to hold at that point, or we may not know either way. This semantics is described more carefully in [1]. The failure of the excluded middle in this semantics is useful for representing problems where we have only partial information. All of the propositions in the domain theory are believed necessarily. Temporal propositions are believed necessarily at all points throughout their observation intervals. Inference from projection and temporal implication result in the addition of new tokens to the time map, representing belief in propostions holding for new intervals of time. Persistence is captured in a preference over models: those in which the appropriate facts persist are preferred over those in which they don't. Conflicts in these preferences result in ambiguous situations, where no single set of inferences can be preferred to all others.

The theory including the time map and causal rules is intended to support the following kinds of inference.

- holds(t1, t2, P): P is true in all possible worlds.
- holds_m(t1, t2, P): P is true in some possible world.
- Inferences about necessary and possible temporal relations.
- Boolean combinations of these.

The first two kinds of inference concern belief in quantifications of temporal formulas over possible worlds consistent with the user-supplied domain theory. The simplest form of ambiguity in the domain theory that can lead to multiple possible worlds results from a set of temporal relations that defines only a partial order on the set of time points.

2.2 Sources of Disjunction

There are several sources of disjunction in the TMM. There is one source of disjunction we have explicitly removed: there is no way to assert an explicit disjunction in the domain theory. You can say that proposition P is true at time t, and that point t1 is ordered before point t2. You cannot, for example, say that t1 and t2 cannot occur simultaneously (*i.e.*, they are definitely ordered one way or the other).

This leaves us with two main classes of disjunction to deal with. The first is the temporal uncertainty resulting from the fact that we do not require time points to be totally ordered. Actually, there is additional metric uncertainty: we can specify the distance between two time points only as a range without that meaning that there is any uncertainty in ordering anywhere in the time map. Metric temporal uncertainty is straightforward to deal with. It affects no inference more complicated than directly determining whether a proposition holds at a point. Partially ordered points are a more complex problem because ordering affects which inference rules fire. For either projection or temporal implication, whether the rules fire is based solely on ordering relationships: all the possible assignments to temporal relations consistent with a given total order are equivalent, as far as which causal rules will "fire." For this source of disjunction, the "possible worlds" are the total orders consistent with the given partial order. Deciding

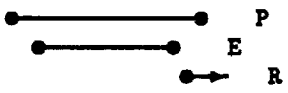
whether a proposition holds at a point necessarily, possibly, or not at all becomes a question of quantifying over the set of total orders. In Section 3, we discuss how this is accomplished (approximated, actually) in the TMM.

The other source of disjunction we must consider is a direct result of the semantics we impose on the system: the persistence assumption. Nonmonotonic reasoning has been recognized by many people at many times as a source of ambiguity and unintended conclusions (most relevant to our work is Hanks and McDermott's paper on applying nonmonotonic logic to temporal reasoning [6]). Unfortunately, it appears to be too useful to dispense with. Simply stated, the persistence assumption says that things tend not to change unless something changes them. If I walk into a room, see that the light is on, and walk out again, it seems both reasonable and useful to conclude that the light was on before I got there, and again after I left. [2] Contradictory information (*e.g.*, walking into the room at a later point and noticing that the light is off) will cause the system to draw different conclusions. The persistence assumption can lead to ambiguous conclusions in a wide variety of situations, a representative sampling of which are discussed in Section 4.

In the examples in the following sections, we represent time maps as follows: A time point is represented by a dot: ●. An observation interval is represented by two time points connected by a line: ●——●. Temporal ordering is from left to right, and all points are drawn with respect to a given frame of reference. When a time point is connected to a solid line, we know its relation with respect to the reference exactly. A dashed line as in ●- - -● indicates uncertainty about the point's location. Forward and backward persistence are represented by forward- and backward-pointing arrows: ←, →. We label tokens with the corresponding propositions and we label time points when we need to refer to them: ●. A lone timepoint with a proposition label is a zero-length observation interval: ● P. A single time point with a persistence symbol is a persistent version of the same thing: ●→ P.

To illustrate, here is a simple time map situation demonstrating the firing of a projection rule. Relevant textual information is displayed above the time map.

project(P, E, R)



3 Partial Orders

The problem with partially-ordered time maps is that inference such as projection and temporal implication depend on what facts hold at a given point. This relation is defined only for totally ordered points, and so we are reduced to determining what facts might possibly or necessarily hold at a point, in some or all of the total orders consistent with the given partial order. With even a very simple causal model, this is an NP-complete problem [4]. The solution we have implemented (first presented in [3]) is to approximate the necessary quantification.

β-TMM includes two holds definitions which together provide a sound-and-incomplete temporal reasoning algorithm which executes in polynomial time. Each definition approximates a quantification over the possible worlds consistent with the domain theory. $holds_S$ (*strong* holds) is a sound-and-incomplete approximation to holds. We use $holds_S$ to identify a *subset* of all necessarily believed temporal propositions. $holds_W$ (*weak* holds) is a complete-and-unsound approximation to $holds_m$. We use $holds_W$ to identify a *superset* of all possibly believed temporal formulas. In the presence of inference such as projection, the strong version requires the weak version: a proposition necessarily holds over an interval unless there is a possibly-derived token (the result of a projection rule, or added by the user), which possibly contradicts (clips) that proposition for some part of that interval.

$holds_S$ is *incomplete* in two ways:

- It avoids combinatorics by looking for a single token to span the query interval for *all* possible worlds. It will fail in a case where the interval is spanned by different tokens in different total orders.
- It relies, ultimately, on the over-achieving $holds_W$ to defeat the strong tokens' persistences.

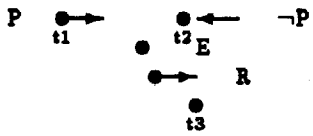
$holds_W$ is *unsound* in two ways:

- It avoids combinatorics by checking for a conjunction of possibilities rather than a possible conjunction. It succeeds sometimes when the conjuncts are not mutually satisfiable.
- It relies, ultimately, on the under-achieving $holds_S$ to defeat the weak tokens' persistences.

Some of these points are illustrated in the following examples.

Example 1: Incompleteness in $holds_S$ can arise directly from opposing contradictory persistences.

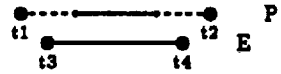
project(P, E, R)
project(¬P, E, R)



Our semantics says that the persistences for P and ¬P clip at some point between t1 and t2l, but not where. One of P or ¬P covers E in all total orders, so holds(t3, R). We are limited to $holds_W$(t3, R).

Example 2: Unsoundness in $holds_W$ can arise directly from partially ordered timepoints.

distance(t1, t2, 3)
distance(t3, t4, 5)



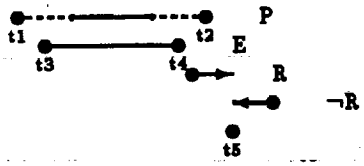
P does

[2] How "reasonable" persistence is, is context-dependent. Consider the same example where I see a cat sleeping on a chair, or a newspaper on a seat on a train.

not cover E in any possible world, so $\neg holds_m(t3, t4, P)$—but the conjunction of possible temporal relations in $holds_w(t3, t4, P)$ is satisfied, and it succeeds, unsoundly. Even though we do not have $(t1 \leq t3 \wedge_m t2 \geq t4)$, we do have $(t1 \leq_m t3 \wedge t2 \geq_m t4)$.

Example 3: Incompleteness in $holds_s$ can arise indirectly, through weakly and unsoundly derived defeaters.

distance(t1, t2, 3)
distance(t3, t4, 5)
project(P, E, R)



From Example 2 above, we know the token for R is weakly and unsoundly derived, and we should have holds(t7, ¬R). But ¬R is defeated weakly and unsoundly and we are limited to $holds_w(t7, \neg R)$.

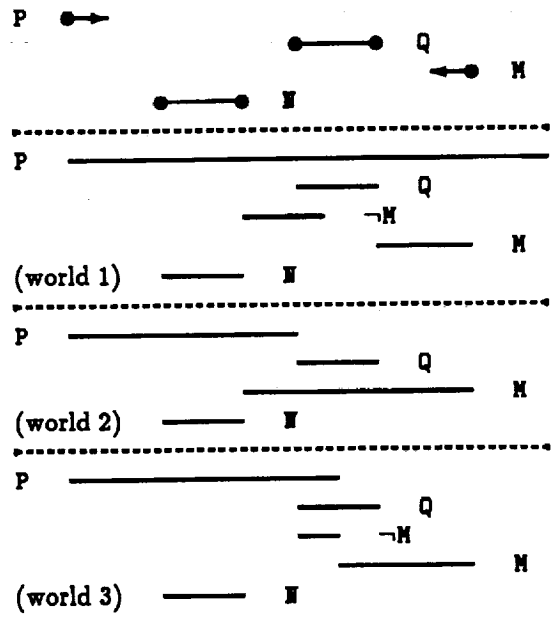
While strong inference ($holds_s$) is incomplete in a well-defined and limited sense (checking a single token), the approximate nature of weak inference ($holds_w$) is less precise. There are tradeoffs that can be made. For example, it is possible to add or omit a check on the maximum possible extent of a given token, rather than just the ordering of the endpoints. Adding such a check would result in a system that handled Example 2 correctly. At an additional computational expense, of course.

4 Ambiguous Models Resulting From Persistence

The persistence assumption combines with temporal implication or projection to generate situations in which there are several possible models for a given domain theory. In other words, we can construct theories in which P is true at some time T in some models (possible worlds) and false in others. These situations arise even if we limit ourselves to theories where all temporal relations are precisely specified for every point in the time map. In the following scenario, there are two temporal implication rules and four tokens specified in the domain theory (the dashed line on the right hand side separates a picture of the initial conditions from three different "possible worlds" corresponding to different models that can be constructed).

Example 4: Temporal implication with persistence can be ambiguous.

R1: (and P Q) $\Rightarrow_t$ ¬M
R2: (and M M) $\Rightarrow_t$ ¬P



In the first possible world (below the first dashed line) we maximize the extent of P's persistence. The result of the temporal implication rule R1, forces us to clip the persistence of M just after the end of Q. This world will be preferred to any world that is the same as this world except that P stops being true at some point after the end Q due to the persistence assumption: we prefer for P to persist as long as possible. Multiple models, and thus ambiguity or disjunction, result when there are several models none of which is preferred over any of the others. [3] There is a symmetric case, in which M's persistence is maximized. In the second model, the persistence assumptions for P and M are maximized with respect to each other. Neither of the rules come into play in this interpretation. They are maximal with respect to each other in the sense that if you extended either, the others' extent would be reduced. Finally, consider a case where P (or symmetricaly M) is allowed to persist to some point within the extent of Q (M). The third picture shows one of an infinite number of possible worlds that can be obtained in this way. In each of these worlds, the persistence of P and Q are maximized with respect to each other in the same sence as described above.

It is not difficult to come up with similar scenarios involving projection and backward persistence; or temporal implication and forward persistence. In fact fairly complex scenarios can be created using chains of projection rules, temporal implication rules, and persistence. There is an easily-identifiable condition of the causal theory that is necessary but not sufficient condition for theories to entail these kinds of ambiguities. Basically, we look for certain kinds of cycles using static analysis of the rules. Consider a DAG created from the rules as follows:

- Create a node in the DAG for each unique antecedent and consequent proposition
- For each rule create an arc from each antecedent node

[3] For a more careful discussion of the use of model preference to model persistence see *e.g.*, [8; 1]

to the consequent node
- For each consequent node create an arc to each contradictory antecedent

If any cycles exist in this DAG then our theory may entail the kind of non-monotonic disjunction describe above.

We have identified two approaches to implementing a practical system that deals with this kind of disjunction:

- Don't deal with it at all. Use the static rule analysis technique described above to reject rule sets that may entail this kind of disjunction.
- Use an approximation that is sound and incomplete. The idea is to be extremely conservative when looking for possible ambiguities. Any time there is a rule that may participate in a cycle of the sort described above, prohibit any backward persistence from being used as an antecedent.

Both approaches are rather heavy-handed: the analyze-and-complain approach leaves the user either without functionality or without predictability; both approaches overreact to prevent situations that may not occur, on the grounds that specific situation detection is too expensive. This will be a further source of incompleteness in the inference the system does. The complaining approach can be turned into a warning approach that goes on to do weak clipping.

5 Summary

In this paper, we have identified the sources of disjunction that must be considered in a temporal reasoning system that handles partially-ordered time points, forward and backward persistence, and two simple forms of causal reasoning. These sources can be grouped roughly into two classes, one corresponding to problems arising from temporal uncertainty (partial orders), the other the result of the nonmonotonic persistence assumption. There is actually a third source of disjunction that we have finessed by restricting the expressive power of the system: we do not permit the expression of explicit disjunctive propositions.

We have demonstrated three general classes of methods for dealing with disjunction, and proposed specific fixes for specific problems. Where possible, we have described implemented solutions from our work on the TMM. This paper presents the first clear characterization of the sources of incompleteness in the sound-and-incomplete decision procedure described in [4].

The techniques we have developed for managing disjunction are crucial to our implementation of an efficient temporal reasoning system. In particular, the representation of a set of disjunctions by some simpler description of a larger set including those disjunctions is a powerful technique that has found repeated use for handling disjunctions with a wide variety of sources and characteristics. With a little care, the resulting system retains the property of soundness, which we regard as crucial to the implementation of a useful system for temporal reasoning.

References

[1] Boddy, M.S., *et al.*, Semantics for Pratical Temporal Reasoning, *in preparation.*

[2] Dean, T.L., Temporal Imagery: An Approach to Reasoning about Time for Planning and Problem Solving, Yale University, University Microfilms 1986.

[3] Dean, T.L. and Boddy, M.S., Incremental Causal Reasoning, *Proceedings AAAI-87,* 196–201.

[4] Dean, T.L. and Boddy, M.S., Reasoning about Partially Ordered Events, *Artificial Intelligence* 36 (1988) 375–399.

[5] Dean, T.L. and McDermott, D.V., Temporal Data Base Management, *Artificial Intelligence* 32 (1987) 1–55.

[6] Hanks, S. and McDermott, D., Default Reaosning, Nonmonotonic Logics, and the Frame Problem, *Proceedings AAAI-86,* 328–333.

[7] Kauts, H., The Logic of Persistence, *Proceedings AAAI-86,* 401–405.

[8] Shoham, Y., *Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence* MIT Press, 1988.

Scheduling of an Aircraft Fleet

Massimo Paltrinieri (*) Alberto Momigliano (**) Franco Torquati

Bull HN Italia
Direzione Sistemi Esperti
Pregnana Milanese, Milano
Italia

Abstract

Scheduling is the task of assigning resources to operations. When the resources are mobile vehicles, they describe routes through the served stations. To emphasize such aspect, this problem is usually referred to as the routing problem. In particular, if vehicles are aircraft and stations are airports, the problem is known as aircraft routing. This paper describes the solution to such a problem developed in OMAR (Operative Management of Aircraft Routing), a system implemented by Bull HN for Alitalia. In our approach, aircraft routing is viewed as a Constraint Satisfaction Problem. The solving strategy combines network consistency and tree search techniques.

1. Introduction

Two of the main concerns for a major airline are flight planning and aircraft routing.

Flight planning involves both technical and market issues, such as the choice of the cities to be served and the weekly frequency of flights. It produces an aircraft rotation, valid for a whole season, which we shall refer to as the *virtual plan* (see fig. 1); it consists of a periodical time table where flights are organized in lines, one for each *virtual* aircraft, an hypothetical resource that could perform them in absence of technical and maintenance constraints.

Aircraft routing assignes tail numbers - the identifiers of the aircraft - to flights, usually for a time window of 24 hours. This process, called *predictive routing*, is trial and error: routes are drawn on the virtual plan, performing switches, i.e. connections between flights on different lines of the plan, to satisfy the constraints that prevent an aircraft to cover the next flight on the same line. When there are no more tasks available for the given aircraft, an assignment to an already scheduled task is possibly invalidated. If the scheduler is not able to cover all the activities with the available resources, maintenance are delayed or, in some extreme cases, flights are dalayed or even cancelled. The schedule produced by predictive routing is coded in the *routing plan*, which differs from the virtual plan in replacing virtual with actual aircraft and arranging programmed maintenance. The routing plan is often modified in real time to avoid or contain, propagation of delays. Such an activity is said *reactive routing*.

This paper describes the Prolog kernel of OMAR (Operative Management of Aircraft Routing), an interactive system designed to provide predictive and reactive routing of the Alitalia fleet. Routing is formulated as a Constraint Satisfaction Problem (CSP): each variable (task) has a domain of possible values (aircraft) while constraints (relations between variables) are used to restrict such domains. Since the refined domains are not in general single-valued, solutions must be found by search, iteratively selecting an aircraft and assigning it to a set of consecutive flights. Aircraft selection is driven by the first fail principle: the most constrained aircraft is scheduled first. A controlled form of backtracking is implemented to partially recover from heuristics flaws while maintaining predictable response time.

Present addresses:
(*) Stanford University - Department of Computer Science - Stanford, CA 94305 - palmas@cs.stanford.edu
(**) Carnegie Mellon University - Department of Philosophy - Pittsburgh, PA 15213 - am4e@andrew.cmu.edu

2. Problem Definition

In this section we give a formal definition of both predictive and reactive aircraft routing.

The constraints of the problem are captured by the function *label*, that associates to each task the set of aircraft that can perform it. The function $start_{qs}$ returns the airport from which an aircraft has to depart after time q_S, the start time of the scheduling window.

Predictive Routing

Input

set T of tasks
set AP of airports
set AC of aircraft
set Q of times
schedule start time q_s and schedule end time q_e
total order $\leq$ on $Q \cup \{q_s\} \cup \{q_e\}$ s.t. $\forall q \in Q$, $q_s \leq q \leq q_e$
total function departing time, dt: T -> Q
total function arrival time, at: T -> Q
total function departing airport, da: T -> AP
total function arrival airport, aa: T -> AP
total function label, label: T -> 2^{AC}
total function $start_{qs}$, $start_{qs}$: AC -> AP

Output

an *aircraft routing*, i.e a total function s: T -> AC, s.t.

(i) $\forall t \in T$, $s(t) \in label(t)$
(ii) if $s^{-1}(ac)$ is not empty, then its elements can be ordered in a sequence (the routing path of ac)

$r_{ac}=<t_{ac,0},t_{ac,1},...,t_{ac,n}>$ such that

$da(t_{ac,0}) = start_{qs}(ac)$
$aa(t_{ac,i-1}) = da(t_{ac,i})$ i=1,..,n
$at(t_{ac,i-1}) < dt(t_{ac,i})$ i=1,..,n

Reactive Routing

Input

aircraft routing as defined above
an unexpected event

Output

an aircraft routing that copes with the unexpected event and most closely conforms to the given routing.

3. Aircraft Routing as a Constraint Satisfaction Problem

A task is said *programmed* if its departure and arrival airports and times are fixed. Flights, as well as main maintenance, are programmed, whereas secondary maintenance not necessary. The duration of each task is a given constant. Let us assume that we have a set $T = \{T_h, h=1,...,m\}$ of programmed tasks to be scheduled in a time window of 24 hours.

Two tasks T_h and T_k are said to be *connectible* (denoted T_h -> T_k), if the following Prolog clause holds:

```
connectible(Th,Tk):-
        task_arrival_airport(Th,Airp),
        task_departure_airport(Tk,Airp),
        task_arrival_time(Th,MinArrT),
        task_departure_time(Tk,MaxDepT),
        ground_time(Airp,GrT),
        ArrT0 is MinArrT + GrT,
        ArrT0 < MaxDepT.
```

In other words, task T_h is connectible to task T_k iff the arrival airport of the former is equal to the departure airport of the latter and the arrival time of the former plus the ground time precedes the departure time of the latter. The graph of the connectibility relation is said the *connection graph*. It is directed and acyclic. Fig. 2 shows the connection graph for the portion of virtual plan in fig. 1.

We say that T_h *precedes* T_k and write $T_h < T_k$ iff (T_h,T_k) is in the transitive closure of ->. If neither $T_h < T_k$ nor $T_k < T_h$, then T_h and T_k are said *incompatible*, denoted T_h >/< T_k: incompatible tasks cannot be assigned to the same aircraft. A *routing path* P is a finite sequence of elements from T

$$P = <T_1, T_2,...,T_n>$$

such that T_h -> T_{h+1} for each h, $1 \leq h < n$. A path S is *operable* by aircraft Ac if each task in the path is operable by Ac, i.e. there are no technical reasons that forbid the assignment to Ac.

An initial state for the fleet is a one-to-one map from Acs, the set of aircraft in the fleet, to a subset of T, the set of programmed tasks. The image of Acs under such map is the set of *initial* tasks of T, which correspond to those nodes in the connection graph with no entering arcs. The set of *final* tasks is the set nodes in the connection graph with no exiting arcs. In the following, paths will have an initial task as first element of the sequence; the idea is that paths are the formalization of the routes that an individual aircraft may cover, starting from its initial state.

We look at the elements of T as variables which take their values from the domain Acs. As already mentioned, a label of a task is the set of aircraft that can perform it. This concept can be extended to the set of all tasks: the *labeling* of the set T is a map l : T -> P(Acs), where P(Acs) is the powerset of Acs.

Constraints are relations in Acs x P(T) that are used to refine the labels of tasks. They come in two types: a *commitment* constraint between aircraft Ac and tasks $T_1,...,T_n$ requires that Ac executes at least one of those tasks; an *exclusion* constraint between an aircraft Ac and and tasks $T_1,...,T_n$ requires for Ac to be excluded from those tasks.

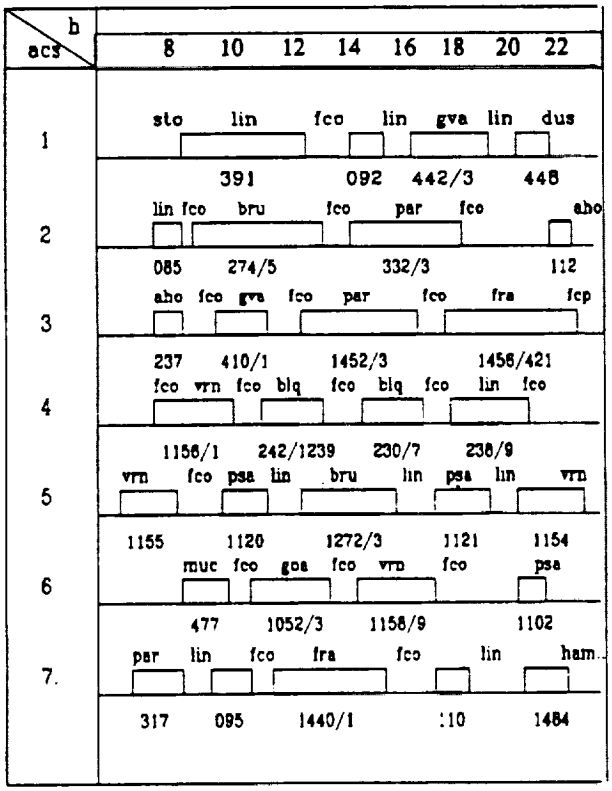


Fig. 1. A portion of about one-fourth of the virtual plan for the DC-9 fleet.

Each singleton labeling that satisfies all the constraints is an *aircraft routing*, i.e. a solution to the routing problem formalized in sect. 2. Such a singleton labeling generates a partition of the set T of tasks such that each element of the partition is a *routing path* for a distinct aircraft.

4. Routing Process

The routing process implemented in OMAR starts loading the state of the fleet and the relevant information on the tasks to be scheduled from the Alitalia database. A necessary, but not sufficient, condition for the existance of a fleet routing is checked, namely whether the number of resources available to be assigned to each task is always greater than or equal to zero. We briefly describe the algorithm, linear in the number of tasks, that tests such condition.

Each airport airport served by the fleet identifies a sequence of chronologically ordered events belonging to one of two classes: departures or arrivals. Each task entails two events, its arrival and departure, unless it is initial, in which case we consider only the arrival. A resource counter representing, at each time, the balance between arrivals and departures, is associated at every airport. The resource counter is initially set to 0 and is incremented or decremented, at each flight arrival or flight departure, respectively. If, scanning the whole plan, the counter of some airport becomes negative, the necessary condition is not satisfied and no routing exists. On the other hand, if the counters are always grater than or equal to zero, then the condition is satisfied and the system enters its next stage.

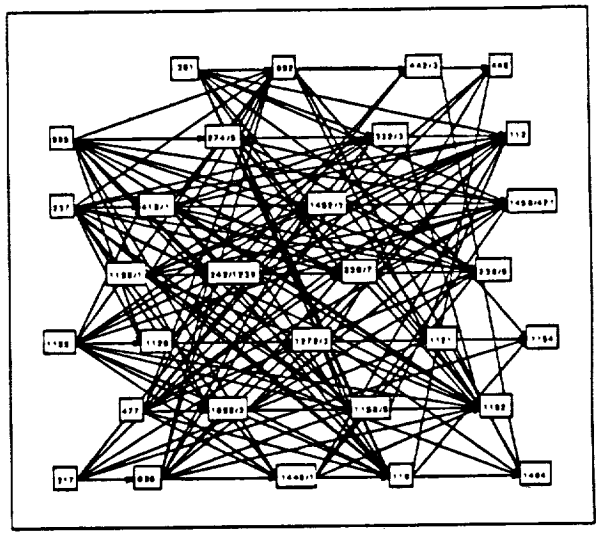


Fig. 2. The connection graph for the virtual plan in fig. 1.

A sample list of events at Linate airport is shown below.

Time	Event	Flight	Resource Level
17:50+0	d	448	0
17:25+35	a	267	1
17:45+35	a	074	2
18:30+0	d	316	1

Observe that the arrival of flight 267 at 17:25, given the ground time of 35 minutes, follows the departure of the flight 448 at 17:50.

The constraint satisfaction algorithm refines the labels so that most dead-ends are avoided and expiry maintenance requirements are implicitly satisfied: this means that aircraft planned for the latter tasks are excluded by those routes that do not lead to the set of airports where maintenance jobs are possible.

If the network is not found consistent, no complete routing exists and the control goes to the human scheduler who relaxes the constraints. It is our opinion that this kind of expertise cannot be adequately simulated by a computer, since the knowledge required to recognize the causes of an inconsistent situation and suggest a solution is too extended and fuzzy. If, on the other hand, everything is succesfull, the system is ready to schedule.

The aircraft are sorted in decreasing order according to the number of occurrences inside the labeling; the idea is that the aircraft coming first in this order are the most constrained ones, since they have a smaller number of tasks on which they can be enrouted. Routes are then created according to such an order by the Prolog procedures sketched below.

```
route_gen([Ac|Acs],Lab,NewLab):-
        pathgen(Ac,Lab,TmpLab),
        !,
        route_gen(Acs,TmpLab,NewLab).
route_gen([],Lab,Lab).

path_gen(Ac,Lab,NewLab):-
        last_started(Ac,Task),
        path_gen(Ac,Task,Lab,NewLab).

path_gen(Ac,Task,Lab,NewLab):-
        select(Ac,Task,Lab,NextTask,TmpLab),
        path_gen(Ac,NextTask,TmpLab,NewLab).
path_gen(_Ac,_Task,Lab,Lab).
```

The recursive procedure *route_gen/3* terminates when the list of aircraft to be scheduled is empty. It searches for a solution in depth-first mode, generating a descendant of the most recently expanded node and backtracking if some dead end is reached. If we relied exclusively on backtracking, the process duration would be unpredictable. Fortunately, we have developed some criteria that help us to discard paths likely to fail. On each aircraft *Ac*, *route_gen/3* calls *pathgen/3*, passing as parameters the aircraft *Ac* and the labeling *Lab* and returning a new labeling *TmpLab* in which the tasks assigned to *Ac* are the generated path. The procedure *pathgen/4* builds a path recursively, task after task, starting from the first one returned by *last_started/2*.

A limited amount of backtracking is allowed: different choices are considered only during the coupling of a task with one of its direct offsprings. Yet paths cannot be invalidated after its completion (note the use of the cut sign '!' after *pathgen/3*). In case of failure, the interaction with the user is more effective. In our experience, after the relevant modifications have been performed, another run of the scheduler is usually sufficient to achieve a complete solution.

Let us analize the path generation process in more detail. The problem is not trivial, since there are both local and global optimizations which influence the choice at various extents, often in opposite directions. For instance, we could always choose the first task departing after the given one (local optimization), but this could generate a new line switch hard to manage in the overall routing (global optimization).

```
select(Ac,Task,Lab,NextTask,NewLab) :-
        propose(Ac,Task,Lab,NextTask),
        check_rc(Task,NextTask),
        update_lab(Ac,NextTask,Lab,NewLab).

propose(Ac,Task,Lab,NextTask):-
        get_methods(Ac,Task,Methods),
        member(Method,Methods),
        offsprings(Task,Offs),
        choose(Method,Ac,Offs,Task,NextTask).

get_method(Ac,Task,Methods):-
        rule(Condition,Methods),
        apply(Condition,Ac,Task).

rule(open_switch, [close_switch,straight,closest,stop]).

rule(default, [straight,open_switch,closest,stop]).
```

The basic step of the path generation process is performed by the Prolog procedure *select/5* shown above. Given an aircraft *Ac*, just assigned to a flight or maintenance (*Task*), *select/5* extends the path of *Ac* to a new flight or maintenance (*NextTask*). The procedure *propose/4* returns *Nextask*, then *check_rc/2* checks whether the resource counter becomes negative: in such a case it fails, otherwise it succeedes and the labeling is updated, aircraft *Ac* being assigned to *NextTask*. The path of *Ac* is extended with *NexTask* by *propose/4* as follows: first, a list *Methods* of methods compatible with *Ac* and *Task* is selected by *get_methods/3*; then, one *Method* is chosen nondeterministically from such a list; after, the offsprings of *Task* in the connection graph are returned by *offsprings/2* and finally, one of them, *NextTask*, is returned by *choose/5*, which basically applies *Method* to the given *Ac* and *Task*.

A method is a technique to choose the next task that extends a given path. Methods are gathered in lists and are associated to conditions. The relation between conditions and lists of methods is defined by *rule/2*. Two sample rules are shown above for the *open_switch* (remember that an aircraft opens a switch when its path is extended on a different row) and the *default* conditions. Given *Ac* and *Task*, if a condition is applicable to *Ac* and *Task*, which is checked by *apply/3*, a list of methods is returned by *get_methods/3*. Such methods are tried in the same order as they appear in the *Methods* list, the first one being the most desirable. For any possible *Ac* and *Task* there is at least one rule whose condition is satisfied, thus a list of methods is always selected, eventually by the *default* rule. In such a case, the list of methods tries to extend the path on the same line of the virtual plan with the *straight* method, which is considered optimal, otherwise a switch is opened by *open_switch*; if it is not possible to open a switch, the closest flight is selected by *closest* to minimize the consumption of the resources; if even this method is not applicable, the path is terminated by *stop*.

5. Conclusions

Aircraft routing is a problem for which no exact solution is known. Consequently, all models are heuristic and research is now concentrating on the systematic interaction between human and computer.

OMAR is an interactive system for the routing of the Alitalia fleet. Its kernel is presently composed of 20,000 lines of Quintus Prolog source code, and the system's response time is satisfactory. Once the derived structures have been computed from the primary database, the fleet routing is returned nearly in constant time (approximatively 30 seconds for a fleet of 26 aircraft with 170 flights).

Moreover, if the constraints are compatible with complete schedules, there is a very high probability that the system succeeds finding one of them. Of course, we cannot expect that the solution perfectly matches the user's expectations. According to our experience, however, an intervention by the user modifying, on average, five assignments, is suffucuent to reach such an accomplishment.

In the tests supplied by Alitalia so far, OMAR's solutions can be compared with those of a senior scheduler.

References

[Da] Davis E., Constraint Propagation with Interval Labels, *Artificial Intelligence*, 32, 1987, 281-331.

[De&Pe] Dechter R. & Pearl J., Network-Based Heuristics for Constraint Satisfaction Problems, *Artificial Intelligence*, 34, 1988, 1-38.

[Et&Ma] Etschmeier M.M. & Mathaisel D.F.X., Aircraft Scheduling: the State of the Art, *XXIV AGIFORS Symposium*, Strassbourg, 1984,181-225.

[Ha&El] Haralick R.M. & Elliot G.L., Increasing Tree Search Efficiency for Constraint Satisfaction Problems, *Artificial Intelligence*, 14, 1980, 263-313.

[Na] Nadel B.A., Tree Search and Arc Consistency in Constraint Satisfaction Problems, in Kanal & Kumar (eds), *Search in Artificial Intelligence*, Springer-Verlag, 1988.

[Ri] Richter H., Optimal Aircraft Rotations based on Optimal Flight Timing, *VIII AGIFORS Symposium*,1968,34-69.

[Ste&Sha] Sterling L. & Shapiro E., *The Art of Prolog Programming*, MIT Press, Cambridge, Massachussets, 1986.

[Wal] Waltz D., Understanding Line Drawings of Scenes with Shadows, in *The Psychology of Computer Vision*, edited by P. H. Winston, McGraw-Hill Company, 1975.

Adaptive Planning For Applications With Dynamic Objectives

Khosrow Hadavi
Wen-Ling Hsu
Siemens Corporate Research

Michael Pinedo
Columbia University

Abstract

Planning is commonly viewed as a task to devise a course of action or a *plan* that conforms as much as possible to a set of goals before acting. The plan will then be used to guide the activities. Most classic planning systems assume a static environment for the planning agents. In a static environment, states remain unchanged between actions, and the outcomes of actions are assumed to be deterministic. In reality, however, most applications are dynamic and stochastic in nature. External events, not caused by controlled actions, may occur; outcomes of actions may differ from expectations; new constraints may be introduced; and a new set of goals may evolve in response to the changes. Recently, we have proposed a multi-modal framework for adaptive planning in a *dynamic* environment with multiple objectives having the following characteristics:

- some of the objectives of the planning process may be conflicting
- some objectives may be ill-defined or difficult to measure quantitatively
- the objectives may change over time

The task domain of production planning and scheduling is a typical example of such an environment. The scheduling objectives typically include the following: meeting due dates; reducing lead times; reducing work-in-process and finished goods inventories; maximizing resource utilization and the throughput of the system; and minimizing the sensitivity of the schedule to random events. These objectives are sometimes in conflict with each other. In our previous work, we developed a real-time distributed scheduling system [1] that observes its environment from different perspectives. These perspectives stem from the different objectives, and the system can react to events as they occur while monitoring the various objectives. This multi-perspective monitoring helps our system achieve better control of the environment. During our study, we discovered that although these global objectives may not change over time, the relevance of each objective is actually a function of time and the state of the system. For example, given a set of N objectives O_1, O_2, ... O_n, at time t_1, objective O_2 may be significantly more important than O_1, whereas at another instance

[1] For a detailed description of the system, please read the attached paper titled "An Architecture for Real Time Distributed Scheduling" to appear in "Applications of AI in Manufacturing," published by AAAI Press, edited by Dana S. Nau.

of time t_2, objective O_1 may become most important. Furthermore, each heuristic implies a set of reactive strategies that move the system toward some objectives but *away* from other objectives (due to the conflicting nature of these objectives).

In our current research, we devise a qualitative control layer to be integrated into a real-time multi-agent reactive planner. The reactive planning system consists of distributed planning agents attending to various perspectives of the task environment. Each perspective corresponds to an objective. The set of objectives considered are sometimes in conflict with each other. Each agent receives information about events as they occur, and a set of actions based on heuristics can be taken by the agents. Within the qualitative control scheme, we use a set of *qualitative feature vectors* to describe the effects of applying actions. A *qualitative transition vector* is used to denote the qualitative distance between the current state and the target state. Given a target state and a set of heuristics, we have an algorithm to test the *reachability* of the target state. We will then apply on-line learning at the qualitative control level to achieve adaptive planning. Our goal is to design a mechanism to refine the heuristics used by the reactive planner every time an action is taken toward achieving the objectives, using feedback from the results of the actions. When the outcome is compared with expectations, our prior objectives may be modified and a new set of objectives (or a new assessment of the relative importance of the different objectives) can be introduced. Because we are able to obtain better estimates of the time-varying objectives, the reactive strategies can be improved and better prediction can be achieved.

Global planning of several plants

Sylvie Bescos

BIM sa/nv
Kwikstraat,4
B-3078 Everberg (Belgium)
e-mail: sb@sunbim.be

Abstract

This paper discusses an attempt to solve the problem of planning several pharmaceutical plants at a global level. The interest in planning at this level is to increase the global control over the production process, to improve its overall efficiency and to reduce the need for interaction between production plants. In order to reduce the complexity of this problem and to make it tractable, some abstractions have been made. Based on these abstractions, a prototype is being developed within the framework of the EUREKA project PROTOS, using Constraint Logic Programming techniques.

Introduction

This paper describes the development of a prototype "global planning tool" within the framework of the EUREKA project PROTOS [PROTOS90]. PROTOS aims at the application of Prolog-based techniques to real-life planning and scheduling problems. The problem addressed by this prototype was proposed by one of the PROTOS partners, which is a large swiss pharmaceutical company.

The whole production of this company is split over several plants. The aim is to compute a global production plan for all these plants. Up to now, there is no such global plan, and all the coordination and adjustments of the production process between the different plants is achieved through phone calls between plant managers; there is no global control. This scheme works because of the experience and know-how of the plant managers, but the result is far from optimal.

If a good global plan could be provided, ensuring that no major coordination problem should occur, then each plant could make local optimisations as long as the constraints imposed by the global plan are respected; also the resulting production process would become much closer to optimality. As a side effect, this global plan would also reduce the need for the phone call based coordination, although it is not expected to suppress it totally.

As it is far too complex to take into account all details of the local data of each individual plant, the considered global planning tool is based on an approximation of the local reality. Thus, the output of this tool is only a "rough" global plan, that will then be further refined at each plant, by the local scheduling tool (in this case a job-shop scheduling tool).

The implementation tool chosen was the Prolog III system [Col90], in order to take advantage of the recent advances in the Constraint Logic Programming field [Coh90, VH89].

1 Problem description

Scheduling problems are known to become quickly intractable, because of combinatorial explosion. This gets even worse when trying to compute a global plan for several plants, as it is practically impossible to consider all details of each plant. This problem has to be simplified somehow.

The work described here is based on one approximation of the local reality, which is the abstraction of individual machines in *machine groups*.

In order to define a machine group, some terms have to be introduced:

- the word "product" designates both intermediate and finished products.
- several production steps are needed to go from one or several intermediates to the product of the next upper level; all these steps are grouped in a single "production order".

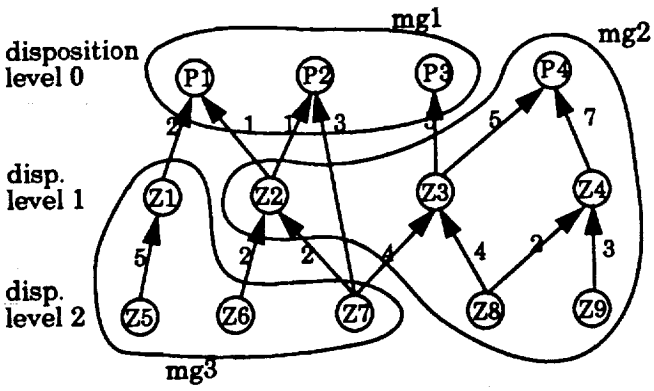
A machine group is a set of machines located physically close to each other, and each order can be completely executed using only machines within one machine group.

Also, at the global planning level, the different production steps of one order are abstracted in only one production task. Thus, one order is considered as being one task using one resource.

The global planning tool takes as input:

1. demands for finished products, a demand[1] being a pair (amount, due date),
2. the allocation of machine groups to products (each product is considered as being always produced on the same machine group),
3. *the dispositive bill of materials*:

   E.g. dispositive bill of materials with machine group allocation to products:



   Legend:

   * circles are products,
   * an arrow from A to B means that product A is an input to the production of B,
   * a number *n* near an arrow between A and B means that *n* units of product A are needed to produce 1 unit of product B,
   * shapes round products represent machine group allocation: e.g., products P1,P2 and P3 will be produced on machine group mg1.

4. stock data,
5. eventually, existing machine group allocations to some orders.

The requirement is to generate time windows for all finished and intermediate products appearing in the dispositive bill of materials, from the demands of finished products. For this, a convenient sequence for the production of the required finished and intermediate products has to be found.

The prototype has to perform *backward scheduling* where planning starts from the finished products and the allocations are made as late as possible. Backward scheduling in this way tends to minimize stocks.

While it is hoped that a conflict-free solution can be found in most cases, this might not always be the case because of the abstractions/approximations made. When no conflict-free solution exists, the global planning tool has to generate the best imperfect solution (i.e. featuring some conflicts on resource allocations). This best imperfect solution can then be used at the local scheduling level, which still has some flexibility that does not appear at the global planning level, and which could possibly solve conflicts.

The complexity of the problem not only comes from the number of demands to plan, but also from the handling of stocks and residuals:

- stocks may be available at the beginning of the planning period;
- additional stocks are likely to be generated during the production process because of some production constraints: it is not possible to produce less than a minimum quantity of a product at once (minimal lot size);
- residuals can be regenerated during the production process: e.g. the production of Z3 regenerates a certain amount of Z7, that could be used as input for the next demand of Z3 (not shown on the dispositive bill of materials drawn above).

This results in a "chicken and egg" problem:

- to find a sequence between the production tasks, it is needed to know the amounts to be produced, as the duration of a production task depends on the amount to be produced;
- the amounts to be produced depend on stocks, and the stocks evolve with time during the planning period depending on the chosen sequence of production.

2 Cutting the complexity

2.1 Decomposition of the planning horizon into sub-periods

To solve this "chicken and egg" problem, a further approximation was introduced in the planning process model. This approximation divides the planning horizon into several "sub-periods". This means that stocks are taken into account as if they were available only at the frontiers between these sub-periods.

1. In the following, "order", "production task", "demand" will be used indiscriminately.

In this way, it is still possible that more is produced during a sub-period than is strictly necessary: some stocks created during this sub-period (because of minimal lot size constraints) could have been used to reduce some demands for the same products occurring later in the same sub-period. However these stocks are likely to be used during the next sub-period, as a particular product is often produced again several times in the year[1]. Thus stock levels over the whole planning horizon should remain relatively stable.

It is not necessary to actually perform the planning of a sub-period in order to know how much stocks will be available at the end: all the demands in this sub-period will be produced, so it is not needed to know the exact sequence to compute the global result in terms of stocks available at the end.

It is then sufficient to:

- group the demands into sub-periods, according to their due dates;
- rearrange the demands, within each sub-period, taking into account stocks available at the end of the preceding sub-period, and compute the new stock levels at the end of the current sub-period.

This process is repeated for each disposition level in turn, starting with level 0 (i.e. finished products). The reason for starting with disposition level 0 is simply that initially, there are only demands for finished products, from which demands for intermediate products have to be successively derived.

2.2 Decomposition of the problem according to machine groups

The planning problem consists in making choices about a sequence and precise dates for all the demands to be produced. This search space is too wide to expect reasonable computation times. It is then needed to decompose the search space into several sub-spaces that can be treated independently.

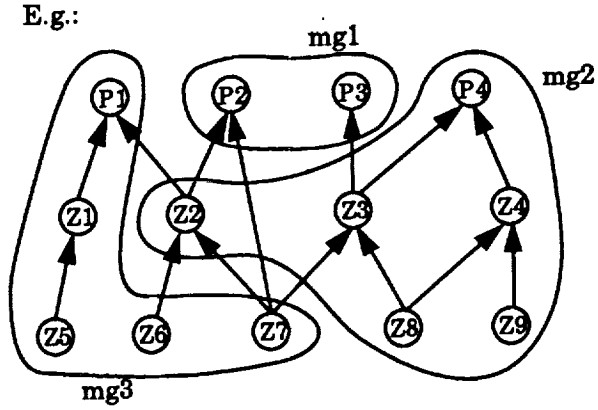
The machine groups serves as a basis for this decomposition:

- the list of machine groups is ordered according to dependency links, to obtain a so-called *machine group graph* (this more or less reflects the fact that a machine group is allocated to lower or higher levels of the dispositive bill of materials),
- for each machine group in turn:
  - * a sequence and particular dates for the tasks are chosen;
  - * these choices are committed;
  - * due dates and earliest beginning dates[2] for tasks allocated to the remaining machine groups are propagated.

2.3 Cycles in the machine group graph

There is a cycle in the machine group graph when, between two production tasks that are allocated to the same machine group, there exists one or several intermediate production tasks to be performed on other machine groups. According to the experts of the pharmaceutical company, this is unusual, and it is acceptable in such cases if the result is not as good.

E.g.:



In this example, there is a cycle between mg2 and mg3, because of the links between P1 and Z2, Z2 and Z6, and Z2 and Z7. If mg2 is treated before mg3, the demand on Z2 coming from that on P1 will be planned as late as possible with respect to the precedence constraints, which will eventually result in no freedom being left for the demand of P1. If mg3 is treated before mg2, then the demands on Z2 and even Z3 will eventually be too constrained.

Such cycles must be cut. The minimum number of links in the dispositive bill of materials that have to be cut in order to eliminate the cycle are marked (in the above example, the link Z2 → P1 is cut rather than the links Z6 → Z2 and Z7 → Z2). When there is a dependency between two production tasks along one of these links, these tasks are

1. Regulations require a pharmaceutical company to have several years of stocks, so external demands are not customer-driven. For most finished products, the yearly demand is split into several ones with due dates distributed over the year.

2. The earliest beginning date of a demand is the date when all input products are available.

further constrained so that the planning freedom is equally shared out among these tasks.

3 The program

The program has been implemented using Prolog III, a prolog interpreter with integrated constraints over rationals, booleans, and lists.

The basic algorithm is:

- first the machine group graph is computed from the dispositive bill of materials and the machine group allocation to products;
- then the data structure, which is a network of demands linked by constraints, is constructed;
- a schedule is computed;
- finally, the resulting plan is shown in a graphical form.

The construction of the data structure and the planning process will now be described:

Data structure

The data structure is a list of demands/orders represented each by a term:

```
[ id, product, machine group, due date,
  duration, end date, dependency info ]
```

It is constructed starting from the highest level of the dispositive bill of materials (i.e. finished products) going to the lower levels. At each level, for each product:

1. demands are grouped into sub-periods according to their due dates;
2. for each of these sub-periods in turn:
   a. demands are rearranged according to minimal lot sizes constraints, residuals and stocks available at the beginning;
   b. stocks that will be available at the end are computed;
3. from all these rearranged demands (over the whole planning horizon), demands for intermediate products are derived, and data about residuals is updated.

During the construction of this data structure, several kinds of constraints are enforced:

- precedence constraints,
- stocks availability constraints:
  - stocks of a product are considered to be available only after the end of the last allocation for this product during the preceding sub-period.
  - a demand that will take some amount of an input product from stocks is constrained to begin later than this date.
- residuals availability constraints: the use of residuals is allowed only if the demand is already constrained to begin later than the end date of the residuals production.

Planning, making choices

Even after decomposing the problem according to machine groups, the search space still needs to be reduced in order to make the program reasonably efficient. As it seems sensible to treat together demands that are close in time, sub-periods will be introduced again here. Choices will be committed after planning each sub-period.

However this decomposition into sub-periods implies some additional constraints. In order to express these constraints, it is needed to define the "planning limit" for a machine group as the latest end date of all allocations of this machine group for the demands of the previous sub-period. The additional constraints are that no allocation for the current sub-period can be made before this planning limit (to reduce the complexity, otherwise it would be needed to check disjunction with allocations of preceding sub-periods).

For each machine group in turn (starting from the machine groups allocated to the higher levels of the dispositive bill of materials):

- the demands are grouped into sub-periods, according to their due dates;
- for each sub-period:
  - if it is possible to find a conflict-free sequence, a maximisation of the minimum of all end dates is performed (so that the whole set of production tasks is planned the latest as possible);
  - if no conflict-free solution exists, conflicts are progressively allowed but minimised. This minimisation has to be based on a conflict evaluation. However, finding a convenient cost function of conflicts is a problem in itself, and one of the objectives of this prototype is to experiment with different ones. Up to now, the implemented measure is simply a count of the number of days in overlaps.

These optimisations are local to one machine group during one sub-period because a global opti-

misation would be too expensive in computation time.

The resulting plan contains precise dates for each production task instead of just time windows, as was requested at the beginning. In fact, this result can be viewed as a particular "fully instantiated" solution of the problem. In order to leave some freedom to the local plants, a more general solution could be retrieved, by deducing time windows from these precise dates and from the dependency information which was kept in the data structure.

4 Computational results

Two versions of the program exist:

- a coarse one for getting a rough idea of the resulting plan quality allowed by a given machine group allocation,
- a finer (but slower) one for getting the best possible plan for a given machine group allocation.

There is currently a dearth of representative examples (the extraction of the machine group information from the detailed description of each plant is still an open problem being tackled by people from the pharmaceutical company), and so no figures are yet available.

However, what has been learned from the development of the current prototype is the adequacy of the CLP approach for prototyping. The CLP approach allowed a switch from one version of the algorithm to alternative ones in a very short time, because of the declarativity and expressiveness of CLP languages.

Conclusion

The validation of the approach described in this paper can only come from the experimental use of this prototype together with several instances of a local plant scheduling tool, in order to check whether feasible plans are obtained. Such experiments have not yet been possible because of the difficulty in extracting the machine group information from the detailed data.

Up to now, the main interest in this work has been the refinement of the approach during discussions with experts from the pharmaceutical company. These discussions were based on hypothetical examples and on the successive versions of the program which have lead to the one presented here.

When representative examples become available, this research will go on by using this prototype to experiment with different evaluation functions of conflicts, and to investigate about the validation of the resulting plan.

Acknowledgements

I would like to thank all PROTOS partners for many fruitful discussions. I would also like to thank Pierre-Joseph Gailly and Paul Massey for their helpful comments on earlier drafts of this paper. Partial funding for this work was provided by the ESPRIT II project 5246 PRINCE (PRolog INtegrated with Constraints and Environment for industrial and financial applications).

Bibliography

[PROTOS90] The EUREKA Project PROTOS. H.-J. Appelrath, A. B. Cremers, and O. Herzog Ed., Zurich, Switzerland, April 9, 1990.

[Coh90] Jacques Cohen. *Constraint Logic Programming Languages*. Communications of the ACM, Vol.33, No.7, July 1990.

[Col90] Alain Colmerauer. *An Introduction to Prolog III*. Communications of the ACM, Vol.33, No.7, July 1990.

[VH89] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. Logic Programming Series, The MIT Press, Cambridge, MA, 1989.

The MICRO-BOSS Scheduling System: Current Status and Future Efforts

Norman M. Sadeh

Center for Integrated Manufacturing Decision Systems
The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 - U.S.A.

sadeh@ri.cmu.edu

1. INTRODUCTION

[1]Over the past few years, several approaches to scheduling have been proposed that attempt to reduce tardiness and inventory costs by *opportunistically* (i.e. dynamically) combining a resource-centered perspective to schedule bottleneck resources, and a job-centered perspective to schedule non-bottleneck operations on a job by job basis. Rather than relying on their initial bottleneck analysis, these schedulers reexamine the problem each time a resource or a job has been scheduled. This enables them to detect the emergence of new bottlenecks during the construction of the schedule. This ability has been termed *opportunistic scheduling* [3]. Nevertheless, the opportunism in these systems remained limited, as they required scheduling large resource-subproblems or large job-subproblems before allowing for a change in the scheduling perspective (i.e. before permitting a revision in the current scheduling strategy). For this reason, we actually refer to these approaches as *macro-opportunistic* techniques.

In reality, bottlenecks do not necessarily span over the entire scheduling horizon. Moreover they tend to shift before being entirely scheduled. A scheduler that can only schedule large resource subproblems will not be able to take advantage of these considerations. Often it will overconstrain its set of alternatives before having worked on the subproblems that will most critically determine the quality of the entire schedule. This in turn will often result in poorer solutions. A more flexible approach would allow to quit scheduling a resource as soon as another resource is identified as being more constraining[2]. In fact, in the presence of multiple bottlenecks, one can imagine a technique that constantly shifts attention from one bottleneck to another rather than focusing on the optimization of a single bottleneck at the expense of others. Therefore, it seems desirable to investigate a more flexible approach to scheduling, or a *micro-opportunistic* approach, in which the evolution of bottlenecks is continuously monitored during the construction of the schedule, and the problem solving effort constantly redirected towards the most serious bottleneck. In its simplest form, this micro-opportunistic approach results in an *operation-centered* view of scheduling, in which each operation is considered an independent decision point and can be scheduled without requiring that other operations using the same resource or belonging to the same job be scheduled at the same time.

Section 2 describes a micro-opportunistic factory scheduler called **MICRO-BOSS** (Micro-Bottleneck Scheduling System). Section 3 describes an empirical study that compares

[1]This research was supported, in part, by the Defense Advanced Research Projects Agency under contract #F30602-88-C-0001, and in part by grants from McDonnell Aircraft Company and Digital Equipment Corporation.

[2] [1] describes an alternative approach in which resources can be resequenced to adjust for resource schedules built further down the road. This approach has been very successful at minimizing makespan. Attempts to generalize the procedure to account for due dates seem to have been less successful so far [6].

MICRO-BOSS against a macro-opportunistic scheduler that dynamically combines both a resource-centered perspective and a job-centered perspective. A summary is provided in Section 4, along with a brief discussion of current research efforts.

2. A MICRO-OPPORTUNISTIC APPROACH

In the micro-opportunistic approach implemented in MICRO-BOSS, each operation is considered an independent decision point. Any operation can be scheduled at any time, if deemed appropriate by the scheduler. There is no obligation to simultaneously schedule other operations upstream or downstream within the same job, nor is there any obligation to schedule other operations competing for the same resource.

MICRO-BOSS proceeds by iteratively selecting an operation to be scheduled and a reservation (i.e. start time) to be assigned to that operation. Every time an operation is scheduled, a new *search state* is created, where new constraints are added to account for the reservation assigned to that operation. A so-called consistency enforcing procedure is applied to that state, that updates the set of remaining possible reservations of each unscheduled operation. If an unscheduled operation is found to have no possible reservations left, a *deadend state* has been reached: the system needs to *backtrack* (i.e. it needs to undo some earlier reservation assignments in order to be able to complete the schedule). If the search state does not appear to be a deadend, the scheduler moves on and looks for a new operation to schedule and a reservation to assign to that operation.

In MICRO-BOSS, search efficiency is maintained at a high level by interleaving search with the application of consistency enforcing techniques and a set of look-ahead techniques that help decide which operation to schedule next (so-called *operation ordering heuristic*) and which reservation to assign to that operation (so-called reservation ordering heuristic).

1. **Consistency Enforcing (or Consistency Checking):** Consistency enforcing techniques prune the search space by inferring new constraints resulting from earlier reservation assignments [2, 5].

2. **Look-ahead Analysis**: A two-step look-ahead procedure is applied in each search state, which first optimizes reservation assignments within each job, and then, for each resource, computes contention between jobs over time. Resource/time intervals where job contention is the highest help identify the critical operation to be scheduled next (operation ordering heuristic). Reservations for that operation are then ranked according to their ability to minimize the costs incurred by the conflicting jobs (reservation ordering heuristic). By constantly redirecting its effort towards the most serious conflicts, the scheduler is able to build schedules that are closer to the global optimum. Simultaneously, because the scheduling strategy is aimed at reducing job contention as fast as possible, chances of backtracking tend to subside pretty fast too.

The so-called opportunism in MICRO-BOSS results from its ability to constantly *revise its search strategy and redirect its effort towards the scheduling of the operation that appears to be the most critical in the current search state.* This degree of opportunism differs from that displayed by other approaches where the scheduling entity is an entire resource or an entire job [3], i.e. where an entire resource or an entire job needs to be scheduled before the scheduler is allowed to revise its current scheduling strategy.

3. PERFORMANCE EVALUATION

MICRO-BOSS was compared against a variety of scheduling techniques, including popular combinations of priority dispatch rules and release policies suggested in the Operations Management literature [5].

This section outlines a study comparing MICRO-BOSS against a macro-opportunistic scheduler that dynamically combined both a resource-centered perspective and a job-centered perspective, like in the OPIS scheduling system [3]. However, while OPIS relies on a set of repair heuristics to recover from inconsistencies [4], the macro-opportunistic scheduler of this study was built to use the same consistency enforcing techniques and the same backtracking scheme as MICRO-BOSS[3]. The macro-opportunistic scheduler also used the same demand profiles as MICRO-BOSS. When average demand for the most critical resource/time interval was above some threshold level (a parameter of the system that was empirically adjusted), the macro-opportunistic scheduler focused on scheduling the operations requiring that resource/time interval, otherwise it used a job-centered perspective to identify a critical job and schedule some or all the operations in that job. Each time a resource/time interval or a portion of a job was scheduled, new demand profiles were computed to decide which scheduling perspective to use next. Additional details on the implementation of the macro-opportunistic scheduler can be found in [5].

In order to compare the two schedulers, a set of 80 scheduling problems was randomly generated to cover a wide variety of scheduling conditions: tight/loose average due dates, narrow/wide due date ranges, one or two bottleneck machines. Each problem involved 20 jobs and 5 resources for a total of 100 operations (see [5] for further details).

[3]An alternative would have been to implement a variation of MICRO-BOSS using the same repair heuristics as OPIS. Besides being quite time-consuming to implement, such a comparison would have been affected by the quality of the specific repair heuristics currently implemented in the OPIS scheduler.

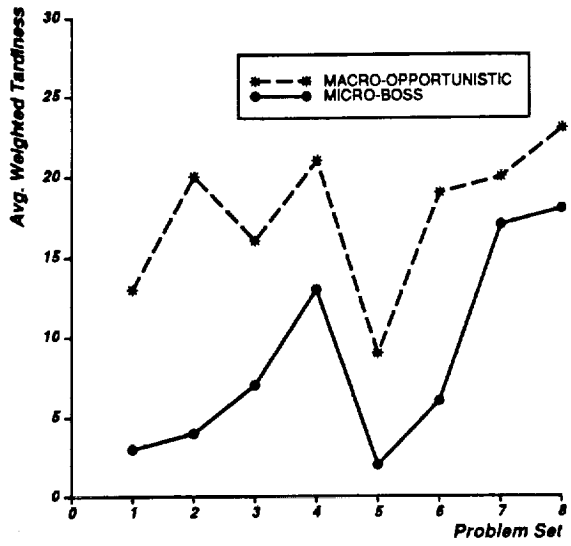


Figure 3-1: Tardiness performance of MICRO-BOSS and the macro-opportunistic scheduler on eight different problem sets.

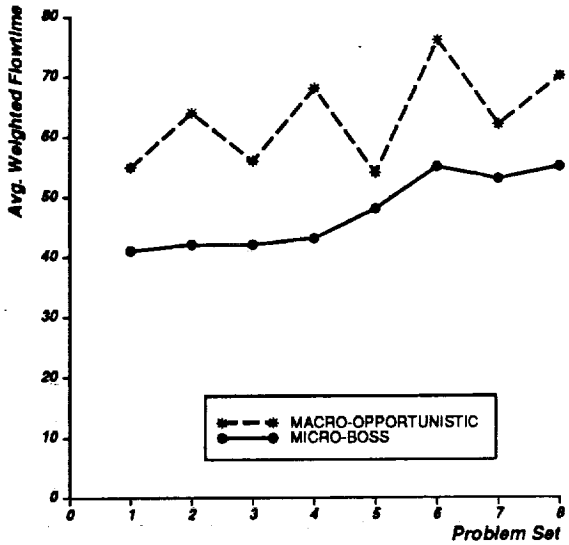


Figure 3-2: Flowtime performance of MICRO-BOSS and the macro-opportunistic scheduler on eight different problem sets.

Figures 3-1, 3-2 and 3-3 summarize the results of the comparison between MICRO-BOSS and the macro-opportunistic scheduler[4]. The macro-opportunistic scheduler was consistently outperformed by MICRO-BOSS (under all eight scheduling conditions) both with

[4]The results presented in this section correspond to the 69 experiments (out of 80) that were each solved in less than 1,000 search states by the macro-opportunistic scheduler.

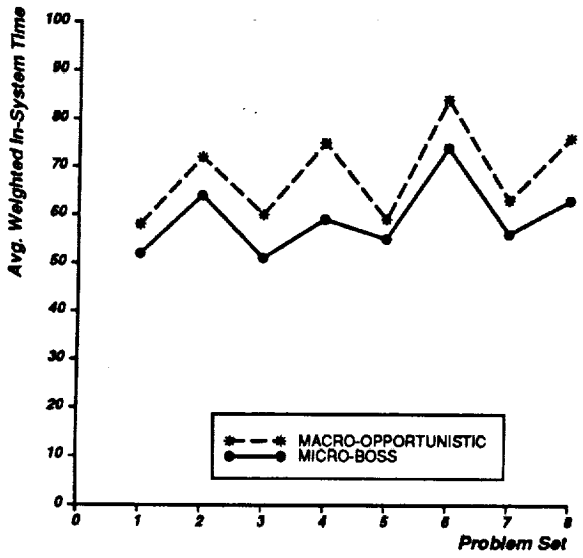


Figure 3-3: In-system time performance of MICRO-BOSS and the macro-opportunistic scheduler on eight different problem sets.

respect to tardiness, flowtime (i.e. work-in-process) and in-system time (i.e. total inventory, including finished-goods inventory). *More generally, these results indicate that highly contended resource/time intervals can be very dynamic, and that it is critical to constantly follow their evolution in order to produce quality schedules.*

In most problems, MICRO-BOSS achieved a search efficiency of 100% (computed as the ratio of the number of operations to be scheduled over the number of search states that were visited), and required about 10 minutes of CPU time to schedule each problem. The current system is written in Knowledge Craft, a frame-based representation language built on top of Common Lisp, and runs on a DECstation 5000.

4. CONCLUSIONS

In this paper, a micro-opportunistic approach to factory scheduling was described that closely monitors the evolution of bottlenecks during the construction of the schedule, and continuously redirects search towards the bottleneck that appears to be most critical. This approach differs from earlier opportunistic approaches, such as the one described in [3], as it does not require scheduling large resource subproblems or large job subproblems before revising the current scheduling strategy. This micro-opportunistic approach has been implemented in the context of the MICRO-BOSS factory scheduling system. A study comparing MICRO-BOSS against a macro-opportunistic scheduler suggests that the additional flexibility of the micro-opportunistic approach to scheduling generally yields important reductions in both tardiness and inventory.

Current research efforts include:

- Adaptation of MICRO-BOSS to deal with sequence-dependent setups
- Development of micro-opportunistic reactive scheduling techniques that will enable the system to patch the schedule in the presence of contingencies such as machine breakdowns, raw materials arriving late, job cancelations, etc.

APPENDIX: PROBLEM SETS

Problem Sets			
Number of Bottlenecks	Avg. Due Date	Due Date Range	Problem Set
1	loose	wide	1
1	loose	narrow	2
1	tight	wide	3
1	tight	narrow	4
2	loose	wide	5
2	loose	narrow	6
2	tight	wide	7
2	tight	narrow	8

REFERENCES

1. J. Adams, E. Balas, and D. Zawack. "The Shifting Bottleneck Procedure for Job Shop Scheduling". *Management Science 34*, 3 (1988), 391-401.

2. A.K. Mackworth and E.C. Freuder. "The Complexity of some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems". *Artificial Intelligence 25*, 1 (1985), 65-74.

3. Peng Si Ow and Stephen F. Smith. "Viewing Scheduling as an Opportunistic Problem-Solving Process". *Annals of Operations Research 12* (1988), 85-108.

4. P.S. Ow, S.F. Smith, and A. Thiriez. Reactive Plan Revision. Proceedings of the Seventh National Conference on Artificial Intelligence, 1988, pp. 77-82.

5. Norman Sadeh. *Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling*. Ph.D. Th., School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, March 1991.

6. P. Serafini, W. Ukovich, H. Kirchner, F. Giardina, and F. Tiozzo. Job-shop scheduling: a case study. In *Operations Research Models in FMS*, Springer, Vienna, 1988.

Iterative Refinement Scheduling*

Eric Biefeld
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109-8099
U. S. A.

Abstract

We present a heuristics-based approach to deep space mission scheduling which is modeled on the approach used by expert human schedulers in producing schedules for planetary encounters. New chronological evaluation techniques are used to focus the search by using information gained during the scheduling process to locate, classify, and resolve regions of conflict. Our approach is based on the assumption that during the construction of a schedule there exist several disjunct temporal regions where the demand for one resource type or a single temporal constraint dominates (bottleneck regions). If the scheduler can identify these regions and classify them based on their dominant constraint, then the scheduler can select the scheduling heuristic.

1 Introduction

Scheduling science experiments for such projects as Viking, Voyager, and Spacelab consumes a large amount of time and manpower. Whenever the Voyager spacecraft encounters a planet, the science experiments must be preplanned and ready to execute. This is a difficult scheduling problem due to the number and complexity of the experiments and the extremely limited resources of a spacecraft.

Since very few opportunities for space science exist, the major goal of mission scheduling is to maximize the number of science experiments that can be performed using the limited resources of the spacecraft. The total amount of requested experiments can be several times the amount that the project can accomplish.

Not only are schedules oversubscribed, they are also dynamic. Although the Voyager spacecraft was built and launched years ago, the flight rules governing the use of the spacecraft have changed. As the scientists learn more about their objectives, the experiment requests are updated. Thus, the mission schedule is a dynamic entity.

*This research was done at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored through an agreement with the National Aeronautics and Space Administration.

The Jet Propulsion Laboratory has performed mission scheduling for many years with a variety of deep space flight projects. The effort in scheduling an entire project such as Voyager can be measured in mancenturies. Because of this huge cost, JPL has been researching advanced software scheduling systems for several years (e.g. Deviser, Plan-It, Switch, Ralph, OMP).

Our current research, the Operations Mission Planner (OMP), is centered on minimally disruptive (nonnervous) replanning and the use of heuristics to limit the scheduler's search space. This paper addresses some of the problems pertinent to mission scheduling. It then defines iterative refinement, one of the basic design goals of our current research. This work has been greatly influenced by discussions with and the observations of the expert mission schedulers for the Viking, Voyager, and Spacelab projects.

2 Definitions

2.1 Resource/State

A resource/state (here after shorten to resource) tracks how a variable describing a state of the system changes through time and the steps which presently reserve this resource. An example is a pooled resource which tracks how many pieces of equipment out of a limited pool is being used at any moment in time. Another example is the direction of an antenna which is a continuous-state resource.

There are five fundamental types of resources: capacity, consumable renewable, continuous-state, and discrete-state [Starbird, 1987]. A capacity resource is basically a pooled resource but can have non-integer value and may have a time varying initial capacity.. Steps allocate a amount of the resource for their duration and then free up the resource for other activities. A consumable resource is one for which there is a limited supply, and once it is used by a step, it is no longer available (e.g. spacecraft fuel). A renewable is a generalization of a consumable, where the resource can be replenished (e.g. storage tape; it is used up during recording, and "replenished" during playback). A state resource represents a resource whose state (configuration, position, etc.) must be a certain value in order to support an activity. A continuous- state resource is one in which the state of the resource can best be described by a continuous vari-

able (e.g. the direction that an antenna is pointing). A discrete-state resources, on the other hand, are represented by discrete values (e.g. on/off, low-gain/medium-gain/high-gain).

Most domain resources can either be directly mapped into these resource types or be modeled by combining these type of fundamental resources to form a special meta-resource. A ground based Deep Space Network Antenna could be modeled as a meta-resource which combines two continuous- states and a renewable resource. The two continous-states would model the azimuth and declination while the renewable would model the number of times the antenna cables are wrapped around the antenna pedestal.

While the four fundamental types of resources can be used to model most of the resources we have encountered there exist a domain specific resources which could not be easily modeled. An example is the Voyager tape recorder which is a four track tack wire tape recorder. To schedule the tape recorder the schedulers build tape maps of what data is at what physical location on which track. This information is used to determine the order in which data can be removed and how long it takes to position the tape head to the beginning of a particular data track.

Associated with each type of resource is its definition of conflict. A conflict for a capacity resource occurs if the system reserves more then the limit of the pool at any moment in time. The resource is in conflict at the temporal interval for which a oversubscription occurs. A discrete-state is in conflict if either a step "reserves" a state that is not compatible with the state of the resource during the duration of the step or if the resource changes states without having an appropriate state changing step occurring.

2.2 Step/Activity

A step is a temporal interval which "reserve" resources where the meaning of reserve depends on the type of resource. While the resources model the state of the system over time steps model changes or constraints on the system. Along with resource reservations a step can contain constraints that either directly limit the range of choices possible in scheduling a step or links a step to other steps. The most common type of linking constraints are temporal predecessor and successor relations.

An activity is a set of steps and a set of constraints that link the steps together. The temporal constraints are the "glue" that bind the steps into a logical unit. The most common type of temporal linkage is the predecessor and successor relations. Along with the steps and constraints between the steps, an activity includes constraints that act on upon all the steps within an activity. This includes any global temporal windows and other global scheduling preferences like a priority for the request.

The users views an activity as the "primitive" action that must be scheduled to satisfy a user scheduling request. When a user issues a "request" the system finds the one or more activities that satisfy the request. The scheduler heuristically selects one of the activities and schedules the entire activity as a logical unit. Unlike resources the scheduler does not violate the constraints within an activity.

Since the activities interact only through the resource timelines, in some sense the activities are independent. It is possible to modify a previously schedule activity without backtracking or updating any other scheduled activity. Modifying a previously scheduled activity may cause some resource conflicts, but at certain stages of the scheduling process that is acceptable. The scheduler has the ability to note the conflicts for resolution at latter stages in the processing.

3 Focused Iterative Refinement

3.1 Expert Iterative Refinement

Iterative refinement is a technique used by expert spacecraft schedulers. The expert user first lays out the highly constrained activities over which he has little or no control. This forms a background against which the rest of the scheduling is done. The expert user then places the activities which impact large portions of the schedule. These may, for example, be a series of activities that have to be performed at exactly one-hour intervals over a large portion of the schedule. Any changes to this type of activity would cause changes to most of the schedule. If the scheduler gets stuck trying to place such an activity, he may elect to move it, but only as a last resort. Next, the expert user positions the high-priority activities, minimizing the number of conflicts. Finally, to complete the initial loading process, the expert user places the remaining activities on the schedule. If, at this point, some of the lower-priority activities do not fit easily, the expert user may simply ignore them.

After the loading process is done, the schedule is 80sense that most activities are in their final position on the schedule), although some resource contentions may still exist. The expert user has only spent about 20user will spend the remaining time trying to fit a few more activities into the schedule and trying to resolve resource contentions.

Up to this point in the scheduling process the scheduler has been task oriented [Smith and Ow, 1985]. Now the scheduler becomes resource oriented. The expert user focuses on the activities which are causing resource contentions on a particular resource and in a particular time region. After this area is fixed the expert user moves to another. Using this type of planning, the expert user iterates over and over again on the schedule, each time refining it a little more. After each pass through the schedule, the scheduler is willing to do a deeper search on any single activity because the total number of activities needing to be searched will decrease.

By focusing on just one area at a time the expert user may fix a portion of the schedule just to cause conflicts when the next portion of the schedule is processed. After several iterations, a small set of activities will circulate through the problem areas of the schedule. In this stage of scheduling, the expert user once again becomes task oriented. The expert user focuses on this

small set of hard-to-place activities and performs the deepest search. The expert user addresses any chain reactions resulting from moving a specific activity. In Voyager scheduling this reasoning recurses about three levels down. In SpaceLab science scheduling the depth cut off is about four levels down. It is important to realize, however, that at this point the expert user has a small list of activities to try. The scheduler also restricts the impacted activities to those that seem flexible.

In the final stage of processing, the expert user looks for under-utilized areas of the schedule. The expert user checks the list of unscheduled activities looking for an activity that could use these resources. This unscheduled activity will, most likely, not fit directly into the schedule without causing some conflicts. Otherwise, the activity would have been scheduled earlier in the process. The scheduler tries to adjust some of the activities in the under-utilized areas in order to make room for the unscheduled activity. This may involve a series of shifts, but since both the activity and the under-utilized areas have been identified, it is a tightly focused search.

The schedule is then evaluated by the mission scientists for its total science return. The scientists negotiate with one another and with the scheduling team about which activities to include in the final sequence. The results of the negotiations must be reflected in the schedule. Therefore, the evaluation process following the generation of the initial schedule often results in requests to change the schedule, and hence the requirement for the replanning capability discussed earlier.

3.1.1 Phases of Iterative Refinement

Iterative planning consists a series of techniques. Each technique is responsible for a different aspect of the overall planning process. The first of these techniques roughs out the plan and identifies areas of high resource-conflict. The later techniques use the knowledge of the resource conflicts to refine the plan and solve many of the schedule problems. The final techniques try to solve the last of the conflicts and "optimize" the plan.

The OMP Load Phase is responsible for drafting an initial schedule. During this phase, the scheduler focuses on the requested activities, fitting them into the schedule with minimal concern for conflicts and levels of oversubscription.

During the Resource Centered Phase, OMP becomes resource oriented [Smith and Ow, 1985]. The scheduler focuses upon a resource region which contains conflicts and uses quick and simple techniques to fix these regions before processing another resource. It is during this phase that the bulk of the schedule is roughed out.

By focusing on just one resource region at a time the scheduler may fix one portion of the schedule but create additional conflicts in other regions. The scheduler discovers the bottlenecks by tracking these interactions between the separate regions. Once a bottleneck has been identified, it is classified and OMP attempts to resolve that bottleneck using techniques specialized for the type of bottleneck.

Once the conflict regions of the schedule have been resolved (which, since this is an oversubscribed domain will involve deleting some activities from the schedule), OMP takes another look at the high priority activities which have been deleted from the schedule and tries to fit them in. At this point, OMP will perform its deepest search in an effort to schedule just one more activity (extremely important in a domain such as deep space mission scheduling where opportunities to perform interplanetary experiments are rare). This phase is called the Optimization , although it doesn't produce a truly optimal schedule as would be defined in an operations research sense. Rather, it refers to fitting in additional activities after a conflict-free schedule has been produced. According to Spacelab scheduling experts, an optimal schedule is one where no one can suggest an improvement [Japp, 1986].

By specializing the planning techniques, each technique can be made more efficient. For example, the first techniques will use shallow searches over a broad spectrum of activities. Later techniques will use deeper searches but the search will only be applied to a limited number of activities. They will use knowledge about the particular schedule (i.e. the current resource conflicts, which activities have changed most often in the scheduling process) to constrain the search space. The techniques will employ either a shallow and broad search or a deep and narrow search. If a planner must perform a broad and deep search, it will not be able compute the schedule in any reasonable time.

3.1.2 Self-Reflective Iterative Refinement

The basic concept of self-reflective search is focusing the search by using knowledge gained from monitoring the search process. The OMP architecture, operating as outlined in the previous section, provides the mechanisms for supporting self-reflective search: the chronologies gather the raw information, the assessment heuristics analyze the information and feed the results to the control heuristics which focus the dispatch heuristics.

During the scheduling process, OMP keeps a chronology [Biefeld and Cooper, 1989] of the effort expended to resolve resource conflicts. In OMP, the chronologies are composed of a set of course grain resource timelines which record the scheduling effort level associated with a given region of the schedule, one measure of which is the number of times the scheduler attempts to resolve conflicts in that region.

During the resource centered phases, OMP focuses on a temporal interval within a given resource that is in conflict. Simple heuristics (which either change the resource used by an activity or temporally shift an activity out of the focus region [Biefeld and Cooper, 1991]) are used to reduce the level conflict in the focus region. The chronologies keep track of the effect of these actions within the region and on other regions which are changed as a result of the scheduling actions.

The system first attempts to find a set of resource assignments which reduces the total amount of conflict in the entire schedule. If the system can not lower the total conflict then it will increase the effort level for the focus region. The system retries the search, again attempting to reduce the conflict level in the region of focus, how-

ever this time it can increase the conflict level in other temporal regions for which the effort level is less than the focus region's effort level.

The above process will eventually cause OMP to cycle through the same regions. When the effort level for these regions exceeds the preset threshold, OMP exits the resource centered phase and begins the bottleneck centered phase. The assessment heuristics search through the chronologies and find the regions that have recently been raised to a high effort level. These regions are then collected into a bottleneck. The assessment heuristics then classify the bottleneck depending on its temporal size and its degree of oversubscription.

The current assessments heuristics in OMP distinguish bottlenecks by: 1) the amount of subscription compared to the bottleneck capacity; 2) the temporal extent of the bottleneck regions; and 3) the number of resources the bottleneck spans. Using these ratings the assessment heuristics classify the bottlenecks as either: 1) largely oversubscribed; 2) close to capacity but large in extent; or 3) close to theoretical capacity and small in extent.

If a bottleneck is largely oversubscribed then OMP's control heuristics will delete the low priority activities from the bottleneck region until the demand is only slightly larger then the capacity of the bottleneck. If a bottleneck is close to capacity but large in extent the control heuristics will split the bottleneck into several smaller regions. The first step is to distribute the tasking uniformly across the bottleneck and to reduce the demand slightly by shrinking the duration of the activities. The control heuristics will then focus on the smaller regions and use dispatch heuristics that emphasize local modifications over the global modifications used in the Resource Centered Phase. During this processing the assessment heuristics closely monitor the chronologies to identify small bottleneck regions. OMP processes each of the small bottlenecks as it locates them.

When processing a small bottleneck OMP uses it's most complicated heuristics. They use localized modifications to position one more activity onto the schedule. If the region in conflict is temporally small, the heuristics will either try to clip some activity whose start or end time is near the conflict, or the heuristics will split some activity into two separate activities with a gap equal to the conflict duration. If the conflict region is slightly larger, the heuristics clip and form gaps in a series of activities and align these gaps in such a manner as to reduce the conflict over the focus region.

Some heuristics, such as those for antenna handoff, are domain specific. A antenna handoff is when an activity splits its requirement for an antenna between two or more antenna resources. In the OMP demonstration domain, an activity may use one antenna for the first part and a second antenna for the second part but there must be a period of overlap during which it is using both antennas. In the OMP demonstration domain, if a bottleneck either spans two antennas or the temporal regions on the two antennas are near but do not completely overlay, then a antenna handoff may be practical. The dispatch heuristics attempt to split the activity into two activities and assign the antennas and temporal overlap to reduce resource contention.

This is an example of not only domain specific ways of expanding an activity but also where domain specific heuristics are needed to suggest when and how to try a particular activity expansion. Since the durations of the handoff overlap and the duration that an activity must spend on any single antenna is relatively small compared to the entire duration of an activity, the total number of ways an activity can be sliced up using antenna handoffs is quite large and in most cases not very useful. By identifying the bottleneck regions and then using domain specific heuristics to find particular patterns in the bottleneck regions the search process can be restricted, while still finding most cases were there special configuration tricks are useful.

4 Summary

This iterative planning approach to scheduling arose from attempts to heuristically control the search space of mission scheduling. The source of the heuristics were the human schedulers of Voyager, Viking, and SpaceLab who provided information on the stages of the scheduling process. Earlier stages are concerned with "roughing out" the schedule, placing most of the tasks, and identifying the trouble areas. Later stages then use scheduling heuristics to refine the existing schedule.

Most of these heuristics assume that the scheduler knows which resources are the bottlenecks and which tasks are causing the most difficulty for the scheduler. The best way to identify these critical resources and tasks is from the schedule produced by the earlier stages. In order to know what to try next one must already know what the schedule will be like.

Iterative planning assumes that the information gained by earlier techniques can be used by the later techniques to constrain the search space. Iterative planning also assumes that the schedule will not be changed dramatically by the later techniques. These assumptions seem to hold for the mission scheduling domain, which is extremely under- constrained. There exist many possible schedules for a single set of requested tasks. Two different human schedulers will produce two very different but equally acceptable schedules, given the same set of requested tasks. If, however, one human scheduler must modify another person's schedule, the basic structure of the schedule will not be modified. Therefore, expert schedulers normally perform non-nervous replanning.

References

[Biefeld and Cooper, 1989] Eric Biefeld and Lynne Cooper. Comparison of Mission and Job Shop Scheduling. *Proceedings of the Third International Conference on Expert Systems and the Leading Edge in Production Planning and Control*, pages 483–494, Hilton Head Island, South Carolina, May 1989.

[Biefeld and Cooper, 1989] Eric Biefeld and Lynne Cooper. Scheduling with Chronology-Directed Search. *Proceedings of the AIAA Computers in Aerospace VII*

Conference, pages 1078–1087, Monterey, California, October 1989.

[Biefeld and Cooper, 1991] Eric Biefeld and Lynne Cooper. Bottleneck Identification Using Process Chronologies. *Proceedings of the Twelfth International Joint Conference On Artificial Intelligence*, pages 218–224, Sydney, Australia, August 1991.

[Dean, 1986] Thomas Dean,. Intractability and Time Dependent Planning. *Proceedings of Workshop on Planning and Reasoning About Action*, pages 143–164, June 1986.

[Dean and McDermontt, 1987] Thomas Dean and Drew McDermott. Temporal Data Base Management. *Artificial Intelligence Journal*, 32(1):1–55, 1987.

[Dean *et al.*, 1987] Thomas Dean, R. James Firby, and David Miller. Hierarchical Planning involving Deadlines, Travel Time, and Resources. *Computational Intelligence*, 4(4), November 1988.

[Eskey and Zweben, 1990] Megen Eskey and Monte Zweben. Learning Search Control for Constraint-Based Scheduling. *Proceedings of the Eighth National Conference on Artificial intelligence*, page 908–915, Boston, Massachusette, July 1990.

[Fox and Smith, 1984] Mark Fox and Stephen Smith. ISIS: A Knowledge-Based System for Factory Scheduling. *Expert Systems*, 1(1):25–49, July 1984.

[Japp, 1986] John Japp. Mission Timeline Analysis Demonstration. NASA/Marshall Space Flight Center, June 3, 1986.

[Johnson and Roadifer, 1986] Craig Johnson and James Roadifer. A Look-Ahead Strategy for Heuristic Activity Scheduling. *Joint Conference of the Operations Research Society of America and the Institute of Management Sciences*, October 1986.

[Le Pape and Smith, 1987] Claude Le Pape and Stephen Smith. Management of Temporal Constraints for Factory Scheduling. *Proceedings of the Working Conference on Temporal Aspects in Information Systems*, May 1987.

[McLaughlin and Wolff, 1989] W. I. McLaughlin and D. M. Wolff. Automating the Uplink Process for Planetary Missions. *AIAA 27th Aerospace Science Meeting*, Reno, Nevada, January 1989.

[Miller, 1988] David Miller. A Task and Resource Scheduling System for Automated Planning. *Annals of Operations Research*, 12(1–4):69–198, February 1988.

[Minifie and Davis, 1986] J. Roberta Minifie and Robert Davis. Survey of MRP Nervousness Issues. *Production and Inventory Management*, 27(3):111–121,1986.

[Muscettola and Smith, 1987] Nicola Muscettola and Stephen Smith. A Probabilistic Framework for Resource-Constrained Multi- Agent Planning. *Proceedings of the Tenth International Joint Conference On Artificial Intelligence*, page 1063–1066, Milan, August 1987.

[Ow and Smith, 1988] Peng Si Ow and Stephen Smith. Viewing Scheduling as an Opportunistic Problem-Solving Process. *Annals of Operations Research*,12(1-4):85–108, February 1988.

[Smith *et al.*, 1986] Stephen Smith, Mark Fox, and Peng Si Ow. Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-Based Factory Scheduling Systems. *AI Magazine*, 7(4):45–61, 1986.

[Smith and Ow, 1985] Stephen Smith and Peng Si Ow. The Use of Multiple Problem Decompositions in Time Constrained Planning Tasks. *Proceedings of the Ninth International Joint Conference On Artificial Intelligence*, pages 10130–1015, Los Angeles, California, August 1985.

[Starbird, 1987] Tom Starbird. Space Flight Operations Center Sequence Subsystem (SEQ) Functional Requirements Document for Planning. JPL Internal Document D–4697; NASA, Jet Propulsion Laboratory, California Institute of Technology, Pasadena California, August 1987.

[Vere, 1983] Steven Vere. Planning in Time: Windows and Durations for Activities and Goals. *IEEE Transactions on Machine Intelligence PAMI-5*, No. 3, pages 246–267, May 1983.

[Zweben *et al.*, 1990] Monte Zweben, Micheal Deal, and Robert Gargan. Anytime Rescheduling. *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 251–259, San Diego, California, November 1990.

CABINS : Case-Based Interactive Scheduler

Kazuo Miyashita
miyashita@cs.cmu.edu

Katia Sycara
katia@cs.cmu.edu

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

1. Introduction

Although there has been a lot of progress in knowledge-based scheduling [5, 4], there is still a need for schedule improvement and repair through interaction with a human scheduler. There are several reasons for this. First, a user's preferences on the schedule are context dependent (e.g., may depend on the state of the scheduling environment at a particular time). Also, interactions among preferences and effective tradeoff very often depend on the particular schedule produced. This means that generally a user of the scheduling system can't fully specify his/her preferences a priori before getting the scheduling results from the system. By looking over the obtained schedule results, the user often thinks of additional preferences. Consider, for example a situation where a human scheduler does not like to use MACHINE-A which is substitutable for MACHINE-B but is of lower quality than MACHINE-B for processing ORDER-X. The reason high quality results are desired is that ORDER-X belongs to a quite important client. Suppose, however, that the schedule indicates that ORDER-X is tardy by an amount above an acceptable tardiness threshold due to demands on MACHINE-B (by orders more important than ORDER-X). Then, the human scheduler may decide to use the less preferable machine, MACHINE-A for the less important order, ORDER-X. If the tardiness was below the threshold, he/she may prefer to allow a tardy order. It is very difficult to elicit this type of preference and preference thresholds from the human scheduler independent of the presence of a particular context.

The second reason interactive schedule repair is desirable is that it is impossible for any given knowledge based scheduling model to include all the constraints that may be relevant. Current advanced scheduling systems can exploit very complicated models to represent the factory, orders and user's preferences. But no matter how richly the model is constructed, there are always additional factors which may influence the schedule but had not been represented in the model. For example, for a certain foundry it may be good to decrease usage of a sand casting machine during the summer, because the combination of heat and humidity of the weather may make it slower than usual. But how should the model of the scheduling system represent the season, weather or humidity? And isn't it necessary for the model to represent time of the day, strength of wind or health of a machine operator and so on? [2]. Nevertheless these factors, that an experienced human scheduler learns to take into consideration, could have a big influence on schedule quality but it is very difficult to represent in a principled manner so they can be used by an automated scheduling system.

The third reason interactive schedule repair is desirable is that factories are dynamic environments. Unexpected events, such as operator absence, power failure and machine breakdowns frequently happen. Therefore, it is necessary for the scheduling system to adapt to the events in the factory environment as soon as possible by reactively repairing the existing schedule. Although initial progress has been made in automatic schedule repair [3], human intervention may be necessary as a result of the reasons given (context dependent user preferences, and difficulty of representing all relevant constraints).

Another consequence of the above is that *local repair rather than re-scheduling* is more desirable, since re-scheduling will suffer from the same ills as the initial scheduling. In addition, it is in general desirable [3] to minimize disruption to the shop floor. If re-scheduling from the point of failure is attempted, the new schedule may be drastically different from the original schedule, thus necessitating disruption of the work flow in the shop, and new work allocation. The new schedule, moreover may solve the current problem but introduce new problems that have to be solved.

One extremely beneficial side effect of interactive schedule repair is the insight that the user obtains into his/her scheduling preferences and their context of applicability. The process of interactive repair requires the human scheduler to analyze the current problem, repair it by clarifying or modifying his/her preferences and finally evaluate the result. This gives the human scheduler good opportunities to understand his/her criteria in diverse situations. So later when he/she encounters a problem that is similar to a previous one, he/she can be reminded of the applicable previous repair and re-use it in the current situation.

1.1. Why case-based repair?

Case-based Reasoning (CBR) is a recent AI problem solving paradigm [1]. A CBR system tries to solve a problem by (1) retrieving the most similar case with the current problem from its case base, (2) modifying it to adapt to the current situation and (3) applying it to the current problem. At the end of problem solving, the new solved problem is stored as a new case in the case memory. As a computational model the first feature of CBR is its method of knowledge acquisition. In CBR the unit of knowledge is the case, which is an experience encountered during problem solving. This makes it easier to articulate, examine and evaluate the knowledge. The second feature is its learning capability. A CBR system can remember its performance and modify its behavior to avoid repeating prior mistakes. The third feature is its adaptive power. By reasoning from analogy with the past experiences, a CBR system should be able to construct solutions to novel problems. These features make CBR very attractive for interactive schedule repair.

Because a case describes a particular specific experience, the factors that were deemed relevant to this experience can be recorded in the case. This description fully captures the dependencies among features and their context. So if a similar situation is encountered, the system can re-use the repairing method which is stored in the retrieved case. In addition, a case serves as a knowledge structuring mechanism so that all relevant factors are local to a case rather than distributed through the system (as happens with rule based systems). Even when the result of applying the repairing method of the retrieved case turns out to be failure, if the user can explain the failure, then the system can create a new case based upon this failure experience and store it as a new case along with the associated explanation. Thus, as the case base is enriched with successful and failed experiences, the system becomes more robust for various type of schedule defects that would have been difficult to predict in advance. This enables the replacement of expert users with novices that rely on the system's experiences.

2. Case-based Interactive Scheduler (CABINS)

Based upon the above discussion, we are developing the Case-based Interactive Scheduler (CABINS) whose goal is to support interactive schedule repair. A CABINS user is envisioned to be a person who is responsible for making schedules in advance of production. In making an initial schedule, the user may be assisted by an automated scheduling system. If the user identifies undesirable features of the current schedule, he/she uses CABINS for schedule repair, so as to improve the current schedule. CABINS finds defects in scheduling results and repairs them by patching locally or modifying part of its model (resources, orders, shifts and user's preferences).

2.1. System Architecture

After the initial schedule is made, it is examined by the user and the defect detector (a rule-based system) to find undesirable parts in the existing schedule. If some defects are detected, the information about the defects are passed to the repairer. If local repairing is determined to be feasible by the repairer, resource reservations in the current schedule are directly modified or canceled by the repairer and the scheduler is asked to re-schedule the conflicting operations whose reservations were canceled. When local repair turns out to be impossible, the repairer modifies the scheduling model and re-scheduling is attempted based on the modified model. The overall goal of CABINS is to make repairs as cheap as possible trying at the same time to minimize interfering side effects of these repairs on the current schedule. Figure 2-1 depicts the architecture of CABINS.

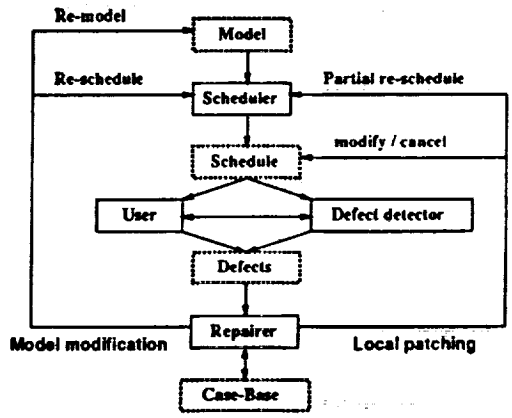


Figure 2-1: Architecture of CABINS

2.2. Schedule Repairing Process

The processing of CABINS has four stages:

- defect detection
- defect selection
- selection of repair strategy
- selection of repair tactics

Currently defect detection and defect selection are performed by the user who finds the most important defect and identifies the features associated with the defect. These features are used as indices into the case memory to find similar past defects. Out of the retrieved similar past defects, the *least critical* is selected. To determine defect criticality, the system uses the cost of repairing the defect as a measure: the lower the repair cost, the less critical the defect. Low repair cost is usually associated with local patching whereas high cost means that more changes are made to the overall schedule. So, beginning with the lowest cost repair is a good heuristic since the defect can be potentially fixed cheaply.

CABINS uses two level of repairs: repair strategies and repair tactics. A repair strategy is associated with a particular high level description of classes of defects. Each repair strategy has a variety of repair tactics associated with

it. The repair tactics are appropriate for particular specializations of the defect classes. We have identified two general types of repair strategies: local patching and model modification.

To select a strategy for repairing important defects, CABINS looks for the most similar case to the current situation in the case base and selects the same strategy which succeeded in the past case. The system has several alternative strategies for each defect and one of them is selected based on the feature similarity of the current situation and the past experience. Some of the features that we are currently using for case retrieval are various defect types, such as order tardiness and various schedule characteristics, such as schedule tightness, inter-order slack, and machine idle time. For example, if the type of defect is "tardy order", there are seven repair strategies:

1. Reduce the slack between operations in the tardy order
2. Reduce the idle-time of resources needed by operations in the tardy order
3. Relax due-date constraint of orders (the tardy order or interfering orders)
4. Relax release-date constraint of orders (the tardy order or interfering orders)
5. Reduce the shop load
6. Increase shifts
7. Increase resource capacity.

The first two strategies belong to the general category "local patching" and the rest to the category "model modification".

In general, we have presented the repair strategies in order of expensiveness (from the cheaper --strategy 1 to most expensive --strategy 7). For tardiness repair, the discriminating feature between selecting cases with repair strategies in classes 1 to 2 and selecting cases with repair strategies 3 to 7 is the tightness of the current schedule. If the current schedule is not very tight (i.e., there are a lot of idle intervals on resources needed by operations of the tardy order), CABINS will select cases where tardiness was repaired by local patching. Whether cases with repair-strategy-1 or repair-strategy-2 will be selected depends on whether, beside enough idle interval, there is also slack between adjacent operations of the tardy order. If there are, then cases where strategy-1 was used will be selected. Tactics associated with strategy-2 could be to move every operation of the tardy order upstream (left shifting) on the time line if enough idle interval is available for the operation.

If the current schedule is tight, then cases that prescribe model modification rather than local patching will be retrieved. If there are no discriminating features to determine the applicability of strategies 3 to 7, CABINS uses the default ordering: use strategies in ascending cost. The cheapest model modification is relaxing due-date constraints of the tardy order or interfering orders (strategy-3). This is cheap since it is easily accomplished and has no side effects on the shop floor environment. On the other hand, reducing the factory load (strategy-5) (e.g., by subcontracting orders) and re-scheduling is in general more expensive than relaxing due dates of interfering orders because one must determine the orders to be subcontracted out, price of subcontracting, possible delays etc. An additional concern is that the resulting schedule might not be entirely satisfactory and may need to be repaired anew. Similarly, strategies 6 and 7 are increasingly expensive, since additional investments in paying overtime or buying new machines are needed.

Although strategy-3 is the cheapest of the repair strategies of type "model modification", it may not always be desirable. To determine applicability of strategy-3, CABINS retrieves cases where application of strategy-3 has failed. If other features of the current situation match features of the past failures of strategy-3 (e.g., the tardy order has a stiff penalty for tardiness), then CABINS is warned that strategy-3 is not applicable. Similarly, if there are no discriminating features to distinguish among the application of strategies 4 to 7, retrieval of previous cases where the strategy under consideration has failed gives the system additional discriminating information. Thus, CABINS uses the default ordering of repair strategies as well as successful case application as necessary conditions of the applicability of particular repairs; it uses past failures as sufficiency conditions. As more cases are encountered, both the necessary and sufficiency conditions are refined. Therefore, it is hoped that CABINS can improve its performance over time.

For each repair strategy, there could be a variety of repair tactics that are applicable. For repairing order tardiness, there is a variety of appropriate tactics for local patching. Below, we present some of these tactics.

1. left-shift on same resource: move the operation as much to the left as possible, while maintaining the amount of disruptions as small as possible.
2. left-shift on substitutable resource: if the operation that is desired to be moved has a substitutable resource, then move the operation as much to the left as possible, while maintaining the amount of disruptions as small as possible.
3. swap on same resource: find another operation which is to the left of the operation to be moved on the same resource and whose duration is approximately equal to the duration of the current operation and swap the two operations.
4. swap on substitutable resource: if the operation that is desired to be moved has a substitutable resource, then find another operation which is to the left of the operation to be moved on the substitutable resource and whose duration is approximately equal to the duration of the current operation and swap the two operations.

The last two tactics may result in tardiness of other orders but this may be allowable.

For model modification, possibly applicable tactics along with the associated repair strategy are:

1. relax-due-date-of-tardy-order (strategy-3)
2. find-most-interfering-order with the current tardy order and make it tardy (strategy-3)
3. relax-release-date-of-tardy-order (strategy-4)
4. find-most-interfering-order with the current tardy order and make it start earlier (strategy-4)
5. subcontract-least-profitable-order to create more slack (strategy-5)
6. subcontract-most-interfering-order to create more slack (strategy-5)
7. overtime-work on weekday (2 hours) (strategy-6)
8. overtime-work on weekend (8 hours) (strategy-6)
9. increase-capacity-of-most-critical-resource (strategy-7)
10. capacity-of-substitutable-resource-of-most-critical-resource (strategy-7)

Each retrieved case has been repaired by possibly using a combination of repair strategies and tactics. Upon recognition of similarities in schedule defects and defect context, the appropriate repair plan could be applied. If the application of a repair step leads to failure, the user is asked to supply a possible explanation of the failure. The failure is then stored in memory so it can be retrieved and help the user avoid similar failures in the future.

3. Example

In this chapter we explain how CABINS works by using a simple example. In the example we make a schedule of 4 orders on 5 resources. Each order has a client, fixed release-date and fixed due-date. Every order is composed of 5 operations (ope-1 to ope-5), which should be ordered in that order. Each operation has fixed duration and requires one resource which may or may not have a substitutable resource. The detail specifications of the example problem are depicted in figure 3-1. In Figure 3-2 we show the result of the original scheduling. Each rectangle represents the reservation of each operation over the time-interval on the machine. The small number inside each rectangle shows the order to which the operation belongs. In scheduling the 4 orders, the scheduler failed to meet the due-date of order-3 by 130. (The due-date of order-3 is 790, while order-3 is scheduled to finish on 920.) Suppose that the client of order-3 has had the late shipment of his orders several times, s/he is sure to cancel her/his contract as a result of our more tardy shipment. Therefore, finding and fixing this situation is critical. A human scheduler at the factory tries to fix this problem by consulting with CABINS.

First, CABINS considers the current problem as a case by compiling the current scheduling results with respect to the tardiness of order-3. A human scheduler gives additional contextual information to it if s/he finds it's necessary or helpful for finding the solution of the current problem. The vocabulary of this information is maintained by CABINS and a human scheduler can update it by adding/deleting terms. Figure 3-3 shows the contents of this example problem case.

Then, CABINS tries to retrieve the case most similar

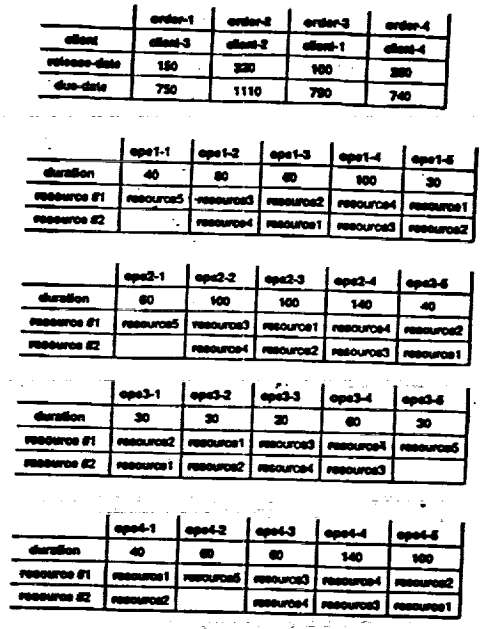


Figure 3-1: Problem Specifications

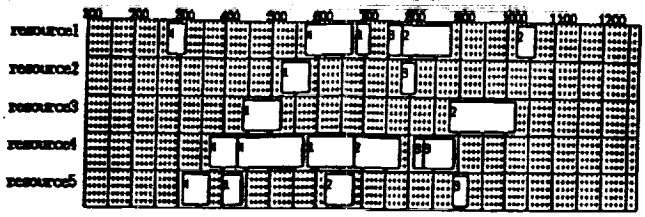


Figure 3-2: Initial Schedule

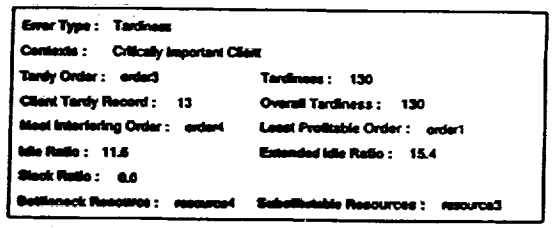


Figure 3-3: Current Problem Case

cases to the current problem case from its case-base library. The retrieved case includes not only the problem situation description but also repairs and repair outcomes. For repair strategy selection, every solution includes the information of the selected strategy, the result of applying the strategy and the explanation of why it succeeded or failed. The explanation of the solution outcome is added to the case by a human scheduler only when s/he thinks it is necessary for credit or blame assignment of the selected strategy. Figure 3-4 depicts the retrieved case to solve this example problem.

After display of the retrieved cases, a human scheduler examines whether s/he can apply the same solution method

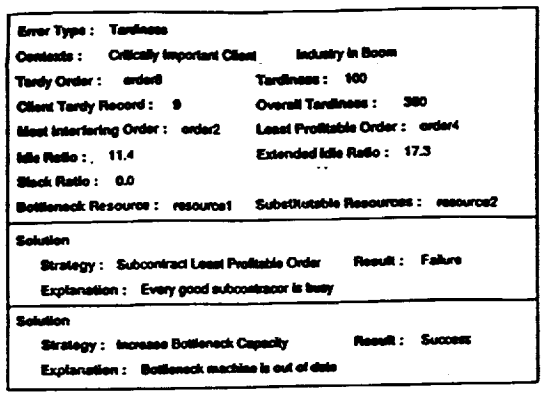


Figure 3-4: Retrieved Case

to the current problem. Even when the result of the solution in the retrieved case was failure, the solution may be worth trying if the explanation of failure given in the previous case does not hold in the current situation. On the other hand, a human scheduler should also check the validity of the explanation of a successful previous solution before s/he applies it to the current problem. In this example, even though the first solution failed when it was applied in the precedent case, a human scheduler can try to apply it, because the explanation of the failure given ("Every good subcontractor is busy") is apparently related to the description of the context of the problem ("Industry in Boom"). Therefore the explanation is not necessarily true in the current situation which doesn't share the same context. Note that those judgments are done by a human scheduler. However, by retrieving and displaying previous similar cases, CABINS gives her/him useful information to help making her/his decision. Moreover, the greater the number of new cases that are added into the case-base library, the more likely CABINS is to retrieve the case which is close enough to the current problem. Therefore, it becomes progressively easier through CBR to decide whether the solution of the retrieved case is applicable or not.

After determining the solution method, a human scheduler can execute it by interacting with the scheduling system. Figure 3-5 depicts the result of rescheduling order-3 after subcontracting the least profitable order (order-1) in this example. It shows that order-3 meets its due-date, i.e. the repair was successful.

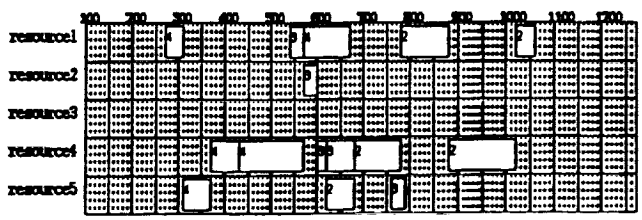


Figure 3-5: Repaired Schedule

4. Concluding Remarks

In this paper we discuss the need for interactive factory schedule repair and improvement, and identify case-based reasoning (CBR) as an appropriate methodology. Case based reasoning is the problem solving paradigm that relies on a memory for past problem solving experiences (cases) to guide current problem solving. Cases similar to the current case are retrieved from the case memory, and similarities and differences of the current case with past cases are identified. Then a best case is selected and its repair plan is adapted to fit the current problem description. If a repair solution fails, an explanation for the failure is stored along with the case in memory, so that the user can avoid repeating similar failures in the future.

So far we have identified a number of repair strategies and tactics for factory scheduling and have implemented a part of our approach in a prototype system, called CABINS. As a future work, we are going to scale up CABINS to evaluate its usefulness in a real manufacturing environment.

References

[1] Kolodner, J., Simpson, R. and Sycara, K.
A Process of Case-Based Reasoning in Problem Solving.
In *Proceeding of the Ninth International Joint Conference on Aritificial Intelligence*, pages 284-290. IJCAI, Los Angeles, CA, 1985.

[2] K.Mckay, J.Buzacott, F.Safayeni.
The Scheduler's Knowledge of Uncertainty: The Missing Link.
In *Proceedings of IFIP Working Conference on Knowledge Based Production Management Systems*. Galway, Ireland, 1988.

[3] P.S. Ow, S.F.Smith, A.Thiriez.
Reactive Plan Revision.
In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 77-82. AAAI, St-Paul, Minnesota, 1988.

[4] Norman Sadeh.
LOOK-AHEAD TECHNIQUES FOR MICRO-OPPORTUNISTIC JOB SHOP SCHEDULING.
PhD thesis, School of Computer Science, Carnegie Mellon University, 1991.

[5] Stephen F.Smith, Peng Si Ow, Nicola Muscettola, Jean-Yves Potvin Dirk C.Matthys.
AN INTEGRATED FRAMEWORK FOR GENERATING AND REVISING FACTORY SCHEDULES.
Journal of the Operational Research Society, 1990.

Scheduling Lessons Learned from the Autonomous Power System

Mark J. Ringer
Sverdrup Technology Inc.
NASA Lewis Research Center Group
Cleveland, Ohio 44135
Ringer@mars.lerc.nasa.gov

Abstract

The Autonomous Power System (APS) project at the NASA Lewis Research Center is designed to demonstrate the applications of integrated intelligent diagnosis, control and scheduling techniques to space power distribution systems. The project consists of three elements: the Autonomous Power Expert System (APEX) for Fault Diagnosis, Isolation, and Recovery (FDIR); the Autonomous Intelligent Power Scheduler (AIPS) to efficiently assign activities start times and resources; and power hardware (Brassboard) to emulate a space-based power system.

The AIPS scheduler has been tested within the APS system. This scheduler is able to efficiently assign available power to the requesting activities and share this information with other software agents within the APS system in order to implement the generated schedule. The AIPS scheduler is also able to cooperatively recover from fault situations by rescheduling the affected loads on the Brassboard in conjunction with the APEX FDIR system.

AIPS served as a learning tool and an initial scheduling testbed for the integration of FDIR and automated scheduling systems. Many lessons were learned from the AIPS scheduler and are now being integrated into a new scheduler called SCRAP (Scheduler for Continuous Resource Allocation and Planning). This paper will serve three purposes: an overview of the AIPS implementation, lessons learned from the AIPS scheduler, and a brief section on how these lessons are being applied to the new SCRAP scheduler.

1. Introduction and Motivation

Future NASA spacecraft and planetary surface installations will require larger and more sophisticated infrastructure systems and living environments. Such systems will consist of dozens of resources and hundreds of attached loads. The electrical power system on the Space Station Freedom, a Lunar base, or Martian base represents a critical portion of such a system. The APS project explores intelligent hardware and software architectures for efficient system operation and scheduling of an electrical power system [Ringer 1991].

1.1 The Need For (Automated) Scheduling

Onboard a complex spacecraft many activities must be performed, each competing for a multitude of temporal positions and limited resources. A scheduler must assign start times to each activity without violating any resource or temporal constraints. The resources onboard such a spacecraft will be vastly oversubscribed, having many times more resource requests than available resources. This makes it a paramount objective to efficiently utilize the available resources in order to complete as many activities as possible.

Current NASA space-based systems rely on ground-based human-intensive scheduling methods. Humans provide the main scheduling intelligence for constructing schedules. These schedules are then transmitted to the spacecraft to be executed. If the scheduling expertise and computers are ground-based, every anomaly that occurs onboard the spacecraft that incurs a schedule modification would cause significant time delays and efficiency losses. With the advent of more complex space-based systems such as the Space Station Freedom and beyond, a more efficient automated scheduling paradigm is necessary [Britt 1988].

1.2 The APS Project Scheduling Goals

The goal of the APS project is automated scheduling for space systems with proof-of-concept demonstrations on a power system testbed. In this process only the high level goals of the system are stated by the human operators, that is, which activities should be performed. This information is taken and the scheduler attains the goal of activities executed. The scheduler must

not only know how to generate the schedule, but must also know how to implement the schedule, and how to recover from system or load induced deviations in the schedule.

2. AIPS Implementation

Since scheduling cannot take place in a vacuum, the scheduler must be able to interact with other agents as well as cope with many operational concerns. The scheduler must be able to generate an initial schedule, it must have domain specific knowledge of how to implement the schedule, it must be able to reactively modify the schedule in the case that the assumed information of the state of the system changes, and it must be able to do this within metric time constraints.

2.1 What is being scheduled

The APS Brassboard is a power system testbed that contains a set of power supplies, switchgear, and loads that emulate a space-based power system. This hardware is controlled by a set of embedded controllers capable of configuring the state of the Brassboard. These controllers are then used to configure the Brassboard to supply power to the loads designated by the scheduler. Figure 1 shows the current configuration of the APS Brassboard. RBI's and RPC's are remote controlled switches and an *L* represents a load attached to the system.

The loads attached to the Brassboard are resistive load banks. In order to more closely emulate a space-based power system each load is given a set of attributes resembling those of a space-based system. Each activity (load) has a time varying profile of power demand, earliest start time and latest completion time constraints, priority, and temporal placement preference.

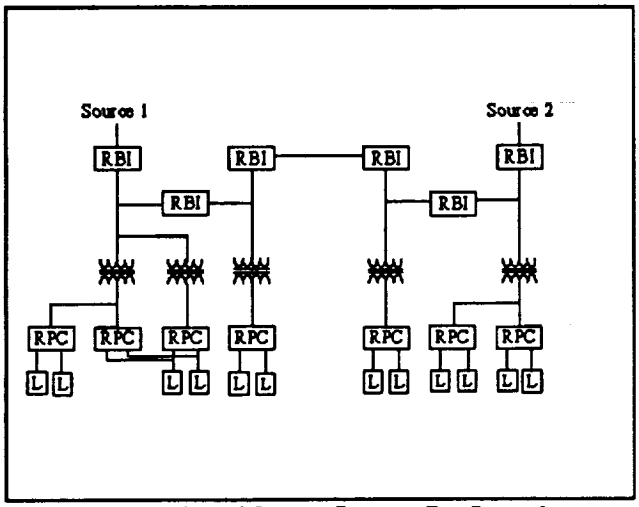


Figure 1 Brassboard Power System Configuration

2.2 Cooperation Between AIPS and APEX

AIPS is responsible for assigning the power requesting activities attached to the Brassboard temporal positions and resources without overallocating the available power. APEX is responsible for the implementation of the schedule generated by AIPS. In order to adequately model the interaction between APEX and AIPS, a set of protocols was developed to communicate different scheduling and rescheduling procedures. Protocols were developed to generate an initial schedule and modify executing schedules. Figure 2 shows a graphical representation of a schedule generated by AIPS. A chart showing the interaction between the three portions of the APS project is given in Figure 3.

2.3 Scheduling Methods Used

Two modes of schedule generation are needed for any integrated scheduling system. The ability to generate an initial schedule and the ability to modify (reschedule) an already executing schedule in the case of an anomaly. In the former case, a metric amount of time is allocated to the scheduler after which a solution must be returned. In the latter case, the rescheduling results are usually needed as soon as possible.

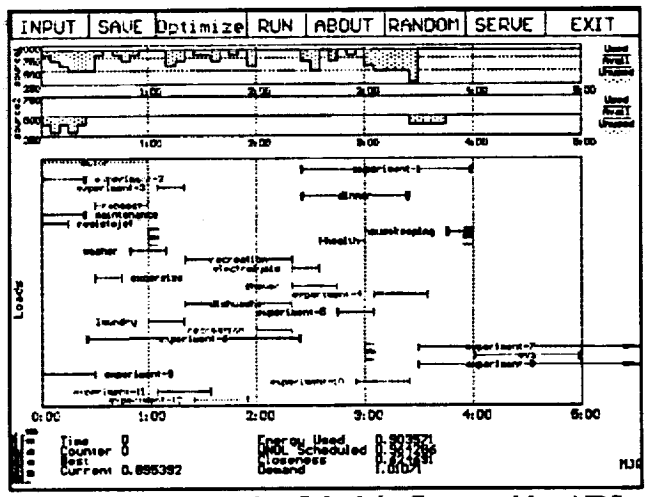


Figure 2 Representative Schedule Generated by AIPS

The AIPS scheduler has two modes of schedule generation used for scheduling and rescheduling. The scheduling engine is an incremental scheduler that uses a set of activity selection and placement heuristics [Sadeh 1989]. These heuristics are used to construct a schedule by taking each activity one by one, and determining where to place the activity on the timeline. These heuristics also form the basis of the rescheduling engine.

When the scheduler is given more time, it will use the same basic heuristics along with a Monte-Carlo type optimization method to generate multiple schedules

based on the heuristics. Since the heuristics use local goodness information, they do not produce globally good schedules. Small perturbations to these heuristic decisions will often improve the efficiency of the generated schedule. Each schedule is rated based on a goodness rating and when time to generate a schedule has run out, the best schedule (that has been saved in memory) is returned. With a relatively huge state space of solutions this method works quite well probing many portions of the state space that look promising based on the heuristics.

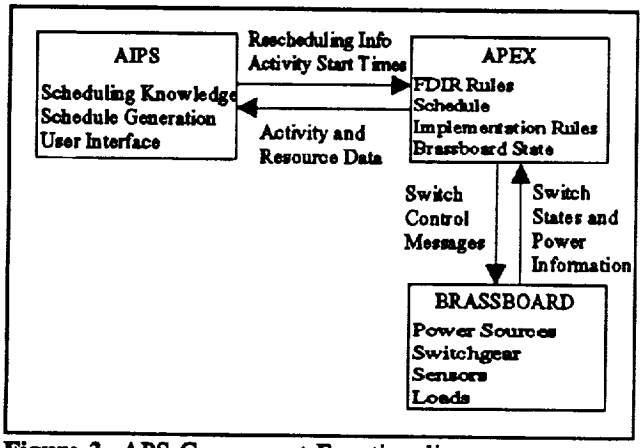


Figure 3 APS Component Functionality

Rescheduling must also be accomplished "non-nervously", that is, with as little deviation to the original schedule as possible [Biefeld 1990, Zweben 1990]. In systems with human interaction the original schedule should be followed as closely as possible in order to not disturb the humans interacting with the system. To accomplish non-nervous rescheduling, AIPS uses a set of heuristics that judge the amount of perturbation caused by a schedule modification versus the change of goodness of the new schedule.

3. Lessons Learned

Lessons were learned from the design of the scheduler, implementing a scheduler in a real system, and integrating scheduling with an FDIR system. Some of the lessons learned represented shortfalls in the original AIPS scheduler while others represented ideas for the improvement of the overall efficiency of the scheduling system.

3.1 Retrospective

Many of the concepts implemented in the AIPS scheduler worked quite well. Time was broken down into smaller scheduling horizons in order to make the problem solution feasible. Priorities were used to delineate between the relative value of activities. Time was partitioned at a granularity of five minutes. This was a reasonable simplification since the time for APEX and the Brassboard to be configured was on the order of one minute. The ability to schedule within metric time constraints was incorporated. A graphical interface was available for both schedule display and human-scheduler interaction.

The largest assumption made about the environment was that all temporal durations and resource requests are exact. In a real-life situation, if an activity requests 100 watts for one hour the probability of the activity using a constant 100 watts or lasting exactly one hour is quite small. The problems incurred may include undervoltage/overcurrent conditions caused by higher than expected demands as well as propagation of temporal constraints among activities caused by an extension of an activity's duration. The need for some type of temporal or resource padding is necessary. This padding decreases schedule efficiency although it may improve overall implemented schedule efficiency since the schedule will not have to be modified as often with the padding added.

3.2 Perspective

Much was learned about scheduling, but even more was learned about implementing a schedule in an automated domain. The whole object of scheduling is to produce the best overall system efficiency. In order to increase the efficiency of the implemented schedule, most new ideas point to the need for the ability for real-time reaction in the scheduler [Johnston 1989]. Here are three examples.

Conventional schedulers use temporal padding to increase the probability of executing a schedule. Temporal uncertainties cause the forward propagation of predecessor/successor constraints and resource availability. If activities are padded, and this padding is not used, it is wasted. It may be possible, however, to assign this temporal position/resource to another activity. This would entail moving another activity forward in time to fill the temporal position/resource left unused by the previous activity. This demonstrates a need for reaction in the scheduler.

Suppose a 500 watt cooling fan operates only when the experiment temperature rises above a certain threshold. This may only operate 10% of the time (10% duty cycle). How can the resources be allocated to prevent oversubscriptions? If 500 watts are continuously allocated 90% of this energy will be wasted (of course, a conflict free schedule is guaranteed). Energy balancing between multiple duty cycle activities can be used, but problems arise if all these activities turn on at the same time. Reaction is needed to delay some of these events if they desire to consume power when it is not immediately available. In addition, there is the possibility of

performing energy balancing in the power system domain. With energy balancing however, it is necessary to use a storage type resource such as a battery.

When using reaction, think about the "moveability" of an activity. In a Space Station domain, a dishwashing activity is much more moveable than a medical experiment using two crew members and various ground-based experts. The dishwashing activity has very few attached dependencies while the medical experiment would require the movement of many human interactors. The dishwashing activity is easier to move and a small temporal position change will not affect it as long as the dishes are washed before the next meal. This information can be used to make reactive modifications to the schedule without impacting the humans who will have to interact with the system.

The ideas of temporal padding usage, duty cycle balancing, and activity moveability will allow for more efficient use of limited resources. Of course, a scheduler (and testbed architecture) that allows these ideas to be implemented remains to be built and tested. The next section will briefly describe this new scheduler.

4. Implementing the Lessons Learned

The SCRAP scheduler is currently under development. General improvements in the representation of the SCRAP scheduler include multiple resources, multiple resource types (capacity, consumable, and storage), one second time granularity, activities broken into tasks, and multiple levels of schedule abstraction. Since the previous section showed a need for reaction, a scheduling paradigm that makes reaction easier would be beneficial.

4.1 Prediction vs. Reaction

Two general categories can be delineated in scheduling: predictive and reactive systems. Predictive scheduling allows the efficient allocation of available resources to activities by generating schedules based on predicted knowledge of the activity and resource states. This type of scheduling works well in static domains but is often hard to implement and less efficient in complex, uncertain, and dynamic domains. Reaction provides easier implementation in dynamic domains, but sacrifices resource usage efficiency caused by the lack of knowledge used to generate schedules.

In most real world problems a combination of static and dynamic domains exist. For example, a completely reactive scheduler might have no information on predicted resource demands of an activity, while a completely predictive scheduler would assume exact temporal durations and resource requests. Usually, a combination of these methods are used with a predicted resource level that provides an allowance for deviations from that level. This would point to the use of a combination of reaction and prediction. All schedulers that operate in a real domain actually combine the two, but the idea of SCRAP is to provide a framework that allows these ideas to be implemented efficiently.

4.2 How to Combine Prediction and Reaction

Even though building an initial schedule is computationally intense, the need to continuously modify the schedule during execution is even more difficult because of the tighter time constraints in the rescheduling domain. When rescheduling, all temporal and resource constraints propagate forward causing even more conflicts in the schedule, also known as the *ripple effect*. Propagating temporal and resource constraints during a reschedule clobbers previously computed future portions of the schedule. If rescheduling occurs often, the entire precomputed schedule may be recomputed by the rescheduling engine. This is an extreme case but proves the point that it may not be necessary to construct the initial schedule with a great level of detail. Therefore it may be wise to schedule far term activities with less effort or detail than near term activities. In the SCRAP scheduler this is accomplished by using multiple levels of abstraction when scheduling activities. Further into the future the schedule is constructed abstractly, while nearer to the execution time more precision is used. Also, more in-depth scheduling methods are used for times nearer to the execution time than for times further into the future.

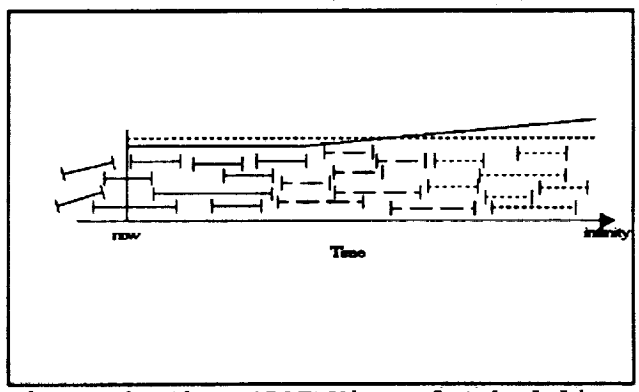


Figure 4 The SCRAP View of Scheduling

Multiple abstractions based on temporal distance from the execution time will allow for more efficient forward temporal propagation of constraints in the schedule since less information is used for the future portions of the schedule. The future portions of the abstractly generated schedule serve as a partially computed schedule when it comes time to actually schedule at a more precise level. The scheduling timeline can be looked at as a rolling horizon, with the future coming closer to

the present as time ticks during execution.

Figure 4 graphically shows the general idea of SCRAP. The timeline moves in conjunction with the movement of "real" time. Time "now" is the current execution time of the schedule. Time "infinity" is some time very far in the future. The gantt chart shows I-beams at different levels of scheduling abstraction. The solid lines are precisely scheduled, the dashed lines are scheduled at a medium abstraction, while the dotted lines are abstractly scheduled. Resource oversubscriptions are allowed in the future since the schedule in those areas has not been computed more abstractly. Nearer to the time of execution, more scheduling effort and precision is used and these resource conflicts will be eliminated.

Many of the reactive situations stated in the lessons learned section can be more easily implemented using the SCRAP paradigm. In an automated domain the scheduler has much more control over the executing schedule. This control along with the ability to efficiently modify the schedule during execution will allow for an overall implemented schedule efficiency increase.

5. Conclusion

The Autonomous Power System project at the NASA Lewis Research Center is an ongoing effort to demonstrate the use of knowledge-based diagnosis and scheduling software in advanced space-based electrical power systems. The APS project has completed one development iteration. A scheduling system was developed for the APS project and integrated with an FDIR system and hardware. The original AIPS scheduler was successful as a learning tool and a new improved scheduler is being developed. Many new ideas for increasing the implemented schedule efficiency will be realized using the SCRAP paradigm. The SCRAP scheduling paradigm will allow for more efficient use of the available resources.

Acknowledgements

This works was performed under NASA contract NAS3-25266 with Jim Kish as Technical Coordinator.

References

[Biefeld 1990] Biefeld, E., Cooper, L., "Operations Mission Planner: Final Report", JPL Publication 90-16, March, 1990.

[Britt 1988] Britt, D.L., Gohring, J.R., Geoffrey, A.L., "The Impact of the Utility Power System Concept on Spacecraft Activity Scheduling", Proc. 23rd Intersociety Energy Conversion Engineering Conference, 1988.

[Johnston 1989] Johnston, M., "Knowledge-Based Telescope Scheduling", In Knowledge-Based Systems in Astronomy, Springer-Verlag, 1989.

[Ringer 1991a] Ringer, M.J., Quinn, T.M., and Merolla, T., "Autonomous Power System: Intelligent Diagnosis and Control", Proceedings of the NASA Goddard Conference on Space Applications of Artificial Intelligence, 1991.

[Ringer 1991b] Ringer, M.J., "Autonomous Power System: Integrated Scheduling", Proceedings Space Operations, Applications, and Research Symposium, NASA Johnson Space Flight Center, Houston, Texas, July 1991.

[Sadeh 1989] Sadeh, N., and Fox, M.S., "Focus of Attention in an Activity-Based Scheduler", In Proc. NASA Conference on Space Telerobotics, Pasadena, California, 1989.

[Zweben 1990] Zweben, M., Deale, M., and Eskey M., "Anytime Rescheduling", NASA Ames Artificial Intelligence Branch Technical Report, February 1990.

Planning for the Semiconductor Manufacturer of the Future

Hugh E. Fargher & Richard A. Smith
Semiconductor Process Development Center
Texas Instruments, Inc.
P.O. Box 655012, MS 3635
Dallas, TX 75265

Introduction

Texas Instruments (TI) is currently contracted by the Air Force Wright Laboratory and the Defense Advanced Research Projects Agency (DARPA) to develop the next generation flexible semiconductor wafer fabrication system called Microelectronics Manufacturing Science & Technology (MMST). Several revolutionary concepts are being pioneered on MMST including new single-wafer rapid thermal processes, in-situ sensors, cluster equipment, and advanced Computer Integrated Manufacturing (CIM) software. The objective of the project is to develop a manufacturing system capable of achieving an order of magnitude improvement in almost all aspects of wafer fabrication [1]. TI was awarded the contract in October, 1988, and will complete development with a fabrication facility demonstration in April, 1993.

An important part of MMST is development of the CIM environment responsible for coordinating all parts of the system. The CIM architecture being developed is based on a distributed object oriented framework made of several cooperating subsystems. The software subsystems include: Process Control for dynamic control of factory processes; Modular Processing System for controlling the processing equipment; Generic Equipment Model which provides an interface between processing equipment and the rest of the factory; Specification System which maintains factory documents and product specifications; Simulator for modelling the factory for analysis purposes; Scheduler for scheduling work on the factory floor; and the Planner for planning and monitoring of orders within the factory.

This paper first outlines the division of responsibility between the Planner, Scheduler, and Simulator subsystems. It then describes the approach to incremental planning and the way in which uncertainty is modelled within the plan representation. Finally, current status and initial results are described.

Planner/Scheduler Division of Responsibility

One role of the Planner is to plan and predict work completion dates, given a required confidence level, set of plan goals and the current state of the factory. This requires that the plan representation model factory resource utilization over time, and that the plan be continually updated to reflect unexpected events such as machine failure. This role is not provided by the Scheduler, which performs more locally based decision making.

As part of this role, the Planner is able to warn the user of the impact of unexpected events. For example, the Planner can determine whether work completion dates are slipping, well in advance of their quoted delivery dates. The user can also be warned of any work which has been automatically replanned due to unexpected events, so that they may request changes to the plan if required. Automatic replanning of work will remain an option to be invoked if desired by the user.

The ability to request plan changes is another key Planner role which is not provided by the Scheduler. 'What-if' plan changes refer to requests such as putting a machine on hold or introduction of new work.

Finally the Planner constrains work release into the factory, based on the current plan being executed. This is important since early release of work carries the penalty of increased WIP and early completion of work is undesirable. The high level plan representation does not allow the Planner to determine the precise moment for work release, which may be based on low level factory data such as machine queue sizes. This is an important role for the Scheduler, since work released early will only increase WIP by placing work on a queue. Work release is accomplished by the Scheduler requesting more work from the Planner, with the Planner satisfying the request as best as possible given the work planned for release over the next chosen time interval.

Another role of the Scheduler is to make sequencing decisions for work on the factory floor, based on details such as queue sizes, machine setups, and so forth. Although such decisions may be based on currently planned ship dates, this service cannot be provided by the Planner (which does not distinguish between identical resources in the plan representation). Finally, the Scheduler is responsible for tracking work in process.

- The Planner influences the schedule being executed by constraining work release and predicting work completion dates, which may be used in Scheduler dispatch decisions. However, work released into the factory cannot be directly influenced by the Planner. The Scheduler provides important feedback to the Planner by tracking work in process. This can be used to update cycle time estimates used by the Planner, and to warn of tardy work which may cause replanning.

Planner/Simulator Division of Responsibility

Both the Planner and Simulator systems provide the user with the ability to determine the consequences of 'what-if' requests. However, the allowed requests differ fundamentally between the Planner and Simulator.

Planner 'what-if' requests may be made on a single plan only, and result in incrementally updating the existing plan to satisfy the request. Typically, the existing plan reflects the current state of the factory. Rapid feedback is required, since the requests may refer to the effect of putting a machine down in the near future for maintenance, or the effect of introducing a new hot lot onto the factory floor. These requests must be rapidly evaluated if a manager is to fully benefit, since they may require immediate attention. The ability to have multiple 'what-if' plans open simultaneously will also be important if possible plan options are to be compared.

In contrast to this, Simulator 'what-if' requests are typically performed by running a suite of simulations, using factory conditions possibly selected at random from a set of work release or machine failure distributions. Feedback is not required immediately since simulation results typically refer to changes which are not immediately put into practice. Example requests may include the effect of introducing new machines into the factory, or re-training several of the operators.

The Planner system may interact with the Simulator in two distinct modes. First, by providing a static work release plan, generated using some initial factory status, which provides the Simulator with a work release time table. This is particularly important for verifying the plan model and algorithms, since simulated work completion should match plan predictions if the Planner is correctly predicting processing capacity. Second, by providing a dynamic release plan, which is updated in response to simulated events (such as machine failure) during simulation execution. This is important for verifying Planner response times, which must remain small if the Planner is to be truly 'reactive'.

Approach to Incremental Planning

A plan representation has been chosen which models the manufacturing environment in enough detail to achieve the planning functions, while allowing incremental updates due to replanning. The following section outlines the representation, along with the search algorithm used to generate and update plans.

Modelling the Plan

The plan representation is based on the processing capacity of resource groups within the factory, divided into contiguous time intervals. Each resource group has an associated set of processing capabilities which every member of the group is able to perform. Since a single semiconductor manufacturing machine may perform several different processes, a machine may be a member of several different resource groups. Each resource group is represented over contiguous time intervals, where the planned processing commitment and remaining capacity is recorded.

The plan representation does not distinguish which resource, within a resource group, is planned to process a particular piece of work represented within a plan. The representation simply commits processing time for the whole resource group to a particular piece of work. Furthermore, the plan representation does not sequence processing within each time interval, only between time intervals. In this way, the level of detail modelled by the plan is a function of both resource groups and time interval sizes. If resource groups contained only one resource, and all time intervals were shorter than the shortest processing step, the plan representation would reduce to a Gantt chart describing the processing schedule for each resource. If, on the other hand, the entire plan were covered within a single time interval, the representation would reduce to the model frequently used for planning within semiconductor manufacturing [2]. The 'time-phased' representation outlined above lies somewhere between the two extremes.

The plan representation must accurately reflect factory capacity, projected forward from the current clock time. To ensure this, all planned processing for the earliest time interval is removed from the plan representation when the clock time exceeds the time interval upper bound. Planned processing is then compared with the current state of the factory (via the WIP tracking system) and the system user is warned of any work which appears tardy on the factory floor. Finally, the processing capacity of resource groups within the first plan time interval reduce linearly with time, to reflect the constantly increasing clock time.

The Planning Algorithm

The planning algorithm is divided into two parts, that of determining the sequence of work to be planned (given its due-date, customer priority, etc), and incorporating the required processing into the plan representation (given the current resource group commitments, type of planning requested, and constraints imposed on which time intervals processing may be planned for). Planning may use the existing plan representation as a starting point, or some user defined

variation if multiple 'what-if' plans are to be explored.

Deciding the sequence of work to be planned ultimately determines the overall product mix, and is determined by an ordered list of goals in which the first unsatisfied plan goal is used to sequence work for planning. The ordered goal list may be thought of as defining the Planner 'strategy'. Each goal sequences work using its associated heuristic, which is designed to guide plan generation in favor of satisfying the goal. All goals have numerical values, which must be met by the plan if the goal is to be satisfied. Once a goal is satisfied, processing moves to the next unsatisfied goal. By 'interleaving' similar goals in the ordered list, the Planner strategy can be used to satisfy several different goals, while ensuring that the plan never deviates much from satisfying any one goal [3].

Once work has been sequenced for planning, it must be incorporated into the time-phased plan representation. The resources required for each processing step must be committed over some time interval so that no resource group is overutilized and all constraints on processing are satisfied. Plan independent constraints, such as processing times and required resource groups, are determined by querying the Specification system. Within these constraints, the planning search algorithm determines precisely in which time interval to commit resource groups for each processing step.

The planning search algorithm uses a work representation in which wafer processing is divided into discrete segments, where each segment represents processing on resources which may be completed within one time interval of the plan representation. Division of wafer processing into segments is performed by calculating which segment each processing step would lie in if processing were distributed evenly over the entire wafer cycle time. Since the wafer cycle time is greater than the minimum theoretical processing time, such a representation accounts for the expected queue time during wafer processing. Each search operation either inserts or removes segments from the plan representation, terminating when all required segments for processing work have been inserted, or when no further processing capacity remains.

The search algorithm uses a modified beam search with chronological back-tracking. Maximum beam width is determined by the ratio of measured wafer cycle time to minimum theoretical cycle time, since the greater the ratio, the greater the choice of time intervals for planning each processing segment. The search space is further reduced by constraining the beam width to increase linearly with search depth. One advantage of this is that solutions which appear unpromising at an early stage in the search are quickly discarded, whereas those which appear more promising are more thoroughly searched. Another advantage is that 'disjoint' plan representations, in which no resources may be available for an extended period of time due to factory shut-down, do not prevent new work from being planned, as long as sufficient processing capacity exists while the factory is operational.

Replanning due to unexpected resource failure requires reasoning at both the goal list and the search algorithm level. To ensure that resource groups are not overutilized in the plan representation when a resource goes down, currently planned work must be sequenced for replanning. This is performed by removing work until resource utilization levels are not exceeded, and then replanning this work to be released at a later date.

Results

Table 1 illustrates performance when using this algorithm to plan new work into an existing plan. The table shows the fraction of successful search nodes (for which a processing segment was successfully inserted into the plan representation), failed nodes (for which there was not enough processing capacity in the attempted time interval), and backtracked nodes. The results illustrate that even for a highly utilized factory the search required to plan new work, for which there is processing capacity available, is not prohibitive. Furthermore the percentage of backtracked nodes does not continue to increase with committed utilization. In a semiconductor fabrication facility an average of 80% utilization across all machines is considered very high. The results in this case assume that human operators are not a bottleneck resource.

Table1:

Committed Utilization Percent	Successful Node Percent	Failed Node Percent	Backtracked Node Percent
10%	100%	0%	0%
20%	100%	0%	0%
30%	47%	40%	13%
40%	44%	44%	12%
50%	36%	50%	14%
60%	35%	52%	13%
70%	32%	56%	12%
80%	30%	58%	12%

Approach to Modelling Uncertainty

The plan representation must be able to model the uncertainty inherent in work cycle-times, since such cycle-times often form the best available data for planning. The following section outlines the approach taken to representing uncertainty in the planning process.

Domain Uncertainty

Two areas of uncertainty are tackled by the Planner, both corresponding to data which is represented by a probability distribution. The first is wafer yield, which is recorded as the probability of manufacturing n good chips given the starting number. The second is cycle time, which is recorded as the probability of completing all manufacturing steps on a wafer in a given time.

This section outlines how cycle time distributions are used within the Planner.

The objective of the Planner is to predict work completion dates to within some given confidence, which may be used to negotiate with customers. For example, an order may be represented within the plan so that it completes processing on Friday to within a 50% confidence level, but on the following Monday to within an 80% confidence level.

Modelling Uncertainty

Uncertainty is modelled within the Planner by reinterpreting the plan representation in terms of fuzzy sets [4]. Resource group utilization for a given piece of work has a degree of membership within each time interval, which reflects the expected utilization of resources for this work during the time interval. For example, the total cycle time distribution for wafer processing may be interpreted as the probability distribution for completing the final processing step at a given time. This can be modelled within the plan representation by assigning degrees of membership between time intervals to match the given probability distribution for the final processing step. The advantage gained by this interpretation is two-fold. First, computation on fuzzy sets is much less expensive than on probability distributions. Second, cycle time uncertainty within the time-phased representation means that resources committed to processing a given set of wafer steps within one time interval will very likely process some of those steps within other time intervals. This closely matches the concept of membership degree within fuzzy set theory.

To enable the Planner to reason at this level of detail, knowledge of the total processing cycle time distribution is required, as well as some estimate of the distributions required to complete each time interval's worth of processing. Intermediate processing steps for which data is recorded in semiconductor manufacturing are traditionally referred to as 'log-points'. If log-point data were available for processing steps within each Planner time interval, this data could be used to model the distributions for required processing over all time intervals. However, this log-point data may not be available for all processing steps, only the final cycle time. For this reason, the Planner uses an algorithm to estimate log-point cycle times, given the final cycle-time which is available as a distribution.

The algorithm attempts to decompose the final cycle time probability distribution into cycle time distributions for each successive time interval throughout a wafer's processing. This is done so that:

- Interval cycle time distribution variance increases with successive intervals, to reflect increasing future uncertainty.
- Interval cycle time variance is bounded by the final cycle time variance.
- The final computed interval cycle time distribution matches the input cycle time distribution.

The algorithm represents distributions using fuzzy numbers and performs all calculations using fuzzy arithmetic. This approach is based on the job shop scheduling system FSS [5] which also uses fuzzy arithmetic to model increasing uncertainty in generating future schedules. A key advantage with this approach is that calculations on distributions can be performed extremely rapidly. The algorithm has been tested against simulated results, as described in the next section.

Once time interval cycle time distributions have been calculated for a given wafer processing route, they are used to 'fuzzify' the resources committed to processing steps during each time interval of the plan representation. This is achieved by using the fuzzification operator (defined for fuzzy set theory) and results in resource utilization being 'smeared out' within the plan representation. This reflects the uncertainty in the time at which planned processing will actually take place in the factory.

Once work has been planned for a wafer with a given processing route, the final cycle time distribution is used to quote the completion date to within a given confidence level. For example, if 50% of the final time interval processing has been planned to complete by Friday, the wafer may be quoted to complete on Friday with a 50% confidence level. In fact, the confidence level associated with any delivery date may be quoted.

Finally, measured cycle time distributions provide one important method for feedback to the Planner from the outside world. Cycle time distributions may be updated incrementally as wafers complete processing for each type of manufactured technology. Furthermore, since cycle times are closely related to WIP and product mix, distributions used for planning should be chosen to reflect current conditions. However, planning work in semiconductor manufacturing has shown the difficulty in predicting cycle times up-front, which are highly sensitive to conditions such as resource status and WIP levels.

Results

Table 2 illustrates the cycle time mean and variance, for part of a processing sequence completing during a given time interval, calculated using simulation and the proposed fuzzy arithmetic algorithm. The simulated CT mean and variance were calculated by performing a series of simulations, forward in time, based on known time interval cycle time distributions. The resulting final cycle time distribution (at time interval number 5) was then plugged into the algorithm to generate the set of estimated intermediate time interval cycle time distributions. The algorithm estimated time interval distributions were then compared with the simulated distributions by measuring their mean and variance. Time units are measured in numbers of time intervals. Agreement between simulated and

fuzzy means remains close, while agreement between simulated and fuzzy variance improves over several time intervals. Agreement improves as CT variance increases due to the greater number of members in the fuzzy number used to represent the distribution. We intend to explore several possible variations on the algorithm in an attempt to improve agreement.

Table2:

Time Interval	Simulated Mean	Fuzzy Mean	Simulated Variance	Fuzzy Variance
1	1.11	1.00	0.10	0.00
2	2.21	2.04	0.20	0.04
3	3.30	3.10	0.28	0.16
4	4.40	4.07	0.37	0.37
5	5.48	5.48	0.45	0.45

Current Status

A prototype CIM system was built as one of the first tasks of the CIM program. This helped with the overall system design, as well as provide a platform in which to plug prototype subsystems and get feedback from potential users. However, only small parts of each subsystem had been designed at this stage.

All CIM subsystems have now been designed and documented, and are currently being implemented in Smalltalk. The MMST Planner is currently about 25% of the way through the development phase. Interfaces between subsystems have not yet been completed, so many of the results shown above have relied on 'stubbing' subsystem functionality external to the Planner. Functionality has been stubbed to match the expected external system performance as closely as possible, and is based on a detailed scenario analysis for MMST [6]. In particular, wafer processing requirements and resources have been chosen to reflect those described in the analysis.

The Planner mechanism that requires the most development is the 'what-if' capability. Several design approaches have been documented, although determining the best approach (for example, in terms of speed of response) will require experimental measurements which may only be obtained by implementation.

Finally, full CIM installation and integration within a TI fabrication facility remains as the final stage in the MMST program.

Conclusion

A reactive planning system for semiconductor wafer fabrication has been designed and partially implemented, as part of the MMST program, jointly funded by TI, Air Force Wright Laboratory and DARPA. The planning system has been designed to maintain a plan which is constantly up to date with the factory environment, and which can reason with uncertain data such as processing cycle time distributions. The planning algorithm generates plans using a variation on the traditional beam search, and models uncertainty using a fuzzy set approach. Initial results indicate that the system is able to incorporate new work into an existing plan without incurring a large amount of computationally expensive backtracking. However, further work will be required to verify plan results in an existing wafer fabrication environment, and to integrate the Planner with the rest of MMST.

Acknowledgements

This work was sponsored in part by the Air Force Wright Laboratory and DARPA Defense Science Office under contract F33615-88-C-5448.

References

[1] J.McGehee, D.Johnson & J.Mahaffey: 'Semiconductor manufacturing: a Vision of the Future', Texas Instruments Technical Journal, vol.8, no.4, pp.14-26, 1991

[2] PMDS Technical Report CSC-TR89-004, Texas Instruments internal report, 1989

[3] PMDS Memo 91-DR-01, Texas Instruments internal report, 1991

[4] A.Kaufmann & M.Gupta: 'Introduction to Fuzzy Arithmetic', Van Nostrand Reinhold Company, New York, 1985

[5] R.Kerr & R.Walker: 'A Job Shop Scheduling System based on Fuzzy Arithmetic', Proc. of 3rd Int. Con. on Expert Systems & Leading Edge in Prod. & Operations Man. pp.433-450, 1989

[6] J.McGehee: 'Scenario Analysis', Texas Instruments internal report, 1991

Uncertainty Management by Relaxation of Conflicting Constraints in Production Process Scheduling

Jürgen Dorn – Wolfgang Slany – Christian Stary

Christian Doppler Laboratory for Expert Systems
E184/2, TU Wien, A-1040 Vienna, Austria, Europe
Phone: +43-1-58801/{6127|6123|6124}
Fax: +43-1-5055304
E-Mail: {dorn|wsi|stary}@vexpert.dbai.tuwien.ac.at

Abstract

Mathematical-analytical methods as used in Operations Research approaches are often insufficient for scheduling problems. This is due to three reasons: The combinatorial complexity of the search space, conflicting objectives for production optimization, and the uncertainty in the production process. Knowledge-based techniques, especially approximate reasoning and constraint relaxation, are promising ways to overcome these problems.

A case study from an industrial CIM environment, namely high-grade steel production, is presented to demonstrate how knowledge-based scheduling with the desired capabilities could work. By using fuzzy set theory, the applied knowledge representation technique covers the uncertainty inherent in the problem domain. Based on this knowledge representation, a classification of jobs according to their importance is defined which is then used for the straightforward generation of a schedule.

A control strategy which comprises organizational, spatial, temporal, and chemical constraints is introduced. The strategy supports the dynamic relaxation of conflicting constraints in order to improve tentative schedules.

1 Introduction

The task of scheduling jobs and resources in a factory is difficult for mainly three reasons. First, one has to deal with the combinatorial complexity due to multiple ways of job accomplishment [6]. Second, conflicting objectives may hinder the definition of an undisputed optimality measure [11]. Finally, there is uncertainty in the execution of jobs due to the lack of knowledge about the exact physical facts underlying the production process. Thus, it becomes senseless to compute exact scheduling solutions. Often reactive scheduling is proposed as a solution to these problems [10]. To illustrate the situation, an existing scheduling task is described in the following.

In a joint project between the Alcatel-Elin Research Center Vienna and the CD-Laboratory for Expert Systems, an expert system was developed. It supports the technical staff of the Böhler steelmaking plant in generating weekly schedules for steel heats [2]. Side conditions are the same as for the approach proposed in this paper, with the difference that no attempt to handle uncertainty was made in this first expert system. Böhler is one of the most important European producers of high-grade steel. The plant produces tool steel, high-speed steel, and stainless steel. There are hundreds of different kinds of steel, with 42 chemical elements varying in their specification. The requirements concerning steel quality are very strong.

One problem in scheduling is that residuals of one heat in the electric arc furnace may pollute the next heat. As a general rule of thumb, it can be said that 3% of a chemical element in a heat remain on the electric arc furnace's wall, and 3% of the difference of this element in the first heat and the second heat will be assimilated by the second heat. Two heats that have similar shares of the element in question pose no problem. However, if the second heat has a much smaller percentage than the preceding one, the pollution by the residual from the first becomes too large to be compensated by decreasing the amount added to the second heat. This either means that the quality of the second heat will be badly influenced, or if the polluting element is expensive, that it will be wasted, and money is lost. In the following these two constraints are called compatibility rule. The compatibility rule is effective for all 42 chemical elements, but usually only 8 main elements are considered, since the others generally are not expensive, do not vary significantly, or have no great impact on the steel quality. Uncertainty arises because exact values for the chemical elements can very often not be mesured. Further constraints for the scheduling process are temporal, distribution control, spatial, and resource restrictions on and among the aggregates.

2 Uncertainty Management

One objective of the presented strategy is to schedule as many jobs as possible. In order to get the most important jobs scheduled, the evaluation function for an entire schedule must contain a factor representing the importance of jobs. Hence, an evaluation function is defined to assign an importance value to a schedule by adding up the importance values for each job in the schedule. These

No.	Name	Time	Type	Ni	Cr	Co	Mn	Fe	V	W	Mo
h_0	M100	5am		.1	1.2	.005	1.3	95	.1	.005	.005
h_1	A101			12.0	17.8	.25	1.8	69	.005	.005	2.8
h_2	S600			.2	4.3	.0005	.35	78	1.9	6.7	5.2
h_3	K460	11am	CC	.1	.6	.0005	1.15	94	.1	.15	.005
h_4	A506			8.0	17.5	.005	2.0	69	.005	.005	0.05
h_5	M238		BEST	1.2	2.1	.005	1.6	90	.005	.005	.25
h_6	M238		BEST	1.2	2.1	.005	1.6	90	.005	.005	.25
h_7	K116			.1	12	.0005	.4	83	.005	.005	.005

Table 1: Characteristics of given heats in the example

latter values are calculated by considering the resource requirements, due dates, and various other attributes of individual jobs.

A first schedule is generated straightforward by considering most important jobs first. The first schedule may not contain all jobs and still violate some constraints. In these cases, jobs in the schedule will be exchanged to find a proper schedule. A hill climbing search method is used to control this exchange. To compare solutions, an evaluation function based on the given constraints is needed. Fuzzy logic is a sound AI-technique to manage uncertainty as present in this problem [8, 12]. Since [9], and as recently as in [1], fuzzy logic has been successfully applied to knowledge-based scheduling. Our approach generalizes these former ones to include, beside temporal constraints, other kinds like chemical or organizational constraints.

In section 2.1, we propose a method how the given constraints may be represented by fuzzy sets and how an evaluation for a complete schedule is computed. Section 2.2 explains the generation of a preliminary schedule and the search for a better schedule. Such a schedule can only be found if constraints are relaxed, because many constraints are antagonistic. This relaxation will again be based on fuzzy sets.

A small example of the application is described to illustrate the used techniques. The example is restricted to one furnace and the planning horizon is only several hours. Additionally, only a subset of the given constraints is considered in order to reduce the complexity of the example. The existance of a schedule until 5am is assumed. The input is a list of jobs that should be scheduled. The first heat h_0 in the list is the latest job scheduled from the last scheduling process. The main ingredients of each order are given in table 1.

Three heats of table 1 have special characteristics that imply their classification as very important jobs. Heat h_3 is processed on the continuous caster (CC) and has a delivery date. The delivery date is 4pm, the overall treatment takes about five hours, and therefore the processing should start at 11am. Heats h_5 and h_6 shall be cast into big ingots with a special BEST[1]-treatment. This implies that they cannot be produced immediately one after the other. Instead, there should be a time interval of at least ten hours between them.

[1]BEST stands for Böhler Electro Slag Topping.

2.1 Qualitative Representation and Evaluation of Constraints with Fuzzy Logic

The constraints of the given application can be divided into three categories: Constraints on a particular job, temporal constraints, and constraints on the compatibility of jobs.

Constraints on a particular job are constraints based on required resources or aggregates. They are used to describe the importance of jobs. This importance of jobs is used later to control the generation of a preliminary schedule by scheduling the most important job first. In our sense, this importance is a combination of the difficulty to schedule a job in general and its urgency, that is to schedule it for the actual planning horizon. A job that requires a bottle-neck resource like the continuous caster is usually difficult to schedule. A job with a certain delivery date is important, because it must be scheduled in the planning horizon in which the delivery date falls. Jobs that are not important may be shifted to the next planning horizon. To schedule a shifted job eventually, it is necessary that the importance of the job increases over time. The range of fuzzy linguistic variables to represent importance is: *urgent*, *very important*, *important*, *medium*, and *not important*.

The classification of jobs in the list is dependent on the situation in the actual planning horizon. For instance, if for the actual planning horizon many jobs with a high chromium-nickel-alloy exist, then a high percentage of nickel (Ni) is no problem. On the other hand, when there are only few jobs with high nickel percentages, these jobs can be difficult to schedule.

Temporal fuzzy values can be used to describe that jobs are too early or too late. The fuzzy value describes a degree of uncertainty in both direction. One can identify the following linguistic variables: *very early*, *early*, *in time*, *late*, *very late*. For the evaluation of a schedule it makes no difference whether jobs are too early or too late. Therefore, the five variables are mapped onto three: *in time*, *nearly in time*, and *not in time*. Representation of temporal constraints with fuzzy sets is discussed in detail in [1, 3, 4, 9].

The compatibility of two jobs integrates several factors: Different chemical elements, and the work load of workers. The compatibility between two jobs is calculated by first evaluating the compatibility for each factor separately, in order to get restricted compatibility measures. Accordingly, we define six fuzzy sets for the global as well as for each restricted compatibility: *very*

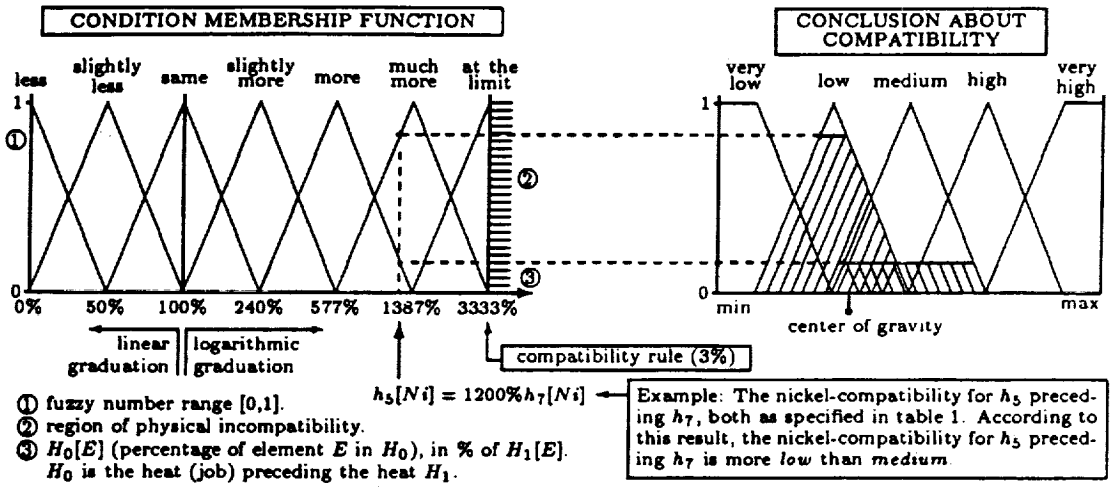


Table 2: Fuzzy inference to compute chemical compatibility between two heats

high, *high*, *medium*, *low*, *very low*, and *no* compatibility. The latter is a special case, since a sequence being classified incompatible can never be scheduled in this order because of hard chemical constraints to be observed.

The compatibility calculation for nickel is shown in table 2. The condition parts of the fuzzy inference rules used for this calculation contain statements about the percentage of some chemical element in the first heat compared to the following heat. In the example taken from table 1, the heat h_5 must contain $h_5[Ni] = 1.2\%$ of the chemical element nickel, whereas heat h_7 should contain only $h_7[Ni] = 0.1\%$. The relative percentage of $h_5[Ni]$ is therefore 1200% of $h_7[Ni]$. The question is, considering only nickel, whether the sequence h_5 preceding h_7 is allowed or not, and if yes, how good this sequence is. To decide this with the given fuzzy inference rules, the linguistic variables and numeric values must be matched. This is done with a fuzzy membership function as defined in table 2, both for the condition and for the conclusion part. In the example, the numeric input of 1200% relates more or less with the linguistic variables *more* and *much more*. Following the dotted lines to the conclusion membership functions for such rules as "IF the percentage of chemical element E in heat H_0 is *more* than in heat H_1, THEN the E-compatibility of H_0 preceding H_1 is *medium*" or "IF the percentage of chemical element E in heat H_0 is *much more* than in heat H_1, THEN the E-compatibility of H_0 preceding H_1 is *low*", membership functions $low_{[Ni]}(h_5, h_7)$ and $medium_{[Ni]}(h_5, h_7)$ appear as a result of the calculation. Their combination is a new membership function defining the nickel-compatibility of h_5 preceding h_7. In order to compare the result with other compatibilities, it must be defuzzified. This can be done by calculating the center of gravity of the surface and then taking the value of its x-coordinate as the result, a standard method in fuzzy calculation [8].

The conditions of the fuzzy inference rules consider only relative values for the percentage of elements like nickel in the two compared heats. Absolute values are of minor interest for the compatibility problem, but could easily be modeled by introducing more complex three-dimensional membership functions. We chose a half-logarithmic graduation to be able to handle those relative values. Since the compatibility rule is asymmetric and only restricts the second heat to a minimal value for a certain chemical element, which must at least be present in this heat, the graduation is asymmetric, too, and only logarithmic on the right half. Beside simplifying the visualization, this logarithmic scale has an additional positive effect, since positions on the right side of the 100% mark that are still near the center, are preferred and get more attention per unit than positions more close to the physical limit on the far right. This reenforces the natural meaning of the fuzzy linguistic variables positively.

The fuzzy inference rules like those used in table 2 give several fuzzy judgements how compatible the heats are. These judgements in form of membership functions can be simplified to the linguistic variable to which the judgement mainly pertains. The resulting fuzzy-values can all be combined by computing a weighted mean of the defuzzified values to get one overall value for the two heats:

$$comp(H_i, H_j) = \sum_{E \in \{Wl, Ni, Cr, \ldots\}} g(E) comp_{[E]}(H_i, H_j)$$

In this formula, $g(E)$ is the normalized weight of a rule and E is a member of the set of all factors influencing the compatibility, namely work load (Wl) and the 42 chemical elements like nickel or chromium. This computation is done for every pair of jobs that may be scheduled. The result is a matrix of fuzzy values where the fuzzy values describe how compatible the sequence of the job of a column after the job in a row is according to all rules. After defuzzifying the matrix, numeric values that can be rematched with the original fuzzy linguistic variables can be written in the matrix.

Table 3 shows the matrix for the example. It will be used for the construction of the preliminary schedule and during the improvement process. To evaluate

$H_0 \setminus H_1$	h_1	h_2	h_3	h_4	h_5	h_6	h_7
h_0	low	medium	high	low	high	high	medium
h_1	–	very low	very low	very high	low	low	very low
h_2	very low	–	very low	low	very low	very low	very low
h_3	medium	high	–	medium	high	high	high
h_4	high	very low	very low	–	medium	medium	medium
h_5	medium	high	high	medium	–	very high	medium
h_6	medium	high	high	medium	very high	–	medium
h_7	medium	medium	low	medium	medium	medium	–

Note: H_0 precedes H_1, e.g., the compatibility of heat h_3 preceding h_2 is *high*, whereas h_2 preceding h_3 is *very low*.

Table 3: Compatibility matrix for heat sequences

compatibility:		high		medium		low		high		very low		low	
	h_0		h_5		h_7		h_3		h_2		h_1		h_6
time:	5am		7am		9am		11am		1pm		3pm		5pm

Table 4: Preliminary schedule for example heats

schedules during improvement steps, it is necessary to compute an evaluation function for the compatibility of the entire schedule. This can be achieved with a fuzzy and-operator.

2.2 Generating a Schedule

To generate a preliminary schedule, the jobs are classified regarding their importance. Then they are scheduled in the sequence of their importances. Scheduling a job means assigning a temporal interval to it. These intervals are spread over the entire planning horizon because of temporal and resource constraints. During the scheduling process, empty intervals are created between scheduled jobs. The compatibilities with the jobs before and behind this empty interval are not considered. If empty intervals with a duration of approximately one job are created, they are filled with compatible jobs as long as there are some available.

Usually, some jobs can not be scheduled, because no interval exists where they would not violate some compatibility constraints. In addition, some empty intervals remain in the schedule, and the compatibility between the jobs adjacent to this interval is usually bad. In order to cope with the given complexity, instead of backtracking to the last scheduling decisions, such a preliminary schedule is repaired or improved by exchanging jobs.

In the list of jobs given in table 1, job h_3 has a delivery date. It will be scheduled first. Thereafter, jobs h_5 and h_6 will be scheduled, because they are very difficult jobs. They include a special treatment and therefore need a long time span between each other. Fortunately, one of them fits well after h_0. h_5 is choosen to be the successor of h_0. The other is scheduled at the end of the planning horizon. The job h_7 is scheduled between h_5 and h_3 to close the empty interval between them. Heat h_2 is another difficult job for the actual planning horizon, because most heats have high percentages of nickel (Ni) and chromium (Cr), and h_2 has only small amounts of both. Moreover, h_2 has large amounts of vanadium (V) and tungsten (W). The best place for h_2 is behind heat h_3. An empty interval remains between h_2 and h_6. There exists no heat in the given list that fits between h_2 and h_6. To fill the interval, h_1 is scheduled between h_2 and h_6. Heat h_4 remains for the next planning horizon. This preliminary schedule is illustrated in table 4.

To improve a schedule, a measure for schedules that evaluates which schedule of two is the better one is needed. Unfortunately, the violation of constraints can have far-reaching consequences. The violation of a temporal constraint can cause the need for more resources such as additional energy, or rescheduling in subsequent plants. The violation of chemical compatibility can result in the loss of a heat which would be a heavy financial damage. On one hand, one must consider hard constraints that may not be relaxed, and on the other hand constraints must be relaxed to a certain degree in order to get a feasible schedule with as many jobs as possible. In order to evaluate all these antagonistic constraints, an evaluation function based on the introduced fuzzy values is needed.

The actual schedule is called the "currently best schedule". To improve a given schedule, a potential constraint violation that could be improved is searched. In the example, such a violation is found between heat h_2 and h_1. Therefore one of them is taken out of the schedule. If h_1 is taken, no other heat is found in the whole list that would fit better. Therefore h_2 is taken out of the schedule and another heat that fits better is searched. h_2 can be replaced by h_4 and one gets the schedule shown in table 5 which is the "current best schedule", because the evaluation function based on fuzzy sets assigns a better value to this schedule than to the old one.

In the next step, the compatibility of h_7 preceding h_3 is found low. Therefore a job that would be a better predecessor of h_3 is searched. Heat h_5 is the best fit. There are two possibilities: a heat that can be processed between h_0 and h_5 can be searched, or h_3 can be simply shifted in time. Regarding only the compatibility constraints, the best solution would be to exchange h_5 and h_7. Unfortunately, another constraint is violated in this

compatibility:	high	medium	low	medium	high	low	
	h_0	h_5	h_7	h_3	h_4	h_1	h_6
time:	5am	7am	9am	11am	1pm	3pm	5pm

Table 5: Intermediate schedule for example heats

compatibility:	high	high	high	medium	high	low	
	h_0	h_5	h_3	h_7	h_4	h_1	h_6
time:	5am	7am	9am	11am	1pm	3pm	5pm

Table 6: Final schedule for example heats

case: The interval between the heats h_5 and h_6 should be at least 10 hours. Therefore heat h_3 will be shifted. Since delivery dates may be shifted up to two hours, heat h_3 can start at 9am and heat h_7 started after h_3. The result is the schedule shown in table 6.

Every exchange of jobs in the schedule can be interpreted as one operator in a search process. The search for better schedules can be guided by heuristics based on our evaluation function. This heuristic search is a kind of hill climbing method. Unfortunately, the disadvantage of a hill climbing method is that it can be caught in local maxima. In [7] a technique called TABU search is described that can be used to overcome this problem.

The search will end if no more constraint violations can be detected, or no further improvement can be achieved. It is not that easy to say that no further improvement can be achieved. Here it makes sense to define a distance function between an optimal schedule where all compatibilities would be very high, and all the other constraints would be observed too. If there is such a distance function, the search effort can be restricted by a ratio between distance and search effort. It would be fruitless to invest much more search effort if only a small distance exists. On the other hand, if the distance is large, one should search longer for a better schedule.

3 Conclusion

Due to highly unreliable knowledge and conflicting objectives in scheduling applications, mathematical-analytical methods as used in Operation Research approaches are insufficient in many cases. We have illustrated this very problem for a steelmaking plant. In order to overcome this deficiency we have developed a solution which combines two sound AI-techniques for problem solving: Approximate reasoning and constraint relaxation.

We believe that, using the described techniques, the development cycle for scheduling expert system becomes shorter, the knowledge representation easier, and better schedules can be generated compared to earlier used techniques.

References

[1] G. Bel, E. Bensana, D. Dubois, J. Erschler and P. Esquirol: "A Knowledge-Based Approach to Industrial Job-Shop Scheduling" In: *Knowledge-Based Systems in Manufacturing, Andrew Kusiak (ed.)*, Taylor & Francis, pp 207–246, 1989.

[2] Jürgen Dorn and Reza Shams: "An Expert System for Scheduling in a Steelmaking Plant" In: *Proceedings of the World Congress on Expert Systems, Orlando Fla.*, Pergamon Press, 1991.

[3] Didier Dubois: "Fuzzy Knowledge in an Artificial Intelligence System for Job-Shop Scheduling" In: *Applications of Fuzzy Set Methodologies in Industrial Engineering, Gerald W. Evans et al. (eds.)*, Elsevier, pp 73–89, 1989.

[4] Didier Dubois and Henri Prade: "Processing Fuzzy Temporal Knowledge" In: *IEEE Transactions on Systems, Man, and Cybernetics, Vol. 19, No. 4*, pp 729–744, July/August 1989.

[5] Mark S. Fox: *Constraint-Directed Search: A Case Study of Job-Shop Scheduling.* Pitman, London, 1987.

[6] Mark S. Fox and Norman Sadeh: "Why Is Scheduling Difficult? A CSP Perspective" In: *Proceedings of the European Conference on Artificial Intelligence*, pp 754–767, 1990.

[7] Fred Glover: "Tabu Search–Part I" In: *ORSA Journal on Computing, Vol. 1, No. 3*, pp 190–206, 1989.

[8] Constantin V. Negoiţă: *Expert Sytems and Fuzzy Systems.* Benjamin/Cummings 1985.

[9] Henri Prade: "Using Fuzzy Set Theory in a Scheduling Problem: A Case Study" In: *Fuzzy Sets and Systems, Vol. 2, No. 2*, pp 153–165, April 1979.

[10] Patrick Prosser: "A Reactive Scheduling Agent" In: *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp 1004–1009, 1989.

[11] Stephen F. Smith, Mark S. Fox and Peng Si Ow: "Constructing and Maintaining Detailed Construction Plans: Investigations into the Development of Knowledge-Based Factory Scheduling Systems" In: *AI Magazine 7(4) Fall*, pp 45–61, 1986.

[12] Lotfi A. Zadeh: "Knowledge Representation in Fuzzy Logic" In: *IEEE Transactions on Knowledge and Data Engineering, Vol. 1, No. 1*, pp 89–100, March 1989.

Experiments with a Decision-Theoretic Scheduler*

Othar Hansson[1,2] and Gerhard Holt[1] and Andrew Mayer[1,2]

[1]Heuristicrats Research Inc.
1678 Shattuck Avenue, Suite 310
Berkeley, CA 94709-1631

[2]Computer Science Division
University of California
Berkeley, CA 94720

Abstract

This paper describes DTS, a *decision-theoretic scheduler* designed to employ state-of-the-art probabilistic inference technology to speed the search for efficient solutions to constraint-satisfaction problems. Our approach involves assessing the performance of heuristic control strategies that are normally hard-coded into scheduling systems, and using probabilistic inference to aggregate this information in light of features of a given problem.

BPS, the Bayesian Problem-Solver [2], introduced a similar approach to solving single-agent and adversarial graph search problems, yielding orders-of-magnitude improvement over traditional techniques. Initial efforts suggest that similar improvements will be realizable when applied to typical constraint-satisfaction scheduling problems.

1 Background

Scheduling problems arise in schools, in factories, in military operations and in scientific laboratories. Although many algorithms have been proposed, scheduling remains among the most difficult of optimization problems. Because of the problem's ubiquity and complexity, small improvements to the state-of-the-art in scheduling are greeted with enormous interest by practitioners and theoreticians alike.

A large class of scheduling problems can be represented as constraint-satisfaction problems (CSPs), by representing attributes of tasks and resources as variables. Task attributes include the scheduled time for the task (start and end time) and its resource requirements. A schedule is constructed by assigning times and resources to tasks, while obeying the constraints of the problem. Constraints capture logical requirements (a typical resource can be used by only one task at a time) and problem requirements (task T_x requires N units of time, must be completed before task T_y, and must be completed before a specified date).

One common approach to finding an assignment for the variables employs a preprocessing stage which tightens the constraints (e.g., by composing two constraints to form a third), followed by a backtrack search to find a satisfying assignment. Figure 1 illustrates the operation of such a search algorithm: searching depth-first until a dead-end is reached, and then backtracking to the nearest choice point to continue the search.

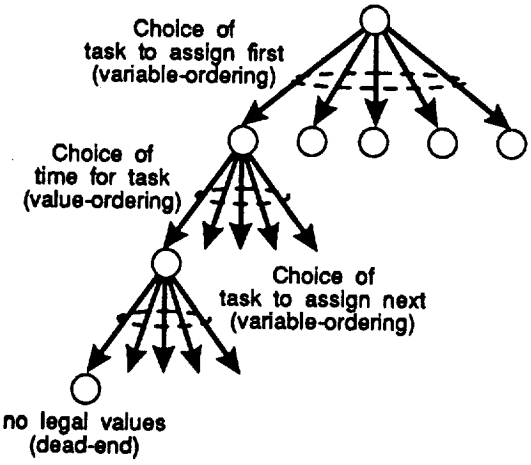


Figure 1: Basic CSP Algorithm

Heuristic functions guide the ordering of variables and values. For example, one heuristic for variable ordering counts the number of possible values for each variable, and chooses the variable with the smallest number of values as the next to instantiate. Typi-

*This research was supported by the National Aeronautics and Space Administration under contract NAS2-13340.

cally, the variable ordering in backtracking algorithms is static, determined prior to search by use of a heuristic function. As heuristics for variable and value ordering form the basis for the algorithm's performance, tremendous effort has been invested in developing good general-purpose heuristics. However, practitioners often bypass the general-purpose heuristics in favor of hand-crafted domain-specific heuristics (e.g., Sadeh's work [8]).

2 DTS Rationale

CSP heuristics are imperfect and exhibit highly domain-specific performance. Although they often provide useful search control advice, the possibility of error introduces uncertainty into the search algorithms which rely on them. Consequently, current techniques are forced to pay a large computational price in cases where the heuristic function makes incorrect classifications. Furthermore, the algorithms will repeat these costly mistakes, as there are no robust learning mechanisms designed to improve a CSP heuristic's performance over time.

Existing heuristic functions encode many different domain attributes. Some estimate the quality of partial schedules while others estimate the difficulty of finding a feasible solution. Unfortunately, there is no *sound* methodology for combining the information provided by an arbitrary number of heuristics for use in controlling a single search. This forces human schedulers to make an unpleasant choice:

- decide *a priori* on a particular heuristic, and thus concentrate on a single domain attribute. This can skew the system's performance at the expense of other domain attributes.
- hand-craft a composite heuristic which captures multiple domain attributes in a single function.

For this reason, the selection of heuristics and problem-solving techniques for any given CSP domain remains an art despite years of comparative study.

DTS, which is derived from previous work on BPS (the Bayesian Problem-Solver), is designed to address these problems. The first area of innovation is the *heuristic error model*: a probabilistic semantics for heuristic information, based on the concept of conditional probability in statistical decision-theory [3]. Heuristics are interpreted by correlating their estimates with the actual payoffs of problem-solving instances. When a problem is solved, the heuristic error model is updated, adapting it to the problem's specific characteristics. Multiple heuristics are combined by correlating payoffs with a *set* of heuristic estimates. This alleviates the human scheduler's dilemma by providing a dominating alternative, a sound framework for combining an arbitrary number of heuristic functions.

The second area of innovation is the use of *multiattribute utility theory*, a formalized method for quantifying preference relationships among a set of uncertain outcomes. An important target application for DTS is experiment scheduling for the Hubble Space Telescope. Figure 2 depicts a partial set of utility attributes, whose non-linear tradeoffs can be encoded by a multiattribute utility function. In contrast to

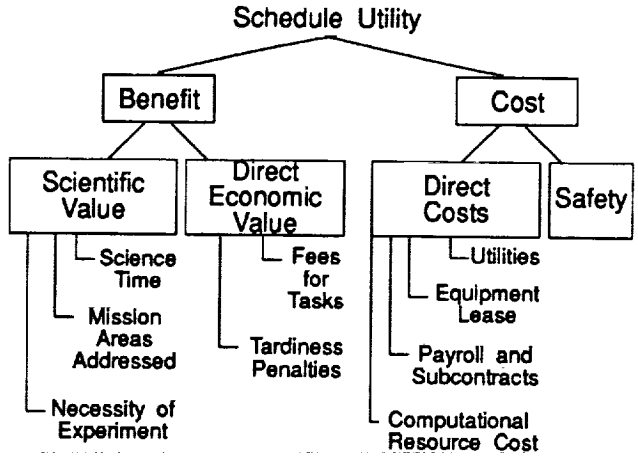


Figure 2: Utility Attributes for Experiment Scheduling

traditional CSP scheduling algorithms, which employ special-purpose control rules, DTS's control rule is the decision-theoretic *rationality* criterion of maximizing expected utility.

In DTS, domain information is encoded in heuristic functions and user preferences are encoded in utility functions. By combining domain-independent and domain-specific heuristics, and then using the user's utility function to make search control decisions, DTS provides a more efficient and flexible alternative to traditional scheduling techniques.

3 DTS: First Results

This section describes empirical results illustrating the performance advantages of these two DTS innovations.

3.1 Combining Heuristics

The primary strength of the DTS prototype is the method for combining information from separate heuristic evaluation functions to improve constraint-satisfaction search control. Experiments with the prototype on the Eight Queens and Bridge-Construction Scheduling [9] problems confirm that the combination of heuristic functions provides more information than any of the heuristics taken individually. This translates into significant reductions in overall search time.

Traditionally, CSP algorithms make use of a variable ordering heuristic and a value ordering heuristic. Figure 3 shows the performance of a standard CSP algorithm using all possible pairs (A1, A2, B1, B2)

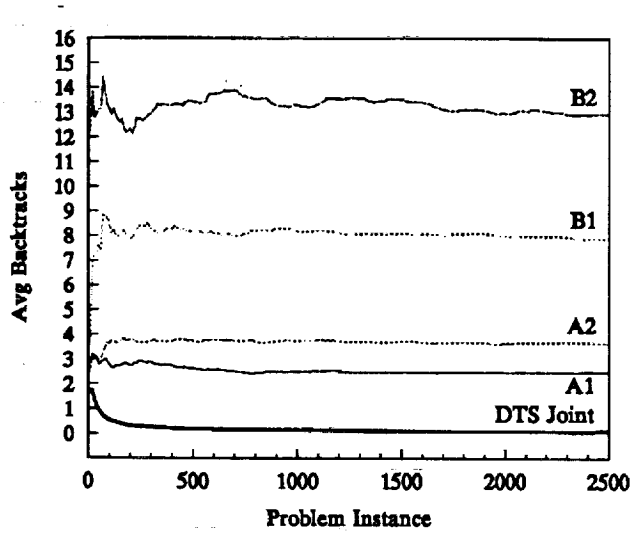


Figure 3: Eight Queens: Combining Heuristics vs. Heuristics in Isolation

drawn from two well-known variable ordering heuristics (Most Constraining Variable (A), Minimum Domain Variable (B)) and two well-known value ordering heuristics (Least Constraining Value (1), Dechter's Value Heuristic (2)[1]). Also shown is the DTS prototype (DTS-Joint), which dominated the competition by using all four heuristics in combination. The horizontal axis plots the number of problem instances solved and the vertical axis plots the running average of search time over the entire experiment. The plot, but not the average, beging with the tenth problem instance.

Figure 4 shows a corresponding graph for the Bridge-Construction Scheduling problem. The variable ordering heuristic used was Minimum Domain Variable and the value ordering heuristics were Least Constraining Value (curve A1) and ASAP, "as soon as possible" (curve A2). Also shown are the corresponding individual DTS performance curves (DTS A1, DTS A2) as well as the combined heuristic performance curve (DTS-Joint).

To summarize both graphs, the improvement is seen to be nearly 50% on average for Bridge Construction Scheduling, and over 95% for the Eight-Queens problem. Note that the sharp downward slope of the DTS-Joint *running average* in Figure 4 demonstrates the performance improvement accrued by learning, unattainable using traditional techniques.

3.2 Learning Heuristic Error Models

Figure 5 displays an example heuristic error model learned over the course of 2500 Eight-Queens problem

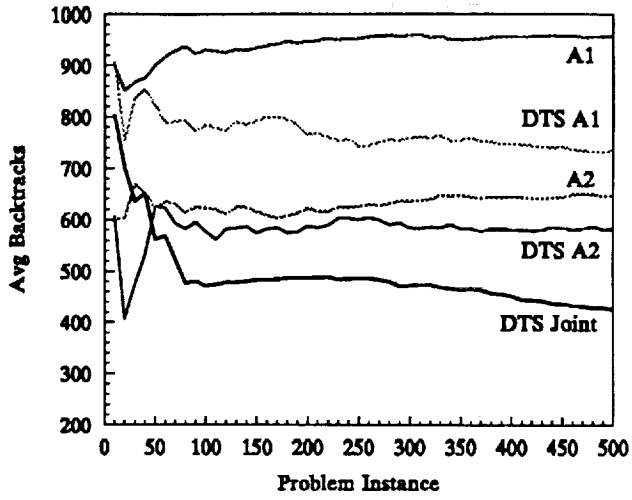


Figure 4: Bridge-Construction Scheduling: Combining Heuristics vs. Heuristics in Isolation

instances (for the Minimum Domain heuristic). The horizontal axis plots the heuristic function estimate and the vertical axis plots the preference for that estimate. In DTS, preference is based upon the expected utility associated with a heuristic estimate (dashed line). In traditional algorithms, the heuristic is assumed to rank-order alternatives perfectly, and therefore, preference is a monotonic function of the heuristic estimate.

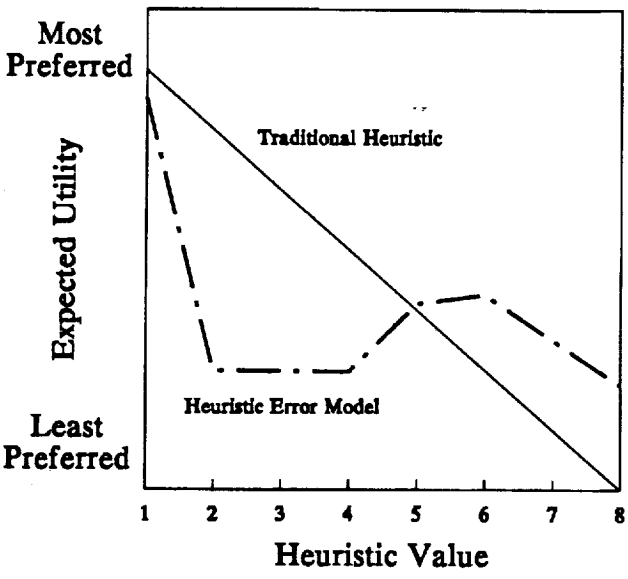


Figure 5: Sample Heuristic Error Model

The discrepancy between the heuristic estimates and the actual utilities explains the poor performance of

traditional approaches, which assume perfect heuristic estimates. Further, it explains why DTS outperforms these techniques, as it does not make this assumption, and instead learns to correct for the discrepancy.

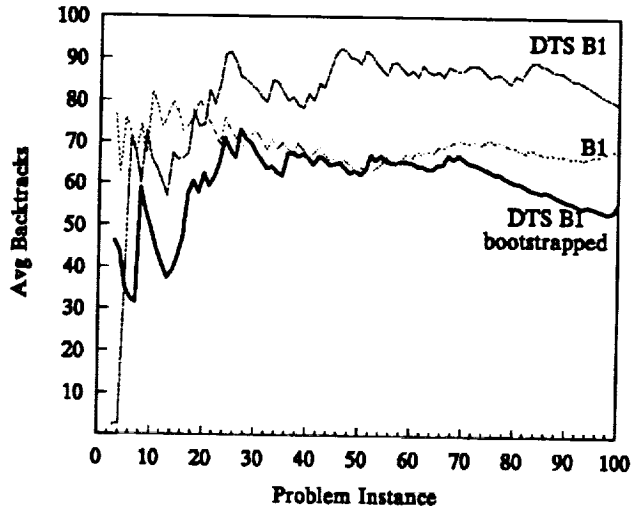


Figure 6: Generalizing Data to Larger Domains

An additional benefit of the heuristic error model is the ability to generalize learned data across domains. For example, Figure 6 depicts the performance of DTS on the Thirty-two-Queens problem with 1) no prior heuristic error model, and 2) a heuristic error model generalized (or "bootstrapped") from the 2500 Eight-Queens examples solved in Figure 3. Generalizing data from the simpler domain has reduced search complexity. This is particularly important as the time required to calibrate heuristic error models increases with problem complexity.

3.3 Decision-Theoretic Backtracking

The DTS prototype employed a simplified decision-theoretic control mechanism which was adapted to a conventional backtracking search algorithm: this allowed for controlled experiments on DTS vs. traditional algorithms. The application of decision theory to backtracking elucidates many important ideas.

The only search control decisions made in traditional backtracking systems are the selections of which subtrees of the search graph to explore next. Once a subtree is selected (by selecting the next variable or value), it is explored exhaustively unless a solution is found. Such an ordering problem can be viewed as a decision-tree. Figure 7 depicts the choice of ordering two subtrees A and B. We have proven a theorem [4] which shows that the system's expected utility (search time to first solution) is maximized if variables (or values) are ordered by the quantity $P(v)/C(v)$, where $P(v)$ indicates probability of finding a solution in the subtree, and $C(v)$ indicates the cost of searching the subtree (whether or not a solution is found). $P(v)$ and $C(v)$ are attributes of the payoff mentioned above. Experiments confirmed that once $P(v)$ and $C(v)$ are learned, this rule outperforms traditional backtracking search algorithms which interpret heuristic estimates at face value. This result indicates that decision-theoretic search-control improves overall system performance. A similar analysis can also be performed for iterative improvement [4].

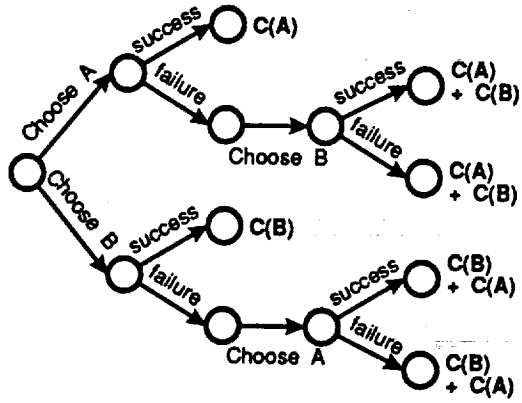


Figure 7: Decision Tree for Value-Ordering Problem (Values A and B)

As is evident from this discussion, DTS must convert raw heuristic estimates at a node into estimates of (1) probability of finding a solution in the subtree under that node, and (2) the cost of search in that subtree. We note here that while heuristics are usually very good at rank-ordering nodes based on (1) and (2) individually, the rank-ordering for the combination is typically incorrect. DTS' heuristic error model corrects for this.

3.4 Implementation Synopsis

The prototype performs a backtracking search, using the standard optimizations of forward-checking and dynamic search rearrangement. The search is ordered by the expected utility selection criteria ($P(v)/C(v)$) discussed above. The estimates of $P(v)$ and $C(v)$ are derived from the heuristic error model, using traditional CSP heuristics. The heuristic error model is updated during and between trials using a bucketed histogram, and interpreted by a Laplacian estimation.

C-2

4 Future Directions

Our initial study of CSP and scheduling domains demonstrates that applying even the simplest modeling techniques of statistical decision theory can yield significant payoffs. There are many other aspects of CSP algorithms which would benefit from a similar decision-theoretic approach. We conclude with two such examples.

4.1 Preprocessing and Caching of Learned Constraints

Our decision-theoretic approach could be applied equally well to the control of scheduling subproblems. For example, Minton [5] has considered a simple utility-based model of the selective caching of learned problem-solving rules.

Minton demonstrated that the caching of too many rules acquired from problem-solving instances leads to a substitution of knowledge-search (searching the rule cache for an applicable rule) for problem-solving search. Similarly, in a CSP problem, any number of implicit constraints can be generated by preprocessing or constraint-recording and cached in the constraint graph. But additional constraints, while reducing problem-solving search, increase the number of consistency checks per search tree node (knowledge search). Choosing to generate and record a constraint is, again, a decision made under uncertainty, and it would be interesting to consider a decision-theoretic approach to the problem. We feel that decision-theoretic modeling and the simple structure of CSPs can provide a firmer theoretical foundation for this area of research.

4.2 Selective Value Generation

A common problem among search algorithms is selective expansion of successors. The textbook description of most search algorithms calls for a full expansion of all successors of a given node. For constraint-satisfaction problems, this is clearly inadequate, as many variables such as task start and end times have an infinite number of infinitesimally-spaced values.

One possible approach employs heuristics for value generation. While we have applied decision theory to search by designing an algorithm which evaluates all successors and then selects among them, it is equally possible to apply these tools to selective expansion of successors. If several heuristics (dispatch rules) can be used to suggest plausible values, our approach can be applied to the heuristics trivially. If no such heuristics exist, one possibility is to employ a tree of values, and perform an auxiliary search of this tree to select a particular value. This brings on a new learning task: clustering values of similar merit into a hierarchy of values.

5 Conclusion

The use of Bayesian probability theory in DTS underscores that scheduling involves decision-making under uncertainty, and illustrates how imperfect information can be modeled and exploited. The use of multiattribute utility theory in DTS underscores that scheduling involves complex tradeoffs among user preferences. By addressing these issues, DTS has demonstrated promising performance in preliminary empirical testing.

References

[1] R. Dechter and J. Pearl. Network-Based Heuristics for Constraint-Satisfaction Problems. In *Search in Artificial Intelligence*, L. Kanal and V. Kumar, eds., Springer-Verlag, New York, 1988.

[2] O. Hansson and A. Mayer. Heuristic Search as Evidential Reasoning. In *Proceedings of the the Fifth Workshop on Uncertainty in Artificial Intelligence*, Windsor, Ontario, August 1989.

[3] O. Hansson and A. Mayer. Probabilistic Heuristic Estimates. *Annals of Mathematics and Artificial Intelligence*, 2:209–220, 1990.

[4] O. Hansson and A. Mayer. Decision-Theoretic Control of Artificial Intelligence Scheduling Systems. HRI Technical Report No. 90-1/06.04/5810, September 1991.

[5] S. Minton. *Learning Effective Search Control Knowledge: An Explanation-Based Approach.* Kluwer Academic, Dordrecht, 1989.

[6] U. Montanari. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Processing Letters*, vol. 7, 1974.

[7] N. Sadeh. Lookahead Techniques for Activity-Based Job-Shop Scheduling. Technical Report TR CMU-RI-TR-89-2, CMU, the Robotics Institute, 1989.

[8] N. Sadeh. *Lookahead Techniques for Micro-Opportunistic Job-Shop Scheduling.* PhD Thesis, CMU, the Robotics Institute, 1991.

[9] P. van Hentenryck. *Constraint-Satisfaction in Logic Programming.* MIT Press, Cambridge, MA, 1989.

[10] M. Zweben, M. Deale and R. Gargan. Anytime Rescheduling. in *Proceedings of the DARPA Planning Workshop*, Morgan Kaufmann, San Mateo, CA, 1990.

Generating Effective Project Scheduling Heuristics by Abstraction and Reconstitution

Bhaskar Janakiraman and **Armand Prieditis**
Department of Computer Science
University of California
Davis, CA 95616

Abstract

A project scheduling problem consists of a finite set of jobs, each with fixed integer duration, requiring one or more resources such as personnel or equipment, and each subject to a set of precedence relations, which specify allowable job orderings, and a set of mutual exclusion relations, which specify jobs that cannot overlap. No job can be interrupted once started. The objective is to minimize project duration. This objective arises in nearly every large construction project—from software to hardware to buildings. Because such project scheduling problems are NP-hard, they are typically solved by branch-and-bound algorithms. In these algorithms lower-bound duration estimates (admissible heuristics) are used to improve efficiency. One way to obtain an admissible heuristic is to remove (abstract) all resource and mutual exclusion constraints and then obtain the minimal project duration for the abstracted problem; this minimal duration is the admissible heuristic. Although such abstracted problems can be solved efficiently, they yield inaccurate admissible heuristics precisely because those constraints that are central to solving the original problem are abstracted. This paper describes a method to *reconstitute* the abstracted constraints back into the solution to the abstracted problem while maintaining efficiency, thereby generating better admissible heuristics. Our results suggest that reconstitution can make good admissible heuristics even better.

1 Introduction

One way to solve a difficult problem is to simplify it by removing certain details, solve the simplified problem, and then use its solution as a guide for solving the original problem. For example, in solving a difficult physics problem, details such as friction might be ignored. Although the simplified problem might be easy to solve, it might ignore precisely those details that are central to solving the original problem. This paper describes a method called *reconstitution* that adds back such ignored details to the simplified problem's solution, thereby providing a better guide for solving the original problem. The ultimate goal of this research is to develop an automatic reconstitution system, thereby shifting some of the simplification and problem-solving from humans to machines.

As a vehicle for exploring reconstitution, we are currently focusing on *project scheduling* problems because they are of practical importance and are difficult to solve. A project scheduling problem consists of a finite set of jobs, each with fixed integer duration, requiring one or more resources such as personnel or equipment, and each subject to a set of precedence relations, which specify allowable job orderings, and a set of mutual exclusion constraints, which specify jobs that cannot overlap. No job can be interrupted once started. The objective is to minimize project duration. Since this objective arises in nearly every large construction project—from software to hardware to buildings—efficient algorithms that obtain that objective are desirable.

Integer linear programming methods have been used to solve project scheduling problems for years [1, 2, 13, 7]. However, these methods are computationally expensive, unreliable, and applicable only to problems of small size. The underlying reason for the computational expense and limited problem size is that such project scheduling problems are NP-hard (see the Appendix). As a result, such problems are typically solved by branch-and-bound algorithms with lower-bound duration estimates (admissible heuristics) to improve efficiency [21, 4]. In addition to improving efficiency, admissible heuristics have other several other desirable properties in various branch-and-bound algorithms such as guaranteeing minimal project duration [16] or guaranteeing a project duration no longer than a certain factor of the minimal one [18].

Several researchers have shown how admissible heuristics can be derived by simplifying the original problem via abstraction (ignoring certain details) and then using the length of a shortest path solution in the abstracted problem as the admissible heuristic [8, 6, 17, 11, 15, 19, 20]. For example, the Manhattan Distance heuristic for sliding block puzzles is derivable by ignoring the blank. For such heuristics to be effective, the abstracted problem that generates them should be efficiently solvable and yet close to the original problem [22, 15, 22]. Typically, the more details that are removed, the easier the problem is to solve and the less accurate the resulting heuristics. This tension between accuracy and ease of solvability makes discovering those abstracted problems that are easy to solve *and* close to the original problem a difficult task [19].

The only published attempt at discovering admissible heuristics with this approach in a scheduling domain yielded poor heuristics [14, 20]. Moreover, the particular scheduling problem (uniprocessor scheduling) to which it was applied did not allow concurrency, which is the *essence* of scheduling. One of the contributions of this paper is to apply abstraction-based heuristic derivation techniques to a scheduling problem where concurrency is allowed (i.e. project scheduling).

The other contribution of this paper is an automatic method to reconstitute an abstract solution, thereby boosting the effectiveness of an admissible heuristic. The idea that abstraction-derived heuristics can sometimes be made more effective by taking into account certain details ignored by the abstracted problem was first expressed by Hansson, Mayer, and Yung [9]. In particular, they hand-derived a new effective admissible sliding block puzzle heuristic

(the LC heuristic) by taking into account those linear tile conflicts (same row or column) ignored by the Manhattan Distance heuristic. We have extended this idea to a problem involving time rather than solution path length: scheduling.

2 Definition of Key Terms

As shown in Figure 1, a scheduling problem can be represented as graph with jobs as vertices, precedences as single-arrowed edges, and mutual exclusions as double-arrowed edges. For example, the figure shows that job I must be completed before job J can start and that jobs J and K cannot overlap. The single number above each job represents the job's duration. For example, job J takes 10 units of time to complete. The letter to the left of each job represents the resource that the job requires; one job's use of a resource cannot overlap with another job's use of that same resource. For example, jobs I and E, which both require resource s, cannot overlap with each other.

A *precedence graph* is a directed acyclic graph consisting only of the precedence relations and no resource constraints. An *early schedule graph* is derived from the precedence graph, where each job is scheduled as early as possible. The numbers within the square brackets near each job in the figure represent the *earliest start time* and the *earliest completion time* of each job. The *critical path* is the longest path in the early schedule graph; it shows the earliest time by which all jobs can be completed.

No job on the critical path can be delayed, although other jobs on the same early schedule can be delayed as long as they do not increase the critical path length. For example, if job J, which is on the critical path, starts later than 33 units of time, the entire project will be delayed. These jobs may have to be delayed in order to satisfy mutual exclusion constraints. The total completion time of an early schedule is therefore equal to the critical path length, which in our case is 43. An *optimal schedule* is an early schedule which takes the least total time among all possible schedules. Note that jobs within an optimal schedule may not be scheduled optimally, according to this definition.

Given only precedence constraints, finding an early schedule reduces to a topological sort of the precedence graph, which can be done in linear time of the number of jobs [10]. Finding the critical path in an early schedule also takes linear time of the number of jobs. Therefore, if all other constraints such as mutual exclusion constraints and resource constraints can be recast as precedence constraints, the problem is easily solvable. For example, the mutual exclusion constraint between jobs J and K can be recast in two ways: either J is completed before K or vice versa. Similarly, for resource constraints each pair of jobs sharing the same resource can be recast as a mutual exclusion constraint between the two jobs. Each mutual exclusion constraint can then be recast as one of two precedence constraints as previously described.

3 Branch and Bound Project Scheduling

The idea of recasting mutual exclusion and resource constraints as precedence constraints suggests the following simple combinatorial algorithm. Explore all recastings, one at a time, that do not create a cycle and find early schedules for all of these recastings; the early schedule with the minimum critical path length is the optimal one. Unfortunately, this brute-force algorithm is combinatorially explosive: n mutual exclusion constraints results in 2^n possible recastings, which is clearly too large a space to explore exhaustively for large n. One way to reduce this combinatorial explosion is to use a branch-and-bound algorithm with lower-bound estimates to prune certain recastings earlier. If the current duration + the lower-bound estimate exceeds a given upper-bound, then that schedule can be pruned.

The critical path estimate of an early schedule, which is efficiently computable, is clearly a lower-bound since any early schedule that satisfies *part* of the constraints is a lower bound on the completion time for any optimal schedule satisfying *all* constraints. Moreover, any additional constraint will not result in a decrease in the critical path length. Notice that the critical path (CP) heuristic results from an abstraction of the original problem: all mutual exclusion and resource constraints are ignored.

Although the CP heuristic is admissible and easily computable and has proved to be valuable in evaluating overall project performance and identifying bottlenecks, it can be far from the actual project duration. In the worst case, it can underestimate the actual project duration by a factor of n, where n is the total number of jobs to be scheduled. This case arises when the only possible schedule is a se-

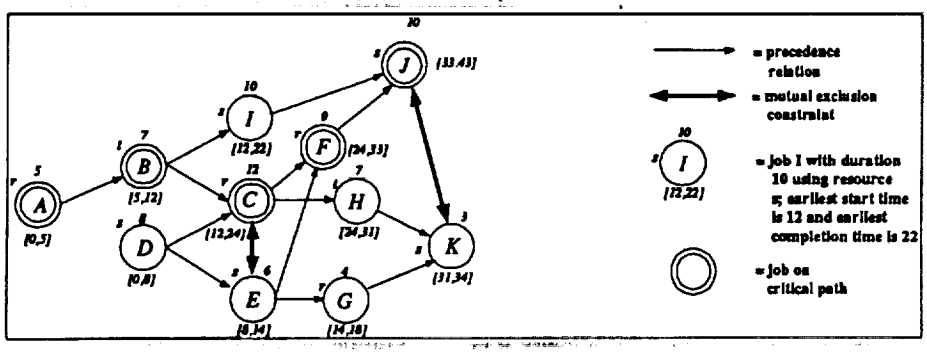


Figure 1 A Project Scheduling Problem

1. Calculate the critical path (CP).
2. Traverse the CP backwards. Assume jobs encountered are $j_k, j_l, \ldots$
3. As each job j_k on CP is encountered, look for unsatisfied mutual exclusion constraints between j_k, and some job j_r, where j_r is not on CP.
4. If in the given schedule, execution of j_k overlaps with j_r, then push j_r ahead so that there is no overlap.
5. Push other jobs ahead if necessary. This is done by listing all jobs j_p such that j_r must come before j_p according to the precedence relation, and rescheduling j_p so that j_r completes before j_p. Repeat this step until all the precedence relations are satisfied.
6. If the length of CP in the new schedule is greater than the original CP, then calculate new bound as original CP length + Amount of overlap between j_k and j_r.
7. Else, repeat till a mutual exclusion constraint is found which increases CP length.
8. If no such constraint is found, return the CP length as the new bound.

Figure 2 An Algorithm to Compute the RCP Heuristic

rial schedule. For example, if a scheduling problem has no precedence constraints and has mutual exclusion constraints between every pair of jobs, then the only possible schedule will be a serial one. For this case, the CP heuristic will return length of the longest job, which underestimates the optimal duration by a factor of n. Also, since the critical path estimate ignores the resource constraints, certain sequencing decisions may be required in the actual schedule that increase the project duration well beyond the critical path estimate.

4 Reconstitution-based Heuristics

What we would like is an admissible heuristic that is as easily computable as the critical path estimate, but that takes into account the resource and mutual exclusion constraints, which the critical path estimate ignores. We would like to reconstitute these ignored constraints back into the critical path somehow. The RCP (Reconstituted Critical Path) heuristic described below does exactly that.

The basic idea behind the RCP heuristic is to extend the critical path by analyzing all unsatisfied mutual exclusion constraints between jobs in critical path and jobs not in critical path. When possible, all jobs with such unsatisfied constraints are rescheduled at a later time while still preserving critical path length. If that is not possible, then the critical path length is increased by a time overlap underestimate between the jobs of each type. For example, consider the project scheduling problem in Figure 1, which has a critical path of J, F, C, B, A. First, we examine job J and check for any mutual exclusion constraints involving it. The only such constraint is the one with job K. Next, we check if J overlaps with K, which in fact it does. The object now is to try to delay job K beyond the completion time of job J, which is at 43 time units. Delaying job K will necessarily increase the length of the critical path by 1 time unit. If the rest of the jobs were ignored, the RCP heuristic would return 44, which is the length of critical path (43) plus the overlap of the earliest start time of job J and the earliest completion time of K $(34 - 33 = 1)$. The general algorithm is shown in Figure 2. (We assume that resource constraints have been recast as mutual exclusion constraints.)

To see that the RCP heuristic is admissible, consider a job j_l on the critical path which has a mutual exclusion constraint with job j_m. In the final schedule, either j_l will be scheduled before j_m or vice versa. Note that neither of the two jobs can be scheduled any earlier since the schedule is already an early schedule. If job j_m cannot be scheduled after j_l without increasing the critical path length in the current schedule by pushing jobs ahead which depend on j_m, then neither can it be scheduled after j_l in the final schedule. The reason is that precedence constraints are always added and never removed at each iteration of the search algorithm and adding more precedence constraints cannot invert an existing scheduling order. If j_l is scheduled after j_m, then the critical path length will be increased by *at least* the minimum of the overlap between the earliest start time of j_l and the earliest completion time of j_m or the earliest start time of j_m and the earliest completion time of j_l.

Although the RCP heuristic takes slightly longer to compute than the CP heuristic, it prunes more of the space than the CP heuristic. As we will see in the next section, the extra time taken in computing the heuristic is more than compensated by the time saved from pruning the search space. If the current critical path length is optimal, then computation of the RCP heuristic takes longer than that of the CP heuristic, since the algorithm has to examine all jobs on the critical path. The worst case complexity of computing the RCP heuristic is $O(n^2)$ for n jobs, since at most $O(n)$ jobs will be on the critical path and $O(n)$ work will be required to process a mutual exclusion constraint involving a job on the critical path. An analysis of the *average* computational complexity is, however, difficult since the heuristic depends on specific mutual exclusion constraints. The degree of complexity can be controlled by reconstituting less mutual exclusion constraints, if desired.

The complexity of the RCP heuristic can be further reduced by computing it incrementally. Since new precedence constraints are added and never removed at each iteration of the search algorithm, the critical path up to the point in the graph where the new precedence constraint is added remains the same and the critical path need only be

Jobs	Precedences	Mutual Exclusions	CP Heuristic			RCP Heuristic		
			States Expanded	*CPU Seconds*	*Bytes*	*States Expanded*	*CPU Seconds*	*Bytes*
30	112	0	0	.25	30820	0	.25	30808
		10	14	5.43	41024	14	8.52	58108
		15	19	5.98	43212	19	9.18	70256
		20	6237	1644.17	45796	49	30.58	109584
		25	6242	1651.53	47188	54	30.67	116068
40	128	10	12	5.15	52928	12	5.63	72228
		20	24	9.73	56988	24	18.52	101936
		30	1084	718.20	71024	431	494.33	194220
		40	1096	727.83	65104	521	521.80	224112

Table 1 Comparative Performance Analysis of the CP and RCP Heuristics with IDA*

recomputed from that point on.

5 Empirical Results

To get some idea of the effectiveness of the RCP and CP heuristics, we implemented the IDA* algorithm [12], which is a standard branch-and-bound algorithm in which to evaluate admissible heuristics, in Quintus Prolog on a Sun Sparstation 1+ and ran it on a set of random solvable (i.e. no cycles) problem instances with various numbers of jobs, mutual exclusion constraints, and precedence constraints. The algorithm works as follows. All partial schedules whose duration exceeds a certain threshold are pruned. Initially, the threshold is set to the value of the admissible heuristic on the initial state. If no solution is found within that threshold, then the algorithm repeats with a new threshold set to the minimum of duration plus heuristic estimate over all the previously generated partial schedules whose duration exceeds the threshold. One important property of IDA* is that it guarantees minimal duration solutions with admissible heuristics.

A state consists of three items:

1. A precedence graph which includes original precedence constraints and a set of precedence constraints originating from mutual exclusion constraints which have so far been recast as one of two precedence constraints.
2. An early schedule satisfying the precedence constraints.
3. A set of unsatisfied mutual exclusion constraints.

The goal state is characterized by an empty mutual exclusion constraint set. A state transition is a recasting of a mutual exclusion constraint into one of two precedence constraints followed by the generation of a new early schedule. Search proceeds from an initial schedule satisfying only the original precedence constraints. (Our implementation assumes that resource constraints have been recast as a set of mutual exclusion constraints.)

We ran two sets of experiments, each with a fixed the number of jobs and precedence constraints and a variable number of mutual exclusion constraints since problem complexity grows as the number of mutual exclusion constraints increases: one with 30 jobs with 112 precedence constraints and the other with 40 jobs with 128 precedence constraints. For the first set, we varied the number of mutual exclusion constraints between 0 and 25; for the second, between 10 and 40. We chose these problems because they were the largest ones we could generate that still could be solved in a reasonable amount of time on our machine.

Table 1 summarizes the results of running IDA* on these two problem sets. For each problem set, the table lists the number of mutual exclusion constraints, the number of states expanded, the CPU time, and the amount of run-time memory used. As the table shows, for problems with few mutual exclusion constraints, the number of states expanded in both cases remain the same and CP consistently takes less time than RCP, since RCP does more work each time. However, for all problems where RCP resulted in a saving in terms of states expanded, RCP always takes less CPU time. RCP also uses slightly more run-time memory in all examples, but always within a factor of 4 when compared to CP. In summary, RCP works better than CP in all cases where the critical path length is not optimal, which is typically the case in real-world (non-artificial) problems, where it is highly probably that constraints other than precedence constraints play a major role in dictating the total project duration. Therefore, RCP will result in better performance in most real-world cases.

6 Conclusions and Future Work

This paper has described an instance of a general three step problem-solving paradigm: abstract, solve, reconstitute. Certain details of the original problem are removed by abstraction. Next, the abstracted problem is efficiently solved. Finally, the abstracted details are reconstituted back into this solution. This reconstituted solution is then used as a guide for solving the original problem. We applied this paradigm to project scheduling problems and obtained

a novel effective heuristic (the RCP heuristic). The general idea of reconstitution is to boost the informedness of an admissible heuristic by adding back previously abstracted details and maintaining efficiency.

This approach as applied to project scheduling has several shortcomings. First, complex project scheduling problems often involve resource constraints with fixed limits for each job, typically specifying the *number* of fixed resource units that cannot be exceeded, rather than the absolute resource constraints as in our model; it is not clear to us how to recast such resource constraints as mutual exclusion constraints. However, Davis and Heidorn [3] show a branch-and-bound solution to the problem. They describe a preprocessor algorithm that expands a job with duration k into a sequence of k unit duration jobs each successively linked with a "must immediately precede" precedence relation. After this expansion, a standard branch-and-bound project scheduling algorithm can be run. Unfortunately, such expansion can result in enormous project networks in projects with long duration jobs.

A second shortcoming is that not all scheduling constraints can be recast as precedence constraints. For example, a constraint that a particular job must start only after a certain time cannot be recast as a precedence constraint. Effective admissible heuristics that reflect such general constraints would be an important contribution to scheduling.

Finally, although this paper has described a method for generating better admissible heuristics from existing ones, the process of discovering heuristics such as the RCP heuristic is far from automatic. We are currently extending this method to job-shop scheduling problems of the sort described in [5]. In a job-shop problem, n jobs are to be scheduled on m machines with varying durations per job per machine. We hope to develop a set of general principles that practitioners in the scheduling field can follow to derive effective heuristics and eventually to automate the discovery process.

References

[1] M. L. Balinski. Integer programming: Methods, uses, computation. *Management Science*, November 1965.

[2] J. D. Brand, W. L. Meyer, and L. R. Schaffer. The resource scheduling problem in construction. Technical Report 5, Dept. of Civil Engineering, University of Illinois, Urbana, 1964.

[3] E. W. Davis and G. E. Heidorn. An algorithm for optimal project scheduling under multiple resource constraints. *Management Science*, 17(12), August 1971.

[4] M. Dincbas, H. Simonis, and P. V. Hentenryck. Solving large combinatorial problems in logic programming. *Journal of Logic Programming*, 8(1 and 2):75–93, 1990.

[5] M. S. Fox. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling*. Pitman, 1984.

[6] J. Gaschnig. A problem-similarity approach to devising heuristics. In *Proceedings IJCAI-6*, pages 301–307, Tokyo, Japan, 1979. International Joint Conferences on Artificial Intelligence.

[7] D. Graham and H. Nuttle. A comparison of heuristics for a school bus scheduling problem. *Transportation*, 20(2):175–182, 1986.

[8] G. Guida and M. Somalvico. A method for computing heuristics in problem solving. *Information Sciences*, 19:251–259, 1979.

[9] O. Hansson, A. Mayer, and M. Yung. Criticizing solutions to relaxed models yields powerful admissible heuristics, 1992. To appear in Information Sciences.

[10] E. Horowitz and S. Sahni. *Fundalmentals of Data Structures*. Computer Science Press, Inc, Rockville, Maryland, 1978.

[11] D. Kibler. Natural generation of heuristics by transforming the problem representation. Technical Report TR-85-20, Computer Science Department, UC-Irvine, 1985.

[12] R. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(2):97–109, 1985.

[13] C. L. Moodie and D. E. Mandeville. Project resource balancing by assembly line balancing techniques. *Journal of Industrial Engineering*, July 1966.

[14] J. Mostow, T. Ellman, and A. Prieditis. A unified transformational model for discovering heuristics by idealizing intractable problems. In *AAAI90 Workshop on Automatic Generation of Approximations and Abstractions*, pages 290–301, July 1990.

[15] J. Mostow and A. Prieditis. Discovering admissible heuristics by abstracting and optimizing. In *Proceedings IJCAI-11*, Detroit, MI, August 1989. International Joint Conferences on Artificial Intelligence.

[16] N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, Palo Alto, CA, 1980.

[17] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem-Solving*. Addison-Wesley, Reading, MA, 1984.

[18] I. Pohl. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *Proceedings IJCAI-3*, pages 20–23, Stanford, CA, August 1973. International Joint Conferences on Artificial Intelligence.

[19] A. Prieditis. *Discovering Effective Admissible Heuristics by Abstraction and Speedup: A Transformational Approach*. PhD thesis, Rutgers University, 1990.

[20] A. Prieditis. Machine discovery of effective admissible heuristics. In *Proceedings IJCAI-12*, Sydney, Australia, August 1991. International Joint Conferences on Artificial Intelligence.

[21] F. J. Radermacher. Scheduling of project networks. *Journal of Operations Research*, 4(1):227–252, 1985.

[22] M. Valtorta. A result on the computational complexity of heuristic estimates for the A* algorithm. *Information Sciences*, 34:47–59, 1984.

Mutual Exclusions are NP-Hard

Finding a minimum duration schedule for a project graph with only mutual exclusion constraints and unit length job duration is equivalent to solving a graph coloring problem. In the project scheduling problem, the object is to partition jobs into a minimum number of sets such that each job is in exactly one set and no two jobs in a set have a mutual exclusion edge between them. Since all jobs in each set can be scheduled in parallel, the final schedule's duration is simply the number of sets. In the graph coloring problem, the object is to color the nodes of a graph such that no two nodes connected by an edge have the same color and the minimum number of colors are used. Since there is a 1-1 correspondence between the two problems and the graph coloring problem is NP-Hard, so is the project scheduling problem with mutual exclusion constraints. Furthermore, since resource constraints can be recast as mutual exclusion constraints, the problem of scheduling with resource constraints is also NP-Hard and adding non-unit length job durations only makes the problem harder. Notice that adding precedence constraints will not affect this result. We thank Charles Martel for suggesting the basic idea behind this proof.

Real-time Scheduling Using Minimin Search

Prasad Tadepalli and **Varad Joshi**
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331-3202

Abstract

In this paper we consider a simple model of real-time scheduling. We present a real-time scheduling system called RTS which is based on Korf's Minimin algorithm. Experimental results show that the schedule quality initially improves with the amount of look-ahead search and tapers off quickly. So it appears that reasonably good schedules can be produced with a relatively shallow search.

1 Introduction

Job shop scheduling is one of the most computationally intensive parts of flexible manufacturing systems. Scheduling in the real world is complicated by several factors including the resource contention, unpredictability of events, multiple agents with mutually conflicting goals, and the sheer combinatorial explosiveness of the task. In this paper, we simplify the real world scheduling problem to a great extent and focus exclusively on one aspect of the problem, namely its real-time character.

This paper looks at detailed job shop scheduling at the level of individual machine operations. The scheduling problem is treated as assigning the job-steps to individual machines and ordering them so that (a) the precedence and resource constraints are satisfied, and (b) the schedule is "good" in some measurable objective sense.

Most approaches to scheduling are static in that the scheduling is done all at once and not during the production process. Static scheduling has several obvious drawbacks: First, optimal static scheduling is computationally prohibitive in any realistic manufacturing system, which involves hundreds of jobs and machine operations. Second, since the static scheduler has to make decisions based on predicted information, it has no way of recovering from incorrect predictions even after they were proved wrong. Thus, it is unable to readjust to or recover from changes in the production environment, including machine failures, new jobs, or machine delays.

Real-time scheduling prevents the above two pitfalls of static scheduling by requiring that after every constant time, some real world action is taken. This not only prevents the system from losing itself in a combinatorially explosive search space, but also makes it possible to continually readjust to the changing environment.

In this paper we present a system called RTS (Real-time Scheduler) which uses the Minimin algorithm of Korf [Korf, 1990] to do real-time scheduling. Minimin is similar to the Minimax algorithm extensively used in games. We view scheduling as a state space search where states represent partial schedules. Minimin performs a fixed depth look-ahead search from the initial state, and applies a heuristic evaluation function to the partial schedules at the leaves of the search tree to estimate the cost of the schedule. This value is backed up to the root of the tree and the system takes the most promising scheduling action, i.e., it assigns a job-step to a machine which leads to a schedule with the best estimated cost.

Since RTS relies on heuristic estimates, the schedules the system produces are not guaranteed to be optimal. However, our experimental results show that the schedule quality initially improves with the amount of look-ahead search and tapers off quickly. So it appears that reasonably good schedules can be produced with a relatively shallow search. We conclude that our approach to real-time scheduling based on Minimin is promising and can be extended in several directions, including learning better evaluation functions, and doing variable depth search.

2 Previous Work

One approach to scheduling is based on expert systems [Fox and Smith, 1984]. However, expert systems approach to scheduling seems inadequate because of the dynamic nature of the scheduling problem, which is due to changes to job loads, availability of machines and labor, introduction of new machines and manufacturing processes, changes in the inventory space, etc. For this reason, there are no experts in this domain, and even if there were, they would be quickly outdated [Kempf et al., 1991].

Many AI-approaches to scheduling are constraint-based [Fox, 1987, Sadeh, 1991, Smith et al., 1986, Zweben and Eskey, 1989]. Here scheduling is viewed as finding a schedule (assignment of machines to various job-steps) which satisfies a set of constraints, including precedence relationships between job-steps and global resource constraints. However, most of these approaches

assume a static scheduling problem, and are not easily adaptable to real-time scheduling.

Traditionally, the "dynamics" of the manufacturing process is handled by local greedy dispatch rules [Vollmann et al., 1988]. One dispatch rule, for example, recommends to schedule the job with Least Processing Time (LPT) first, while another rule uses Earliest Due Date (EDD) to prioritize jobs. While computationally cheap, such local dispatch rules are too short-sighted, and do not guarantee efficient schedules except in very special cases [Kempf et al., 1991].

In summary, static optimal scheduling is computationally prohibitive and is not sufficiently responsive to change. On the other hand, local dispatch rules are too short-sighted to be generally effective. The expert systems approach is plagued by the dynamics of the scheduling problem and paucity of experts. In this paper, we propose an approach based on real-time search which attempts to address each of the above problems.

3 Problem Description

The problem we address can be characterized as scheduling the job-steps in a set of jobs on various machines in real time. We make the following assumptions.

1. Each job consists of a sequence of job-steps that must be performed serially.
2. There may be several machines of each machine type.
3. Each job-step requires a machine of a particular type to perform it.
4. Each machine can only process one operation at a time.
5. Each job may require the same machine (or machine type) more than once. In other words, we have a "job shop" situation rather than a "flow shop" situation [Vollmann et al., 1988].
6. The machine type required for each job-step and the time for each job-step is known in advance.
7. The real-time constraint means that the time for deciding which job-step to schedule next is "small," and should not depend on the number of jobs and job-steps.

For example, each job in Figure 1 consists of a sequence of job-steps. The task of the scheduler is to incrementally add new job-steps to the current machine queues. As the machine queues are filled from the back by the scheduler, they are emptied from the front by the machines executing the job-steps. In addition, the job-step must wait until its predecessor job-step in its job is executed. For example, in Figure 1, job-steps S-11, S-22, and S-42 are in the queue for machine M1 in that order. In addition, S-11, S-12, S-13, and S-14 must also be processed sequentially, because they are all part of a single job.

Since scheduling is done while the jobs are getting executed, the scheduler has only a limited time to decide what job-step to schedule next, and on what machine.

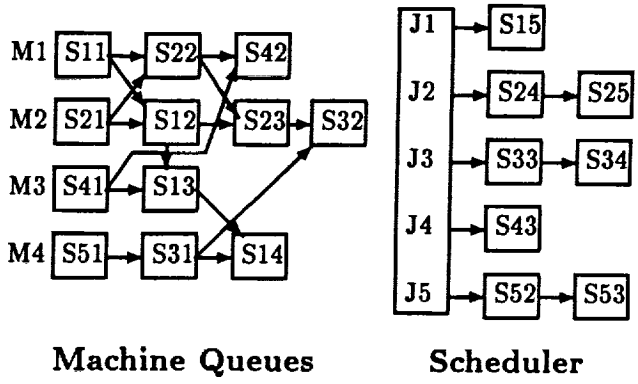


Figure 1: Scheduler assigns job-steps to machine queues.

4 Scheduling as State Space Search

We formulate the scheduling problem as a state space search problem. States in the scheduling task correspond to partial schedules represented as queues of job-steps for the machines. The search problem is characterized by an initial state, where there are no jobs scheduled, and a final state, where all the jobs are scheduled. In any state, there are several alternative assignments of the job-steps to machine queues. A job-step is "ready" when all its precedent job-steps have completed. Scheduling operators or "moves" assign job-steps to one of the machines of the required machine type. In other words, they can be placed on any one of the possible queues of the appropriate machine type. Each such placement creates a new state. The scheduling problem is to find a best assignment of job-steps to machine queues according to some measure of goodness (objective function). For example, we may use the total time for the schedule or the sum of the inventory and shortage costs as an objective function.

The static scheduling problem corresponds to finding the best path in the state space from the initial state to a final state. However, static scheduling suffers from the combinatorial explosion due to deep searches and is not sufficiently responsive to the dynamics of the manufacturing domain. In the following, we describe our approach to scheduling that addresses these problems.

4.1 Minimin search

Our approach to scheduling consists of a real time search method called "Minimin search" [Korf, 1990]. Minimin is similar to minimax search in two-person games, except that instead of alternating Min and Max nodes, the search tree only contains Min nodes.

Minimin works by a fixed depth look-ahead search followed by a real-time action. The search terminates after a small depth called "search horizon," after which the leaves of the tree are evaluated using a heuristic evaluation function. The evaluation function applied at the leaves estimates the minimum total cost of any solution that begins with a partial path ending with that leaf. It is backed up to the root using the Min function. In other words, the value of any node is the minimum of all the values of its children, and the move that results in

that value is the "best move." After searching for a fixed look-ahead depth, Minimin chooses the first best move, executes it, updates the state and once again starts look-ahead search from that point.

The "knowledge" of the Minimin algorithm lies in its heuristic evaluation function f. The more closely it follows the real cost of the solution, the more optimal the algorithm's current decision is going to be. An evaluation function is "admissible" if it never overestimates the real cost of a solution. An evaluation function is monotonic, if its value is monotonically non-decreasing along any single path of the search tree.

When the evaluation function of the Minimin search is monotonic, it is amenable to an effective branch and bound technique called α-pruning. α-pruning works by pruning the branches whose estimated cost is more than the current best estimated cost. Like α-β pruning, α-pruning is guaranteed to preserve the outcome of the look-ahead search.

Each time the Minimin algorithm is called it returns the best next state and its estimated evaluation. The main program then takes the corresponding action in the "real world" and updates its current state to this new state. After this, the program repeats its cycle again by calling the Minimin algorithm.

4.2 Real-time Scheduling

We noted that in Scheduling the states correspond to partial schedules and operators correspond to scheduling actions. In order to complete the mapping of the real-time scheduling problem to Minimin search, we need to specify how a schedule is evaluated.

Several optimality criteria might be used to evaluate the schedules. One of the criteria is the sum of the shortage and the inventory costs. Another criterion is the total length of the schedule from the beginning to the end, also called "make-span." In our system, we currently use the make-span criterion to evaluate schedules. The smaller the make-span, the better the schedule. In Minimin search, the cost of the schedule must be estimated after only a small number of steps are scheduled, i.e., much before the full schedule is known. To do this effectively, we should necessarily rely on heuristic estimates of the schedule cost. A good heuristic evaluation function must approximate the optimality criterion as closely as possible.

As discussed earlier, there is an implicit precedence relationship between the job-steps in the same machine queue, and between the job-steps that belong to the same job. For any job-step s, let $PRE(s)$ be the set of job-steps which are immediate predecessors of s, in that they need to be performed before s is done. In Figure 1, PRE(S-12) = {S-11, S-21}.

Our estimate of the make-span is done as follows: first, we compute the time T_i by which each machine M_i finishes its current queue. Assuming that the expected time $ET(s)$ for each job-step s is known in advance, this can be calculated exactly. Let the expected start time and the expected finish time of a job-step s be denoted by $ES(s)$ and $EF(s)$ respectively. The expected start and finish times of any job-step can then be calculated using

```
Minimin(CurrentState, depth, α)
  If depth = SearchHorizon return (f(CurrentState));
      %Alpha Pruning
  If f(CurrentState) ≥ α return (α + 1)
  S := job-steps which are "ready";
  M := {m | ∃s ∈ S that needs a machine of m's type };
  Pick m ∈ M s.t. its current queue finishes earliest.
  For each job-step s ∈ S which matches m's type, Do
     Begin
     NewState := Assign(s,m);
     Val := Minimin(NewState, depth + 1, α);
     If Val < α
        Begin
        α := Val;
        BestNextState := NewState;
        End;
     End;
  Return(α, BestNextState);
End Minimin;
```

Table 1: Minimin Applied to Scheduling

the following recurrence relations.

$$ES(s) = Max_{r \in PRE(s)}\{EF(r)\}$$
$$EF(s) = ES(s) + ET(s)$$

Let T_i be the time by which machine M_i finishes the last job-step in its current queue. The goal of the Minimin search is to find the best next job-step to add to the current queues by doing a look-ahead search of fixed depth in the space of partial schedules (machine queues).

A job-step is considered "ready" if all its predecessors are either already executed or present in one or the other of the machine queues. At any given state, RTS first filters its machines by discarding those machines which do not have any ready job-steps waiting for their machine type. It then chooses the machine M_i which is expected to finish its queue the earliest, i.e., with a minimum T_i, and considers scheduling various job-steps on it. Each "ready" job-step s whose type matches that of machine M_i is a possible choice. For each such possible choice, Minimin creates a new state by assigning s to M_i, and updates the expected finish time of M_i's current queue using the above recurrence relations. RTS proceeds in depth first search in this manner until it reaches the search horizon.

At the leaves of the look-ahead search tree, the total time required to complete the remaining schedule must be estimated. Since none of the job-steps in the remaining schedule is assigned to a machine yet, their expected finish time cannot be exactly estimated. It is here that we rely on a heuristic lower bound.

Let T_K be the maximum of T_i of all machines M_i of type K. Let W_K be the total work remaining on machines of type K, i.e., the total expected time of all job-steps that need a machine of type K. Assume also that there are N_K machines of type K. Ignoring all the precedence constraints between the job-steps, the work remaining on machines of type K can be distributed as

follows. First fill each machine of type K until they reach the level T_K. This does not increase the make-span because it anyway takes that long to wait for the current queue to finish. This reduces the remaining work on machine of type K to $W_K - \Sigma_i\{T_K - T_i\}$, which may be distributed evenly among all the machines of type K in the best possible case. Hence, we observe that the time for completing the schedule must at least be as high as the following two bounds.

1. $Max_{K \in Machine-Types}\ (T_K + \frac{W_K - \Sigma\{T_K - T_i\}}{N_K})$
2. $Max_{J \in Jobs}(\Sigma_{s \in J} ET(s) + Min(T_i))$

The second lower bound above is obtained by noting that the job-steps in a single job should be executed sequentially. To the total time needed to execute any job, the minimum expected finish time of all machine queues is added. The finish time of the schedule is estimated to be the maximum of the above two bounds.

The above evaluation function is both admissible (never overestimates the true cost) and monotonic (monotonically non-decreasing along any path). This follows because, adding job-steps to the machine queues can only increase but never decrease the delays, by introducing more constraints. Since each step in the search adds a new job-step to the queues, the expected completion time is monotonically non-decreasing. The monotonicity is exploited by RTS by maintaining the current estimate α of the best schedule and evaluating f at internal nodes even before the search horizon is reached. Because f is monotonically non-decreasing, any path whose current estimate of the schedule cost exceeds the current value of α is guaranteed to yield only a worse solution and hence need not be pursued further. In other words, α-pruning would not sacrifice solution quality.

The estimated time for completion is backed up to the internal nodes from the leaves and finally to the root of the look-ahead search tree. The path that promises the lowest make-span is considered the best. An assignment of the first job-step in this path is made as suggested by this path. After this assignment, which corresponds to an action in the "real world," RTS takes a fresh look at its environment and starts a new cycle all over again.

4.3 Experimental Results

The problem specification is a 5-tuple. It consists of the number of jobs, the number of machines, the number of types of machines (this has to be less than the number of machines) and two numbers which specify the upper bounds on the number of steps for any job and the processing time for any step. Number of steps for each job is generated randomly, bound by the upper bound given in the problem specification. Each job-step is randomly assigned a machine type. Each job-step is also assigned some processing time randomly, bound from above as given in the problem specification.

We tested RTS on a sample of 39 randomly generated problems. Each problem had about 4-6 machines divided into 3-4 types, and 4-6 jobs each of which had about 5 steps, each step taking up to 6 units of time. We then ran the system with different look-ahead depths, and measured the total time to execute the whole schedule

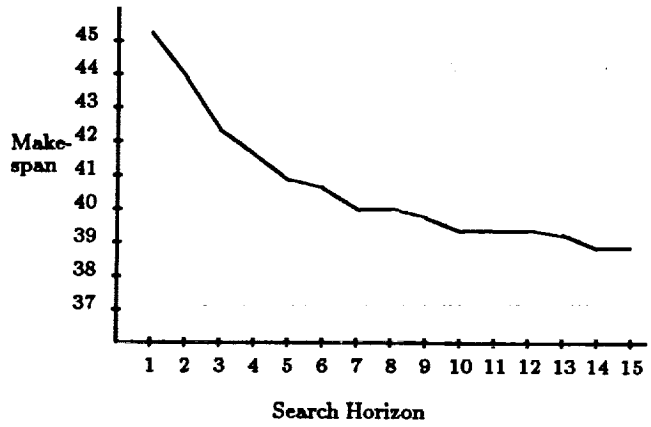


Figure 2: Solution quality improves with search horizon.

(make-span). We plotted the search horizon on the X-axis and the average make-span on the Y-axis.

The results show that the solution quality generally improves with search horizon, as expected. This tradeoff of search for solution quality was very favorable in the beginning, and tapered off toward the end. Although deeper searches resulted in better solutions on the whole, they also required exponentially larger number of nodes, taking exponentially longer time. In our context, the results indicate that a search horizen of 7 to 10 would achieve reasonably good schedules without extravagant search.

In general, it appears that a shallow look-ahead search would suffice to improve solution quality in this domain, which means that deep expensive searches may not be needed.

5 Future Work

The work reported here is preliminary and a lot remains to be done to make the ideas more practical and applicable in a real-world setting. A few of the promising directions to pursue are listed below.

Reactivity: One of the major reasons for building "real-time" systems is that they are more responsive to changes in their environment. This is especially crucial in the manufacturing domain, where unexpected events such as machine break-downs and tool failures are common. We believe that our system would respond better to such changes than a static scheduler. Indeed, it is possible to completely change the machine and job configuration before every cycle of the Minimin algorithm. The system should still be able to make locally optimal decisions with respect to its changed configuration. However, we expect that the system's behavior degrades gradually as the dynamics in the system configuration increases. It might also be expected that the usefulness of the look-ahead search decreases with increased dynamism. These hypotheses need to be experimentally verified.

Variable Depth Search: We assumed that the search horizon is fixed. However, this need not be the

case, and it is possible to change the search horizon across problems and even within the same problem. For example, a method called "Singular Extensions" proved very effective in game domains by focusing the search along narrow paths which appear significantly more promising than their nearest competitors [Anantharaman et al., 1990]. It seems possible to adapt this technique to real-time scheduling and search deeper at places in the search tree which appear promising. We can also add the iterative-deepening capability to Minimin, so that more time can be spent searching for a better schedule if time is available [Korf, 1985]. This also makes it an *anytime* algorithm in the sense of [Dean and Boddy, 1988], in that it can be interrupted at any time during its computation and asked to schedule the next job-step. The utility of the system's decisions is expected to increase with the time available to make the decision.

Learning: The performance of the system at a given search horizon depends mostly on the goodness of the evaluation function used to estimate the optimality of the schedule. Although our current evaluation function performed fairly well on the problems that we tested it on, it does not take into account factors such as bottleneck resources, which are crucial for a good scheduler. However, it is time-consuming and laborious to encode sophisticated evaluation functions. Besides, good evaluation functions are sensitive to the scheduler's environment, and hence may not be generally effective. Hence we plan to apply machine learning to learn effective evaluation functions [Lee and Mahajan, 1988]. There have already been some machine learning methods applied to scheduling domains [Kim, 1990, Shaw et al., 1990]. We think that significant improvements beyond current scheduling techniques can be achieved using machine learning.

6 Summary

In this paper we described a real-time scheduling system based on the Minimin algorithm and showed that it is effective and capable of producing good schedules with reasonably small effort. In particular, we showed that the schedule quality improves with increased look-ahead, confirming some of the results of Korf on Real-time Search in the scheduling domain. The future work includes evaluation function learning, variable depth searches, and demonstration of the reactivity of the system. Although much remains to be done, the preliminary results reported in this paper appear promising.

Acknowledgments

We thank Logen Logendran, Toshi Minoura, and Vijay Srinivasan for many fruitful discussions, and the department of computer science for financial support.

References

[Anantharaman et al., 1990] T.Anantharaman, M. Campbell, F. Hsu. Singular Extensions: Adding Selectivity to Brute Force Searching. *Artificial Intelligence*, 43(1), 1990.

[Dean and Boddy, 1988] T. Dean and M. Boddy. An Analysis of Time-dependent Planning. AAAI-88 Proceedings, Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[Fox and Smith, 1984] M.S. Fox and S.F. Smith. ISIS: A Knowledge-based System for Factory Scheduling. *Int. J. Expert Systems*, 1(1), 1984.

[Fox, 1987] M.S. Fox. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling.* Morgan Kaufmann, Los Altos, CA, 1987.

[Kempf et al., 1991] K. Kempf, C.L. Pape, S.F. Smith, and B.R. Fox. Issues in the Design of AI-Based Schedulers: A Workshop Report. *AI Magazine*, 11(5), 1991.

[Kim, 1990] S. Kim. *Schedule-Based Material Requirements Planning: An Artificial Intelligence Approach* M.S. Thesis, Department of Industrial and Manufacturing Engineering, Oregon State University, Corvallis, OR, 1990.

[Korf, 1985] R. Korf. Depth-first Iterative-deepening: An Optimal Admissible Tree Search. Artificial Intelligence, 27, 1985.

[Korf, 1990] R. Korf. Real Time Heuristic Search. Artificial Intelligence, March, 1990.

[Lee and Mahajan, 1988] K.F. Lee and S. Mahajan. A Pattern Classification Approach to Evaluation Function Learning. Artificial Intelligence, 36, 1988.

[Sadeh, 1991] N. Sadeh. Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling. Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, available as CMU-CS-91-102, 1991.

[Shaw et al., 1990] M.J. Shaw, S.C. Park, and N. Raman. *Intelligent Scheduling with Machine Learning Capabilities: The Induction of Scheduling Knowledge.* Technical Report, Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign, Urbana, IL, 1990.

[Smith et al., 1986] S. Smith, M.S. Fox and P.S. Ow. Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-based Factory Scheduling Systems. *AI Magazine*, Vol. 7, No. 4, 1986.

[Vollmann et al., 1988] T.E. Vollmann, W.L. Berry, and D.C. Whybark. Manufacturing Planning and Control Systems. Irwin, Homewood, IL, 1988.

[Zweben and Eskey, 1989] M. Zweben and M. Eskey. Constraint Satisfaction with Delayed Evaluation. IJCAI-89 Proceedings, AAAI Press, Menlo Park, 1989.

Planning, Scheduling, and Control for Automatic Telescopes

Mark Drummond Sterling Software
Keith Swanson NASA
John Bresina Sterling Software
Andy Philips Sterling Software
Rich Levinson Recom Software

NASA Ames Research Center
Mail Stop: 269-2
Moffett Field, CA 94035

1 Introduction

Making observations through telescopes is an activity of central importance to NASA. Whether a telescope is located on the Earth, is in orbit around the Earth as a satellite, is located on the moon, or is even on another planet, it presents an exciting and sometimes unique opportunity for gathering data about various astronomical phenomena. Telescopes have always been a scare resource, and astronomers have had to make do with extremely limited access. Further, an astronomer has been expected to be physically present at a telescope in order to gather data. Restricted access and local operation have limited the amount of data that can be gathered, and thus have directly contributed to fewer scientific results than might otherwise be expected.

Recent work by the Fairborn Observatory and AutoScope Corporation has freed astronomers from the need to be physically present at the telescope site. These organizations, working with astronomers, have designed and built control systems and associated hardware for the management and control of photoelectric telescopes; for a review of these Automatic Photoelectric Telescopes, or APTs, see Genet and Hayes (1989). While existing automation deals primarily with photoelectric telescopes, other sorts of telescope and other sorts of science are currently under investigation. The key point is that there is a perceived need, *within the astronomy community*, that the automation of local telescope control is desirable. Existing automation does not address all needs of all astronomers, but it does provide an excellent starting point. The eventual goal is what we call a "simplified management structure". The term refers to an approach to the management and control of telescopes that minimizes the number of people that must come between an astronomer's scientific goals and the telescopes required to realize those goals. A simplified management structure requires significantly more sophisticated telescope automation than is currently possible.

The Entropy Reduction Engine (ERE) project, carried out at the Ames Research Center, is focusing on the construction of integrated planning and scheduling systems. Specifically, the project is studying the problem of integrating planning and scheduling in the context of closed-loop plan use. The results of this research are particularly relevant when there is some element of dynamism in the environment, and thus some chance that a previously formed plan will fail. After a preliminary study of the APT management and control problem, we feel that it presents an excellent opportunity to demonstrate some of the ERE project's technical results. Of course, the alignment between technology and problem is not perfect, so planning and scheduling for APTs presents some new and difficult challenges as well.

This paper presents an argument for the appropriateness of ERE technology to the planning, scheduling, and control components of APT management. The paper is organized as follows. In the next section, we give a brief summary of the planning and scheduling requirements for APTs. Following this, in section 3, we give an ERE project precis, couched primarily in terms of project objectives. Section 4 gives a sketch of the match-up between problem and technology, and section 5 outlines where we want to go with this work.

2 APT problem summary

An Automatic Photoelectric Telescope is a telescope controlled by a dedicated computer for the purpose of gathering photometric data about various objects in the sky. While there are many sorts of photometric techniques, we focus on the technique known as *aperture photometry*. An excellent overview of aperture photometry is given by Hall and Genet (1988). In aperture photometry, and for current purposes, a *group* is the primitive unit to be scheduled. A group is a sequence of telescope and photometer commands defined by an astronomer. Any given astronomer has certain scientific goals, and he or she uses the group as the primary unit of instruction to an APT in order to achieve those goals. The language used to define groups is called ATIS (for Automatic Telescope Instruction Set); ATIS is an ASCII-based language for communicating with APTs (the *de facto* standard).

The communication process between astronomer and APT proceeds roughly as follows. First, an astronomer who wishes to use an APT forms a set of groups consistent with his or her scientific goals. These groups are written specifically in terms of a given telescope: since each telescope can vary slightly (instruments, optical characteristics, mechanical characteristics, location on the Earth), groups must be formulated in a telescope-specific manner. For any given APT there is a single person who

acts as a central clearing-house for usage requests; such a person is known in the vernacular as the APT's *Principal Astronomer*, or PA. Thus, once an astronomer has assembled his or her set of ATIS groups, they package the groups off to the appropriate PA. The PA collects together such sets from a variety of astronomers, attempts to ensure that the telescope is not overloaded, and then sends the complete set of groups off to the correct telescope. Actual communication between PA and APT is carried out by using personal computers, modems, and phone lines, but the particular technology isn't critical for the current discussion. The important aspect of the communication is that the PA can be located anywhere on the planet (in principle), and need only have access to an appropriate communication link.

The PA sends a set of groups to an APT, with the intention that these groups should be run for some time; eventually, the PA requests from the telescope the results that have been obtained under the execution of the given groups. The elapsed time varies, and depends on the telescope, the groups, the PA, and a variety of other factors. Of course the goal is to worry the astronomers (and the PA) as little as possible about the picayune details of day-to-day telescope management. Thus, the telescope is often left alone for significant periods of time (weeks, perhaps months). However long the telescope operates unattended, it is eventually asked for data, and this is returned to the PA as a "results file". The results file is also in the ATIS language, and it contains the groups that were executed, relevant observing parameters to help with data reduction, and the actual data obtained from the observations. The PA breaks this results file into the pieces that are relevant for the astronomers and sends each astronomer the results of his or her requested observations. Thus the cycle of group submission, compilation, execution, and data return can begin again when the astronomers discover that the data they've been given doesn't really tell them what they wanted to know (such are the joys of real science).

Of course, the interesting part of this process is the part that we've completely ignored so far; that is, the process by which the groups are accepted and executed by the local telescope controller. *This* is the interesting part, and it is with respect to this process that our planning and scheduling work can make a real difference. Currently, a program called ATIScope manages the execution of a file of groups. ATIScope runs locally at the given telescope, using observatory and telescope sensors to determine when to execute the provided groups. ATIScope has a variety of responsibilities, but we focus specifically on only one of these; namely, *group selection*.

At the core of ATIScope is a test that attempts to find a "currently" executable group. Roughly, a group is executable if the logical preconditions established by its astronomer-creator are met. Typically, these preconditions relate to the current date and time and to whether the moon is up or down. Additionally, an astronomer can specify a group *priority*, used by ATIScope to sort the groups in order of importance. There are other pseudo-preconditions that have to do with frequency of group execution, but we can safely ignore these for now.[1] Roughly, the core of ATIScope is a sense-check-execute loop. In sensing, all relevant environmental parameters are determined (date, time, moon status). ATIScope next checks to see which of the various possible groups are enabled according to the match between the current sensor values and the astronomer-provided preconditions. Let's call the set of groups that pass this matching test the *enabled* groups. The set of enabled groups is winnowed by the application of *group selection rules*. These rules express heuristic knowledge relating to the wisdom of executing any particular group before any other. In scheduling parlance, this scheme is sometimes called *heuristic dispatch*, since at any point in time, some task (here, a group) is "dispatched" for execution, and the selection of a task is determined, purely locally, by the application of some domain-specific heuristics. The information content of the heuristics used by ATIScope isn't critical for the current discussion (however, see Genet & Hayes, 1989, pp. 207–210). In the current context, heuristic dispatch is used to transform the set of enabled groups into a (hopefully) single group that is executed. If the heuristic group selection rules fail to winnow the set of enabled groups down to a single candidate, then the first group in the given list is selected (this, however, almost never happens, as the group selection rules normally produce a single preferred group). Following selection, the lucky group is executed, at which point telescope control is largely surrendered to the astronomer who wrote the group. Of course, there are safety checks to ensure that the astronomer's commands don't damage equipment, but if the commands are well-behaved (and if the weather cooperates), group execution finishes normally, and ATIScope is free to perform another iteration through its sense-check-execute loop.

How well does ATIScope do, in terms of schedule quality, by using this heuristic dispatch technique? One way of answering this question is to recall the old adage about an incredible dancing dog: the question of the quality of the dog's dancing needn't really be raised; one should instead be happy that the dog dances at all. ATIScope does, of course, provide an acceptable level of performance for some astronomers. There is no question, however, that the level of telescope performance can be dramatically improved by better group scheduling. With the heuristic dispatch technique, all decisions are *local* in the sense that no temporal look-ahead is performed to evaluate the ramifications of executing a given group. The system also has no memory of what it has done on previous nights, so groups cannot be selected with respect to some desired frequency of execution. Other scheduling techniques, such as those based on *temporal projection* (Drummond & Bresina, 1990), consider the impact of a given action by looking ahead in time to see how the current local choice impacts global objectives. Look-ahead is only sensible when astronomer objectives can be clearly and precisely formulated. Assuming that this can be done, it seems clear that a look-ahead scheduler

[1] The main factors that influence frequency of execution are a group's *probability* and *number of observations*; see Genet & Hayes (1989), p. 208.

can outperform the current ATIScope heuristic dispatch method. ATIScope, however, provides us with an existing level of performance against which all would-be contenders can be gauged.

3 ERE goals

The design of systems that can synthesize plans has been a long standing research topic in the field of Artificial Intelligence (AI). Such systems, called *planners*, are given a description of the problem at hand, and can synthesize a plan to solve that problem. Of course, a plan is merely a specification of a solution, and so must be executed to *actually solve* the given problem. Various sorts of "execution system" are possible; for instance, a plan might be executed by a manufacturing system, by a group of people, or by a robotic device; all that is required is a system that is capable of instantiating the plan's actions and thus producing the desired result. The design of these automatic planners has been addressed in AI since its earliest days, and a large number of techniques have been introduced in progressively more ambitious systems over many years. In the AI research branch at NASA Ames, the Entropy Reduction Engine (ERE) project is our focus for extending these classical techniques in a variety of ways. In this section we present the ERE project's overall goals; for more detail on the architecture itself, see Bresina & Drummond (1990), Drummond & Bresina (1990a, 1990b), and Drummond, Bresina, and Kedar (1991).

The Entropy Reduction Engine project is a focus for research on planning and scheduling in the context of closed-loop plan execution. The eventual goal of the ERE project is a set of software tools for designing and deploying integrated planning and scheduling systems that are able to effectively control their environments. To produce such software tools, we are working towards a better theoretical understanding of planning and scheduling in terms of closed-loop plan execution. Our overall project has two important sub-goals: first, we are working to integrate *planning* and *scheduling*; second, we are studying plan execution as a problem of *discrete event control*. Let's consider these complementary goals in a bit more detail.

Integrate planning and scheduling. Traditional AI planning deals with the selection of actions that are relevant to achieving given goals. Various disciplines, principally Operations Research, and more recently AI, have been concerned with the scheduling of actions; that is, with sequencing actions in terms of metric time and metric resource constraints. Unfortunately, most of the work in scheduling remains theoretically and practically disconnected from planning. Consider: a scheduling system is given a set of actions and returns, if possible, a schedule composed of those actions in some specific order. If the scheduler cannot find a satisfactory schedule, then it simply fails. The business of planning is to *select* actions that can solve a given problem, so what we need is an integrated planning and scheduling system to overcome the problems of scheduling alone. An integrated planning and scheduling system would be able to consider *alternative* sets of actions, unlike the stand-alone scheduler, which is unable to deviate from its given action set. We are working towards such an integrated system by incrementally constructing a unified theory of planning and scheduling that can be computationally expressed as practical software tools.

Study plan execution as a control theory problem. Most planning and scheduling work assumes that the job of the automatic system is done when a plan or schedule has been generated. Of course, one of the first things that you learn about plans is that they are rarely ever perfectly predictive of what will happen. As Dwight D. Eisenhower observed, "Plans are nothing, planning is everything". We agree with this view, since it tells us that the importance of planning does not lie in the existence of a *single* plan, but rather in a system's ability to re-plan and predictively manage plan execution failures in light of feedback from the environment. In the ERE project, we view plan execution as a problem in discrete event control; specifically, we formalize a plan as a simple type of feedback controller, and this gives us a new view on plan execution. Traditionally, plans have been executed by executing each component action in sequence. Our plans are functions that map from current sensor values and a desired goal into a set of acceptable control actions. The interpretation of the function is that any of the actions, if executed in the current situation, constitute an acceptable prefix to a sequence of actions that eventually satisfies the goal.

4 The match, in the abstract

The previous two sections have, in rough terms, explained the APT problem and overall ERE project goals. In this section, we consider how ERE technology promises to address key APT planning and scheduling issues. This section is optimistic and is, by necessity, "promissory", in the sense that some of what we suggest has yet to be rigorously demonstrated. This section reflects what we currently perceive as opportunities for using ERE technology on the APT planning, scheduling, and control problem.

First, the obvious: ERE is an architecture for producing systems that look ahead into the future, and by so doing, choose actions to perform. We feel that the ERE architecture is well-suited to the APT planning and scheduling problem in this regard. ATIScope currently does no look-ahead, so assuming that our system does, it should be able to produce better schedules. In fact, one of our research interests is the relationship between the cost of looking ahead and the increased "quality" of the system's actual behavior. In the APT domain, the quality of system behavior is determined by the amount and quality of the data returned by a given set of observations, and by the fairness of telescope allocation to the various astronomers' groups. Now ATIScope currently achieves a particular level of quality, and we expect to be able to increase this through some amount of look-ahead. But at what cost? When does look-ahead actually give rise to better system performance? ATIScope, while perhaps not producing the highest quality behavior, does so with great alacrity. A scheduling system that does any amount of look-ahead consumes more computational re-

sources than ATIScope, so the behaviors it produces had better be worth the increased cost. Of interest here is the impact of environmental factors on the underlying requirement for look-ahead: if the environment is completely predictable, and if a great deal of time is available in advance, then a scheduler that looks ahead extremely far into the future is apparently what's required. However, if the environment can change quickly, and change in unpredictable ways, then much of the work done by a look-ahead scheduler is wasted. The correct balance between look-ahead and heuristic dispatch is truly a function of the domain. There has been little empirical study of this issue in general, and we feel that APT planning and scheduling provides an excellent test case.

We have an algorithm for incremental, "anytime", planning (Drummond & Bresina, 1990) that we think will be useful in the APT context. While our algorithm has only been tested on relatively simple planning problems, we think that many of the underlying ideas transfer to scheduling as well. The essential idea is as follows: if a system has a limited amount of time to plan, and, having planned, is allowed to plan no further, then it makes sense for the system to make the best use of the available time by incrementally improving its current plan until time runs out. Our algorithm, called *traverse and robustify*, does this. It uses information about possible execution outcomes to predictively patch errors, before they actually occur. By doing this the algorithm attempts to maximize the probability that the plan it finds will satisfy the user's objectives. This algorithm promises to be useful in a scheduling context, and APTs provide an appropriate test-domain. If we think of the scheduler as running during the day (remote from the telescope, in the PA's place of work), and imagine that the finished schedule will be shipped to the telescope for overnight execution, then one would like the schedule produced to be of the highest possible robustness given the available time, so our algorithm seems appropriate.

5 Objectives

First and foremost, we must define an appropriate objective function for APT observation schedules. How well can this objective function be formalized? How will we notate it? That is, what will be our language for writing down the objective function? For the problems we have studied to date, our language of *behavioral constraints* has been adequate. The current behavioral constraint language allows a user to give arbitrary conjunctions and disjunctions of predicates that must be maintained true (or prevented from being true) throughout an interval of time (see Drummond & Bresina, 1990, for more detail). Is this language adequate for expressing the sorts of goals that astronomers have? Will we need to drop into the language of arbitrary mathematics? Of course, this *is* what most of decision analysis does, so should we expect to do any better? We hope to devise a new sort of behavioral constraint language, specifically designed to allow astronomers to define APT observation schedule preferences. Even with such a specially-designed language, there's a remaining second-order problem: the PA (or other user) must be able to define what constitutes a fair and equitable tradeoff of telescope and instrument allocation between different astronomers. Of course, we don't want a person (the PA or other user) to have to specify the *specific* tradeoff for each given scheduling instance, but the *general form* of the tradeoff function used must be defined by a user. These and other interesting issues lurk in the vicinity of schedule objective functions.

We are fortunate to have access to several APT experts. One expert is an original APT architect who has founded a firm to commercially produce APTs. The other experts are experienced photometric astronomers, one of whom is an active APT user and has acted as a principal astronomer in the past. It is our hope that by working directly with this diverse and experienced group of APT developers and users, we will be able to produce planning and scheduling tools of use to a large number of photometric scientists.

In the short term (6 months), we plan to produce an interactive scheduling tool for use by ourselves, with our APT user acting as a local domain expert. The tool will help a user analyze a given set of groups by interactively determining the best sequence in which the groups should be run, providing help with the selection of the best sequence, but leaving the user free to intervene should he or she so desire. The system will automatically compile out a set of group selection rules that will produce the desired set of group execution sequences. Essentially, our system will be used to compile a set of scheduling dispatch rules that are designed specifically for the target set of groups, to be run on the target telescope, for a particular night of observations. We have studied the problem in some detail and are confident that our existing techniques for compiling such rules will work on the APT problem (see Drummond, 1989).

We have access to an APT simulator and will use this to evaluate our system's evolving capabilities. Of course, the eventual goal of this research is to remove humans from the control loop, so this first short term objective might not appear to be a tremendous step forward. It is, in fact, best construed as a step "sideways", prefatory to a giant *leap* forward. We will use our interactive scheduling tool to gain experience with the APT planning and scheduling problem; our eventual goal is to entirely automate the decisions still made by a human user. This first sideways step towards a decision support system is thus not an end in itself, but only a means to a bigger, more important end.

In the medium term (1 year), we plan to produce a better, incremental scheduler designed to replace the ATIScope system. Our new scheduler would be based on experience gained with building our look-ahead scheduling decision-support system. Our scheduler, like ATIScope, would accept a set of groups from the PA (or various astronomers, thus freeing the PA entirely from any scheduling responsibilities), and would schedule and execute these in a flexible manner. This first prototype automatic scheduler would not provide a very sophisticated language of scientific objectives; instead, it would allow a user or users to specify a set of groups, and would attempt to better the current level of performance obtained by ATIScope by doing temporal projection (look-

ahead) and history recording (remember-behind).

Our long term plan (2 years) is to extend the language of objectives to allow users to specify interesting scientific objective functions. The first test case would be a facility for filling out a desired light curve. Other test cases will be established in conjunction with our APT experts. The extra functionality offered at this stage of development will be that of *planning*, as opposed to pure *scheduling*. It is at this point that our system really begins to offer increased scientific power over that of the traditional ATIScope-style system. Until now, we have only sought to increase the "quality" of the group execution sequences. Here, we seek to increase the expressiveness of the language that is used by an astronomer to specify scientific objectives.

Once individual APTs are routinely being used by remotely located astronomers, with nearly all scheduling conflicts being resolved automatically, many new opportunities arise. For instance, at this point it becomes practical to consider a network of relatively inexpensive telescopes, located around the world, which are able to provide continuous observation of astronomical objects. While possible now for exceptional events (supernova), the logistical overhead precludes wider practice.

We are purchasing and intend to operate a 16-inch APT. This telescope will be located in northern California, and will be made available to members of the scientific community, with the focus being on educational institutions. We will make our system available over the InterNet, such that remotely located astronomers can simply Email request files to our system. Our system will accept a number of requests from various users, schedule them, and download the set of groups and group selection rules to the telescope. Users will receive their requested data via return Email or will be given access to an FTP site where their data may be recovered. This system will provide the first example of a totally automated telescope planning, scheduling, and control system. We plan to have the system operating totally autonomously as soon as possible.

We hope that our demonstration of fully automatic telescope operations will serve as groundwork for new applications of simplified telescope operations. Of particular interest is the possibility of placing a number of small telescopes on the moon (Genet *et al*, 1992). Such a telescope facility would be an excellent test of our "simplified management structure". We feel that ERE can provide a solid base for the development of integrated telescope planning, scheduling, and control systems that help to make this simplified management structure a reality.

Acknowledgements

Comments from Russ Genet, David Genet, Butler Hine, and Bill Borucki have been extremely useful; to each of them, our thanks.

References

[1] Bresina, J., Drummond, M., and Kedar, S. *Forthcoming.* Reactive, Integrated Systems Pose New Problems for Machine Learning. To appear in the volume on Learning in AI Planning and Scheduling Systems; Langley, P., and Minton, S. (eds).

[2] Bresina, J., and Drummond, M. 1990. Integrating Planning and Reaction: A Preliminary Report. *Proceedings of the AAAI Spring Symposium Series* (session on Planning in Uncertain, Unpredictable, or Changing Environments).

[3] Drummond, M. 1989. Situated Control Rules. *Proceedings of Conference on Principles of Knowledge Representation & Reasoning.* Toronto, Canada.

[4] Drummond, M., and Bresina, J. 1990a. Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction. In proc. of *AAAI-90.*

[5] Drummond, M., and Bresina.J. 1990b. Planning for Control. In proc. of *Fifth IEEE International Symposium on Intelligent Control*, published by the IEEE Computer Society Press, Philadelphia, PA. pp 657–662.

[6] Drummond, M., Bresina, J., and Kedar, S. 1991. The Entropy Reduction Engine: Integrating Planning, Scheduling, and Control. *Proceedings of the AAAI Spring Symposium Series* (session on Integrated Intelligent Architectures).

[7] Hall, D.S., and Genet, R.M. 1988. Photoelectric Photometry of Variable Stars. Wilmann-Bell, PO Box 35025, Richmond, VA (2nd edition).

[8] Genet, R.M, Genet, D.R., Talent, D.L., Drummond, M., Hine, B., Boyd, L.J., and Trueblood, M. 1992. Multi-Use Lunar Telescopes. A chapter in "Robotic Observatories in the 1990's". Edited by Alexei V. Filippenko, published by the Astronomical Society of the Pacific Conference Series.

[9] Genet, R.M., and Genet, D.R. 1991. Dynamic Scheduling of Astronomical Observations By Intelligent Telescopes (An Informal Discussion of the Problem). Draft Paper, AutoScope Corporation, Mesa, AZ.

[10] Genet, R.M., and Hayes, D.S. 1989. Robotic Observatories: A Handbook of Remote-Access Personal-Computer Astronomy. Published by the AutoScope Corporation, Mesa, AZ.

O-Plan2: The Open Planning Architecture

Brian Drabble, Richard Kirby and Austin Tate
Artificial Intelligence Applications Institute
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
United Kingdom

The O-Plan2 Project at the Artificial Intelligence Applications Institute of the University of Edinburgh is exploring a practical computer based environment to provide for specification, generation, interaction with, and execution of activity plans. O-Plan2 is intended to be a domain-independent general planning and control framework with the ability to embed detailed knowledge of the domain.

- A hierarchical planning system which can produce plans as partial orders on actions.
- An agenda-based control architecture in which each control cycle can post pending tasks during plan generation. These pending tasks are then picked up from the agenda and processed by appropriate handlers (*Knowledge Sources*).
- The notion of a "plan state" which is the data structure containing the emerging plan, the "flaws" remaining in it, and the information used in building the plan.
- Constraint posting and least commitment on object variables.
- Temporal and resource constraint handling. The algorithms for this are incremental versions of Operational Research methods.
- O-Plan2 is derived from the earlier Nonlin planner from which extended the ideas of Goal Structure, Question Answering and typed conditions.
- We have extended Nonlin's style of task description language Task Formalism (TF).

O-Plan2 could be applied to the following types of problems:

- planning and control of space probes such as VOYAGER, etc.
- project management in large scale construction projects.
- planning and control of supply logistics.

The Scenario

- A user specifies a task that is to be performed through some suitable interface. We call this process *job assignment.*
- A *planner* plans and (if requested) arranges to execute the plan to perform the task specified.
- The *execution system* seeks to carry out the detailed tasks specified by the planner while working with a more detailed model of the execution environment.

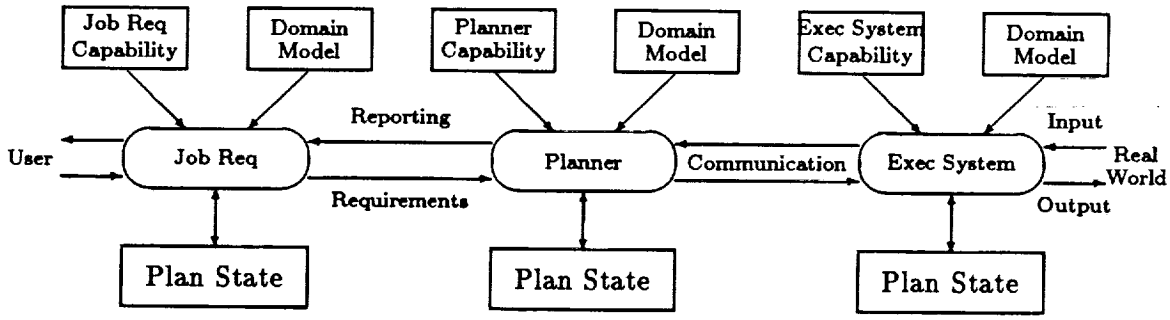


Figure 1: Communication between Central Planner and Ex. Agent

We have deliberately simplified our consideration to three agents with these different roles and with possible differences of requirements for user availability, processing capacity and real-time reaction to clarify the research objectives in our work.

A common representation is sought to include knowledge about the capabilities of the planner and execution agent, the requirements of the plan and the plan itself either with or without flaws (see Figure 1).

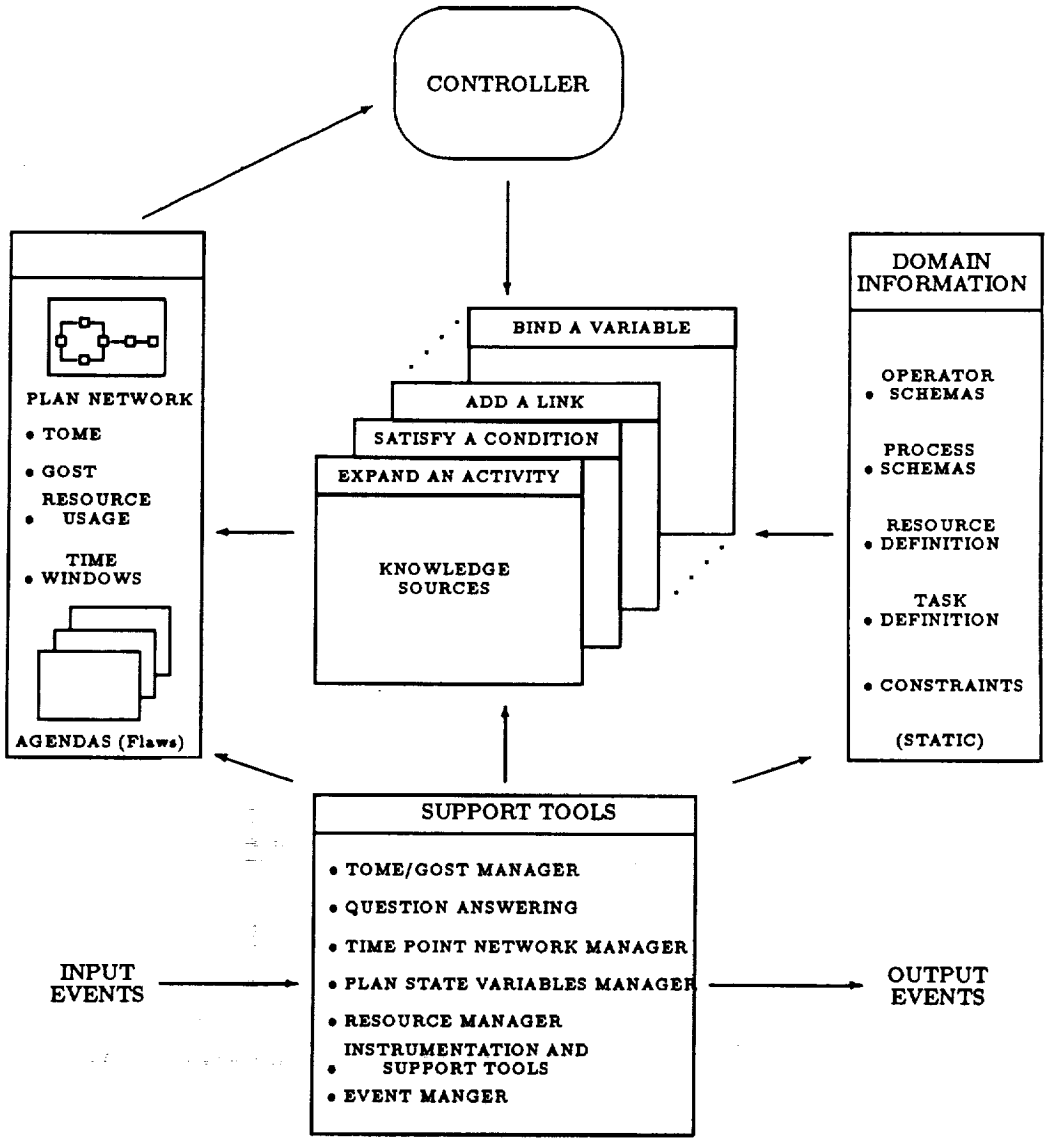


Figure 2: O-Plan2 Architecture

Developer Interface

O-Plan2 is implemented in Common Lisp on Unix Workstations with an X-Windows interface. It is designed to be able to exploit multi-processors in future and thus has a clear separation of the various components (as shown in Figure 2). Each of these may be run on a separate processor and multiple *platforms* may be provided to allow for parallelism in knowledge source processing. A sample screen image as seen by the O-Plan2 developer or an interested technical user is shown in Figure 3.

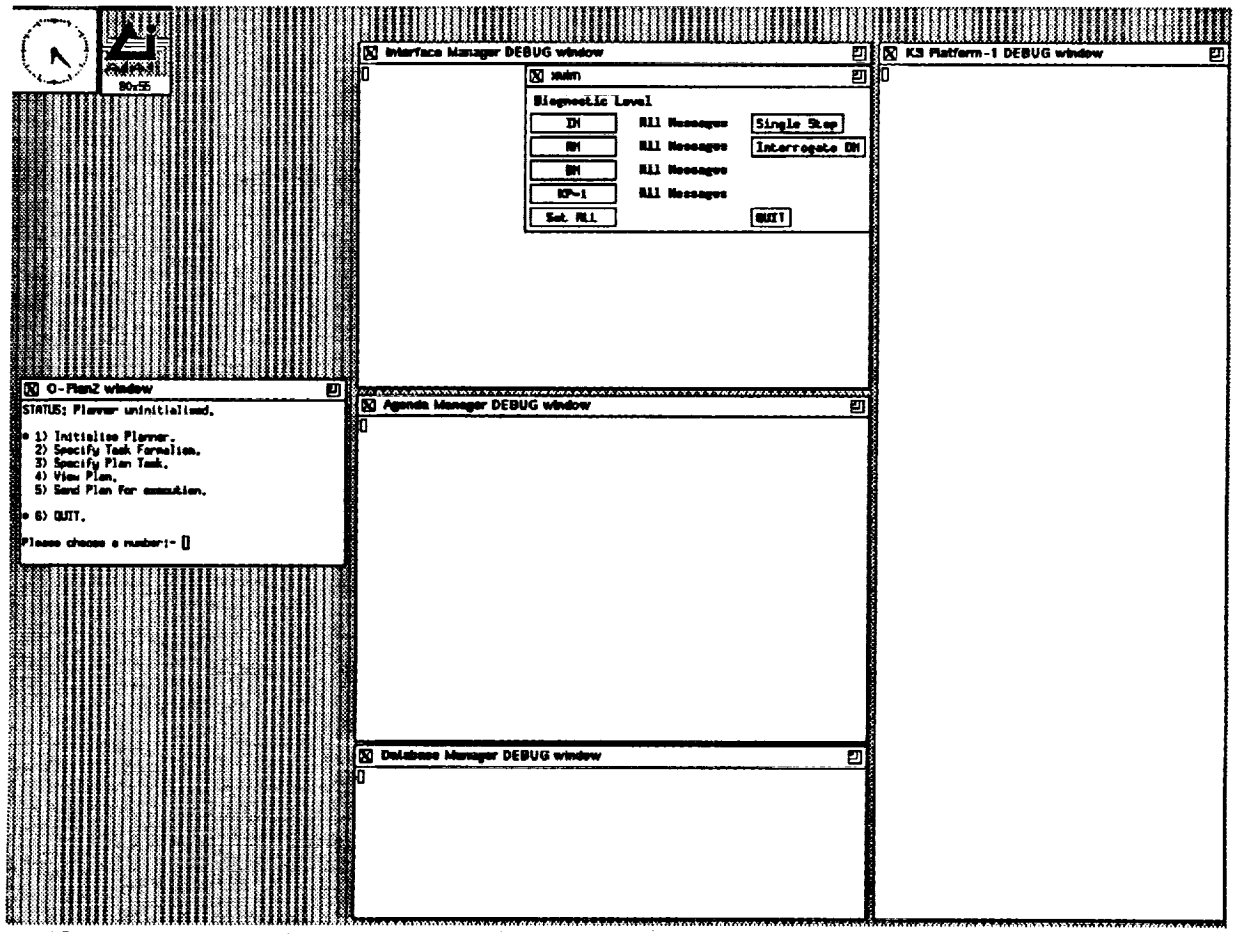


Figure 3: Example Developer Interface for the O-Plan2 Planning Agent

User Interface

AI planning systems are now being used in realistic applications by users who need to have a high level of graphical support to the planning operations they are being aided with. An interface to AutoCAD has been built to show the type of User Interface we envisage (see Figure 4). The lower window draws the plan as a graph, and the upper right window can be useed for simulations of the state of the world at points in the plan.

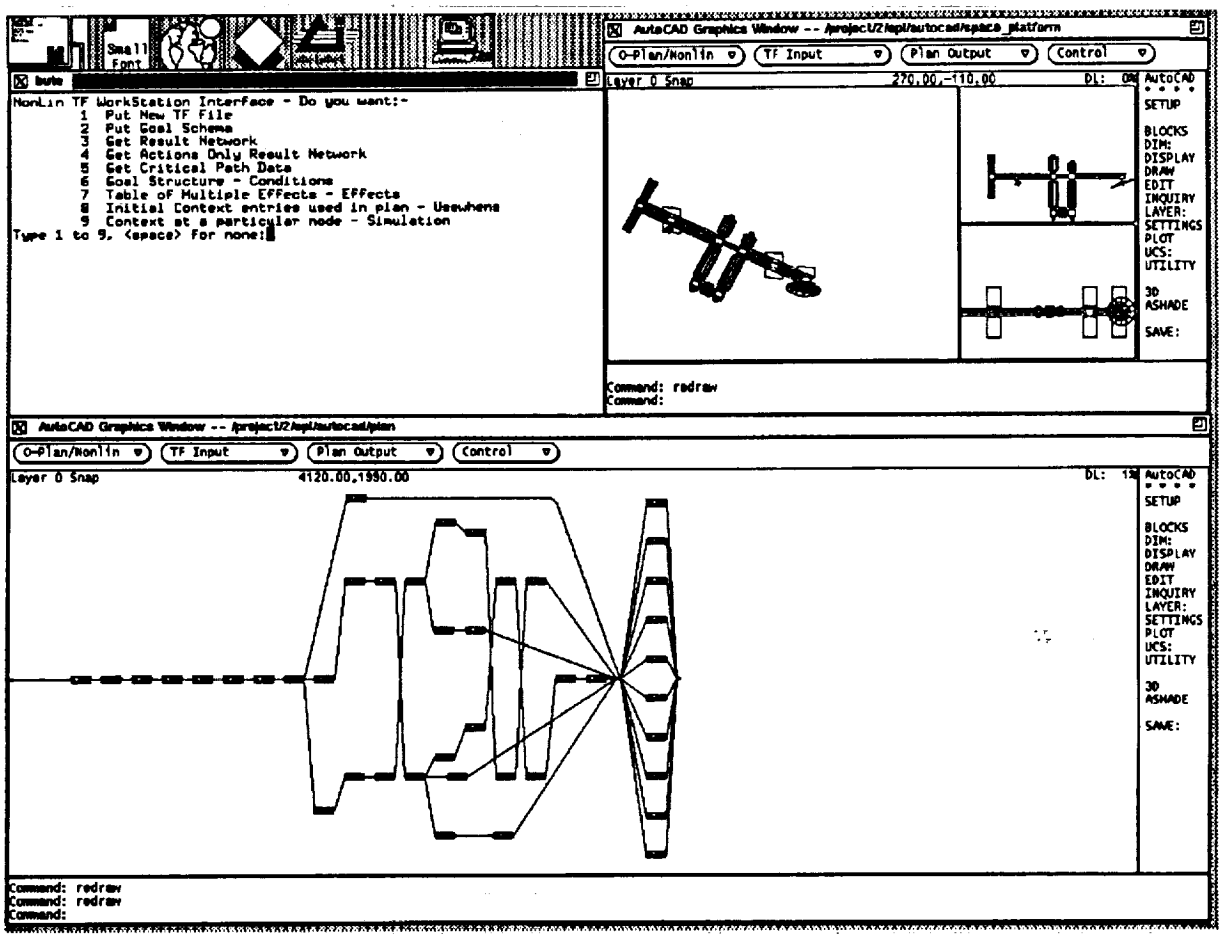


Figure 4: Example Output of the AutoCAD-based User Interface

Rescheduling with Iterative Repair

Monte Zweben
Eugene Davis*
Brian Daun†
Michael Deale‡
NASA Ames Research Center
M.S. 269-2
Moffett Field, California 94035

Abstract

This paper presents a new approach to rescheduling called *constraint-based iterative repair*. This approach gives our system the ability to satisfy domain constraints, address optimization concerns, minimize perturbation to the original schedule, and produce modified schedules quickly. The system begins with an initial, flawed schedule and then iteratively repairs constraint violations until a conflict-free schedule is produced. In an empirical demonstration, we vary the importance of minimizing perturbation and report how fast the system is able to resolve conflicts in a given time bound. These experiments were performed within the domain of Space Shuttle ground processing.

Introduction

Space Shuttle ground processing encompasses the inspection, repair, and refurbishment of space shuttles in preparation for launch. During processing the Kennedy Space Center (KSC) flow management team frequently modifies the schedule in order to accommodate unanticipated events, such as lack of personnel availability, unexpected delays, and the need to repair newly discovered problems. If the Space Shuttle ground processing turnaround time could be shortened, even by a small percentage, millions of dollars would be saved. This paper presents GERRY, a general scheduling system being applied to the Space Shuttle ground processing problem.

As originally put forth in [Smi85], rescheduling systems should satisfy domain constraints, address optimization concerns, minimize perturbation to the original schedule, and produce modified schedules quickly. GERRY [Zwe90] is a novel approach to rescheduling that addresses these concerns and gives the user the ability to individually modify each criteria's relative importance. In an empirical demonstration of the system, we vary the importance of minimizing perturbation and report how fast the system is able to converge to a conflict-free schedule (or a near-conflict-free schedule) in a given time bound.

*Recom Technologies
†Lockheed Artificial Intelligence Center
‡Lockheed Space Operations Company

Problem Class: Fixed Preemptive Scheduling

Scheduling is the process of assigning times and resources to the tasks of a plan. Scheduling assignments must satisfy a set of domain constraints. Generally, these include temporal constraints, milestone constraints, and resource requirements. The Space Shuttle domain also requires the modeling of state variables. State variables are conditions that can change over time; examples include the positions of switches, the configuration of mechanical parts, and the status of systems. Tasks might be constrained by the state conditions (a *state requirement*) and they might cause a change in state condition (a *state effect*).

Preemption is an additional complicating factor introduced by the Space Shuttle problem. In preemptive scheduling, each task is associated with a calendar of legal work periods that determine when the task must be performed.

Preemption effectively splits a task into a set of subtasks. Resource and state constraints are annotated as to whether they should be enforced for each individual subtask (and not during the suspended periods between subtasks) or during the entire time spanning from the first subtask until the last (including suspended periods). Preemptive scheduling requires additional computational overhead since for each task the preemption times must be computed and appropriate constraint manipulation for each time assignment must be performed.

Rescheduling

Rescheduling is necessitated by changes that occur in the environment. Systems can respond in three ways: schedule again from scratch, remove some tasks from the schedule and restart from an intermediate state, or repair the schedule where the changes occurred.

Scheduling from scratch reconsiders the scheduling problem in light of exogenous events. In [Ham86], [Sim88] and [Kam90], the authors argue that it is

more efficient to modify flawed plans than to plan from scratch. Moreover, since scheduling from scratch will generate a new schedule without considering any values from the previous solution, a high amount of perturbation is likely to occur.

To schedule from an intermediate state, all tasks affected by the exogenous events are first removed from the schedule; scheduling then is resumed considering the exogenous events. For example, suppose T_1, T_2, T_3, and T_4 are tasks in a schedule that are constrained to be sequential in the order shown. If T_3 is delayed, then only T_3 and T_4 would be removed from the schedule before restarting, because the other tasks are unaffected by the delay. This approach is complex, because a dependency analysis is required to determine whether a schedule modification could affect any particular task. Further, even though a task is unaffected by an exogenous event, it may be possible to provide a better schedule by reconsidering its assignments.

GERRY adopts the third approach, which is to repair the constraints that are violated in the schedule.

Constraint-Based Iterative Repair

Constraint-based iterative repair begins with a complete schedule of unacceptable quality and iteratively modifies it until its quality is found satisfactory. The quality of a schedule is measured by the *cost* function: $Cost(s) = \sum_{c_i \in Constraints} Penalty_{c_i}(s) * Weight_{c_i}$, which is a weighted sum of constraint violations. The *penalty* function of a constraint returns an integer reflecting its degree of violation. The *weight* function of a constraint returns an integer representing the importance or utility of a constraint.

In GERRY, repairs are associated with constraints. Local repair heuristics that are likely to satisfy the violated constraint can then be encoded without concern for how these repairs would interact with other constraints. Of course local repairs do occasionally yield globally undesirable states, but these states, if accepted (see below), are generally improved upon after multiple iterations.

Repairing any violation typically involves moving a set of tasks to different times: at least one task participating in the constraint violation is moved, along with any other tasks whose temporal constraints would be violated by the move. In other words, all temporal constraints are preserved after the repair. We use the Waltz constraint propagation algorithm over time intervals [Wal75, Dav87] to carry this out (thus enforcing a form of arc-consistency [Mac77, Fre82]). The algorithm recursively enforces temporal constraints until there are no outstanding temporal violations.[1] This scheme can be computationally expensive, since moving tasks involves checking resource constraints, calculating preemption intervals, etc.

[1]Note that all temporal constraints are also preserved (using the same Waltz algorithm) whenever the user manually moves tasks.

At the end of each iteration, the system re-evaluates the cost function to determine whether the new schedule resulting from the repairs is better than the current solution. If the new schedule is an improvement, it becomes the current schedule for the next iteration; if it is also better than any previous solution, it is stored as the best solution so far. If it is not an improvement, with some probability it is either accepted anyway, or it is rejected and the changes are not kept. When the changes are not kept, it is hoped that repairs in the next iteration will select a different set of tasks to move and the cost function will improve.

The system sometimes accepts a new solution that is worse than the current solution in order to escape local minima and cycles. This stochastic technique is referred to as simulated annealing [Kir83]. The escape function for accepting inferior solutions is: $Escape(s, s', T) = e^{-|Cost(s)-Cost(s')|/T}$ where T is a "temperature" parameter that is gradually reduced during the search process. When a random number between 0 and 1 exceeds the value of the escape function, the system accepts the worse solution. Note that escape becomes less probable as the temperature is lowered.

In GERRY the types of constraints that can contribute to the cost function include the resource, state, and perturbation constraints.

Resource Constraints The penalty of a resource capacity constraint is 1 if the resource is overallocated. If K simultaneous tasks overallocate the resource, then all K tasks are considered violated. One of these tasks will be selected in an attempt to repair as many of the K violations as possible. The heuristic used to select this task considers the following information:

Fitness: Move the task whose resource requirement most closely matches the amount of overallocation. A task using a significantly smaller amount is not likely to have a large enough impact on the current violation being repaired. A task using a far greater amount is more likely to be in violation wherever it is moved.

Temporal Dependents: Move the task with the fewest number of temporal dependents. A task with many dependents, if moved, is likely to cause temporal constraint violations and result in many task moves.

Distance of Move: Move the task that does not need to be shifted significantly from its current time. A task that is moved a greater distance is more likely to cause other tasks to move as well, increasing perturbation and potentially causing more constraint violations.

For each of the tasks contributing to the violation, the system considers moving the task to its *next earlier* and *next later* times such that the resource is available, rather than exploring many or all possible times.

This reduces the computational complexity of the repair and, like the "distance to move" criterion above, tends to minimize perturbation.

Each candidate move is scored using a linear combination of the *fitness*, *temporal dependents*, and *distance to move* heuristic values. The repair then chooses the move stochastically with respect to the scores calculated. After the repair is performed, the Waltz algorithm moves other tasks in order to preserve temporal constraints.

State Constraints The penalty of a state constraint is 1 if the required state is not set. To repair a state constraint, the task with the violated state requirement is reassigned to a different time when the state variable takes on the desired value. Similar to the resource capacity constraints, the system considers only the next earlier and next later acceptable times and selects between these randomly. We are currently investigating improvements to this repair and expect to extract more useful heuristics from our experts. One effort underway is the development of a repair that can introduce new tasks into the schedule, thus yielding a behavior generally associated with AI planning systems.

Perturbation Constraint The penalty function of the perturbation constraint returns the number of tasks that differ from their original temporal assignments. Since the weighted penalty of this constraint contributes to the cost of a solution, schedules with significant perturbation tend to be rejected at the close of an iteration. We are in the process of experimenting with repairs for this constraint that augment the information provided by its penalty and weight. Below we show how varying the weight of this constraint can affect convergence speed and solution quality.

Experiments

The problem domain for the experiments consisted of the tasks, resources, temporal constraints, and resource constraints from the STS-43 Space Shuttle ground processing flow. A rescheduling problem was generated by taking the original conflict-free schedule and randomly moving ten tasks. Five such problems were generated for the results reported below. The first and last tasks of the original schedule were anchored in time so repairs could not extend the duration of the entire flow.

In the experiments, we maintained the resource constraint weight at ten, and varied the perturbation constraint weight from zero (perturbation was of no concern) to 50 (perturbation was extremely important). The system terminated its search when all resource constraints were satisfied or when its run time exceeded ten minutes. Upon termination, the system returned the best solution found. Each rescheduling run was performed with the same settings 20 times in order to minimize stochastic variance.

Figure 1 presents the results of our experiments on the five problems from three different perspectives. The first graph plots the number of perturbations for the returned solution against the weight of the perturbation constraint. As expected, with a higher perturbation weight, the best solution has fewer perturbations.

The second plot shows the quality of a returned solution (measured as the number of violated resource constraints), as a function of the perturbation weight. As the graph shows, GERRY has more difficulty satisfying resource constraints as perturbation becomes more important.

Finally, the third plot shows the convergence time (in cpu seconds) as a function of the perturbation weight. Average time to solution generally increased as the perturbation weight increased.

It is interesting to note that for smaller weights on the perturbation constraint (< 20), the increase in resource violations is small while the drop in number of perturbations is fairly large. As the perturbation weight increases beyond 20, resource violations rise quickly, and the drop in perturbations slows.

In summary, our algorithm is interruptible, restartable, and outputs a solution when terminated. As demonstrated in Figure 2, the solution quality increases as a step-function of time. These runs are representative of the system's general performance.

Related Work

Our work was heavily influenced by previous constraint-based scheduling [Fox87, Fox84, Sad89] and rescheduling efforts [Ow,88].

ISIS [Fox87] and GERRY both have metrics of constraint violation (the *penalty* function in GERRY) and constraint importance (the *weight* function in GERRY). In contrast with our repair-based method, ISIS uses an incremental, beam search through a space of partial schedules and reschedules by restarting the beam search from an intermediate state.

OPIS [Fox84, Ow,88], which is the successor of ISIS, opportunistically selects a rescheduling method. It chooses between the ISIS beam search, a resource-based *dispatch* method, or a repair-based approach. The *dispatch* method concentrates on a bottleneck resource and assigns tasks to it according to the dispatch rule. The *repair* method shifts tasks until they are conflict-free. These "greedy" assignments could yield globally poor schedules if used incorrectly. Consequently, OPIS only uses the dispatch rule when there is strong evidence of a bottleneck and only uses the repair method if the duration of the conflict is short. In contrast, GERRY uses the simulated annealing search to perform multiple iterations of repairs, possibly retracting "greedy" repairs when they yield prohibitive costs.

Our use of simulated annealing was influenced by the experiments performed in [Joh90a, Joh90b]. In contrast with our constraint-based repair, their re-

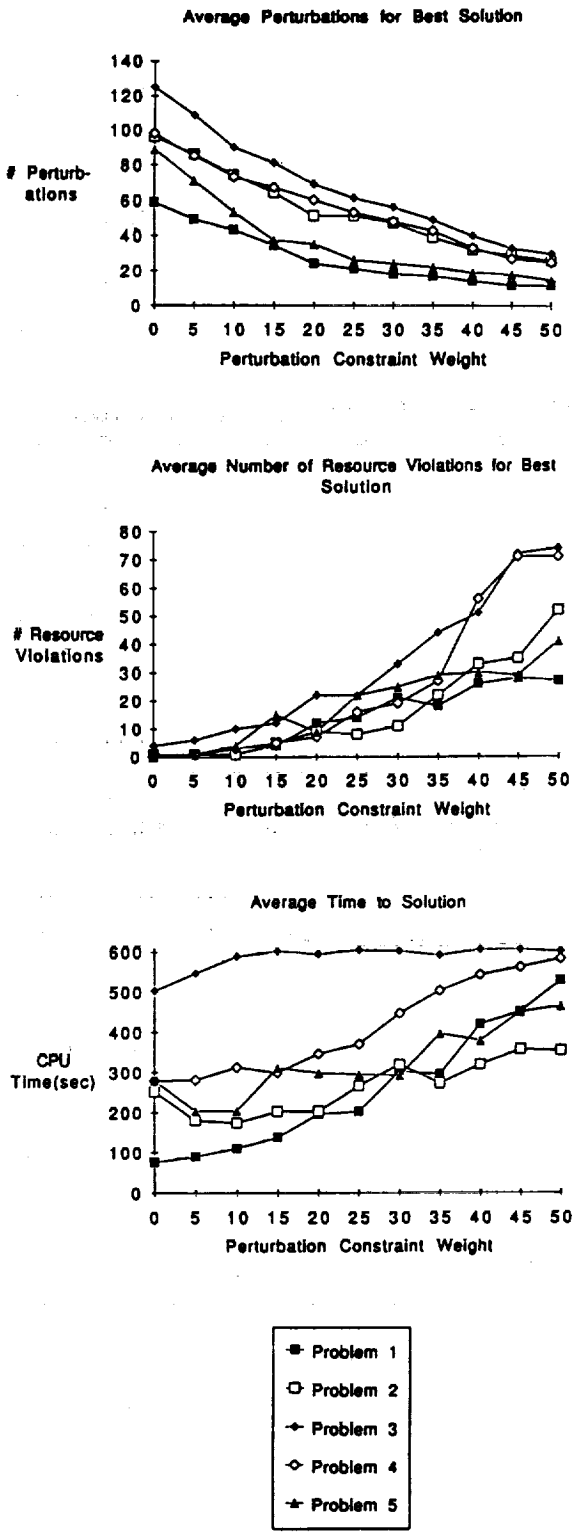


Figure 1: Experimental Results: number of perturbations versus perturbation constraint weight; number of resource violations versus perturbation constraint weight; average run time versus perturbation constraint weight

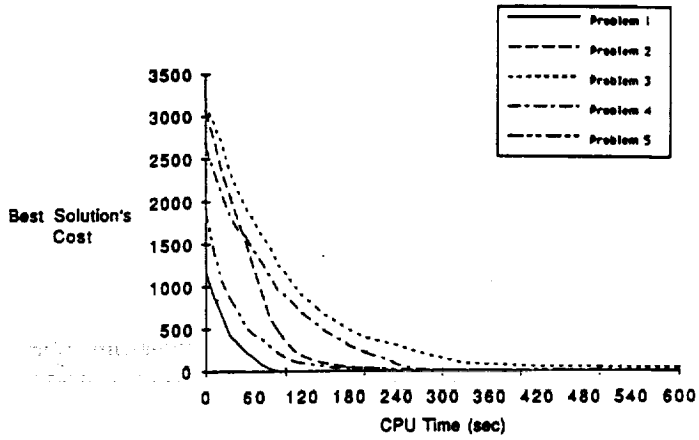


Figure 2: Best Cost versus Run Time

pairs were generally uninformed. In [Zwe92b] we show that constraint repair knowledge improves convergence speed.

The repair-based scheduling methods considered here are related to the repair-based methods that have been previously used in AI planning systems such as the "fixes" used in Hacker [Sus73] and, more recently, the repair strategies used in the GORDIUS[Sim88] generate-test-debug system, and the CHEF cased-based planner [Ham86].

In [Min90], it is shown that the *min-conflicts* heuristic is an extremely powerful repair-based method. For any violated constraint, the *min-conflicts* heuristic chooses the repair that minimizes the number of remaining conflicts resulting from a one-step lookahead. However, in certain circumstances this lookahead could be computationally prohibitive. In [Zwe91], the authors investigate the tradeoff between the informedness of a repair and its computationally complexity. There it is shown that the resource repair described above outperformed a lookahead heuristic on the STS-43 Space Shuttle problem. However, on smaller problems the lookahead heuristic was superior.

Our technique is also closely related to the Jet Propulsion Laboratory's OMP scheduling system [Bie91]. OMP uses procedurally encoded patches in an iterative improvement framework. It stores small snapshots of the scheduling process (called *chronologies*) which allow it to escape cycles and local minima.

[Mil88], [Bel85], and [Dru90] describe other efforts that deal with resource and deadline constraints.

Conclusions and Future Work

Our experiments suggest that our constraint framework and the knowledge encoded in this framework is an effective search tool that allows one to adjust the importance of schedule perturbation and other objective criteria. The framework is modular and extensible

in that one can declare new constraints as long as their weight, penalty, and repair functions are provided.

In future experiments, we hope to better characterize the components of repair informedness and computational complexity. We are currently evaluating candidate metrics of problem difficulty that could be used to guide the selection of repair heuristics. Additionally, we are developing machine learning techniques that allow systems to learn when to dynamically switch between heuristics [Zwe92a].

With respect to the Space Shuttle application, the system is expected to be in daily use sometime this year. Our most significant barrier is gathering accurate models of tasks in an electronic form. We also plan to develop constraints that minimize weekend labor.

References

[Bel85] Bell, C.,Currie, K., and Tate, A. Time Window and Resource Usage in O-Plan. Technical report, AIAI, Edinburgh University, 1985.

[Bie91] Biefeld, E. and Cooper, L. Bottleneck Identification Using Process Chronologies. In *Proceedings of IJCAI-91*, Sydney, Austrailia, 1991.

[Dav87] Davis, E. Constraint Propagation with Interval Labels. *Artificial Intelligence*, 32(3), 1987.

[Dru90] Drummond, M. and Bresina J. Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction . In *Proceedings of AAAI-90*, 1990.

[Fox84] Fox, M. and Smith, S. A Knowledge Based System for Factory Scheduling. *Expert System*, 1(1), 1984.

[Fox87] Fox, M. *Constraint-Directed Search: A Case Study of Job Shop Scheduling*. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.

[Fre82] Freuder, E. C. A Sufficient Condition for Backtrack-Free Search. *J. ACM*, 29(1), 1982.

[Ham86] Hammond, K. J. CHEF: A Model of Case-Based Planning. In *Proceedings of AAAI-86*, 1986.

[Joh90a] Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C. Optimization By Simulated Annealing:An Experimental Evaluation, Part I (Graph Partioning). *Operations Research*, 1990.

[Joh90b] Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C. Optimization By Simulated Annealing:An Experimental Evaluation, Part II (Graph Coloring and Number Partioning). *Operations Research*, 1990.

[Kam90] Kambhampati, S. A Theory of Plan Modification. In *Proceedings of AAAI-90*, 1990.

[Kir83] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P. Optimization by Simulated Annealing. *Science*, 220(4598), 1983.

[Mac77] Mackworth, A.K. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1), 1977.

[Mil88] Miller, D., Firby, R. J., Dean, T. Deadlines, Travel Time, and Robot Problem Solving. In *Proceedings of AAAI-88*, St. Paul, Minnesota, 1988.

[Min90] Minton, S., Phillips, A., Johnston, M., Laird., P. Solving Large Scale CSP and Scheduling Problems with a Heuristic Repair Method. In *Proceedings of AAAI-90*, 1990.

[Ow,88] Ow, P., Smith S., Thiriez, A. Reactive Plan Revision. In *Proceedings AAAI-88*, 1988.

[Sad89] Sadeh, N. and Fox, M. S. Preference Propagation in Temporal/Capacity Constraint Graphs. Technical report, The Robotics Institute, Carnegie Mellon University, 1989.

[Sim88] Simmons, R.G. Combining Associational and Causal Reasoning to Solve Interpretation and Planning Problems. Technical report, MIT Artificial Intelligence Laboratory, 1988.

[Smi85] Smith, S. and Ow, P. The Use of Multiple Problem Decompositions in Time Constrained Planning Tasks. In *IJCAI-85 Proceedings*, 1985.

[Sus73] Sussman, G.J. *A Computational Model of Skill Acquisition* . PhD thesis, AI Laboratory, MIT, 1973.

[Wal75] Waltz, D. Understanding Line Drawings of Scenes with Shadows. In P. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, 1975.

[Zwe90] Zweben, M., Deale, M., Gargan, R. Anytime Rescheduling. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning and Scheduling*, 1990.

[Zwe91] Zweben, M.,Minton, S. Repair-Based Scheduling: Informedness versus Computational Cost. In *The First International Conference on AI Planning Systems*, volume Submitted, 1991.

[Zwe92a] Zweben, M., Davis, E., Daun, B., Drascher, E., Deale, M., Eskey, M. Learning To Improve Constraint-Based Scheduling. *Artificial Intelligence*, To Appear, 1992.

[Zwe92b] Zweben, M., Davis, E., Deale, M. Iterative Repair for Scheduling and Rescheduling. *IEEE Systems, Man, and Cybernetics*, To Appear, 1992.

Realization of High Quality Production Schedules : Structuring Quality Factors via Iteration of User Specification Processes

Takashi Hamazaki

Department of Artificial Intelligence, University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN, United Kingdom
E-mail:T.Hamazaki@edinburgh.ac.uk

Abstract

This paper describes an architecture for realizing the high quality production schedules.

Although quality is one of the most important aspects of production scheduling, it is difficult even for a user to specify precisely. However it is also true that the decision whether a schedule is good or bad can be taken only by a user.

This paper proposes;

- The quality of a schedule can be represented in the form of quality factors, i.e. constraints and objectives of the domain, and their structure;
- Quality factors and their structure can be used for decision making at local decision points during the scheduling process;
- They can be defined via iteration of user specification processes.

1 Introduction

Production scheduling is a *hard* problem in general because of the large search space, large number of factors lead to combinatorial explosion, and also its ill-structured (or ill-defined) nature. The primary concern of this paper is realizing high quality schedules, which is one of the major difficulties of production scheduling.

Since the schedule should be evaluated by several often *conflicting* aspects, it is a common approach to expect the user to specify a single evaluation function, i.e. satisfaction level of each aspect and priority among them.[6, 7, 13] However, it is difficult even for a user to define an evaluation criterion for a schedule precisely.[1] The methods that a system can use to decide an evaluation criterion, and to produce a schedule which optimizes that criterion, are important issues in this domain.

Although it is difficult for users to define an evaluation criterion, at the same time, it is also true that the decision whether a schedule is good or bad can be taken only by a user. (Please compare other aspects, e.g. the performance of a system can be evaluated by an absolute measure – i.e. time.) It follows that the schedule should be evaluated on domain specific information relating to a definition of quality given by the user. Furthermore, the quality information should be used for search guidance during scheduling, since the primary goal of search is affected by the decision of what a good/bad schedule is.

Quality is determined by the combination of the extent to which constraints are satisfied and how well objectives are achieved. Since constraints and objectives can be regarded as atomic factors of the quality of a schedule, I call them *quality factors* in this paper.

This paper describes an architecture which can acquire quality information from the user and reflect the information on the resulting schedule via iteration of user specification processes. This work is currently in progress.

2 Analysis of quality factors

2.1 Structure of quality factors

Before attempting classification, this section will concentrate on the relationships among quality factors.

Since quality factors are defined by a user, they are often interrelated of each other. Some *include* other quality factors, and some *cause* another factor. For instance, a prohibition against changeover at the same time (due to a limitation on operators) can be divided into two levels below.

1. changeover itself – namely, a condition of changeover (defines this quality factor as QF1)

2. simultaneous occurrence of QF1 (QF2)

QF2 can be thought of as a meta-level quality factor.

This information relating to the relationships among quality factors is quite useful for scheduling, and they are defined as a *structure* of quality factors in this paper.

2.2 Classification of quality factors

Quality factors can be classified from several points of views; for examples the function in a real plant[12, 14], the influence on scheduling.[3] In this section I attempt to classify quality factors based on the relationship with the scheduling algorithm. This classification is more detailed than others in order to use it for acquiring additional information about quality factors from the user.

hard – soft The first dimension of classification is based on the strictness of satisfaction/violation ;

- *hard factor* : one which must be satisfied, i.e. cannot be relaxed any more.
- *soft factor* : one which is preferable to satisfy. As all quality factors are preferable to satisfy, *soft factor* can be defined as the complement of *hard factor* more strictly.

Job and Resource The next dimension of classification is based on parameters which a factor contains. The parameters of a factor are either/both of *Job* and *Resource*.

They can be broken down further according to the necessity for the reference to other objects during an evaluation of the quality factor as follows;

Job –

inter-lot need to refer to operations in other lots,

intra-lot need to refer to other operations in the same lot and

no-interaction(no-int) no need to refer to other operations, and

Resource –

inter-machine need to refer to other machine,

intra-machine no need to refer to other machines.

For instance, the parameters of a changeover(QF3) are Job and Resource and detailed class of Job part is *inter-lot* and that of Resource is *intra-machine*. Therefore, this quality factor, "a changeover"(QF3), can be classified as (*inter-lot Job* , *intra-machine Resource*) type.

Global and local The last dimension is based on the applicability at a local decision point. Intuitively, *global* factors are those which can be used to evaluate a full schedule, while *local* factors are those which can be used to evaluate a partial schedule. However, many quality factors can be applied to the evaluation of candidates at a local decision point (even if it looks like *global* one) by using an estimation of resulting value. For instance, although QF2 in previous examples cannot necessarily always be applied at local decision points as it is, it might be possible if the probability of changeover for each product type could be estimated.

It follows that the revised version of the definition is

A global factor is one for which a user cannot define an estimation function at all.

A local factor is the complement of global factor, i.e. those which a user can define so that they can be applied at local decision points.

3 Scheduling via quality factors

3.1 Iterative user specification processes

In the previous section, the concept of quality factors was introduced. This concept makes it possible to characterize the user's evaluation of a schedule, mentioned earlier , as follows.

Suppose as an example two schedules are compared.

1. apply hard quality factors to every item – presumably operations – of each schedule.

   IF violation has occurred in either of two schedules ⟶ unacceptable schedule

   ELSE ⟶ next step

2. apply most important soft quality factors to
   – every item of the schedule – if the quality factor is local
   – the whole schedule – if the quality factor is global

IF a sufficient difference between them is identified in any of the quality factors ⟶ decide

ELSE ⟶ next step

3. apply next level of soft-quality factors
   ... same as above.

If the importance of every QF could be categorized and exact values for a *sufficient* difference could be defined beforehand, this process could be done automatically and it might be possible (apart from realistic processing speed) to optimize a schedule. However, specification – especially the criterion for sufficiency – is difficult (or close to impossible) to define precisely in advance. Consequently, it will be indispensable to adopt some sort of trial and error process for deciding a good schedule. From this view, the following procedure should be an acceptable method for acquiring the information from a user.

1. user specifies each quality factor
   user specifies priority among quality factors in as much detail as possible.

2. system produces a schedule based on quality information acquired so far.

3. user analyzes the resulting schedule produced in the previous step.
   user judges whether the schedule is satisfactory or not.

   IF satisfactory ⟶ end.

   ELSE ⟶ specify which QF should be improved.

4. system re-structures quality information
   goto 2.

3.2 Search guidance by quality factors

Schedule production,i.e. step 2 in the procedure defined in the previous section, is accomplished by a repetition of target selection , i.e. operation, and a reservation for it, i.e. resource and start time. In the each repetition cycle, it is desired to rate candidates appropriately which results in a *good* overall schedule. The next section focuses on this rating step.

3.2.1 Rating by quality factors

When a schedule can be evaluated by a function – defines it as E – two dimensions can be viewed as rating methods.

local optimization(LO) how good will the quality of a partial schedule be
— measures candidates by $E(Pn)$: where Pn is a partial schedule after adopting candidate-N

predicted global optimization(PGO) how good is the quality of the final schedule likely to be, in other words from the opposite perspective, how difficult will expected problems be.[1]
— measures candidates by $E'(Pn)$: where E' is a probabilistic function which expresses the value likely to be achieved.

As described in section 3.1 , E is realized by a series of applications of quality factors and filtering out in each quality factor application. E' is similar in general, since a *predicted* final schedule should be evaluated in the same manner. However a probability among quality factors should be also taken into account as well as the priority among them. For instance, if it is is known that QF1 frequently identifies a bad value, it might be better to apply QF1 prior to other QFs, even though the priority of QF1 is not highest.

The application of quality factors is stopped when a sufficient *difference* among candidates is identified. This *difference* is described as a *threshold* in this paper.

3.3 Feedback from the result

In this section, we describe how the system re-structures quality factors in reaction to the schedule produced,i.e. step 4 in the procedure defined in Section 3.1. This process can be accomplished both automatically and manually.

3.3.1 Manual feedback

As described earlier, the user should judge whether the resulting schedule is satisfactory or not. Generally speaking, a user can communicate with the system via quality factors and structures among them. That is, if the schedule is not satisfactory for a user, the reason why its schedule is not satisfactory is expressed by indicating which quality factors' values should be improved. This feedback from the user will influence structures of quality factors.

In order to support a user in detecting problems, the system provides information, as follows:

verification of assumption It is unrealistic to expect that a user can specify the structure of

[1] There are two heuristics for realizing this method; i.e. variable ordering and value ordering.[5]

quality factors precisely from the early stages of scheduling generation. Consequently, a system should assume some information about structure. The information assumed by a system should be verified at the end of scheduling generation process by the user. The user is informed of assumed structures at the feedback stage.

evaluation by global factors Resulting schedules can often be evaluated at a gross level by global factors, like overall utilization, although all quality factors should be involved for a precise evaluation. Furthermore, since global factors are considered only via causal factors during the scheduling generation process, verification is indispensable. Statistical information based on global factors is provided by the system automatically.

evaluation by specific factors It is quite usual that a user knows which quality factor is critical in the specific application/domain. Statistical information is also provided in response to the user.

3.3.2 Automatic feedback

When scheduling has not been completed, i.e. there remain unassigned operations, the system analyzes its reasons and restructures quality information based on some heuristics, which include

- If there are quality factors in which unassigned operation got the best value
  ⟶ decrease threshold of those quality factors
- If there are quality factors in which unassigned operation was the next candidate
  ⟶ increase threshold of those quality factors

4 System structure

The system consists of mainly four parts, namely; Scheduler, Generator, Analyzer and Data-Base manager and six system files, namely; Quality Data Base(QDB), Evaluation Procedures File(EPF), Scheduling results File(SF), Decision history file(DHF), Order file(OF) and Knowledge-Base(KB).

The general flow of this system is as follows (this can be thought of as a detailed version of the iterative procedure described in Section 3.1.);

1. user specifies initial information
   - order data (presumably from other system) ⟶ OF
   - domain information, e.g. factory, machine ⟶ KB
   - quality factor[2] ⟶ KB
   - attribute of quality factors, e.g. global/local , hard/soft ⟶ QDB
   - structure of quality factors (in as much detail as possible) ⟶ QDB
2. Generator generates evaluation procedures, which can be used in the rating of candidates during scheduling process, based on QDB information and output ⟶ EPF
3. Scheduler generates a schedule based on OF, KB and EPF and output
   - resulting schedule(including unassigned operations) ⟶ SF
   - history of rating by quality factors at every local decision points ⟶ DHF
4. Analyzer analyzes SF and DHF and queries the user if necessary and restructures QDB.
   goto 2 if not satisfactory

5 Future work

This system uses a traditional algorithm as its scheduling mechanism, since the scheduling algorithm itself is not the major concern. However, it is obvious from the analysis in Section 3.2 that quality information which is acquired from a user and scheduling algorithm have tight connection. It follows that the ideas proposed here are restricted by this algorithm. It is required to analyze validity on other algorithms, e.g. distributed scheduling system[4] , as well and extend these ideas.

This system is now being implemented and will be evaluated using real problems, although it is based on my experiences in developing practical production scheduling systems.[9, 8]

6 Conclusion

Although quality is one of the most important parts of production scheduling, it is difficult even for users to define precisely. The first step in realizing a high quality production schedule is to clarify what "high quality" means.

[2] The system requires a user to specify function which represents the goodness of the selected candidate for every quality factor. At the same time, the current system also requires a probabilistic function for each quality factor, although this should be eventually supported by the system.

This paper proposed;

- The quality of production schedules can ultimately be evaluated/measured only by a user, and his intention can be represented in the form of quality factors and their structures defined by him/her. (global evaluation)
- Quality factors and their structures can be used for decision making at local decision points during the scheduling process. (local evaluation)
- They can be refined via iteration of the user specification process. (iterative process)

7 Acknowledgement

This research has been supported by Hitachi Limited(Japan). The author of this paper wishes to acknowledge the support of the following people: Tim Duncan and Howard Beck of AIAI and Peter Ross of DAI.

References

[1] Mostafa El Agizy. Design of systems for replenishing stocks of materials. In Haluk Bekiroglu, editor, *Computer Models for Production and Inventory Control*, pages 61–. The Society for Computer Simulation, California, 1988.

[2] H. Atabakhsh. A survey of constraint based scheduling systems using an artificial intelligence approach. *Artificial Intelligence in Engineering*, 6(2):58–73, April 1991.

[3] Pauline M. Berry. *A Predictive Model for Satisfying Conflicting Objectives in Scheduling Problems.* PhD thesis, University of Strathclyde, 1991.

[4] Iain Buchanan, Peter Burke, John Costello, and Patrick Prosser. A Distributed Asysnchronus Hierarchical Problem-Solving Architecture applied to Plant Scheduling. In G. Rzevski, editor, *Proceedings of the forth International Conference on the Applications of Artificial Intelligence in Engineering*, pages 107–114, Cambridge, UK, July 1989.

[5] Rina Dechter and Judea Pearl. Network-Based Heuristic for Constraint Satisfaction Problems. *Artificial Intelligence*, 34(1):1–30, 1988.

[6] Mark S. Fox and Katia P. Sycara. The CORTES Project:A Unified Framework for Planning Scheduling and Control. In *Proceedings of the 1990 DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 412–421, 1990.

[7] M.S. Fox and S.F. Smith. ISIS:A Knowledge-Based System for Factory Scheduling. *Expert Systems*, 1(1):25–49, July 1984.

[8] T. Hamazaki, T. Kameda, S. Okuide, M. Koroku, and K Kozuka. Approach to Planning and Scheduling Expert Systems. *The Hitachi Hyouron*, 72(11):41–46, November 1990. Japanese.

[9] N. Irisawa, T. Hamazaki, and T. Yamanaka. Production Scheduling by using Expert Systems. *Communication of the operations research society of Japan*, 36(3):141–145, 1991. Japanese.

[10] Richard E. Korf. Search:A Survey of Recent Result. In Howard E. Shrobe, editor, *Exploring Artificial Intelligence*, pages 197–237. Morgan Kaufman, California, 1988.

[11] P. A. Newman. Scheduling in CIM systems. In Andrew Kusiak, editor, *Artificial Intelligence:Implication for CIM*. Bedford IFS, 1988.

[12] BC Niew, BS Lim, and NC Ho. Knowledge Based Master Production Scheduler. In *First International Conference on Expert Planning Systems*, pages 88–93, London, 1990. IEE.

[13] F. Stephen Smith, Peng Si Ow, Nicola Muscettola, Jean-Yves Potvin, and Dirk C. Matthys. OPIS:An Integrated Framework for Generating and Revising Factory Schedules. In *Proceedings of the 1990 DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 497–507, 1990.

[14] S. Smith, M. Fox, and P.S. Ow. Constructing and Maintaining detailed Production Plans: Investigations into the Development of Knowledge-based Factory Scheduling Systems. *AI Magazine*, 7(4), 1986.

[15] S. F. Smith and P. S. Pw. The use of multiple problem decomposition in time constrained planning tasks. In *Proceedings of IJCAI 85*, pages 1013–1015, 1985.

Scheduling Revisited Workstations in Integrated-Circuit Fabrication

Paul J. Kline
Semiconductor Process and Design Center
Texas Instruments Incorporated
P.O. Box 655012, M/S 3635
Dallas, TX 75265
kline@csc.ti.com

Abstract

The cost of building new semiconductor wafer fabrication factories has grown rapidly, and a state-of-the-art fab may cost $250 million or more. Obtaining an acceptable return on this investment requires high productivity from the fabrication facilities.

This paper describes the Photo Dispatcher system which has been developed to make machine-loading recommendations at a set of key fab machines. Dispatching policies that generally perform well in job shops (e.g., Shortest Remaining Processing Time) perform poorly for workstations such as photolithography which are visited multiple times by the same lot of silicon wafers.

The Photo Dispatcher evaluates the history of workloads throughout the fab and identifies bottleneck areas. The scheduler then assigns priorities to lots depending on where they are headed after photolithography. These priorities are designed to avoid starving bottleneck workstations and to give preference to lots that are headed to areas where they can be processed with minimal waiting. Other factors considered by the scheduler to establish priorities are the nearness of a lot to the end of its process flow and the time that the lot has already been waiting in queue.

Simulations that model the equipment and products in one of Texas Instruments's wafer fabs show the Photo Dispatcher can produce a 10% improvement in the time required to fabricate integrated circuits.

Introduction

Texas Instruments has a number of integrated-circuit (IC) wafer fabs which produce many different chip types. Depending on the type of chip on a wafer, fabricating that wafer will place very different demands on the processing equipment. Planning systems are used to produce weekly wafer start-plans that are within the capacity of the fab equipment. These planning systems do not develop a schedule for when each wafer will visit each machine group; instead they try to ensure that no more than a week's worth of work is started for all machine groups.

While good start plans have helped avoid some of the problems of machine overloading and late orders, these problems have not totally disappeared in the wafer fabs. Machine breakdowns, rework, etc., make it inevitable that production rarely proceeds as smoothly as desired. The manufacturing staff reacts to these disruptions by reprioritizing lots of wafers to expedite lots that are behind schedule or to cure workload imbalances on equipment. We developed the Photo Dispatcher to investigate the possibility of automating the scheduling of a set of key machines in the photolithography area.

The photolithography area was chosen because lots continually revisit this area during their processing, so improved scheduling in this area should have wide-reaching impact on the wafer fab. Figure 1 shows a typical process flow for producing a bipolar device with seven pattern steps. As shown in the figure, all of the pattern steps are performed on the same set of projection printers, Printers Grp. A scheduler for the Printers Grp would impact this device seven times as opposed to a scheduler for Depos Grp 6 which would would impact this device only once.

Previous Research

One approach to scheduling the Printers Grp would be to generate a Gantt chart each shift that shows which lots should be processed on which projection printers at which times. We decided not to take this approach because we felt that these schedules would quickly become obsolete because of projection printer breakdowns, unpredictable lot arrivals, and the unpredictable need to rework lots whose first patterning was unsatisfactory. This approach might also be difficult to scale up to schedule all the machine groups in the wafer fab because of the large number of machines (400+) and lots in process (400+).

An alternative is to wait until it is time to load a free machine and then decide which lot to load based on what is in queue and current fab conditions. Dispatch policies, which have been studied extensively in Operations Research (e.g., Panwalker & Iskander, 1977), are one way to make this decision. First-in-First-Out

Process Step	Step Type	Equipment
First Oxidation	Layer	Furnace Grp 10
DUF Pattern	Pattern	Printers Grp
DUF Diffusion	Dope	Furnace Grp 20
Epitaxial Deposition	Layer	Epi Reactors
Second Oxidation	Layer	Furnace Grp 10
Isolation Pattern	Pattern	Printers Grp
Isolation Diffusion	Dope	Furnace Grp 32
Base Pattern	Pattern	Printers Grp
Base Diffusion	Dope	Furnace Grp 53
Emitter Pattern	Pattern	Printers Grp
Emitter Diffusion	Dope	Furnace Grp 60
Contact Pattern	Pattern	Printers Grp
Aluminum Evaporation	Layer	Alum Evap Grp
Metal Pattern	Pattern	Printers Grp
Deposit Overcoat	Layer	Depos Grp 6
Bonding Pad Pattern	Pattern	Printers Grp
Electrical Test	Test	Tester Grp

Figure 1: All of the pattern steps are performed in the photolithography area of the fab on the same equipment, Printers Grp. Wafers fabricated using the process flow illustrated make seven visits to Printers Grp.

(FIFO) is an example of a simple dispatch policy.

However, for the current application, dispatch policies have the following drawbacks:

1. Many dispatch policies do not work well on revisited machine groups like Printers Grp.
2. The typical dispatch policy is myopic in the sense that it considers only the local situation and does not consider the needs of downstream machine groups.

A dispatch policy such as Shortest Remaining Process Time (SRPT) will lead to problems when applied to a revisited workstation like the Printers Grp. If the queue for Printers Grp is made up of a number of lots with the process flow shown in Figure 1, then SRPT will prefer lots that are at their last pattern step, Bonding Pad Pattern. It will only select lots at the first pattern step, DUF Pattern, if there are no other lots in the queue. This leads to long waiting times at DUF Pattern and alternating starve/glut feeding patterns for DUF Diffusion.

We have investigated other dispatch policies such as Shortest Processing Time and Slack to Due-Date, but our experience has been that they also share the defect of SRPT of being biased in favor of one or another of the pattern steps and neglecting others. The problem seems to be that these policies are "winner take all" policies as opposed to "winner take a bigger share" policies. There is nothing wrong with adding priority to lots near the end of their flows or lots in trouble with their due dates. What causes problems is that "winner take all" schemes based on these factors run the risk that low priority lots may wait forever if higher priority lots arrive fast enough. To get around this problem, the Photo Dispatcher uses a variation of a round-robin scheme which provides a "winner take a bigger share" selection.

The FIFO dispatch policy is not biased in favor of particular pattern steps, but it is myopic in the sense that it will select lots that will just have to sit at their next process step because a key machine is down. Alternatively, it may pass over lots that would help keep a downstream bottleneck machine group from starving. Goldratt's OPT system (1984, 1988) emphasised the importance of bottleneck resources in scheduling. AI scheduling systems that emphasize the importance of bottleneck resources include Smith, Fox, & Ow (1986) and Eskey & Zweben (1990). The Photo Dispatcher avoids myopic decision making by monitoring current and historical workloads throughout the wafer fab and reacting to these workloads to avoid starving bottleneck workstations and avoid sending lots to workstations where they will just sit in queue.

Approach

The Photo Dispatcher makes recommendations about which lot of wafers in the queue should be processed next by Printers Grp. The Photo Dispatcher develops these recommendations in three stages:

1. Establish priorities for processing lots at the different pattern steps.
2. Use the priorities to choose a pattern step to work on next. That is, decide whether to work on a lot that is waiting for DUF Pattern, or Isolation Pattern, etc.
3. Choose a specific lot waiting for that pattern step. There are typically several lots waiting for DUF Pattern and this third stage determines which of these lots should be recommended. While a number of different criteria have been investigated for making this lot selection, none of them have outperformed FIFO, so currently this selection is just based on time of arrival of the lots waiting for DUF Pattern.

Pattern-Step Priorities

Figure 2 gives an example of the calculation of priorities for four pattern steps. Three numbers are summed to determine priorities. The percentage of lots waiting for a particular pattern step (e.g., 20 percent for DUF Pattern) is added to a number that is based on the nearness of that pattern step to the end of the processing flow (e.g., 01, the first two digits of the step id number). Finally, a positive number (e.g., 30) is added if more work is needed at downstream work areas or a negative number is added if there is too much work. The higher the priority number for a pattern step the more lots at this step will be recommended for pro-

```
 0100 DUF PATTERN
% Lots Queued    20
Flow Position    01
Feedback         30   (Send More;
                 ----  Short Wait at
Priority         51    Furnace Grp 20)

 0600 ISOLATION PATTERN
% Lots Queued    04
Flow Position    06
Feedback        -30   (Send Less;
                 ----  Long Wait at
Priority        -20    Furnace Grp 32)

 2300 BASE PATTERN
% Lots Queued    12
Flow Position    23
Feedback         00   (No Adjustment;
                 ----  Average Wait at
Priority         35    Furnace Grp 53)


 6000 CONTACT PATTERN

% Lots Queued    05
Flow Position    60
Feedback         60   (Send Much More;
                 ----  Starving Bottleneck
Priority        125    Alum Evap Grp)
```

Figure 2: Three factors determine the priority of a particular pattern step: the fraction of lots waiting for this pattern step, the nearness of this pattern step to the end of the process flow, and feedback from downstream work areas.

cessing. The rationale for each of the three factors illustrated in Figure 2 will be discussed in turn.

% Lots Queued If the process flow in Figure 1 was used by all devices, there would be roughly equal numbers of lots waiting for the individual pattern steps. However, since there are roughly 300 different process flows, the pattern steps do not occur with equal frequency. Including a factor for the percentage of lots waiting for a particular pattern step ensures that the round-robin scheme does not penalise lots that are waiting for frequently used pattern steps.

Flow Position The second factor, nearness of a pattern step to the end of the process flow, has the effect of reducing work-in-process (WIP). Lou and Kager (1989) recommend that when scheduling revisited workstations in IC fabrication, higher priority should be given to process steps that are later in the process flow. Our experiments confirm the benefits of this practice.

Feedback From Downstream Workstations The feedback to Contact Pattern in Figure 2 shows a bottleneck, Alum Evap Grp, requesting additional work because it is starving. Figure 3 illustrates the computations performed to determine this machine group is a starving bottleneck. A software object called a *Work Monitor* is associated with machine groups in the fab. WORK-FOR-ALUM-EVAP knows how to use the current position of a lot provided by the WIP tracking system to determine whether that lot is at a process step that uses the Aluminum Evaporators. In the case illustrated in Figure 3, there is one lot consisting of 48 wafers arrived at a processing step using the Aluminum Evaporators. WORK-FOR-ALUM-EVAP also can tell if a lot has left photolithography and is on the way to a process step using the Aluminum Evaporators but has not arrived yet. Identifying lots that have been sent to the Alum Evap Grp provides an estimate of upcoming workloads.

WORK-FOR-ALUM-EVAP knows how to translate the number of wafers arrived into the time it should take the Alum Evap Grp to complete processing those wafers. In Figure 3 the 48 wafers arrived are estimated to keep the Aluminum Evaporators busy for the next .58 hrs. The following factors are considered to determine how many hours it will take to complete processing a particular set of wafers:

1. number of machines in Alum Evap Grp and their capacities
2. process time for each wafer; different devices may have different processing times
3. setup times

Work Monitors categorise workloads along four different dimensions (TYPICAL-LOAD, COMPARE-NOW-TO-HISTORY, etc.) and these dimensions are referenced by rules that determine the appropriate

```
WORK-FOR-ALUM-EVAP
Lots sent to Alum Evap Grp: 0 lots, 0 wafers

Lots arrived at Alum Evap Grp:
1 lot, 48 wafers
Est work in queue for Alum Evap Grp:
14 wafers, 0.26 hrs
Est work in process for Alum Evap Grp:
34 wafers, 0.32 hrs
Estimated work available = .58 hrs

The avg amount of work, 10.43 hrs, that
 arrives in this area is HIGH relative to the
 long-run avg (4.44 hrs, sigma=3.19)
 for all areas.
Work currently arrived in this area, .58 hrs,
 is LOW relative to the long-run avg
 (10.43 hrs, sigma=6.28) for this area.
Work currently arrived in this area, .58 hrs,
 is LOW relative to the avg currently arrived
 (3.51 hrs, sigma=3.66) for all areas.
Work currently sent to this area, 0.00 hrs,
 is LOW relative to the long-run avg
 (2.70 hrs, sigma=0.64) for this area.

The classification is:
TYPICAL-LOAD              HIGH
COMPARE-NOW-TO-HISTORY    LOW
COMPARE-TO-OTHER-MACHS    LOW
COMPARE-SENT-TO-HISTORY   LOW,
                so MUCH MORE work is needed.
```

Figure 3: An example of a starving bottleneck. The Aluminum Evaporators typically have 10.43 hrs of work; however, at the time this workload evaluation was performed there was only .58 hrs of work and none on the way. A request for much more work is fed back to Contact Pattern.

feedback (e.g., send MUCH MORE). By changing the feedback rules, a wide variety of workload regulation schemes can be implemented. Simulation experiments were run to evaluate three different workload regulation schemes: Bottleneck Starvation Avoidance, Smallest Next Queue, and Workload Smoothing. Each of these three workload regulation schemes performed well at some WIP levels, but none of the three was superior at all WIP levels. A hybrid of these approaches was devised which performed well at all WIP levels investigated.

Some approaches to scheduling using bottleneck starvation avoidance require that the there is only one bottleneck and its identity is known in advance and provided as input to the scheduler (e.g., Glassey & Resende 1988). This assumption makes it difficult to handle shifting bottlenecks which can arise in wafer fabs because of a change in product mix or the breakdown of key equipment. The work-monitoring mechanism of the Photo Dispatcher determines which workstations are bottlenecks by gathering workload histories. As the bottlenecks change over time the shift in workload histories is tracked by the Work Monitors and the priority of lots is changed accordingly.

Using Priorities to Select a Pattern Step

Priorities are recomputed periodically based on the current workloads at downstream machine groups and the number of lots queued for each pattern step. The priorities influence subsequent selection of pattern steps by controlling a round-robin scheme.

Each time a machine in Printers Grp is ready to be loaded, the round-robin advances to the next pattern step in the cycle. The priority assigned to this pattern step determines whether a lot is chosen from this pattern step or whether the round-robin immediately advances to the next pattern step. Priority numbers are rescaled to range from 5 to 100 and pattern steps with priority 5 have a lot selected one out of every 20 times the round-robin stops there (5%), pattern steps with priority 50 have a lot selected every other time the round-robin stops there, etc.

When the priorities are as shown in Figure 2, a sequence of selections produced by this round-robin might be Contact Pattern, DUF Pattern, Base Pattern, Contact Pattern, Contact Pattern, DUF Pattern, Contact Pattern, ... By interleaving pattern steps in this fashion there is limit on how many times a low priority pattern step can be passed over before it is selected.

Results

The Photo Dispatcher was tested using a simulation of one of Texas Instruments' wafer fabs. The fab in question manufactures a wide variety of Bipolar and BiMOS devices. The simulation modeled all of the 103 machine groups (410 machines) in the fab and all of the process steps needed to fabricate any of the roughly

INITIAL WIP	FIFO		Photo Dispatcher	
	CT	Output	CT	Output
290	339	140	328	140
400	482	132	427	136
675	680	138	604	141
1000	1068	132	975	136

Table 1: In simulation experiments, the Photo Dispatcher produces an improvement over a FIFO policy in the average number of hours needed to complete processing a lot of wafers (i.e., cycle time or CT). The same lot starts were used in all simulations, so WIP levels were manipulated by varying the number of initial WIP lots in the fab at the start of simulation.

300 different ICs produced. There are roughly 50 process steps in the recipe for a typical IC in this fab and these process steps are broken down in the simulation to 165 separate operations each of which requires the use of another machine. The simulation included random machine breakdowns, but this was the only random component as processing times were assumed deterministic, lot transport time was not modelled, and there were no operator limitations.

Table 1 shows the Photo Dispatcher has better average cycle-time performance in simulations than a FIFO policy. FIFO does not separate the decision of which lot to process next on Printers Grp into a pattern step selection followed by a lot selection. Instead it merely finds the lot that has been waiting longest for the printers and starts that lot. This is the default behavior of the simulator and it is also used when exercising the Photo Dispatcher for machine groups other than the Printers Grp.

The size of the cycle-time improvement depends on the amount of WIP in the simulated fab. At WIP levels above 400 lots, the Photo Dispatcher reduced cycle-times by roughly 10% with greater output in terms of finished lots per week. Since the fab being simulated currently operates at WIP levels in excess of 400 lots, a 10% cycle-time improvement was projected from the use of this scheduler. Results to date suggest that flow position is the most important factor in producing cycle-time improvements of the three factors combined in Figure 2.

Summary

The Photo Dispatcher provides an effective scheduling approach for revisited workstations such as photolithography in IC fabrication. It takes advantage of the unique opportunity that these revisited workstations provide to shift workloads from one downstream area to another and to reduce WIP by speeding up processing on lots near the end of their process flows.

While the Photo Disptacher was developed with the intention of installing it in Texas Instruments' wafer fabs, to date it has not been used for real-time scheduling of fab operations. However, its Work Monitor capability has been used to a limited degree to analyse production problems in fabs.

References

Eskey, M. and Zweben, M. Learning search control for constraint-based scheduling. *Proceedings Eighth National Conference on Artificial Intelligence (AAAI-90)*, 1990, pp. 908–915.

Glassey, C.R. and Resende, M.G. Closed-loop job release control for VLSI circuit manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, Vol. 1, 1988,

Goldratt, E. and Cox, J. *The Goal.* North River Press, 1984.

Goldratt, E.M. Computerised shop floor scheduling *International Journal of Production Research.* Vol. 26, 1988, pp. 443–455.

Lou, S.X.C. and Kager, P.W. A robust production control policy for VLSI wafer fabrication. *IEEE Transactions on Semiconductor Manufacturing*, Vol. 2, No. 4, 1989.

Panwalker, S.S. and Iskander, W. A survey of scheduling rules. *Operations Research*, Vol. 25, 1977, pp. 45–61.

Smith, S.F., Fox, M.S., and Ow, P.S. Constructing and maintaining detailed production plans: Investigations into the development of knowledge-based factory scheduling systems. *AI Magazine*, Fall 1986, pp. 45–61.

Multi-Agent Planning and Scheduling, Execution Monitoring and Incremental Rescheduling: Application to Motorway Traffic

Pascal Mourou and Bernard Fade
IRIT, Institut de Recherche en Informatique de Toulouse
Université Paul Sabatier
118 route de Narbonne, 31062 Toulouse Cedex, France
Tel: +33 61 55 66 11 extension 72 63 or 63 30
Fax: +33 61 55 62 39
E-mail: mourou@irit.fr or fade@irit.fr

Abstract

This article describes a planning method applicable to agents with great perception and decision-making capabilities and the ability to communicate with other agents. Each agent has a task to fulfil allowing for the actions of other agents in its vicinity. Certain simultaneous actions may cause conflicts because they require the same resource. The agent plans each of its actions and simultaneously transmits these to its neighbours. In a similar way, it receives plans from the other agents and must take account of these plans. The planning method allows us to build a distributed scheduling system.

Here, these agents are robot vehicles on a highway communicating by radio. In this environment, conflicts between agents concern the allocation of space in time and are connected with the inertia of the vehicles. Each vehicle make a temporal, spatial and situated reasoning in order to drive without collision.

The flexibility and reactivity of the method presented here allows the agent to generate its plan based on assumptions concerning the other agents and then check these assumption progressively as plans are received from the other agents. A Multi-agent execution monitoring of these plans can be done, using data generated during planning and the multi-agent decision-making algorithm described here. A selective backtrack allows us to perform incremental rescheduling.

Keywords

Anytime Planning and Scheduling Algorithms, Execution Monitoring and Incremental Rescheduling, Managing limited computation time, Dependency Analysis and Plan Reuse, Autonomous Agents.

1 Multi-agent worlds

Monitoring a little structured multi-agent environment, such as a highway traffic, is an extension to the problem of monitoring robots in a factory. The agents are assumed to be "high-level" since they must have a great ability to perception and they must communicate with each other to cooperate, coordinate their actions and resolve any conflicts. The resolution of conflicts is the main point of interest. Logic schemata, attempting to model human thinking, have been developed to represent the wishes and beliefs [Bessiere, 84][Wilks and Ballim, 87] which are the mutual basic knowledge needed to resolve conflicts. Persuasion [Rosenschein, 82][Sycara, 89] is the aim of exchanging arguments. Most studies are simplified by assuming that agents cooperate (see [Cammarata et al., 83]). Rosenschein and Genesereth [Rosenschein and Genesereth, 85], on the contrary, attempt to allow for agents which are not necessary "benevolent".

2 The motorway

Unlike Wood [Wood, 83], we do not generate routes but consider the driving of the vehicle (acceleration, lane changes, etc.). We shall use a different approach to Fraichard and Demazeau [Fraichard and Demazeau, 89], who describe a centralized system to generate vehicle trajectories at cross-roads. We use a distributed system in which the number of central units increases as the number of agents increases. The multi-agent world was modelled on this basis (see [Mourou and Fade, 91a] and [Mourou, 90]).

Each vehicle has a co-pilot computer which may either be in an automatic mode, driving the vehicle, or in a supervision mode when it warns the driver or, if necessary, takes over control when an accident is imminent.

When all vehicles are in the "automatic driving" mode, it is simple: the vehicles are considered as autonomous robots which communicate with each other. The supervision mode requires a veritable "execution monitoring" which must be highly flexible and supervise drivers' acts by comparing them with the "ideal" plan generated in the automatic mode.

Co-pilots exchange data via a short-range communication network. The agents must cooperate to guarantee "efficient and safe traffic movement" and must respect the highway code, used as veritable "cooperative strategy"

[Cammarata et al., 83]. A number of objectives are also fixed for each agent, such as "to travel at the mean speed required by the driver". Unlike certain systems analyzed by Davis and Smith [Davis and Smith, 83], no tasks need be shared in the procedure since each agent knows what he must do. The negotiation therefore covers solely how its tasks can be accomplished.

The co-pilot in each vehicle is concerned solely by the N relations which affect the vehicle. The task of the co-pilot will therefore involve selecting the behaviour, which is satisfactory to the N influences to which it is exposed at each time. In considering highway traffic, the "common resource" is the space available on the road. The main task of each agent is to check that the space it needs will be free and, if not, to take appropriate action to reach a free space (acceleration, lane changes, etc.). Conventional problem resolution techniques are not capable of simultaneously managing the N conflicts possible at each instant in the future. Moreover, a "distributed scheduling" technique will be unsuitable since, although automatic control can be considered as a resource allocation problem, the inertia of the various vehicles will make it extremely difficult to break the road down into a series of "areas", each considered as a resource.

The method we describe is more "expert"-oriented, allowing the "rules" in the highway code to be expressed and used as they exist and high-level data exchanges to be used. For example "I'm going to move out and accelerate up to 110 km/h" is a kind of action generated by the planner and broadcasted through the network.

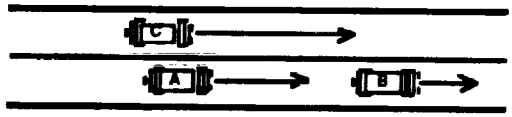
3 Time, influence of other plans and delay

The behaviour of each agent is represented by a linear, non-hierarchical plan. We make the assumption that the agents are synchronised by a common clock broadcast by radio for example. B's "time influence" on A covers all the B's actions and situations around T_i used to plan A's action at T_i. When some of them are missing, A must make assumptions on the actions planned by B and consequently progressively check these assumptions as the actual actions are received. If A's assumption is found to be correct, we shall have saved time. Otherwise, A must replan this action after B has transmitted its decision and no time will have been lost.

4 The "Is there an agent... ?" method

Knowing, or assuming, the actions of other agents, agent A must generate an action (the behaviour for a given step). It can repeatedly pose this kind of question : "Is there an agent preventing me doing this ?". Each question determine whether there is a conflict which prevents one action (method M1).

An example of situation (Example 1) :



can be given by the possible conflicts A will detect :

(C_b) : B is in front of A (Us), which is travelling faster and wants to accelerate even further.

(C_c) : C is on the left of A and prevents A overtaking.

Questions which A could ask before deciding to "slow down" are:

- Can I accelerate ? (agent B imposes the reply "no")
- Can I move out to overtake B ? (agent C imposes the reply "no").

At a first view, it could be difficult to write directly an algorithm capable of taking a decision adapted to A's wishes when exposed to complex influences (see Figure 1).

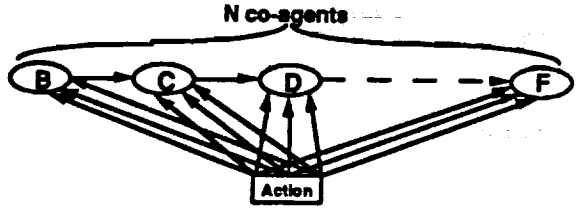


Figure 1. Method M1

5 The dual method: "Is there an action ... ?"

5.1 Definition

We could use tests of the type "Is there an action prevented by this agent ?". The agents would then be reviewed, one after the other, to collect all conflicts to which A is exposed into a "Results" structure (see Figure 2).

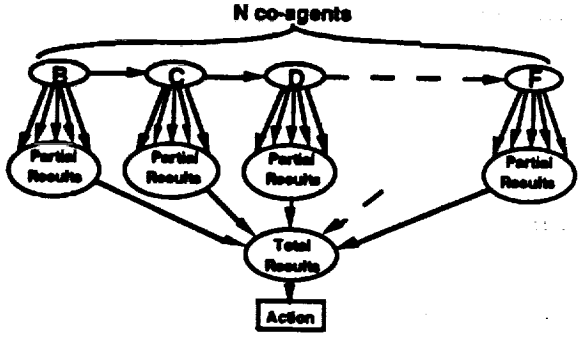


Figure 2. Method M2

- "Is there an action prevented by B ?" : "B prevents me not(slowing down or moving out)" = C_b
- "Is there an action prevented by C ?" : "C prevents me moving out" = C_c

A second phase allows the action to be determined :

- C_b and C_c --> "prevented from not slowing down" = "decelerate"

This second phase, used to find the best possible response in view of all the behaviours that are prevented and the requirements of A, occurs after determining all behaviours that are not possible (method M2). It allows K conflicts to be grouped and assessed simultaneously (K < P: maximum number of conflicts). It could be considered as simplified multi-agent planning which chooses an action in function of the prevented ones. The knowledge required for this reasoning is referred to as "N-agent knowledge".

On the other hand, each question allows assessment of a relationship between two agents. The term "bi-agent" refers to the process and knowledge used for each comparison. The result of a two-agent comparison is known as a "Partial Result".

A "mono-agent" phase may influence the Total Results in function of A's wishes before the series of bi-agent comparison.

5.2 Application to motorway traffic

The Total Results for the Example 1 would be:

Prevent-moving-out	t		
Move-out	t	Move-out-list	(B)
Slow-down	()	Slow-down-list	()

The decision-making rules for the bi-agent and then N-agent phases would be, for example:

- if A is in the right-hand lane and X is in front of A in the right-hand lane and at lower speed and if safety distance has been reached
  then Move-out(X) := t
  Move-out-list(X) := (B)

- if Move-out and Prevent-moving-out then decelerate, choosing vehicles in Move-out-list
- if Move-out then move out
- if Prevent-moving-out then do nothing
- if true then accelerate

5.3 Selective backtrack

The use of Results and the separation of conflict recognition from their overall processing makes a selective backtrack possible. Consequently, new information from agent B concerning an instant T_j, already planned, can be allowed for solely by comparison with B (see Figure 3).

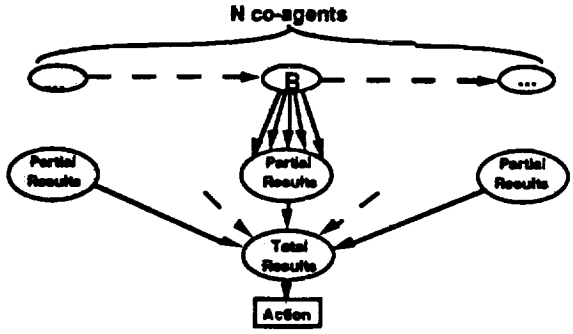


Figure 3. Selective backtrack for agent B

If the new Partial Results for P at instant T_j, designated "new-Partial-B-T_j" equal Partial-B-T_j (i.e. the response to the new influence is the same as that to the previous influence - see Example 2.1) or if "new-Partial-B-T_j" is already part of "Total-T_j" (i.e. the response to the new influence had already been requested by at least one of the agents - see Example 2.2), a total backtrack is pointless since the N-agent phase would produce the same conclusion. This selective backtrack is then sufficient for instant T_j. The same must then be repeated for each instant T_k between T_j and the current planned instant T_i.

Since case 2 covers case 1, there seems to be no point in memorizing the Partial Results but only the Total Results (method named M3*).

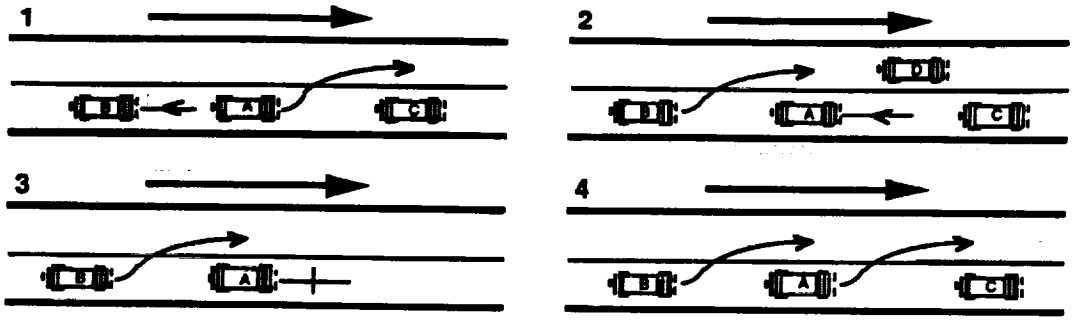
If, for any instant T_k between T_j and T_i, the results are not already included it can only be because a new response has been requested. The N-agent phase must, therefore, be triggered using a new Total Results which can be calculated in two ways:

$$\text{new-Total-}T_k := (\bigcup \text{Partial-X-}T_k \,;\, X \neq B) \oplus \text{new-Partial-B-}T_k$$

$$\text{new-Total-}T_k := \text{Total-}T_k \ominus \text{Partial-B-}T_k \oplus \text{new-Partial-B-}T_k$$

In either case, it would be useful to know certain Partial Results.

The conflict recognition phase is, therefore, avoided for any agent other than B and the backtrack is still not total. If the resultant action is the same as that which would have been generated without the new information (see Example 2.3), we only need to continue selective backtracking on actions for instants after T_k.



1. A's plan : overtake C ; B's plan : decelerate : the constraints B imposes on A are unchanged.
2. A's plan : slow down ; B's plan : overtake A : the new constraint B imposes on A, i.e. forbidden to move-out, was already imposed by D.
3. A's plan : do nothing ; B's plan : overtake A : the new constraint B imposes on A does not affect the action planned by A.
4. A's plan : overtake C ; B's plan : overtake A : the new constraint B imposes on A generates a new action, i.e. slow down. A must replan the following instants.

Example 2. Various selective backtrack levels

However, if the action generated is new (see Example 2.4), a total backtrack from T_{k+1} onwards is necessary since the new action could change the result of all the previous comparisons.

5.4 Execution monitoring

The execution monitoring of the plans generated can be done using the memorized Total Results and the selective backtracking possibilities to check that no agents, cause any infractions.

The real behaviour of human drivers could be monitored as follows:

- If man behaves approximately as the system expects then there will be no problem
- Otherwise:
  - If the man in question is driving our car, check whether the behaviour of the man is included in the prohibited behaviours memorized in the Total Results :
    - If there is infraction of one of these prohibitions, the driver could be warned (for a low-risk situation or a detected intention) or the system could take control to avoid an accident (for a dangerous situation).
    - Otherwise, complete replanning is required to adapt to this new behaviour (once the driver's intentions have been recognized...).
  - If the man is driving another vehicle, which possibly does not have the system, it is necessary to run a selective backtrack for each instant T_k between T_j and the current planned instant T_i to adapt our plan.

6 Theoretical efficiency of the methods

The theoretical costs of each of various methods (among 12 alternatives) were estimated making certain average assumptions about the multi-agent application and the way in which the databases or algorithms are designed (see [Mourou and Fade, 91b] and [Mourou, 92]). These costs are expressed as a mean number of influence tests in function of the number of other agents N, the maximum number of conflicts P and the mean number of influence tests Q used in a M1 conflict test. One result is :

M1	$Q \times N/2 \times P$
M3*	$Q \times N$

M3* requires all possible comparisons to be done while M1 only requires comparisons on request. However, it is more efficient since the influence tests are grouped.

7 Main experimental results

We simulated a highway with two lanes and carrying three vehicles fitted with a co-pilot and 10 other preprogrammed vehicles. The three equipped vehicles are associated with three different processes linked through pipes. 24 rules (10 bi-agent and 14 N-agent rules) are required with M3* to obtain an ideal response in "automatic mode" which respects safety distances and allows for the inertia of vehicles. The knowledge bases made it possible to write that for M1.

In this application, Q = 3 (relative position, relative speed, lane) and P = 3 (number of

booleans in the Results structure). N does not affect the relative performance.

M1 and M3* gave results which matched those from the above formulas : M3* is 30% faster than M1.

In the best case, but which is also the most frequent, when the Partial Results calculated are included in the Total Results, M3* performed its selective backtracks in only 20% of the time required by M1 to completely replan (with N = 10).

Conclusion

The multi-agent planning/scheduling methods described in this article make its possible achieve a more flexible, fast and reactive system. The co-pilot can anticipate the near future by using the available time and without be obliged to wait for its neighbours because it can easily check and integrate a new information.

In execution monitoring, a dangerous situation can be quickly detected. A backtrack of an agent and the selective backtrack of other agents allow to perform incremental rescheduling of the whole system.

References

[Bessiere, 84] Pierre Bessière, "Un Formalisme Simple Pour Représenter La Connaissance Dans un Contexte de Planification Multi-Agents", RFIA 84, Actes du 4ième Congrès de Reconnaissance des Formes et Intelligence Artificielle, 25-27 Janvier 1984, Paris, tome 2, pp. 345-354.

[Cammarata et al., 83] Stéphanie Cammarata, David McArthur and Randall Steeb, "Strategies of Cooperation in Distributed Problem Solving", IJCAI 83, pp. 768-770.

[Davis and Smith, 83] Randall Davis and Reid G. Smith, "Negotiation as a Metaphor for Distributed Problem Solving", Artificial Intelligence 20, pp. 63-109, 1983.

[Fraichard and Demazeau, 89] Th. Fraichard and Y. Demazeau, "Motion Planning in a Multi-Agent World", Workshop on Modeling Autonomous Agents in a Multi Agent World (MAAMAW 89), Cambridge, England, August 1989.

[Mourou, 90] Pascal Mourou, "Représentation et Simulation d'un Agent dans un Monde Multi-Agent: un Véhicule en Circulation Autoroutière", Rapport Interne N°IRIT/90-10, Février 1990, 41 p.

[Mourou and Fade, 91a] Pascal Mourou and Bernard Fade, "Vehicles controlling: Representation of knowledge and Algorithms of a Multi-Agent Decision", Sixth International Conference on Applications of Artificial Intelligence in Enginneering (AIENG 91), 2-4 July 1991, University of Oxford, UK, pp. 683-697.

[Mourou and Fade, 91b] Pascal Mourou and Bernard Fade, "Co-Pilot for the Driver Monitoring", PROMETHEUS PRO-ART Workshop on Intelligent Co-Pilot, December 12-13, 1991, Grenoble, France, pp 183-193.

[Mourou, 92] Pascal Mourou, "Evaluation comparative de techniques de décision multi-agent", Rapport Interne IRIT, à paraître.

[Rosenschein, 82] Jeffrey S. Rosenschein, "Synchronization of Multi-Agent Plans", AAAI 82, pp. 115-119.

[Rosenschein and Genesereth, 85] Jeffrey S. Rosenschein and Michael R. Genesereth, "Deals Among Rational Agents", IJCAI 85, vol 1, pp. 91-99.

[Sycara, 89] Katia P. Sycara, "Argumentation: Planning Other Agents' Plans", IJCAI 89, vol 1, pp. 517-523.

[Wilks and Ballim, 87] Yorick Wilks and Afzal Ballim, "Multiple Agents and the Heuristic Ascription of Belief", IJCAI 87, vol 1, pp. 118-124.

[Wood, 83] Sharon Wood, "Dynamic World Simulation for Planning With Multiple Agents", IJCAI 83, vol 1, pp. 69-71.

REAL-TIME CONTINGENCY HANDLING IN MAESTRO

Daniel L. Britt and Amy L. Geoffroy
Martin Marietta Astronautics Group
P.O. Box 179, ms XL4370
Denver, CO 80201

Abstract

A scheduling and resource management system named MAESTRO has been interfaced with a Space Station Module Power Management and Distribution (SSMPMAD) breadboard at Marshall Space Flight Center (MSFC). The combined system serves to illustrate the integration of planning, scheduling and control in a realistic, complex domain. This paper briefly describes the functional elements of the combined system, including normal and contingency operational scenarios, then focusses on the method used by the scheduler to handle real-time contingencies.

I. Introduction

For the past six years a team at Martin Marietta has been developing an integrated approach to scheduling, resulting in the implementation of a robust prototype scheduling system called MAESTRO [Geoffroy, Gohring & Britt, 1991]. During the same time frame another group at Martin Marietta has been building a hardware/software testbed to study various concepts in the automation of electrical power management, the Space Station Module Power Management and Distribution (SSMPMAD) system. In 1988 an initial version of the SSMPMAD system integrated with MAESTRO was delivered to Marshall Space Flight Center. Since then both the SSMPMAD system and the scheduler have gone through several revisions, and a major delivery of new software occurred in June of 1991. This paper describes that combined system, highlighting those aspects of it that illustrate concepts in integrated planning, scheduling and control. We focus on the replanning and rescheduling processes used in MAESTRO to respond to real-time contingencies, unexpected changes in the state of the power system that cause a schedule currently being executed to become invalid.

Section II defines some terms used in the rest of the paper. Section III describes the functional architecture of the system. In section IV are presented two operational scenarios, one for normal operations and one which describes a possible contingency. Section V provides a description of the processes carried out by the scheduler to effect real-time replanning and rescheduling, including timing issues. In section VI we conclude with indications of possible future directions for this research.

II. Definitions

For the purposes of this discussion we will make use of the following restricted definitions. *Planning* is defined to be the process of specifying goals to be achieved onboard a spacecraft, and further, of specifying the activities which will achieve those goals. This involves determining these activities' structure as well as constraints on the execution of them. *Activities*, in turn, are defined to be sequences of subtasks which accomplish the desired goal. *Scheduling* is defined to be the process of selecting some subset of these activities and specifying exact start/end times and resource assignments for their component subtasks. A *valid schedule* is a specification of start/end times and resource assignments for a set of activities such that the activities may be executed as scheduled. A *contingency* arises when a previously valid schedule becomes invalid as a result of a change in the assumptions upon which that schedule was based. The term *real-time* is used here to mean "during execution of the activities on a schedule". This does not have the connotation from control theory that a real-time event must be responded to within microseconds, but rather is used to differentiate between actions that are occurring at the moment as opposed to those that will occur at some point in the future. A *load* is the use of electrical power by a piece of equipment.

The authors would like to acknowledge John Gohring of Martin Marietta Western Internal Systems and Joel Riedesel of Martin Marietta Astronautics Group for their significant contributions to this report and to the work described herein.

III. Functional Architecture

Figure 1 depicts the functional elements of the combined SSMPMAD/MAESTRO system and relationships among these elements. Briefly, the Activity Editor is used to create definitions for activities which accomplish goals desired by the user. MAESTRO is used to select and schedule a subset of these activities, and to save the resultant schedule(s) out to files. The Transaction Manager (TM) serves as a communications port, facilitating specific types of communications between MAESTRO and the rest of the system during breadboard operation. The Front End Load Enable Scheduler (FELES) creates schedules of power system events (such as closing switches) from saved schedule files. The Communications and Algorithmic Controller (CAC) distributes schedules among Load Centers (LCs), into which are incorporated Lowest Level Processors (LLPs). These LLPs actually control hardware switches on the power system breadboard, as well as monitoring the states of various sensors distributed throughout the system. The Fault Recovery And Management Expert System (FRAMES) performs fault isolation, diagnosis and recovery for the power system, and communicates with the scheduler during real-time contingencies. The Load Priority List Management System (LPLMS) maintains a list of active loads in a prioritized order such that if there is a need to quickly reduce power consumption in a portion of the breadboard, loads can be shed (turned off) in an order that minimizes the impact of this load shedding.

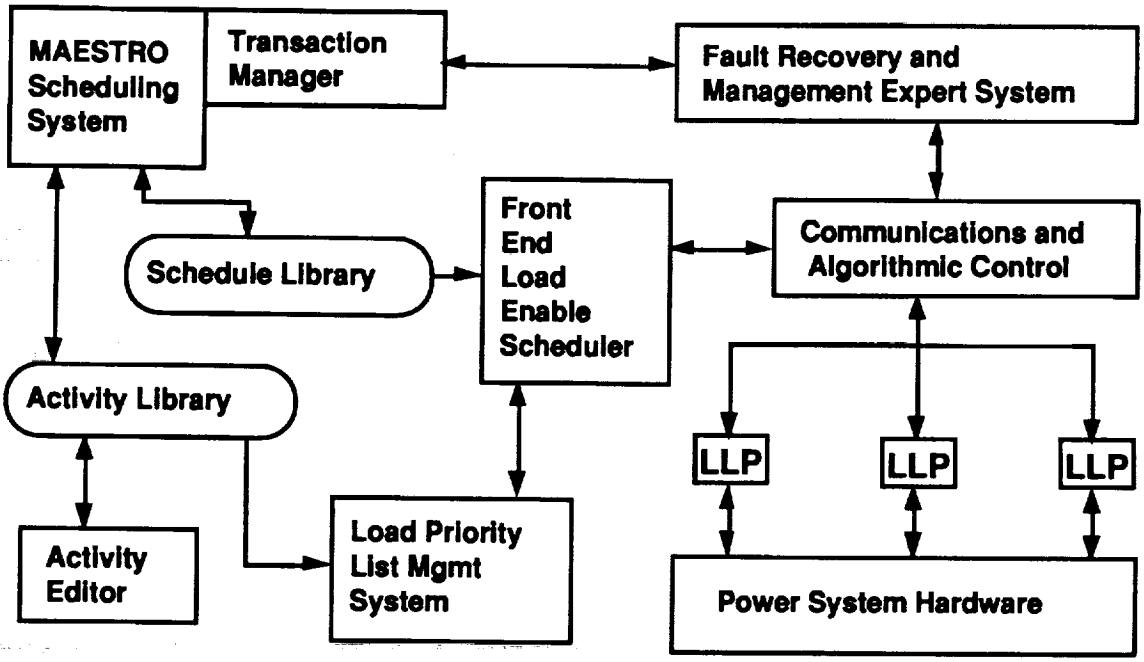


Figure 1. Functional architecture of the MAESTRO/SSMPMAD combined system.

A portion of the actual power circuits on the breadboard is depicted in figure 2. Note that several 1-kilowatt Remote Power Controllers (RPCs) can be attached to a single 3-kilowatt RPC. Thus it is possible to overload an intermediate RPC without overloading any of the lower-level RPCs connected to it. For this reason it is necessary to represent the entire power path for each power-using resource to the scheduling system, rather than just representing total power consumed by each activity.

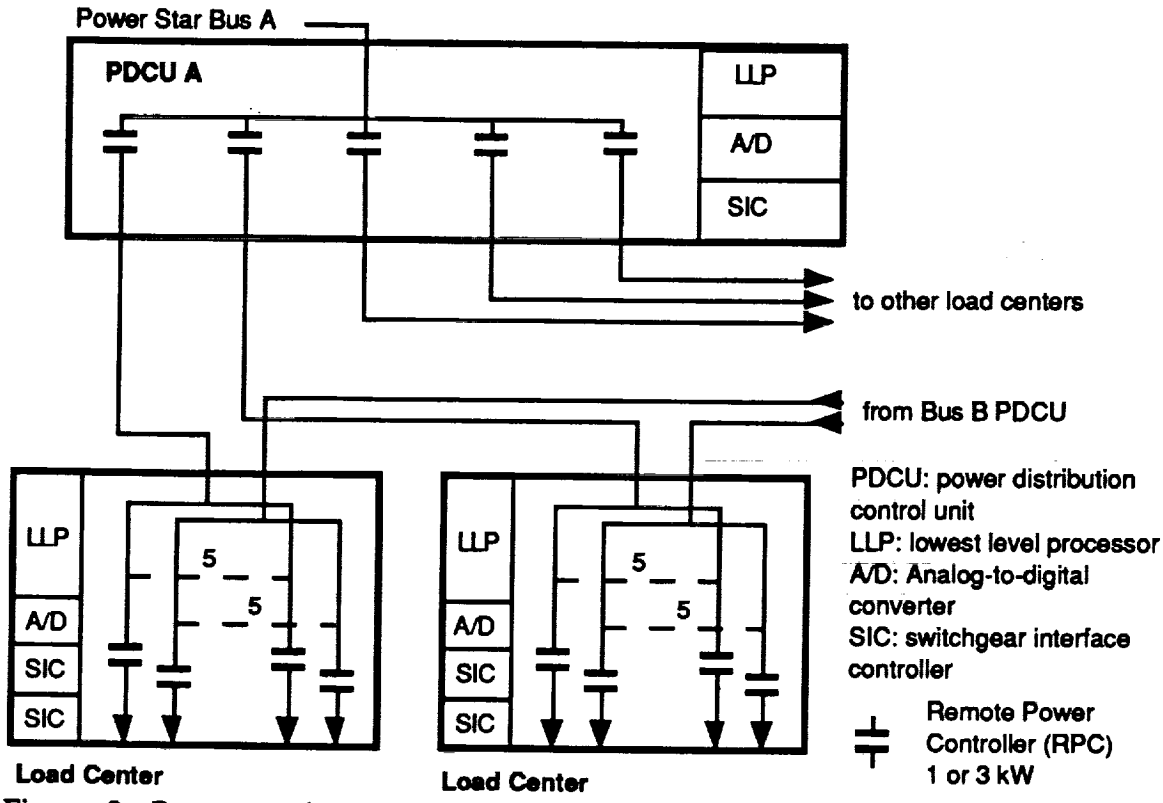


Figure 2. Representative schematic of a portion of the SSMPMAD breadboard.

IV. Operational Scenarios

Normally, a user will interact with the activity editor to create a set of activities to be scheduled, saving these activities' definitions in an activity library. In that or another session, the user will run the scheduler to create one or more initial schedules of these activities. These schedules will be saved into a schedule library. When a user wishes to operate the power system breadboard, s/he uses the SSMPMAD interface to select a saved schedule, initialize the system and execute that schedule. The FELES first obtains a saved schedule and translates a portion of it (roughly one-half hour of activity) into a series of power system events, specifying at what times and power levels each RPC is to be turned on. The LPLMS takes this schedule of power system events and creates a list of loads to shed in an emergency power reduction. The event schedule and priority list are transmitted to the CAC, which distributes them among the LLPs as appropriate. The CAC also maintains a system clock, coordinating timing for the various elements.

Execution of the distributed schedule proceeds with the LLPs directing the RPCs to close and open switches at the times specified by their respective event schedules. The RPCs monitor voltage, current, temperature and other parameters of their operations.

Prior to the expiration of the timeline increment being executed, the FELES will acquire another increment from the saved schedule, translate it into power system events, and transfer it to the CAC, which distributes it to the LLPs. At a specified time, the LLPs stop executing the old increment event list and begin executing the new one.

When an anomalous condition (such as over-current or under-voltage at a switch) is detected by one of the RPCs, it automatically takes a safing action, if possible. The LLP controling it reports this event to FRAMES, which gathers together all available information about the fault, isolates it, and compiles a list of system configuration status changes resulting from the fault. These changes can include a load being switched to a

redundant power source (redundancy switching), an RPC going out of service, the deliberate shutdown of a load to reduce power consumption (load shedding), or a reduction in power available at an RPC and the expected duration of that reduction. This list of changes is then communicated to the scheduler, which revises the activity schedule to reflect the changes and makes the new schedule available to the FELES. It creates a new event list, which is distributed to the LLPs along with a time tag indicating when to begin executing the new schedule.

V. The Real-Time Rescheduling Process

When a power system anomaly occurs, MAESTRO will get a set of information from FRAMES throught the TM. This information will include the current time in addition to redundancy switch, load shed, power availability change, and RPC out-of-service messages. These messages will include the time the event occurred, and if applicable the duration of the change. MAESTRO follows a three-step process to handle these messages and revise the schedule. It 1) modifies the schedule to reflect changes made to it by the power system and to remove resource and temporal constraint violations for activities not yet begun, 2) tries to find ways to create and schedule continuations for interrupted activities, and 3) tries to schedule any activities that can take advantage of the resources released by the interruption of others. The first step results in a valid but possibly not very efficient schedule. It is carried out as quickly as possible to ensure that a workable schedule can be in place soon, reducing the likelihood that adherence by the power system to the old (invalid) schedule will result in a cascade of faults registered by that system. The second and third steps will only be attempted if there is sufficient time to get something useful done. Management of its own computation time is a difficult issue for a real-time rescheduler. It must project a time when it will have a valid schedule available, including the time it takes to transmit that schedule to the entities responsible for carrying it out, then not make changes to the schedule (other than those already made by the power system) that would need to be acted upon before they are received by the power system. For example, if at 10:00 a contingency occurs, and the scheduler determines that an interrupted activity can be continued at 10:05, but this information cannot be transmitted to the power system until 10:08, then the schedule is invalid the moment the system begins to execute it. In this example the scheduler could specify that the activity be continued at 10:08, but not before.

The actual structure used to control the three-step process mentioned above is a prioritized list of command queues. As information comes in from FRAMES, it is routed to one of several command queues, for action as soon as MAESTRO has nothing more important to take care of. Resource availability changes appear in one queue, while redundancy switches are in another and load sheds in a third, for example. MAESTRO will be in a wait state until something appears on one of its command queues, at which time it will process a command from the highest priority queue that has an item, then check all the queues again for new items, returning to the wait state when no items remain.

MAESTRO will add items to its command queues as a result of its own processing. Handling a resource availability change, for example, will cause MAESTRO to add a command to check for resource constraint violations. If a violation is found and an activity interrupted, MAESTRO will add a command to try to plan and schedule a continuation of that activity.

Activity continuation is the single automated planning function within MAESTRO. When initially creating activities, the user specifies ways and conditions under which each subtask may be continued if it is temporarily interrupted. Three continuations are currently represented for each subtask. These are effectively operators that can be selectively applied to achieve the goal of a completed activity performance. First, the unexecuted portion of an interrupted subtask may be skipped, with a parameter stating how much time the subtask must execute prior to the interruption. A data collection subtask could be terminated early and data analysis begun, for

example. Second, a subtask may be continued after a sufficiently brief interruption. Finally, the interrupted subtask may be started over again, making use of states set by previous subtasks but not using the progress gained in the interrupted subtask.

The scheduler will create a new activity model appropriate for a particular type of continuation using information from the interrupted activity and possible continuations specified by the user for that activity. Each of the above continuations has different implications for the rescheduling of the subtasks following the interrupted one, so MAESTRO must try various options in order to find a viable placement for the new activity. MAESTRO can represent temporal constraints between activities, sometimes necessitating the consideration of more than one continuation model at once. This complexity combines with the time limitations on rescheduling to prohibit MAESTRO from finding the "best" way to continue an activity - it simply accepts the first viable continuation found. Attempts are heuristically ordered such that higher-value continuations are tried earlier, however. Note that in many cases no continuation will be possible, in which case the work done to represent the current state of the system is all that can be accomplished for a particular activity. Note also that safing actions are not scheduled but rather are carried out immediately and automatically by the subsystems involved.

As each continuation attempt is made, the system consults the system clock, abandoning further attempts at the point where they would cause changes made to the schedule to be unimplementable. When all continuation attempts have been tried (and there may be none tried), if there is still time, the scheduler will attempt to add new performances of activities to the schedule. System time is checked after each schedule addition, and this process ends when time runs out or no more activities can be added to the schedule. At that point the schedule is made available to the FELES, and schedule execution proceeds as previously described.

VI. Future Directions and Related Work

Work is continuing on MAESTRO, as it is on SSMPMAD. The scheduler needs to be enhanced to manage the timing and consistency issues that arise when a user wishes to alter a schedule that is currently executing. We also intend to enhance the representational as well as computational power of the system. The current methods for finding a way to continue an interrupted activity are cumbersome and depend too much on initial user input into the representation of the subtasks. A more appropriate method would be to have an intelligent system monitoring each experiment or other major activity, with the capability to plan continuations based on an accurate assessment of the state of the activity.

We have begun a task similar to the MAESTRO/SSMPMAD integration for Kennedy Space Center under the Advanced Launch Processing (ALP) contract. In that effort we will build a system executive capable of coordinating the actions of multiple Knowledge-Based Autonomous Test Engineer (KATE) systems [Parrish & Brown, 1991]. These systems are used to monitor and control individual launch vehicle subsystems during testing and launch, but are independent of one another. The system executive will interface with the Kate systems as well as with higher-level launch flow management functions, enhancing integrated vehicle systems tests and reducing launch costs.

VII. References

Geoffroy, A.L. Gohring, J.R. & Britt, D.L. (1991) Sharing intelligence: Decision-making interactions between users and software in MAESTRO. *Telematics and Informatics.* 8 (3/4).

Parrish, C.L. & Brown, B.L. (1991) Knowledge-Based Autonomous Test Engineer (KATE). *Technology 2001 Conference.* NASA. December, 1991, San Jose, CA.

Learning to Integrate Reactivity and Deliberation in Uncertain Planning and Scheduling Problems

Steve A. Chien
Jet Propulsion Lab, M/S 525-3660
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109-8099
chien@aig.jpl.nasa.gov

Melinda T. Gervasio
Beckman Institute
University of Illinois
405 North Mathews Avenue
Urbana, IL 61801
gervasio@cs.uiuc.edu

Gerald F. DeJong
Beckman Institute
University of Illinois
405 North Mathews Avenue
Urbana, IL 61801
gervasio@cs.uiuc.edu

Abstract

This paper describes an approach to planning and scheduling in uncertain domains. In this approach, a system divides a task on a goal by goal basis into reactive and deliberative components. Initially, a task is handled entirely reactively. When failures occur, the system changes the reactive/deliberative goal division by moving goals into the deliberative component. Because our approach attempts to minimize the number of deliberative goals, we call our approach Minimal Deliberation (MD). Because MD allows goals to be treated reactively, it gains some of the advantages of reactive systems: computational efficiency, the ability to deal with noise and non-deterministic effects, and the ability to take advantage of unforseen opportunities. However, because MD can fall back upon deliberation, it can also provide some of the guarantees of classical planning, such as the ability to deal with complex goal interactions. This paper describes the Minimal Deliberation approach to integrating reactivity and deliberation and describes an ongoing application of the approach to an uncertain planning and scheduling domain.

INTRODUCTION

The AI problem of automatically achieving goals has been redefined in the last few years. The classical planning problem can be broadly characterised as finding a set of operators together with sufficient constraints such that when applied to some initial state the resulting state provably satisfies some goal relation. However, this is a narrow view of what is now seen as a more general problem. Recently, there has been a great deal of interest in reactivity as a model of action [Suchman87]. While the classical view of planning has been shown to have computational problems [Chapman87]; from a different perspective one might instead blame our failure to conceive of alternative frameworks for modeling world changes and formalisms for action selection.

Reactivity takes a different, more efficient view of action selection. Pure reactivity fundamentally gives up the idea of projecting the results of actions. Instead an agent reacts to the current state of affairs in the world as directly perceived by sensors. In a sense, reactivity is a hill-climbing action-selection model. The evidence taken into account in the selection of an action is necessarily local (i.e., the current readings of sensors). Based on this purely local information an action is taken that may have resounding global ramifications, fooling the agent into climbing to the top of a locally steep foothill from which state the goal is unachievable.

This phenomenon often occurs in the form of interacting sub-goals both in planning and scheduling. In a planning context, as you exit the parking lot on your way home from work you may prefer a right turn (it more directly leads toward your house, it is less expensive than a left turn across traffic, etc.). However, in the context of a second goal of picking up a loaf of bread, it may be better to turn left, taking you past a supermarket on the way. In a scheduling context, interactions occur through resource contention. A job may finish earlier if allowed to execute one of its subtasks at a certain time, but the overall schedule may suffer. Approaches that address managing such problems of purely reactive systems include: developing a theory of benign environments in which a reactive agent may be more certain that its reactive inclination will meet with success [Agre88, Hammond90]; the integration of classical planning with reactivity [Drummond90, Kaelbling86, Turney89]; and application of machine learning to this end [Gervasio90, Mitchell90, Laird90]. These approaches begin with what is essentially a classical planner and, guided by experience, result in the formulation of reactive components as well.

This research approaches planning and scheduling from a different point of view. Instead of learning to incorporate reactivity into a classical deliberative framework, we propose incorporating minimal classical deliberation into an initially purely reactive system. As failures are encountered, the system utilises its world model to explain why the desired state of affairs was not brought about by the executed actions.

In the case of a failure of a reactive goal, the failure could be due to a faulty set of reactions or due to uncertainty in the effects of actions or schedules. In the case of failure of a deliberative goals, the failure must be due to interference from a reactive goal. In the case of uncertain effects causing the failure of a reactive goal, deliberation can be used to attempt to improve the plan. In the case of reactive interference in a

reactive or deliberative goal, the offending reactions are inhibited by moving the associated goal into the deliberative component, where the negative goal interaction will be considered and avoided.

In this way the purely reactive system adopts just enough deliberation to avoid goal interaction pitfalls. Since deliberation occurs only in reaction to observed failures, (i.e. the resultant plan remains uncommitted on those goals not appearing in the failure trace) this approach will generally retain some level of flexibility by avoiding a rigid classical plan or schedule for all of the goals. Thid flexibility allows the MD approach will retain some of the benefits of reactivity: tolerance of noise, uncertainty, and incomplete knowledge as well as computational efficiency. Yet the MD approach also benefits from its ability to fall back upon traditional deliberative planning. It gains the ability to solve problems which require simultaneous consideration of multiple interacting goals. Additionally, through explanation-based learning(EBL), it gains the ability to cache and generalize decisions made in the plan construction process. As with traditional EBL, the learned deliberation molecules allow a system to find plans more quickly. But more importantly, these deliberation molecules allow a system to avoid repeating the failures resulting from the short-sighted decision of the reactive component.

These benefits of coordinating reactivity and deliberation are relevant to both planning and scheduling issues described in this paper. Reactivity can take advantage of unforseen opportunities. In, planning this is the ability to take advantage of fortuitous conditions in the world state. In scheduling, this is the ability to take advantage of unforseen resource availability. Another strength of reactivity is the capability to deal with uncertainty and noise. In planning this means the ability to deal with uncertain action effects and/or world state. In scheduling this means the ability to deal with uncertain resource consumptions and availabilities. A third strength of reactivity is its computational efficiency due to avoidance of explicit projection. In planning, this means not having to explicitly determine future world states. In scheduling, this means not having to explicitly determine future resource utilization. The principal strength of deliberation is the ability to deal with arbitrary goal interactions by searching the space of possible plans and/or schedules. In planning this means being able to deal with complex precondition and effect interactions between goals. In scheduling, this means being able to deal with difficult resource interactions.

There are a number of assumptions underlying the MD approach. First, we assume that the cost of failures is sufficiently low so that the cost of failures incurred while acting reactively is outweighed by the overall gains in flexibility and efficiency from reactivity. A corollary to this assumption is that the reactive component is sufficiently competent to solve the majority of the goals. Without this constraint, the MD approach would incur the cost of numerous failures only to end up doing primarily deliberative planning. Second, we assume the presence of domain models to allow the system to fall back upon classical planning as well as permitting use of EBL. Third, the system must be allowed multiple attempts to solve a problem.

THE MD ARCHITECTURE

The system architecture advocated by the MD approach is that of an interacting set of components: a deliberative element, a reactive element, and a learning element. The deliberative element is a conventional planner which constructs classical plan/schedule molecules for goal conjuncts requiring deliberation. By analysing the precondition and schedule interactions and performing extensive search deliberation can resolve the goal interactions. The learning element uses EBL [DeJong86, Mitchell86] to learn general plan/schedule molecules which indicate how to achieve a set of goals by designating a reactive/deliberative goal allocation and a set of actions for the deliberative goals.

The reactive element proposes actions using a shallow decision model of reaction rules. Each reactive rule specifies a set of state conditions and resource requirements which specify an action as appropriate to execute. Multiple actions may be executed during a single timestep if resources allow. In most cases, failures in the reactive component will be due to goal interactions. Reaction rules consist of interrupt rules, which cause actions to be executed regardless of the other actions the agent is taking (i.e. actions determined by the deliberative component), and suggestion rules, which are executed when the system has no current pending actions. Thus, interrupt rules represent actions to take advantage of immediate opportunities or avoid dangerous situations regardless of the current deliberative plan, while suggestion rules direct activity when the system is confronted by a set of goals, and does not have a current plan.

Every reaction rule is defined with respect to a goal, and can only apply when its goal matches a reactive goal of the system. Thus, a reaction can be overridden by the deliberative component by removing the triggering goal from the set of reactive goals and planning for the goal deliberatively. Thus, in our architecture, there are three levels of priority: interrupt rules, the action advocated by the current plan, and suggestion rules. Within a given priority level, if more than one action is applicable, the system chooses one arbitrarily but deterministically (e.g. the same set of goals and state will produce the same action). For example, in a delivery domain, interrupt rules might trigger when the truck is at the location of one of its deliveries. This can occur in the midst of executing a decision molecule constructed by the deliberative component, and it results in actions other than those in the decision molecule. An example suggestion rule would be one which causes the truck to move towards the closest delivery site if it does not have a decision molecule to guide it otherwise.

THE MD APPROACH

In the MD approach, a system originally acts based upon a shallow, simple decision model. Through experience, the system gradually acquires a set of decision molecules which allow it to plan past local maxima encountered by the shallow decision model. Because of this progression, we describe the MD approach as "becoming decreasingly reactive", as the proportion of goals the system solves by deliberation increases (where we also consider as deliberative the compiled decision molecules created by the deliberative element). Eventually, for a fixed distribution of problems, the system will learn a set of decision molecules sufficient to allow it to solve the problems occurring in the distribution. Furthermore, because the MD approach uses EBL, the system also learns to avoid a general class of failures relevant to a particular plan, thus reducing the number of failures required to learn a satisfactory set of plans.

A problem consists of a conjunction of goals, and the task of a system in the MD approach is to divide the goals into a deliberative set and a reactive set such that the goals are all achieved with the minimum amount of deliberation and maximum amount of flexibility possible. A plan to solve a conjunction of goals is thus a composite plan/schedule which consists of a decision molecule, constructed by the deliberative component to solve the set of deliberative goals, and a set of reactive goals to be achieved by the reactive component. The MD algorithm is shown below:

```
Given a problem consisting of:

G - a set of problem goals
I - the initial state

REAC := G
DELIB := {}

loop
PLAN := Classical_Planner(DELIB,I)
Execute(PLAN,REAC)
if all goals achieved return SUCCESS
else if REAC = {} return FAIL
else
    for each goal in REAC
        if <goal not achieved> OR
           <reactive action in pursuit of goal
            interfered with another goal G'>
        then
           REAC := REAC - goal
           DELIB := DELIB + goal
go loop

if SUCCESS then generalize successful plan
```

The key to the MD approach is the blame assignment process. In general, failures are due to interactions between subgoals, as the reactive methods are intended to be sufficient to achieve goals without interference. Interference can occur at the planning level (due to an action in service of one goal clobbering a protection in service of another goal) and at the scheduling level (resource expenditures due to one goal causing a resource failure for another goal).

Blame assignment consists of determining which goals are involved and then using this information to reduce future failures due to goal interactions. In goal identification process. there are planning failures and scheduling failures. Each of these failure types (planning, scheduling) can cause a goal to be identified as relevant to a goal analysis. In the first way, a goal G fails, likely due to actions in service of another goal. This goal is called a conflictee and is considered in the analysis described below. This set of circumstances can be detected by checking if goals are achieved at the end of execution (infinite looping is detected by an execution limit). The second relevant goal type is a conflicter goal. A goal G is deemed a conflicter if an action A in service of G caused a failure of another goal H. In the context of planning, this occurs if the conflictee H is a deliberative goal and A clobbers a protection in the plan to achieve H. In a scheduling context a goal G is deemed a conflicter if an action A in service of G was the largest consumer/user of a resource R which caused a scheduling failure for a deliberative goal H.

We now describe how this determination of goal interference is used to modify the allocation of reactive and deliberative goals. If a reactive goal G1 fails without interference, it is moved to the deliberative component and thusly will be achieved by the classical planner and scheduler. A deliberative goal G1 cannot fail without interference as the planner performs full projection. In the case of a goal failing due to interference from a second goal G2, there are four cases, G1 and G2 reactive, G1 reactive and G2 deliberative, G1 deliberative and G2 reactive, and G1 and G2 both deliberative. How each of these cases is treated is described below.

1. Because the deliberative element performs full projection, two deliberative goals cannot interfere, thus the failure case of both G1 and G2 deliberative cannot occur.
2. If G1 is a reactive goal, and G2 deliberative, the MD approach will move G1 to the deliberative goal set and the classical planner will ensure that the negative goal interaction between G1 and G2 will be avoided.
3. If G1 is deliberative and G2 reactive, then due to the blame assignment scheme G2 will be moved into the deliberative component. In the next cycle both G1 and G2 will be delegated to the deliberative component and the interaction will be considered and avoided.
4. If G1 is a reactive goal and it has been thwarted by another reactive goal G2, the blame assignment scheme will move G1 to the deliberative component. If in the next cycle G2 still interferes, it is an example of case 3 above and will be treated accordingly.

Thus the process of moving more goals to the deliberative component continues until the system converges upon a set of deliberative goals for which the planner and scheduler constructs a plan and schedule which in combination with the reactive element achieves all of the problem goals.

This classical plan is then generalised using EBL,

with the reactive goals being generalised to a default level. This resultant plan structure (and reactive/deliberative division) can then be used to solve future problems as follows. When problems are initially posed to MD, it begins by attempting to match the goals and initial conditions to an existing decision molecule. If a matching decision molecule exists, it is used in an attempt to solve the plan. If all such matching molecules fail, the system attacks the problem entirely reactively and the entire MD approach is called from scratch.

EVALUATION

The MD approach has been implemented for a simple delivery planning domain [Chien91]. We have extended the failure analysis algorithm and are currently implementing this newer version of MD for a more complex mathematical planning and scheduling domain. This ongoing implementation is the one described in this paper. In this mathematical domain, each goal can be achieved by the execution of a number of actions. Each action has a randomised number of resource requirements, and possibly state requirement preconditions for each of the resources (e.g., a value for a predicate on the resource). Planning goal conflicts occur through incompatible resource state requirements. Scheduling resource contention occurs through goals competing for resources. Uncertainty exists through a random element in duration of primitives (and thus resource usage).

We plan to test our architecture by generating domain theories which vary a number of parameters which will affect the overall scheduling and planning goal interaction rate. The domain parameters are: 1) the # of resource types (affects resource and conflict rate); 2) average number of resources each action uses (affects resource conflict rate); 3) frequency and types of resource conditions (affects planning conflict rate); and 4) # of preconditions per primitive (affects planning conflict rate). Finally, we plan to vary the amount of action duration uncertainty, which affects the amount of benefit gained by deferring decision-making.

In order to compare with the MD approach, we are currently implementing a fully deliberative planner and scheduler. This comparison classical system simply delegates all of the goals to the deliberative component.

The metrics which we plan to use to evaluate the plans produced by the two systems are: 1) total CPU time required for decision-making; 2) robustness of the schedule (% of goals achieved by deadlines); 3) average time to completion of individual goals; and 4) average time to completion of all goals. These metrics will be evaluated for different combinations of the domain parameters described above.

DISCUSSION

This research is preliminary, and there are a number of outstanding research issues. One difficult issue is determining the correct level of generalization for the reactive portion of any plan/schedule. Because reactive actions are undetermined, analysing generality of the goal achievement methods is difficult. While committing the planner to the same general set of actions used by the reactive component in the current problem would allow EBL on the action trace, it commits the planner to the same general set of actions - losing the flexibility allowed by reactivity and forcing a possibly expensive causal analysis of the example. Yet another approach would be to generalise the reactive portion aggressively and allow later learning to either reduce the level of generality or learn more specific plans which would shadow the over-general plan in cases where it was inappropriate.

One view of the MD approach is that of using deliberation to learn patches to a set of reactive rules. In this view our techniques allow for encoding of a quick and dirty set of reactive rules which solve the majority of problems. Through learning, a set of patches can then be constructed to allow these imperfect rules to solve a given distribution of problems.

Another interesting issue for examination is the tradeoff between reactivity and deliberation in the purely reactive component. Currently, the reactive component does no projection before interrupting the current plan and the deliberative element performs full projection. While ideally both approaches components would be less extreme, the same general mechanisms for integrating deliberation and reactivity would apply.

Another possible approach to integrating deliberation and reactivity is to use the same failure-driven method for splitting goals between the reactive and deliberative component to learn control rules specifying allocation of goals to the deliberative and reactive components. While we feel that the current MD macro-based approach better preserves the notion of a plan/schedule context in that the deliberative actions selected may impact the success of the reactive component, this is a larger issue involving the operationality/generality tradeoff.

Another issue is that of controlling moving interacting goals into the deliberative component. Managing the tradeoff between more expensive (and likely more accurate) failure analyses and more heuristic (and likely less accurate) goal analyses is an issue for future work.

RELATED WORK

Drummond and Kaelbling [Drummond90, Kaelbling86] describe anytime approaches wherein planning is used to constrain the reactions, which are always available for deciding on actions. [Turney89] interleaves planning and execution by allocating some predetermined amount of time to each phase in turn, while [Hanks90] uses the constraints of urgency and insufficient information to determine when to pass control to the reactive component. In these approaches, any goal may thus be addressed reactively or deliberatively. In contrast, a system in the MD approach initially addresses all its goals reactively but incrementally learns which goals

require deliberation to avoid negative interactions and which goals can be addressed reactively without preventing the achievement of other goals. Thus, the MD approach can guarantee the achievement of its goals, which the others in general cannot.

Guaranteed goal achievement is similar to ideas presented in [Gervasio90, Martin90]. In [Gervasio90], the a priori (deliberative) planner must construct an achievability proof for each deferred goal, while in [Martin90], the strategic (deliberative) planner assigns the reactive planner those goals which the reactive planner has proven itself capable of handling. In contrast, in the MD approach, each goal is considered achievable during execution until experience shows otherwise. The MD need not prove achievability but instead incurs failures to determine which goals must be deliberated upon.

In [Mitchell90, Laird90] systems become increasingly reactive by compiling deliberative decisions into stimulus-response rules/chunks. As the decision molecules learned by MD are compiled schemata, MD becomes increasingly reactive in the same sense. However, it becomes decreasingly reactive in the sense that it initially addresses all goals reactively, but gradually learns to address particular goals deliberatively. In contrast, since Theo-Agent and SOAR derive all their rules/chunks from deliberative plans, they always address their goals purely deliberatively.

TRUCKER [Hammond88] learns to optimise its planning from successful opportunistic problem-solving. While in the MD approach, a system learns which goals interact negatively and modifies its planning behavior to deliberate over these goals and avoid the interaction, TRUCKER learns which goals interact positively and modifies its planning behavior to take advantage of this interaction. Other work on learning from failure deals with purely deliberative plans, in contrast to the composite plans in the MD approach.

CONCLUSION

This paper has presented an approach to integrating reactivity and deliberation in planning and scheduling in uncertain domains. In this approach, called Minimum Deliberation (MD), the problem-solver initially attempts to solve all goals reactively. When the system encounters failures it responds by moving reactive goals into the deliberative component. By performing this refinement, the system extends its analysis of the problem minimally until the reactive component can solve the remainder of the goals. Resultant successful plans are then generalised using a combination of EBL and default generalization information. By introducing deliberation minimally, the MD approach retains some of the benefits of reduced computation and flexibility from reactivity while still being able to fall back upon deliberation to solve complex interactions.

Acknowledgements Portions of this research were performed by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Other portions of this work were performed at the Beckman Institute of the University of Illinois and were supported by the Office of Naval Research under Grants N-00014-86-K-0309 and N-00014-91-J-1563.

References

P. E. Agre, "The Dynamic Structure of Everyday Life," Ph.D. Thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, Oct 1988.

D. Chapman, "Planning for Conjunctive Goals", *Artificial Intelligence 32*, 3 (1987).

S. A. Chien, M. T. Gervasio, and G. F. DeJong, "On Becoming Decreasingly Reactive: Learning to Deliberate Minimally," *Proceedings of the Eighth International Workshop on Machine Learning*, Evanston, IL, Jun 1991.

G. F. DeJong and R. J. Mooney, "Explanation-Based Learning: An Alternative View," *Machine Learning 1*, 2 (Apr 1986).

M. Drummond and J. Bresina, "Anytime Synthetic Projection: Maximising the Probability of Goal Satisfaction," *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, Aug 1990.

M. T. Gervasio, "Learning General Completable Reactive Plans," *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, Aug 1990.

K. Hammond, T. Converse and M. Marks, "Learning from Opportunities: Storing and Re-using Execution-Time Optimizations," *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, MN, Aug 1988.

K. Hammond, T. Converse and C. Martin, "Integrating Planning and Acting in a Case-Based Framework," *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, Aug 1990.

S. Hanks and R. J. Firby, "Issues and Architectures for Planning and Execution," *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, San Diego, CA, Nov 1990.

L. P. Kaelbling, "An Architecture for Intelligent Reactive Systems," *Proceedings of the 1986 Workshop on Reasoning About Actions & Plans, Timberline*, OR, Jun 1986.

J. Laird and P. Rosenbloom, "Integrating Execution, Planning, and Learning for External Environments," *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, Aug 1990.

N. G. Martin and J. F. Allen, "Combining Reactive and Strategic Planning through Decomposition Abstraction," *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, San Diego, CA, Nov 1990.

T. M. Mitchell, R. Keller and S. Kedar-Cabelli, "Explanation-Based Generalisation: A Unifying View," *Machine Learning 1*, 1 (Jan 1986).

T. M. Mitchell, "Becoming Increasingly Reactive," *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, Aug 1990.

L. A. Suchman, Plans and Situated Actions, Cambridge University Press, Cambridge, 1987.

J. Turney and A. Segre, "SEPIA: An Experiment in Integrated Planning and Improvisation," *Proceedings of The AAAI Spring Symposium on Planning and Search*, Mar 1989.

Completable Scheduling: An Integrated Approach to Planning and Scheduling

Melinda T. Gervasio and Gerald F. DeJong

Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana–Champaign
405 N. Mathews Ave., Urbana, IL 61801
gervasio@cs.uiuc.edu dejong@cs.uiuc.edu

Abstract

The planning problem has traditionally been treated separately from the scheduling problem. However, as more realistic domains are tackled, it becomes evident that the problem of deciding on an ordered set of tasks to achieve a set of goals cannot be treated independently of the problem of actually allocating resources to the tasks. Doing so would result in losing the robustness and flexibility needed to deal with imperfectly modeled domains. Completable scheduling is an approach which integrates the two problems by allowing an a priori planning module to defer particular planning decisions, and consequently the associated scheduling decisions, until execution time. This allows a completable scheduling system to maximize plan flexibility by allowing runtime information to be taken into consideration when making planning and scheduling decisions. Furthermore, through the criterion of achievability placed on deferred decisions, a completable scheduling system is able to retain much of the goal–directedness and guarantees of achievement afforded by a priori planning. The completable scheduling approach is further enhanced by the use of contingent explanation–based learning, which enables a completable scheduling system to learn general completable plans from example and improve its performance through experience. Initial experimental results show that completable scheduling outperforms classical scheduling as well as pure reactive scheduling in a simple scheduling domain.

Introduction

The planning problem has traditionally been treated separately from the scheduling problem. Planning deals with the determination of an ordered set of actions for achieving a set of goals. In the context of scheduling domains, planning deals with determining an ordered set of tasks for a set of jobs. In contrast, scheduling deals with the actual assignment of tasks to machines and is generally concerned more with finding the best of several alternative task–machine assignments than with finding a particular task–machine assignment. As more realistic scheduling domains have been addressed, however, it has become apparent that planning and scheduling cannot be treated independently. The complexity of real–world domains makes perfect characterizations difficult to construct and often unwieldy. To this end, researchers in both planning and scheduling have investigated reactive approaches which allow for decision–making during execution [Agre87, Firby87, Kaelbling88, Muscettola90, Ow88, Prosser89].

However, the classical approach of first doing planning and then scheduling still remains a problem. Consider giving a classical system in a process planning domain the job of manufacturing a particular part. Its planner must decide a priori on an ordered set of actions or operations which will result in the conversion of the raw material into the desired product. Its scheduler is then given the responsibility of actually allocating resources and carrying out the operations on the machines. However, because the planner commits the system to a particular set of operations, the scheduler may not execute the best plan. For example, the planner may not be able to guarantee that the efficient new milling machine will be available and so choose the older, slower one. However, during execution, the more efficient machine may turn out to be available, but the scheduler does not have the option to alter the plan. Furthermore, an over–constrained classical plan may prevent quick fixes to unpredictable runtime situations. For example, a chosen drill bit may turn out to be unavailable, thus rendering invalid those subsequent actions involving a correspondingly sized bolt and wrench. A simple fix would be to use a different drill bit, and switch to the appropriately sized bolt and wrench, but a scheduler with a completely–determined plan does not have this capability.

A purely reactive approach, with no a priori planning, has its own problems. Most manufacturing domains are fairly well–behaved; there is much information available a priori and fairly accurate predictions can be made about the behavior of the world under particular circumstances. A purely reactive approach which performs no projection cannot take advantage of this information to constrain its actions and prevent thrashing. With the planning problem and the scheduling problem combined, the runtime decision–making problem also becomes a larger and more complex one.

This research began as an attempt to address the problems with classical, a priori planning and pure reactivity. In particular, completable planning was developed as an approach which combined the goal–directedness and provably correct plans of classical planning with the flexibility and ability to utilize runtime information afforded by reactive planning. This enabled a completable planner to more efficiently deal with the problems arising from imperfect a priori information while still retaining the benefits of planning beforehand in relatively well–behaved environments. More recently, we have been investigating scheduling, and we have found that many of the techniques originally developed as part of the completable approach to planning are also useful for solving some of the problems which arise from scheduling in realistic domains where perfect a priori information about the environment is unavailable. For example, in the scheduling scenario above, a completable scheduling system could defer the deci-

sion of which milling machine to use as well as the choice of bit, bolt, and wrench sizes. During execution, it can then use additional information regarding resource availability to address the deferred planning decisions and make the associated scheduling decisions as well.

In this paper, we present an integrated approach to planning and scheduling called completable scheduling. We will first give an overview of the main ideas behind completable planning, and then discuss the extension to scheduling. We will then discuss how completable schedules are learned through an explanation–based learning strategy called contingent EBL. Finally, we will briefly discuss the implementation, including some preliminary results and ongoing experiments.

COMPLETABLE SCHEDULING

Overview of Completable Planning

In completable planning [Gervasio90a, Gervasio90b, Gervasio91], a classical planner is augmented with a reactive component which provides it with the ability to defer planning decisions until execution time. As an augmented classical planning approach, completable planning retains the advantages of classical planning while buying into the advantages provided by reactivity. From classical planning, completable planning borrows the ability to construct provably–correct plans for providing goal–directed behavior. From reactive planning, it borrows planning flexibility and the ability to utilize runtime information in making planning decisions. Completable planning achieves the integration through the achievability criterion, which requires every deferred goal to be proven achievable. Proving achievability requires proving that there exists a plan which will achieve the goal. Our research has shown that proving the existence of a plan does not necessarily entail determining the plan itself, and the intuition is that proving achievability requires much simpler and more readily available a priori knowledge than full–blown planning. A completable planner is thus able to construct provably–correct plans in spite of incomplete a priori information, and in doing so provide goal–directedness to its reactive component while allowing itself to defer decisions and utilize runtime information in addressing deferred goals.

Deferring Decisions

The deferment of scheduling decisions is a powerful tool in dealing with imperfect a priori information. The complexity of real–world domains makes it difficult to construct perfect models. Even when perfect models exist, their use often exceeds reasonable computational bounds. A realistic scheduling system is thus left to contend with imperfect knowledge. There are four types of incompleteness which can result from using classical planning techniques on imperfect information.

First, a schedule may be incomplete due to an unspecifiable parameter setting. With the lack of a fine–grained and tractable world model, a scheduling system may not be able to determine precise parameter settings a priori. For example, the parameters of an operation may be dependent on the properties of a particular object, which may not be known prior to execution. In attaching two parts using a bolt, all that a system may know is that it will be given a bolt of some size, but it may not know precisely what size. However, provided it has access to differently–sized bits and wrenches, it can plan a drilling operation followed by a bolting operation without specifying the precise bit and wrench to use. During execution, when it is given the bolt, it can then determine the appropriate values for bit size and wrench size. Completable scheduling allows the use of conjectured variables to act as placeholders for unspecified parameter settings provided achievability proofs can be constructed for their eventual instantiation. By allowing deferred parameter settings, completable scheduling enables a system to both plan ahead and yet remain flexible enough to deal with some uncertainty.

Second, a schedule may be incomplete due to an undeterminable number of iterations. An imperfect world model may include incomplete characterizations of operations and their effects. Consequently, for an action that requires repetition to achieve some goal, the precise number of iterations needed may not be determinable prior to execution. For example, the depth to which a milling operation cuts through a part is dependent on the face cutter used. Prior to execution, a system may not know which face cutter will be set up on the machine. However, it knows that regardless of which face cutter is used, the desired cut can be achieved by simply repeating the milling operation as many times as necessary. By not tying itself to a particular face cutter and consequently a particular number of iterations, during execution the system can choose to use the current set–up and save the cost of changing set–ups, or it can elect to change to a more efficient set–up. Completable scheduling permits the deferment of iteration decisions provided incremental progress which converges to the goal can be proven. Through this deferment, a completable system can use imperfect operation descriptions as well as make optimizations to a schedule based on runtime information.

Third, a schedule may be incomplete due to an unidentifiable operation choice or task–machine assignment. This case arises when there are multiple ways of achieving the same goal from different states and the system lacks the necessary a priori information for identifying which particular state will be reached. This case also arises when there are multiple ways of achieving a goal, with different situations resulting in different preferences among the various alternatives, and the system does not know a priori which situation will be reached. For example, in planning to shape an object, a system might use some or all of various cutting operations, such as milling, planing, sawing, or grinding. Whether there are several possible states requiring different operations or multiple applicable operations with unknown preferences, a system can use additional runtime information to make a more–informed operation choice. Completable scheduling allows a system to defer operation choice provided it can prove that there exists a way to reach the next state regardless of which of the possible states is reached. This deferment is useful for two reasons. First, it enables a system to use the same schedule to achieve a goal from any of several different states. Second, it allows a system to apply preferences to a set of possible operations using more complete and accurate runtime information.

Fourth, a schedule may be incomplete due to an unorderable set of operations. Imperfect a priori information may result in insufficient constraints for completely ordering a set of operations. For example, in the construction of two parts, the only precedence constraints may be between the milling, drilling, and tapping operations for each part—i.e. the operations for the different parts can be ordered in any way. Depending upon a priori known factors such as the parts involved and the difficulty of changing set–ups as well as a priori unknown factors such as the initial set–up and machine availability, particular orderings will be more desirable than others. By deferring the decision until all the factors are known, a system can utilize runtime information to make decisions for more optimal orderings. Completable scheduling permits the deferment of ordering decisions provided the different orderings are all capable of achieving the goal. In doing so, a completable planner can utilize runtime information in making more–informed ordering decisions for an unconstrained set of actions.

Proving Achievability

While imperfect a priori information is the primary reason for deferring decisions, achievability is the primary criterion for deferment in completable scheduling. By requiring that a deferred goal be proven achievable, completable scheduling enables the construction of incomplete yet provably–correct plans. Previous work on achievability involved finding proofs for the existence of plans to achieve deferred goals. Achievability proofs for deferred parameter settings and number of iterations are discussed in [Gervasio90a, Gervasio90b], and for deferred operator choice in [Gervasio91]. In [Gervasio91], completable planning was also extended to probabilistic domains by relaxing the original criterion of absolute achievability to probable achievability.

Scheduling domains give rise to further new issues in achievability. In planning, the main focus is finding a plan, or sequence of actions, which achieves the goal from a given initial state. In scheduling, the existence of several possible schedules is taken as a given, and the focus is choosing one from among them using some set of preference criteria, maximizing particular performance measures. Examples of performance goals are meeting deadlines and minimizing idle time. Thus, simply defining a goal to be achievable if there exists a plan for it is insufficient for scheduling. Achievability must also be related to the idea of optimization and relative preferences between possible courses of action. For example, proving the achievability of the goal associated with an unordered set of actions is implicit in the construction of a nonlinear plan—i.e. actions are left unordered if there are no constraints requiring precedence relations between them. Thus there exists a plan for achieving the goal. However, there is the interesting issue of deciding on a complete ordering during execution. This involves seeking out additional information for evaluating the different options as well as carrying out the operations themselves. In tying the concept of achievability to optimization, we can also better investigate a primary motivation for combining classical and reactive techniques: the ability to utilize runtime information in planning. Goal–directed, robust behavior in the face of uncertainty is one reason for augmenting a classical planner with reactive abilities. However, another reason to integrate the two approaches, is to take advantage of the wealth of information which becomes available at runtime. This additional information facilitates planning by helping to focus the search for an appropriate action.

LEARNING COMPLETABLE SCHEDULES

Explanation–based learning [DeJong86, Mitchell86] has been demonstrated to be useful in improving the performance of various planning systems [Bennett90, Chien89, Fikes72, Hammond86, Minton85], and in [Gervasio90a, Gervasio91] we present an explanation–based learning strategy called contingent EBL for learning completable plans. Learning completable schedules basically involves learning to distinguish between a priori planning decisions and decisions which have to be made or are better made during execution. Learning when to defer decisions involves first identifying the deferred decision, then constructing an achievability proof for the associated deferred goal. Then a completor for making the deferred decision during execution must be incorporated into the learned general plan.

Identifying Deferred Decisions

A main difference between classical plans and completable plans is the existence of deferred decisions in completable plans. In constructing an explanation for how a given training example achieves a target goal, an EBL system must explain how each action is chosen for execution. In planning, this usually means verifying that previous actions achieve the preconditions necessary for the execution of an action. However, with the addition of reactive abilities and the option to utilize runtime information, a system needs to distinguish between a priori satisfied preconditions and runtime–verified preconditions. Our solution is to allow the system to distinguish between a priori information and runtime–gathered information and to prefer a classical proof of correctness to an explanation of achievability. Thus, in explaining how an action is chosen for execution, a system first attempts to explain its preconditions with a priori available information. If this is unsuccessful, then the action being explained is tagged as a potential deferred decision, and the system attempts to construct an achievability explanation for the precondition. Only if it is successful is the learning process allowed to continue. The final explanation will thus contain the identified deferred decisions as well as their supporting achievability explanations.

Tying the concept of achievability to optimization adds further concerns. An explanation of executability is no longer enough. Explanations for preferences may also need to be constructed, and as with other deferred decisions, the associated runtime verified conditions need to be distinguished from a priori satisfied conditions. As with proofs of correctness and explanations of achievability, explanations of preferences may also be constructed in standard EBL fashion.

Constructing Achievability Proofs

To construct provably–correct plans, a completable planner must construct achievability proofs for the deferred goals of

its incomplete plans. While the mechanics of constructing proofs of correctness vs. proofs of achievability are essentially the same—both use standard EBL on a given domain theory—there are some requirements needed for a domain theory to be used in proving achievability.

There are four types of deferred decisions and each requires particular kinds of information for proving achievability. First, deferred parameter settings must be represented, and this is done using conjectured variables. These variables may only be introduced in the context of the rules used to construct their corresponding achievability explanations, thus guaranteeing that every conjectured variable in an explanation has a supporting achievability proof. Second, a system must be able to reason about the incremental progress achieved by a repeated action. This requires action characterizations to include statements regarding the changes made with respect to some measurable quantity. This can then be used to reason about progress towards the goal. Third, the incompletely known situation requiring a deferred operator choice must be represented in such a way that the system can reason about the space of possibilities. Achievability can then be measured in terms of the coverage provided by the alternative actions over this space. Finally, proving achievability with respect to an unordered set of operations is implicit in the absence of precedence constraints between the operations, which means that any of the possible total orderings will achieve the goal.

The second aspect of achievability, optimality, also imposes certain requirements on the domain theory used to construct explanations. The heuristics to be used in making dispatching or scheduling decisions must be built in to the domain theory. These heuristics can then be used both for constructing a priori explanations and making runtime decisions. In explaining particular decisions made in a training example, a system can then construct explanations incorporating the heuristics and learn general completable plans which will employ the heuristics in future applications.

Incorporating Completors

The final step in learning how to construct a completable schedule is to incorporate completion steps into the learned general plan. There are four types of completors corresponding to the four different types of deferred decision. The first, a monitor, finds a value to replace a conjectured variable—i.e. it determines a specific parameter setting. The second, a repeat loop, repeatedly executes an action until a particular exit condition, the deferred goal, is reached. The third, a conditional, evaluates the current state and determines an appropriate action based on which conditions are satisfied. Finally, the fourth, a dispatcher, determines a complete ordering for an unordered set of operations, based on a given set of heuristics. The achievability proofs constructed for the deferred decisions addressed by these completors are incorporated into the explanations supporting the learned plan. Thus the achievability conditions guaranteeing the existence of a completion are also in the learned plan. Provided these conditions, along with other preconditions, are satisfied in future instances, a completion is guaranteed to be found for the incomplete plan yielded by the learned general plan.

IMPLEMENTATION

A simple scheduling domain theory has been constructed to compare the performance of a completable scheduling system with that of a purely classical scheduling system as well as a purely reactive scheduling system. The domain involves a single machine which can be set up in various ways, each set–up of which is capable of performing some set of tasks. The same task may take different processing times on different set–ups. Furthermore, there is a set–up cost involved in changing set–ups. A job consists of a partially ordered set of tasks, and a scheduling problem involves a set of independent jobs. Initially, the only ordering constraints between tasks are based on deadlines. However, additional precedence constraints may be imposed between the tasks of a job if the a priori planning module of the system determines that one task is needed to establish the preconditions for another task. Uncertainty enters into the picture through an unknown initial state—i.e. the system does not know a priori which set–up will be on the machine when it starts executing its plan. Finally, the goodness of a schedule is measured by the length of time taken by the system to finish a set of jobs.

Preliminary results show that a completable system's ability to adapt to varying initial states enables it to construct more efficient plans/schedules than a classical scheduling, which commits itself to specific set–ups and complete task orderings prior to execution. Furthermore, the completable system needs less time both to learn a general completable schedule as well as construct a specific completable schedule, although it does incur the additional cost of runtime plan completion. The completable system is also able to construct more efficient plans than a reactive system because it is more focused in its search for an applicable action, having determined as many precedence constraints between tasks as it can prior to execution. Although both use the same heuristics for choosing between multiple applicable actions, the reactive system has the additional burden of sorting out precedence relations between tasks during execution. Furthermore, although the completable system initially needs to construct a completable schedule, the use of learning helps reduce the a priori planning cost it incurs over the reactive scheduler. We are currently running experiments to gather more data about the performance of the three approaches given different distributions and different machine/set–up/task–processing profiles. The results are expected to help identify particular domain and problem characteristics which favor the different approaches.

SUMMARY AND CONCLUSIONS

This work integrates planning—the determination of an ordered set of tasks—and scheduling—the assignment of those tasks to resource—through completable plans. Because completable plans are incomplete, additional planning is necessary during execution, when scheduling has begun to dispatch the tasks. Thus, this work differs from reactive approaches, such as those discussed in [Ow88, Prosser89, Smith90, Zweben90], where planning is separated from scheduling, and the main approach to uncertainty in the environment is to replan when the constraints of the original plan are violated. While replanning is a valuable tool which any real system will even-

tually need, our work first focuses on constructing plans which are as flexible as possible to minimize the need for failure recovery. In this sense, it is similar to ideas presented in [Drummond90, Martin90]. Drummond and Bresina present an algorithm for maximizing the probability of goal satisfaction in the case of actions with different possible outcomes, which is one of the problems the conditionals in completable scheduling address. Martin and Allen also prove the achievability of goals deferred to the reactive planner, but they do so using empirical methods, in contrast to the explanation–based methods we use. Completable scheduling may also be viewed as a shallow hierarchical planner, where runtime decisions are at the lowest level. However, unlike other hierarchical planners and schedulers, such as ABSTRIPS [Sacerdoti74], MOLGEN [Stefik81], and ISIS [Fox84], a completable scheduling system uses the achievability constraint to guarantee completability at lower levels. The ordered monotonic hierarchies of ALPINE [Knoblock90] are a similar idea. The difference is that ALPINE performs abstraction based on the deletion of literals, while in proving achievability completable scheduling uses explicitly more general or abstract knowledge regarding the deferred goals and their properties.

The idea of deferred decisions is not a novel one—the least commitment principle is a basic foundation of nonlinear planning, for example. What completable scheduling does is extend the least commitment principle to execution time and in doing so, achieving a well–founded integration of planning and scheduling. Unlike other reactive approaches, in which all decisions are subject to deferment, in completable scheduling only achievable decisions may be deferred. This has two main benefits. The first is that the cost of dynamic decision–making is minimized, since only some goals must be planned for and scheduled during execution. The second is that the robustness and flexibility afforded by reactivity is gained without losing the goal–directedness and guarantees of success afforded by a priori planning. Additionally, the use of contingent EBL enables a completable scheduling system to improve its performance through experience. By learning general completable schedules from example, the system can amortize the cost of constructing a completable schedule over the number of times the learned general schedule is applied in future instances as well as reduce the planning cost incurred by the system's a priori planning module.

Acknowledgments. This research was supported by the Office of Naval Research under grants N–00014–86–K–0309 and N–00014–91–J–1563. We would also like to thank Scott Bennett, Steve Chien, Jon Gratch, and Dan Oblinger for many interesting discussions, as well as Michael Shaw, for introducing us to the domain of process planning and scheduling.

References

[Agre87] P. Agre and D. Chapman, "Pengi: An Implementation of a Theory of Activity," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 268–272.

[Bennett90] S. Bennett, "Reducing Real–world Failures of Approximate Explanation–based Rules," *Proceedings of the Seventh International Conference on Machine Learning*, Austin, TX, 1990, pp. 226–234.

[Chien89] S. A. Chien, "Using and Refining Simplifications: Explanation–based Learning of Plans in Intractable Domains," *Proceedings of The Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 590–595.

[DeJong86] G. F. DeJong and R. J. Mooney, "Explanation–Based Learning: An Alternative View," *Machine Learning 1*, 2 (April 1986), pp. 145–176. (Also appears as Technical Report UILU–ENG–86–2208, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana–Champaign.)

[Drummond90] M. Drummond and J. Bresina, "Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction," *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, August 1990, pp. 138–144.

[Fikes72] R. E. Fikes, P. E. Hart and N. J. Nilsson, "Learning and Executing Generalized Robot Plans," *Artificial Intelligence 3*, 4 (1972), pp. 251–288.

[Firby87] R. J. Firby, "An Investigation into Reactive Planning in Complex Domains," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 202–206.

[Fox84] M. S. Fox and S. F. Smith, "ISIS—a knowledge–based system for factory scheduling," *Expert Systems 1*, 1 (July 1984),.

[Gervasio90a] M. T. Gervasio, "Learning General Completable Reactive Plans," *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, August 1990, pp. 1016–1021.

[Gervasio90b] M. T. Gervasio, "Learning Completable Reactive Plans Through Achievability Proofs," Technical Report UIUCDCS–R–90–1605, Department of Computer Science, University of Illinois, Urbana, IL, May 1990.

[Gervasio91] M. T. Gervasio and G. F. DeJong, "Learning Probably Completable Plans," Technical Report UIUCDCS–R–91–1686, Department of Computer Science, University of Illinois, Urbana, IL, April 1991.

[Hammond86] K. Hammond, "CHEF: A Model of Case–Based Planning," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 267–271.

[Kaelbling88] L. P. Kaelbling, "Goals as Parallel Program Specifications," *Proceedings of The Seventh National Conference on Artificial Intelligence*, Saint Paul, MN, August 1988, pp. 60–65.

[Knoblock90] C. Knoblock, "Learning Abstraction Hierarchies for Problem Solving," *Proceedings of the Eight National Conference on Artificial Intelligence*, Boston, MA, 1990, pp. 923–928.

[Martin90] N. G. Martin and J. F. Allen, "Combining Reactive and Strategic Planning through Decomposition Abstraction," *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, San Diego, CA, November 1990, pp. 137–143.

[Minton85] S. Minton, "Selectively Generalizing Plans for Problem–Solving," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, August 1985, pp. 596–599.

[Mitchell86] T. M. Mitchell, R. Keller and S. Kedar–Cabelli, "Explanation–Based Generalization: A Unifying View," *Machine Learning 1*, 1 (January 1986), pp. 47–80.

[Muscettola90] N. Muscettola and S. F. Smith, "Integrating Planning and Scheduling To Solve Space Mission Scheduling Problems," *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, San Diego, CA, November 1990, pp. 220–230.

[Ow88] P. S. Ow, S. Smith and A. Thiriez, "Reactive Plan Revision," *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, MN, August 1988, pp. 77–82.

[Prosser89] P. Prosser, "A Reactive Scheduling Agent," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 1004–1009.

[Sacerdoti74] E. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence 5*, (1974), pp. 115–135.

[Smith90] S. F. Smith, P. S. Ow, N. Muscettola, J. Potvin and D. C. Matthys, "OPIS: An Integrated Framework for Generating and Revising Factory Schedules," *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, San Diego, CA, November 1990, pp. 497–507.

[Stefik81] M. Stefik, "Planning and Metaplanning (MOLGEN: Part 2)," *Artificial Intelligence 16*, 2 (1981), pp. 141–170.

[Zweben90] M. Zweben, M. Deal and R. Gargan, "Anytime Rescheduling," *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, San Diego, CA, November 1990, pp. 251–259.

IOPS Advisor: Research in Progress on Knowledge-Intensive Methods for Irregular Operations Airline Scheduling*

John E. Borse and Christopher C. Owens
The University of Chicago
Artificial Intelligence Laboratory, Department of Computer Science
1100 East 58th Street, Chicago, IL 60637
borse@cs.uchicago.edu

Abstract

Our research focuses on the problem of recovering from perturbations in large-scale schedules, specifically on the ability of a human-machine partnership to dynamically modify an airline schedule in response to unanticipated disruptions. This task is characterized by massive interdependencies and a large space of possible actions. Our approach is to apply both qualitative, knowledge-intensive techniques relying on a memory of stereotypical failures and appropriate recoveries, and quantitative techniques drawn from the Operations Research community's work on scheduling. Our main scientific challenge is to represent schedules, failures and repairs so as to make both sets of techniques applicable to the same data.

This paper outlines ongoing research in which we are cooperating with United Airlines to develop our understanding of the scientific issues underlying the practicalities of dynamic, real-time schedule repair.

Irregular Operations Scheduling (IOPS)

Airline schedules are highly complex, structured objects, with large numbers of internal interdependencies. Airlines must confront the consequences of uncertainty in the execution of their daily schedules — uncertainty stemming from inclement weather, sick calls from crew members, mechanical problems with aircraft, constraints on airport resources, and other problems. A snowstorm at a key airport, for example, can have devastating consequences on the operations of an airline, effects from which it may take days to recover. The interdependencies among factors like crew scheduling, maintenance routing, and congestion at airports add further complication to the daily planning problem. Because of these interdependencies, even a single disruption and the consequent attempts at recovery typically involve widespread and long-lasting downstream effects. The search space of possible recoveries to a schedule disruption is enormous.

Airlines employ schedule planners who attempt to mitigate the effects of schedule disruptions. Their main goals are to minimize both passenger inconvenience and the cost of implementing the repair, while accounting for crew work rules, aircraft maintenance schedules, and other factors. An additional goal is to minimize the overall complexity of a repair.

Controllers attempt to balance the trade-offs and uncertainties of irregular events, typically using information provided by various decision support systems such as real-time scheduling displays and passenger booking data. However, very few, if any, of these systems provide the planner with decision-making advice in the form of strategies or specific recommendations to counteract the adversity of a particular event. The goal of our research is to develop the scientific foundations for a new class of decision support tool to address this problem.

From the viewpoint of Artificial Intelligence planning and decision support, the key features of the irregular operations planning task are:

- Airline schedules are large, complex, and highly interdependent.
- Solving schedule problems by exhaustive search is generally infeasible.
- Current situations typically share more with past situations than they differ from them.

*This work is supported in part by the Air Force office of Scientific Research under contract AFOSR-91-0112, and in part by the Defense Advanced Research Projects Agency and Rome Laboratory under contract F30602-91-C-0028. The authors gratefully acknowledge the assistance of United Airlines in providing data and observer access to live operations. Nothing in this paper represents any policy, position, or opinion of United Airlines.

- While they may be similar, no two situations are ever entirely identical. This means that simply storing and reusing a "library" of solutions will not suffice.

The size of the search space, together with the recurring nature of typical problems, suggests a solution based on the re-use of plans. But re-using plans means more than just retrieving and replaying old solutions. Because the details of situations change over time, the system will need to be able to notice that a retrieved plan does not exactly fit the current situation, therefore it will need to modify its retrieved plans to fit new situations.

Our approach to plan repair is to provide qualitative expertise in the form of a case library linking descriptions of stereotypical problems with appropriate recovery strategies, and quantitative expertise in the form of optimization techniques drawn from the Operations Research (OR) community. The goal of our research is to develop the scientific foundations for a new class of decision support tool. The IOPS Advisor, currently under development, couples the experiential knowledge of schedulers, which is essential in generating strategies for solving a schedule problem, with the quantitative power of operations research techniques, which are effective in comparing the costs and effectiveness of the potential solutions generated by those strategies. Furthermore, the quantitative models may be responsible for optimizing the details missing from a sketchy solution suggested by a qualitative strategy. For example, if a strategy is "stop to refuel", a quantitative analysis may indicate where to stop and how much fuel to take on.

The IOPS Advisor, currently under development, is intended to represent schedules, failures, and repairs so that both sets of techniques can cooperate using the same representational constructs.

Research Objectives

The primary scientific focus of this work is on representation. Specifically, we are determining how to represent schedules, schedule failures, and repair strategies so as to enable the IOPS advisor to:

- Identify and characterize schedule problems so as to determine the applicability of prior solutions or specific quantitative techniques.
- Acquire new descriptive features as they become necessary to discriminate among otherwise indistinguishable situations.
- Compare the applicability of multiple, competing solutions to the same problem.

Knowledge Representation Issues:

The main knowledge representation issue, and the primary focus of our current activity, is to categorize and represent the heuristic knowledge used by controllers and OR analysts, specifically:

- How problems are detected and described.
- What problem-solving strategies exist.
- What aspects of a problem indicate the applicability of one strategy over another.

In order to gather a realistic set of failures and repairs, we have been observing controllers as they detect, diagnose, and repair schedule problems. Our initial study has suggested to us that controllers build and use sophisticated, high-level repairs from a small number of primitive operators. The primitives form the basic representation vocabulary used to describe actions, and it is anticipated that the list will be stable over time. The higher-level strategies, on the other hand, are more dynamic, and one of our tasks is to model the acquisition of new high-level strategies.

Typical primitive operators represent concrete actions like:

- Cancel a segment
- Delay a segment
- Divert a flight to a different airport
- Substitute one aircraft for another
- Substitute one crew for another
- Ferry an empty aircraft from one airport to another

Higher-level strategies, on the other hand, may involve both primary actions and secondary actions designed to mitigate the side-effects of the primary actions. Or, they might involve a series of steps taken to defer the impact of a problem, in the expectation that an opportunistic solution may present itself in the intervening time. Other high-level strategies include geographically localizing the impact of a problem or, conversely, diluting the impact of a problem by spreading a minor delay across several geographic points.

As we gather more high-level strategies from our observation of controllers and from our encoding of quantitative techniques, our plan is to encapsulate the strategies in knowledge structures that also include descriptions of appropriate situations for the strategies. The IOPS advisor will extract from the user a description of the current situation, propose repair strategies based upon the match between the current situation and the stored descriptions, and quantitatively evaluate the utility of situations generated by competing strategies. As it performs this selection and comparison, it can acquire, from the user, information about features of the world that determine the applicability of one strategy over another. These newly-acquired features can then become part of the selection criteria encoded with the strategies in memory.

Knowledge acquisition

While the list of primitives is expected to remain relatively static, an important aspect of the IOPS Advisor

is that it will be able to acquire new descriptive features as it is used. If the system erroneously suggests a prior case as being a good match to the current situation, the user can correct this by supplying a descriptive feature that would differentiate the current situation from the case stored in memory. The error might have occurred either because the discriminating feature was not mentioned in the description of the current situation, or because it was not mentioned in the stored case. In the latter scenario, it can be added.

In general, a longer-range goal for the IOPS advisor is that, in having a human user interact with a planning tool, we have an opportunity to record information about plan accessing strategies, modification techniques and typical failures that can, in turn, become the heuristics used by a more autonomous system. A system that observed human schedulers in action and recorded their responses to specific planning problems, and which indexed those responses in memory using the functional criteria discussed above, would become a powerful expert assistant — an assistant with a good memory for what worked and what didn't in the past.

Case-Based Planning Issues

While case-based planning addresses many of the qualitative problems in the irregular scheduling domain, much work must be done before a practical system could be put in the hands of a human scheduler. Fortunately, the core idea in case-based planning, that of incremental modification, is one aspect of the technology that could be usefully applied in the near term as a way to deal with the type of changes that have to be made to schedules during execution.

One of the recurring problems of automated planning is the issue of the repairs that have to be made during execution as a result of unforeseen circumstances. There are always unexpected problems that arise. Weather, crew sickness, and equipment failures cannot be predicted. Bottlenecks show up where none was suspected. Each of these classes of problems can be recognized using a specific set of symptoms, and each requires a specific type of repair.

Run-time repair and optimization, while useful, has to be traded-off against the overall stability of an existing plan. If a single aircraft is unexpectedly grounded, one form of optimization might be to rebuild the entire system schedule, minus that aircraft. But even if such a repair were computationally feasible, implementing it would be preposterous. A planner that deals with unexpected changes in the state of the world by completely replanning will be constantly creating new plans that will do little more than confuse the people that are using them. What is needed instead is incremental, local plan repair, coupled with local optimization. One wants to perturb the schedule as little as possible in the achievement of an acceptable response to an unexpected occurrence.

Much of the emphasis of CBR research to date has been on issues of plan indexing, retrieval and modification. While these issues are clearly present in this domain, our emphasis is primarily on plan evaluation through objective analytical (OR) tools which are also under development. Specifically, we are focusing on how to direct the search for relevant cases based on the OR model's assessment of the feasibility or "utility" of previously proposed solutions. Because the two sets of techniques tend to characterize the problems differently, integrating them is a challenge.

Operations Research Issues

Operations research analysts tend to think in terms of opportunities for optimization. One of our preliminary findings is that schedule planners do not readily identify these opportunities. Accordingly, an important aspect of the integrative research is to identify classes of situations in which particular optimization techniques are appropriate, and to select descriptive features that allow the system or planners to differentiate among these classes. We intend to codify this knowledge in the form of cases which couple the relevant optimization techniques with characteristic features of the appropriate class of situation.

Case Study

The following hypothetical case study is based on observations of airline planners. The case illustrates the interplay between qualitative and quantitative reasoning described in this paper. Airports are designated by the following three letter codes: SFO = San Francisco, EUG = Eugene, and MED = Medford.

> A runway construction project at EUG has imposed a weight restriction on departing flights. A departing flight EUG-SFO is over the weight limitation by approximately 20 passengers. The flight is scheduled to depart on time, however, inbound flow control is in effect at SFO (due to fog) and is imposing a 53 minute pre-takeoff delay on the EUG-SFO flight.

The planner generates some alternative solutions:

1. Move the excess passengers to a later EUG-SFO flight.
2. Have a flight enroute to SFO passing nearby EUG stop to pick up the excess passengers.
3. Remove enough fuel to carry the excess passengers, and stop at an intermediate point to refuel.

At this stage, the alternatives are qualitative: they simply match a problem with a strategy. Although in many cases this step of the solution process is trivial (e.g., weather-related IOP forces cancellations), we believe that in general this step is non-trivial and it is one aspect of the planner's job which distinguishes an experienced planner from an inexperienced one.

The next step of the planning process involves evaluating the relative merits of each proposed strategy with

respect to the planner's goals. In this case the planner chose not to solve the problem using strategy (1) because pushing the problem to a later flight would most likely cause weight restriction problems downline and would disservice the excess passengers. Strategy (2) was not chosen since it would involve delaying a large number of passengers on a different flight to accommodate a relatively small number of connecting passengers on the EUG-SFO flight. On further analysis of strategy (3), the controller determined that, since SFO air traffic control had already imposed a 53-minute delay on the inbound flight for reasons of airspace crowding, the flight could in fact refuel at MED and carry all passengers to SFO as planned without incurring additional delays. The cost of landing and departing at MED was considered negligible in comparison to the alternative costs of delaying passengers and causing misconnections of aircraft and people (although this calculation was not performed explicitly).

Notice that the planner's analysis in choosing among alternatives remains highly qualitative. The planner uses various sources of information to determine the viability of each approach, however, he rarely explicitly calculates the cost impact of various strategies. We believe that at this stage the planner could be greatly aided by OR models which:

- provide an objective analysis of the relative merits of each strategy based on utility measures.
- determine optimal implementations of high-level strategies, for example, given strategy (2), choosing an appropriate flight, or, given strategy (3), choosing an appropriate airport.

Anticipated Results

Our key preliminary result is a growing catalogue of stereotypical problems and appropriate repair strategies, which form the backbone of a domain theory of schedule failure repair. We anticipate that a longer-term result of our research will be a working prototype of the IOPS Advisor System. This prototype will embody the failure descriptions and recovery strategies, as well as a set of features characterizing appropriate situations in which to apply specific quantitative optimization tools. The knowledge-based system will suggest strategies, given a description of the problem, while the OR components will be responsible for evaluating the costs and benefits of the proposed strategies and for determining specific implementations of the strategies.

Evaluation

The bases against which we can evaluate the IOPS advisor project are:

- Does the system enable a controller to produce good schedule repairs? In particular, can a controller use the system's prepackaged strategies and OR evaluation methods to improve upon solutions produced using the controller's own judgment?
- How good are the high-level strategies that the experienced planners employ? How often do controllers choose the best strategy? While the strategies obviously work, are they applied inappropriately? Does post-facto analysis repeatedly indicate that some other strategy might have been preferable?
- Are individuals able to make use of the canned strategies? Can one individual recognize and re-use canned strategies? Is there any transfer across individuals, such that one individual can use strategies developed by another? If so, how should the strategies be presented to the user?
- Can novices use the strategies and optimizations from the IOPS advisor to generate expert-like repairs? In general, how do solutions built by novices differ from solutions built by experts? Does the availability of a library of expert solutions improve a novice's performance?
- Does the integrative AI/OR approach provide a better method than either technique applied alone? Is it even possible to model the IOPS problem using either technique alone? What form would these models take (e.g. large scale linear programming, expert-system)? How would each of these approaches compare to the integrative approach?

Summary

The airline irregular operation problem is representative of a general class of scheduling problems. An ideal solution would embody both the best quantitative techniques and the genuine expertise of skilled, experienced controllers. Traditionally, the two classes of solution have been described in such divergent terms as to make integration, or even comparison, difficult. By building a uniform representation of schedules, failures and repairs, our intention is to provide a framework for experimenting with qualitative and quantitative solutions and, ultimately, for integrating the two.

An Extended Abstract: A Heuristic Repair Method for Constraint-Satisfaction and Scheduling Problems

Steven Minton[1] and Mark D. Johnston[2] and Andrew B. Philips[1] and Philip Laird[3]

[1]Sterling Federal Systems
NASA Ames Research Center
AI Research Branch
Mail Stop: 269-2
Moffett Field, CA 94035 USA

[2]Space Telescope Science Institute
3700 San Martin Drive,
Baltimore, MD 21218 USA

[3]NASA Ames Research Center
AI Research Branch
Mail Stop: 269-2
Moffett Field, CA 94035 USA

Introduction

One of the most promising general approaches for solving combinatorial search problems is to generate an initial, suboptimal solution and then to apply local *repair* heuristics. Techniques based on this approach have met with empirical success on many combinatorial problems, including the traveling salesman and graph partitioning problems[10]. Such techniques also have a long tradition in AI, most notably in problem-solving systems that operate by debugging initial solutions [18, 20]. In this paper, we describe how this idea can be extended to constraint satisfaction problems (CSPs) in a natural manner (see also [14] for full paper).

Most of the previous work on CSP algorithms has assumed a standard backtracking approach in which a partial assignment to the variables is incrementally extended. In contrast, our method starts with a complete, but inconsistent assignment and then incrementally repairs constraint violations until a consistent assignment is achieved. The method is guided by a simple ordering heuristic for repairing constraint violations: identify a variable that is currently in conflict and select a new value that minimizes the number of outstanding constraint violations.

We present empirical evidence showing that on some standard problems our approach is considerably more efficient than traditional backtracking methods. For example, on the n-queens problem, our method quickly finds solutions to the one million queens problem[15]. We argue that the reason that repair-based methods can outperform backtracking methods is because a complete assignment can be more informative in guiding search than a partial assignment. However, the utility of the extra information is domain dependent.

The work described in this paper was inspired by a surprisingly effective neural network developed by Adorf and Johnston [1] for scheduling astronomical observations on the Hubble Space Telescope. Our heuristic CSP method was distilled from an analysis of the network. In the process of carrying out the analysis, we discovered that the effectiveness of the network has little to do with its connectionist implementation. Furthermore, the ideas employed in the network can be implemented very efficiently within a symbolic CSP framework. The symbolic implementation is extremely simple. It also has the advantage that several different search strategies can be employed, although we have found that hill-climbing methods are particularly well-suited for the applications that we have investigated.

We begin the paper with a brief review of Adorf and Johnston's neural network. Following this, we describe our symbolic method for heuristic repair.

Previous Work: The GDS Network

By almost any measure, the Hubble Space Telescope scheduling problem is a complex task [11, 17]. Between ten thousand and thirty thousand astronomical observations per year must be scheduled, subject to a great variety of constraints including power restrictions, observation priorities, time-dependent orbital characteristics, movement of astronomical bodies, stray light sources, etc. Because the telescope is an extremely valuable resource with a limited lifetime, efficient scheduling is a critical concern. An initial scheduling system, developed using traditional programming methods, highlighted the difficulty of the problem; it was estimated that it would take over three weeks for the system to schedule one week of observations. This problem was remedied by the development of a successful constraint-based system to augment the initial system. At the heart of the constraint-based system is a neural network developed by Adorf and Johnston, the Guarded Discrete Stochastic (GDS) network, which searches for a schedule[1].

From a computational point of view, the network is interesting because Adorf and Johnston found that it performs well on a variety of tasks, in addition to the space telescope scheduling problem. For example, the network performed significantly better on the n-queens problem than methods that had been previously developed. The n-queens problem requires placing n queens on an $n \times n$ chessboard so that no two queens share a row, column or diagonal. The network has been used to solve problems of up to 1024 queens, whereas most heuristic backtracking methods encounter difficulties

with problems one-tenth that size[19].

The GDS network is a modified Hopfield network[8]. In a standard Hopfield network, all connections between neurons are symmetric. In the GDS network, the main network is coupled asymmetrically to an auxiliary network of *guard neurons* which restricts the configurations that the network can assume. This modification enables the network to rapidly find a solution for many problems, even when it is simulated on a serial machine. Unfortunately, convergence to a stable configuration is no longer guaranteed. Thus the network can fall into a local minimum involving a group of unstable states among which it will oscillate. In practice, however, if the network fails to converge after some number of neuron state transitions, it can simply be stopped and started over.

To solve the n-queens problem with the GDS network, each of the $n \times n$ board positions is represented by a neuron whose output is either one or zero depending on whether or not a queen is located in that position. (Note that this is a local representation rather than a distributed representation of the board.) If two board positions are inconsistent, then an inhibiting connection exists between the corresponding two neurons. For example, all the neurons in a column will inhibit each other, representing the constraint that two queens cannot be in the same column. For each row, a guard neuron is connected to each of the neurons in the row and gives the neurons in that row a large excitatory input, large enough so that at least one neuron in the row will turn on. Thus, the guard neurons enforce the constraint that one queen in each row must be on. The network is updated on each cycle by randomly picking a row and flipping the state of the neuron in that row whose input is most inconsistent with its current output. A solution is realized when the output of every neuron is consistent with its input.

Why does the GDS Net Perform So Well?

Our analysis of the GDS network was motivated by the question: "Why does the network perform so much better than traditional backtracking methods on certain tasks?" In particular, we were intrigued by the results on the n-queens problem, since this problem has received considerable attention from previous researchers. For n-queens, Adorf and Johnston found empirically that the network requires a linear number of transitions to converge. Since each transition requires linear time, the expected (empirical) time for the network to find a solution is $O(n^2)$. To check this behavior, Johnston and Adorf ran experiments with n as high as 1024, at which point memory limitations became a problem.[1]

[1]The network, which is programmed in Lisp, requires approximately 11 minutes to solve the 1024 queens problem on a TI Explorer II. For larger problems, memory becomes a limiting factor because the the network requires approximately $O(n^2)$ space.

Nonsystematic Search Hypothesis

Initially, we hypothesized that it was the nonsystematic nature of the network's search that allowed it to perform much better than systematic depth-first backtracking search. There are two potential problems associated with systematic depth-first search. First, the search space may be organized in such a way that poorer choices are explored first at each branch point. For instance, in the n-queens problem, depth-first search tends to find a solution much more quickly when the first queen is placed in the center of the first row rather than the corner. It would appear that solution density is much greater in the former case[19], but most naive algorithms tend to start in the corner simply because humans find it more natural to program that way. However, the fact that a systematic algorithm may consistantly make poor choices does not completely explain why the GDS network performs so well for n-queens. A backtracking program that randomly orders rows (and columns within rows) performs much better than the naive method, and yet still performs poorly relative to the GDS network.

The second potential problem with depth-first search is more significant and more subtle. Depth-first search can be a disadvantage when solutions are not evenly distributed throughout the search space. As the distribution of solutions becomes less uniform, and, therefore, the solutions become more clustered, the time to search between solution clusters increases. Thus, we conclude that, in a tree where the solutions are clustered, depth-first search performs relatively poorly. In comparison, a search strategy which examines the leaves of the tree in random order is not affected by solution clustering.

We investigated whether this phenomenon explained the relatively poor performance of depth-first search on n-queens by experimenting with a randomized search algorithm, called a Las Vegas algorithm [2]. The algorithm begins by selecting a path from the root to a leaf. To select a path, the algorithm starts at the root node and chooses one of its children with equal probability. This process continues recursively until a leaf is encountered. If the leaf is a solution the algorithm terminates, if not, it starts over again at the root and selects a path. The same path may be examined more than once, since no memory is maintained between successive trials.

The Las Vegas algorithm does, in fact, perform better than simple depth-first search on n-queens. In fact, this result was already known [2]. However, the performance of the Las Vegas algorithm is still not nearly as good as that of the GDS network, and so we concluded that the systematicity hypothesis alone cannot explain the network's behavior.

Informedness Hypothesis

Our second hypothesis was that the network's search process uses information about the current assignment

that is not available to a standard backtracking program. We now believe this hypothesis is correct, in that it explains why the network works so well. In particular, the key to the network's performance appears to be that state transitions are made so as to reduce the number of outstanding inconsistencies in the network; specifically, each state transition involves flipping the neuron whose output is most inconsistent with its current input. From a constraint satisfaction perspective, it is as if the network reassigns a value for a variable by choosing the value that violates the fewest constraints. This idea is captured by the following heuristic:

Min-Conflicts heuristic:
Given: A set of variables, a set of binary constraints, and an assignment specifying a value for each variable. Two variables *conflict* if their values violate a constraint.
Procedure: Select a variable that is in conflict, and assign it a value that minimizes the number of conflicts.[2] (Break ties randomly.)

We have found that the network's behavior can be approximated by a symbolic system that uses the min-conflicts heuristic for hill-climbing. The hill-climbing system starts with an initial assignment generated in a preprocessing phase.[3] At each choice point, the heuristic chooses a variable that is currently in conflict and reassigns its value, until a solution is found. The system thus searches the space of possible assignments, favoring assignments with fewer total conflicts. Of course, the hill-climbing system can become "stuck" in a local maximum, in the same way that the network may become "stuck" in a local minimum.

There are two aspects of the min-conflicts hill-climbing method that distinguish it from standard backtracking approaches for CSP problems. First, instead of extending a consistent partial assignment, the min-conflicts method *repairs* a complete but inconsistent assignment by reducing those inconsistencies. Thus, to guide its search, it uses information about the current assignment that is not available to a standard backtracking algorithm. Second, the use of a hill-climbing strategy produces a different style of search.

We have also found that extracting the method from the network enables us to tease apart and experiment with its different components. In particular, the idea of repairing an inconsistent assignment can be used with a variety of different search strategies in addition to hill-climbing.

[2] In general, the heuristic attempts to minimize the number of other variables that will need to be repaired. For binary CSPs, this corresponds to minimizing the number of conflicting variables. For general CSPs, where a single constraint may involve several variables, the exact method of counting the number of variables that will need to be repaired depends on the particular constraint. The space telescope scheduling problem is a general CSP, whereas the other tasks described in this paper are binary CSPs.

[3] See [14] for an analysis of how different initial assignments can affect the repair phase.

Highlights of Experimental Results

This section contains highlights from experiments in which we evaluate the performance of the min-conflicts heuristic on some standard tasks. These experiments identify problems on which min-conflicts performs well, as well as problems on which it performs poorly. The experiments also show the extent to which the min-conflicts approach approximates the behavior of the GDS network.

The N-Queens Problem

- Min-conflicts hill-climbing approximates the GDS network for n-queens.
- For $n \geq 100$ min-conflicts hill-climbing has never failed to find a solution.
- For min-conficts, the required number of repairs appears to remain *constant* as n increases, and the time to find a solution grows linearly with n.
- Standard backtracking using the "most-constrained first" heuristic quickly grows large: for 100 runs when $n > 1000$ a backtracking program implementing the heuristic took more than 12 hours to complete.
- Min-conflicts hill-climbing solves the million queens problem in less than four minutes on a SPARCstation I.
- N-queens is actually quite an easy problem given the right method.

Scheduling Applications: HST

- Min-conflicts hill-climbing approximates the GDS network for HST scheduling.
- Much of the overhead (particularly the space overhead) in the GDS network is eliminated by using the min-conflicts method.
- Because the min-conflicts heuristic is so simple, a min-conflicts scheduler for HST was quickly coded in C and is extremely efficient.
- The simplicity of the min-conflicts method makes it easy to experiment with modifications to the heuristic and the search-strategy.
- Other telescope scheduling problems have started to use the min-conflicts scheduler developed for HST.

Graph Coloring

- Min-conflicts hill-climbing approximates the GDS network for graph coloring.
- A standard backtracking algorithm employing a Brelaz-like[3] heuristic outperforms min-conflicts hill-climbing.

Summary of Experimental Results

For each of the three tasks we have examined in detail, n-queens, HST scheduling and graph 3-colorability, we have found that the GDS network's behavior can be

approximated by the min-conflicts hill-climbing algorithm. To this extent, we have a theory that explains the network's behavior. Obviously, there are certain practical advantages to having "extracted" this method from the network. First, the method is very simple, and so can be programmed extremely efficiently, especially if done in a task-specific manner. Second, the heuristic we have identified, that is, choosing the repair which minimizes the number of conflicts, is very general. It can be used in combination with different search strategies and task-specific heuristics, an important factor for most practical applications.

Insofar as the power of our approach is concerned, our experimental results are encouraging. We have identified two tasks, n-queens and HST scheduling, which appear more amenable to our repair-based approach than a traditional approach that incrementally extends a partial assignment. This is not to say that a repair-based approach will do better than *any* traditional approach for solving these tasks, but merely that our simple, repair-based method has done relatively well in comparison to the standard traditional methods. We also note that repair-based methods have a special advantage for scheduling tasks: they can easily be used for both overconstrained and rescheduling problems. Thus it seems likely that there are other applications for which our approach will prove useful.

Discussion

The heuristic method described in this paper can be characterized as a *local search* method[10], in that each repair minimizes the number of conflicts for an individual variable. Local search methods have been applied to a variety of important problems, often with impressive results. For example, the Kernighan-Lin method, perhaps the most successful algorithm for solving graph-partitioning problems, repeatedly improves a partitioning by swapping the two vertices that yield the greatest cost differential. The much-publicized simulated annealing method can also be characterized as a form of local search[9]. However, it is well-known that the effectiveness of local search methods depends greatly on the particular task.

In fact, it is easy to imagine problems on which the min-conflicts heuristic will fail. The heuristic is poorly suited for problems with a few highly critical constraints and a large number of less important constraints. For example, consider the problem of constructing a four-year course schedule for a university student. We may have an initial schedule which satisfies almost all of the constraints, except that a course scheduled for the first year is not actually offered that year. If this course is a prerequisite for subsequent courses, then many significant changes to the schedule may be required before it is fixed. In general, if repairing a constraint violation requires completely revising the current assignment, then the min-conflicts heuristic will offer little guidance.

The problems investigated in this paper, especially the HST and n-queens problem, tend to be relatively uniform in that critical constraints rarely occur. In part, this is due to the way the problems are represented. For example, in the HST problem, as described earlier, the transitive closure of temporal constraints is explicitly represented. Thus, a single "after" relation can be transformed into a set of "after" relations. This improves performance because the min-conflicts heuristic is less likely to violate a set of constraints than a single constraint. In some cases, we expect that more sophisticated techniques will be necessary to identify critical constraints[5]. To this end, we are currently evaluating explanation-based learning techniques [4, 13] as a method for identifying critical constraints.

The algorithms described in this paper also have an important relation to previous work in AI. In particular, there is a long history of AI programs that use repair or debugging strategies to solve problems, primarily in the areas of planning and design[18, 20]. This approach has recently had a renaissance with the emergence of case-based[6] and analogical [12, 21] problem solving. To solve a problem, a case-based system will retrieve the solution from a previous, similar problem and repair the old solution so that it solves the new problem.

The fact that the min-conflicts approach performs well on n-queens, a well-studied, "standard" constraint-satisfaction problem, suggests that AI repair-based approaches may be more generally useful than previously thought. However, in some cases it can be more time-consuming to repair a solution than to construct a new one from scratch.

There are many possible extensions to the work reported here, but three are particularly worth mentioning. First, we expect that there are other applications for which the min-conflicts approach will prove useful. Conjunctive matching, for example, is an area where preliminary results appear promising. This is particularly true for matching problems that require only that a good partial-match be computed. Second, we expect that there are interesting ways in which the min-conflicts heuristic could be combined with other heuristics. Finally, there is the possibility of employing the min-conflicts heuristic with other search techniques. In this paper, we considered only one very basic method, hill climbing. However, since the number of conflicts in an assignment can serve as a heuristic evaluation function, more sophisticated techniques such as best-first search are possible candidates for investigation. Another possibility is Tabu search[7], a hill-climbing technique that maintains a list of forbidden moves in order to avoid cycles. Morris[16] has also proposed a hill-climbing method which can break out of local maxima by systematically altering the cost function. The work by Morris and much of the work on Tabu search bears a close relation to our approach.

Conclusions

In this paper we have analyzed a very successful neural network algorithm and shown that an extremely simple, heuristic search method behaves similarly. Based on our experience with both the GDS network and min-conflicts hill-climbing, we conclude that the min-conflicts heuristic captures the critical aspects of the GDS network. In this sense, we have explained why the network is so effective. Additionally, by isolating the min-conflicts heuristic from the search strategy, we distinguished the idea of a repair-based CSP method from the particular strategy employed to search within the space of repairs.

Finally, there are several practical implications of this work. First, the scheduling system for the Hubble Space Telescope, SPIKE, now employs our symbolic method, rather than the network, reducing the overhead necessary to arrive at a schedule. Second, and perhaps more importantly, it is easy to experiment with variations of the symbolic method, which should facilitate transferring SPIKE to other scheduling applications. Third, by demonstrating that repair-based methods are applicable to standard constraint satisfaction problems, such as N-queens, we have provided a new tool for solving CSP problems.

References

[1] H.M. Adorf and M.D. Johnston. A discrete stochastic neural network algorithm for constraint satisfaction problems. In *Proceedings of the International Joint Conference on Neural Networks*, San Diego, CA, 1990.

[2] G. Brassard and P. Bratley. *Algorithmics - Theory and Practice*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[3] D. Brelaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22:251–256, 1979.

[4] M. Eskey and M. Zweben. Learning search control for constraint-based scheduling. In *Proceedings AAAI-90*, Boston, Mass, 1990.

[5] M.S. Fox, N. Sadeh, and C. Baykan. Constrained heuristic search. In *Proceedings IJCAI-89*, Detroit, MI, 1989.

[6] K.J. Hammond. *Case-based Planning: An Integrated Theory of Planning, Learning and Memory*. PhD thesis, Yale University, Department of Computer Science, 1986.

[7] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39:345–351, 1987.

[8] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences*, volume 79, 1982.

[9] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, Part II. *To appear in Journal of Operations Research*, 1990.

[10] D.S. Johnson, C.H. Papadimitrou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37:79–100, 1988.

[11] M.D. Johnston. Automated telescope scheduling. In *Proceedings of the Symposium on Coordination of Observational Projects*. Cambridge University Press, 1987.

[12] S. Kambhampati. Supporting flexible plan reuse. In Minton S., editor, *Machine Learning Methods for Planning and Scheduling*. Morgan Kaufmann, 1992.

[13] S. Minton. Empirical results concerning the utility of explanation-based learning. In *Proceedings AAAI-88*, Minneapolis, MN, 1988.

[14] S. Minton, M. Johnston, A.B. Philips, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint-satisfaction and scheduling problems. Submitted to *Special Issue of Artificial Intelligence Journal on Constraint Satisfaction Problems*.

[15] S. Minton, M. Johnston, A.B. Philips, and P. Laird. Solving large scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings AAAI-90*, 1990.

[16] P. Morris. Solutions without exhaustive search: An iterative descent method for binary constraint satisfaction problems. In *Proceedings the AAAI-90 Workshop on Constraint-Directed Reasoning*, Boston, MA, 1990.

[17] N. Muscettola, S.F. Smith, G. Amiri, and D. Pathak. Generating space telescope observation schedules. Technical Report CMU-RI-TR-89-28, Carnegie Mellon University, Robotics Institute, 1989.

[18] R.G. Simmons. A theory of debugging plans and interpretations. In *Proceedings AAAI-88*, Minneapolis, MN, 1988.

[19] H.S. Stone and J.M. Stone. Efficient search techniques - an empirical study of the n-queens problem. *IBM Journal of Research and Development*, 31:464–474, 1987.

[20] G. J. Sussman. *A Computer Model of Skill Acquisition*. American Elsevier, New York, 1975.

[21] M.M. Veloso and J.G. Carbonell. Towards scaling up machine learning: A case study with derivation analogy in prodigy. In Minton S., editor, *Machine Learning Methods for Planning and Scheduling*. Morgan Kaufmann, 1992.

Combining Constraint Satisfaction and Local Improvement Algorithms to Construct Anaesthetists' Rotas

Barbara M. Smith

Division of Artificial Intelligence
School of Computer Studies
University of Leeds
Leeds LS2 9JT, U.K.

Sean Bennett

Department of Anaesthetics
The General Infirmary at Leeds
Great George Street
Leeds LS1 3EX, U.K.

Abstract

A system is described which has been built to compile weekly rotas for the anaesthetists in a large hospital. The rota compilation problem is an optimization problem (the number of tasks which cannot be assigned to an anaesthetist must be minimized) and has been formulated as a constraint satisfaction problem.

The forward checking algorithm is used to find a feasible rota, but because of the size of the problem, it cannot find an optimal (or even a good enough) solution in an acceptable time. Instead, an algorithm has been devised which makes local improvements to a feasible solution. The algorithm makes use of the constraints as expressed in the CSP to ensure that feasibility is maintained, and produces very good rotas which are being used by the hospital involved in the project.

It is argued that formulation as a constraint satisfaction problem may be a good approach to solving discrete optimization problems, even if the resulting CSP is too large to be solved exactly in an acceptable time. A CSP algorithm may be able to produce a feasible solution which can then be improved, giving a good, if not provably optimal, solution.

The Rostering Problem

Leeds General Infirmary (L.G.I.) is a large teaching hospital in the centre of Leeds. The anaesthetics department consists of 19 consultant anaesthetists and 24 other full-time anaesthetists in more junior grades, who are referred to collectively as junior anaesthetists. The junior grades are primarily training grades, and part of the junior anaesthetists' training is to work alongside a consultant anaesthetist. However, in the U.K., junior anaesthetists also do some work on their own. At the L.G.I., there is a set of operating lists, referred to as junior lists, which are always covered by junior anaesthetists working on their own. Junior anaesthetists may also be required to cover consultant lists on their own if the consultant is away. The consultants work the same pattern of operating lists every week, but a weekly rota is required for the juniors, showing what each will be doing in each of ten weekly sessions (Monday to Friday, a.m. and p.m.).

There are three grades of junior anaesthetist: Senior Registrar (SR), Registrar and Senior House Officer (SHO), in descending order of seniority. The SRs and half the Registrars are assigned for a month or more at a time to a training block, which is a specialty such as paediatrics, in order to improve their skills in that area. Most of the SRs work to a fixed timetable in their own specialty for most or all of the week, assisting a consultant. The Registrars who are on a training block should also work with the consultant in their training specialty for much of the week, although they do not have a fixed training timetable. The remaining Registrars are assigned to General Duties, and are available to cover junior lists, stand in for absent consultants and so on, for most of the week, as are the training block Registrars when not involved in training. The SHOs are not assigned to a particular specialty, but are doing general training in the specialties not covered by the training blocks; the least experienced SHOs should spend most of their time accompanying a consultant, while those with more experience can do some of the junior lists on their own, or stand in for an absent consultant.

One of the SRs is assigned to the 'General Duties/Admin' block, and the administrative part of this is to compile the weekly rota. The Admin SR spends half a day a week compiling the rota for the following week. Since each SR spends a maximum of six months on this block, the Admin SR is only becoming expert at compiling the rota by the time that the next person takes over. The job therefore takes much more time than it would if the same person did it all the time; it is also difficult to ensure consistency. On the other hand, the person compiling the rota needs to be an experienced anaesthetist, in order to know what specialties different people can cope with on their own, and so on. The Admin SR is also responsible for making any adjustments to the rota after it has been compiled, for instance if someone is ill, and needs to be able to judge whether a particular operating list will be relatively straightforward, or requires someone with considerable experience in the specialty. Hence it is not appropriate to entrust the compilation of the rota to a clerk, but it was felt that a system which could produce the initial rota automatically, under the control

of the Admin SR, would be of great benefit. It would also allow more strategic questions to be explored, for instance, how many anaesthetists of each grade are required to cover the operating workload.

The rota varies from week to week, partly because of the on-call rota. This is compiled separately, for a month at a time, and shows for each night of the week, and the weekend, five junior anaesthetists who are on call to deal with emergency work, for instance in obstetrics or the Intensive Care Unit. For Registrars and SHOs, being on call at night governs what they do on the immediately previous and following days. The rota also varies because of staff absences, which result in changes to the work that needs to be allocated in the week. If an SR is away, in many cases the work that he or she would have done has to be assigned to someone else, preferably to the Registrar working in the same specialty, if there is one. If a consultant is absent, sometimes no action need be taken, for instance if an SR would normally assist the consultant, and can take responsibility for the list instead. Often, however, a junior anaesthetist who is capable of doing the list alone must be found. When junior anaesthetists are absent, the work to be done has to be shared amongst fewer people; in some weeks the level of absences means that several operating lists have to be cancelled. Compiling the rota therefore means solving a different problem each week: the work to be done varies ¿From week to week, as do the personnel available to do it.

If the opportunities for Registrar and SHO training are included, it is not possible to compile a weekly rota which covers all the work, and the Admin SR tries to strike a balance between covering as many operating lists as possible and allowing adequate training. The first priority, however, is to cover those lists where the consultant is absent; the junior lists can, if necessary, be left uncovered, in which case the list is cancelled, and it is not essential that juniors should be assigned to all the training lists available.

Compiling the Rota

The first step in compiling the rota for a given week (whether manually or by computer) is to record the planned absences of each junior anaesthetist and their predetermined assignments, i.e. those due to regular commitments or to the on-call rota. This gives a partly completed rota, the gaps showing where the juniors are still available to do the remaining work. The Admin SR then needs to know which other operating lists need to be covered and which training lists are available in that week, given the planned absences of both consultants and juniors. This gives a set of tasks to be done in each session of the week, together with a set of people available to do them. In addition, some anaesthetists must be assigned an half day off during the week: normally, an afternoon off is taken following a night on call, but if an anaesthetist is not on call during the week, an afternoon off has still to be assigned. A half day for the compilation of the next rota must also be set aside for the Admin SR.

The rota compilation system extracts the set of tasks to be done, and the junior anaesthetists available, from its basic information about the department, which does not change from week to week, and from data on absences and the on-call rota, which does need to be input each week. The departmental data includes, for each consultant operating list, the action to be taken if the consultant is away: various strategies are available, for instance, to assign a specific junior anaesthetist if they are available, and failing that one of the different grades of junior, listed in order of preference.

Compiling the rota then consists of assigning an anaesthetist to each task, taking into account the requirements of the different tasks, e.g. some operating lists require a particular grade of anaesthetist, some training lists are only appropriate for the anaesthetist training in that specialty, and so on. At the same time, the additional afternoons off must be assigned. The rota must be optimized, in the sense that the number of tasks left unassigned must be minimized, while a satisfactory balance is kept between training and covering the junior lists.

The number of tasks to be done varies from week to week, but is normally about 90-100, and the number of anaesthetists who can do each task averages about 5.5. The number of anaesthetists who need to be given a half day off is about 4 or 5. The size of the problem can be reduced if we recognize that some of the training lists, i.e. the general training lists which are principally for SHOs, rather than the specialized training lists attached to the training blocks, are of much lower priority than other tasks. Acceptable rotas can be compiled by assigning the other tasks first, and then fitting the general training lists into the remaining gaps. This reduces rota compilation to two separate problems, the second of which is trivial. The first problem then has about 75 tasks to be assigned.

Constraint Satisfaction Problems

The constraint satisfaction problem has been discussed extensively in the Artificial Intelligence literature [see references]; it can be used as a formulation of many problems arising in OR. In a constraint satisfaction problem there are a number of variables, each of which has a discrete set of possible values (its domain). There are also a number of constraint relations, specifying which values are mutually compatible for various subsets of the variables: for instance, the assignment of an anaesthetist to a task is incompatible with the assignment of the same anaesthetist to another task in the same session. A solution to the constraint satisfaction problem is an assignment of values to the variables which satisfies the constraints.

Although the definition of the CSP does not distinguish between solutions, so that all assignments which satisfy the constraints are equally acceptable, it is possible to represent optimization problems as CSPs. The objective is represented as an additional constraint, which changes each time a new solution is found. For instance, in a minimization problem, the constraint is that the value of the objective must be less that its

value in the best solution found so far (or, initially, less than some very large number). This ensures that each solution is better than the previous one, and when all the solutions to the CSP have been found, the last one will be optimal. A similar scheme for representing discrete optimization problems as CSPs is described by van Hentenryck [3].

In general, constraint satisfaction problems are NP-complete, so that although several algorithms exist for solving them ([2], [4]), they are not guaranteed to find a solution in a reasonable time unless the problem is small or has special structure. However, in many cases there is a good chance of finding a feasible assignment quite quickly. Optimization problems, on the other hand, will almost certainly suffer from the exponential worst-case performance, since the search cannot be terminated when the first feasible solution is found. Despite this difficulty, it may still be possible use a constraint satisfaction formulation as a basis for finding good solutions to optimization problems, as demonstrated below.

Nadel [4] surveys the available algorithms for the CSP, and compares their performance on some standard problems. One of the best algorithms in these experiments is the forward checking algorithm, described by Haralick and Elliott [2], and this algorithm is used by the rota compilation system.

The Rota Compilation Problem as a CSP

As mentioned earlier, the first stage in compiling the rota is to record the predetermined assignments and the planned absences for the week. The CSP formulation will only be concerned with the problem of assigning the remaining tasks to those anaesthetists who are still available after this first stage.

The variables of the CSP are used to represent the tasks to be assigned in the given week, and the domain of each variable is the set of anaesthetists who can do that task. In addition, there is a small number of variables which represent a half day off for an individual anaesthetist. The domain of such a variable is the list of sessions in which the anaesthetist could take a half day off.

The domain of each task variable is arranged in priority order, with the best choice of junior anaesthetist for the task appearing first. The forward checking algorithm selects values from the domain in the order in which they appear, and hence the anaesthetist appearing first in the list is the one most likely to be assigned, if available. Although ordering the domains is not guaranteed to give the overall best allocation of anaesthetists to tasks, it does in practice give acceptable results.

In order to express the relative priorities of the different types of task, they are divided into three categories: essential, preference and optional. The essential tasks are those arising from consultant absences: an anaesthetist must be assigned to each of these in order to achieve a feasible solution. (It is extremely unlikely that a situation could arise in practice where consultant absences could not be covered.)

The preference tasks correspond to the junior lists and the Registrar accompanied lists, i.e. those training lists which allow a Registrar to accompany a consultant anaesthetist in their assigned specialty. To allow the algorithm to leave the preference tasks uncovered if necessary, an extra value, NONE, is added as the final element in the domain of each of the corresponding variables. When this variable is considered by the algorithm, this value can be selected, if all the anaesthetists who could do this task have been assigned to something else.

It has been found that a satisfactory balance between covering the junior lists and assigning the Registrars to training lists in their own specialty can be achieved by covering as many of the preference tasks as possible, i.e. the number of preference tasks assigned the value NONE should be minimized. This can be done by using an additional constraint to represent this objective, as described in section 3.

The final category is the optional tasks: these are the training lists for the SHOs, in which they accompany a consultant. These also have the value NONE as the last element of their domain. SHOs can be assigned to these tasks if there is nothing of higher priority which they could do instead; to reflect this, the optional tasks are assigned only after a satisfactory assignment of the essential and preference tasks has been found. The current state of the rota is then fixed and the optional tasks are assigned to those junior anaesthetists who have not so far been allocated to do anything in that session.

The constraints of the CSP firstly arise from the fact that an anaesthetist cannot do two things at once, so cannot be assigned to two task variables in the same session, or to have a half day off at the same time as doing a task. These constraints may be thought of as general rostering constraints; similar constraints expressing the fact that no-one can be assigned to do two tasks at the same time will occur in any rota compilation problem. The anaesthetists' system also has a constraint representing the objective, as already described.

In addition, there are other constraints reflecting particular rostering rules used at the L.G.I., which have in fact changed several times during the course of the project. Currently, for instance, there is a rule that Registrars who are on a training block can be taken off training, and assigned to a junior list instead, at most once during the week. Constraints of this kind are likely to vary from hospital to hospital and, as experience at the L.G.I. has shown, to change over time. The system has therefore been designed in such a way that constraints are easy to express.

Improving a Feasible Solution

Having set up the variables and their domains, the forward checking algorithm is used to find an assignment of the essential and preference tasks and the half-day variables. Very little backtracking is required to find a feasible assignment, because most variables do not represent essential tasks and so can if necessary

be assigned the value NONE, which, at this stage, does not conflict with any other assignment. The algorithm therefore finds a first feasible solution very quickly. However, because of the size of the problem, finding the optimum solution would take a very long time. Often, finding any improvement to the first solution takes far longer than would be acceptable.

It is possible that improvements in the way that the forward checking algorithm is used might achieve a sufficient increase in speed to allow an optimal solution to be found. For instance, there are variable and value ordering heuristics, such as those discussed by Nudel [5] which can be expected to give significant improvements in appropriate cases. Value ordering heuristics cannot be used in this case because the original ordering of the domains must be preserved, and the variables with smallest domains cannot be assigned first, as is commonly advised, because they do not represent tasks which are hard to assign, but rather the Registrar training lists, which should not be given higher priority than other tasks. It is still conceivable that variable ordering rules based on problem knowledge could be developed. However, rather than pursuing this possibility, we have used the forward checking algorithm only to produce a feasible solution, and looked for ways of improving such a solution. This approach produces good results very quickly, and it seems unlikely that an improved forward checking algorithm would be able to do any better.

In order to improve on the best solution that the forward checking algorithm can find quickly, an algorithm has been devised that considers each uncovered task in turn and looks for reassignments of related tasks which will allow it to be covered. This local improvement algorithm was developed through examining feasible but non-optimal rotas, and looking for reassignments that would improve them.

Suppose that there is an uncovered task that we want to try to find an assignment for. This is a variable which has been assigned the value NONE. All the anaesthetists in the variable's original domain must have been assigned to do something else in this session (otherwise the assignment of NONE would not have been made) but it may be possible to free one of these anaesthetists by reassigning the task that they are currently assigned to (a *swap*), or by moving a half day off from this session to another session (a *move*).

The following example (adapted from an actual rota) shows the kind of swaps within a session that can be made in order to improve the solution.

Variable	Original Domain	Assigned
ORTHO-TRAUMA-THU-AM	(R-4 R-6 R-5 SHO-1 SHO-2 NONE)	R-4
CW-II-THU-AM	(R-4 R-6 R-5 SHO-1 SHO-2 NONE)	R-6
OBS-THU-AM	(R-5 R-4 R-6 NONE)	R-5
GARDNER-THU-AM	(R-5 NONE)	NONE
PSU-I/A-THU-AM	(R-4 R-6 R-5 NONE)	NONE

The variables are shown in the order in which the forward checking algorithm considers them, so that the value assigned is the first remaining value in the domain. (Values assigned to other variables representing tasks in this session have been omitted.) The two uncovered operating lists in this Thursday morning session (GARDNER and PSU-I/A) can be covered by making use of SHO-1 and SHO-2 who are so far unassigned in this session. The simpler swap is to assign the ORTHO-TRAUMA list to SHO-1, thus allowing R-4 to do the PSU-I/A list. Covering the GARDNER list entails a chain of two exchanges: SHO-2 takes the CW-II list, R-6 takes the OBS list, and R-5 can then do the GARDNER list.

A simple example of a move is to move an anaesthetist's half day off from a session where there is an uncovered task that this anaesthetist could do to another session where they have not been assigned to do anything. More complicated changes involve a swap, of the kind illustrated above, combined with a move. This is done if moving a half day off would allow an uncovered task to be done by the anaesthetist concerned, and the swap has to be done to free the anaesthetist in the session that the half day off is being moved to.

The local improvement algorithm considers each uncovered task in turn in the current solution, and for each anaesthetist in the original domain of the corresponding variable, each of the above changes is tried, starting with the simpler changes, until a change which will allow the task to be covered is found, or the variable's domain is exhausted. This procedure ensures that the first value in the domain which can be assigned to the task is found, thus observing the preference ordering of the values.

In all cases, potential changes to the current solution are checked against the constraints, so that even when new constraints are introduced (e.g. an upper limit on the number of junior lists a Registrar on a training block can do in a week, as mentioned above), the algorithm still produces a feasible solution.

The local improvement algorithm works through the list of uncovered tasks once, and then presents the resulting solution as the best that it can achieve. The combination of swaps and moves seems to be adequate to produce an optimal rota; so far, we have not been able to see any further scope for reducing the number of uncovered tasks in the rotas produced, except by relaxing the constraints.

Producing the Rota

At this point, the rota will have several gaps, where an anaesthetist has not been assigned to do anything. The final stage in constructing the rota is to assign the optional lists to fill these gaps. The resulting rota is then printed out, with a note of any remaining uncovered junior lists.

The Admin SR may still wish to make changes to the rota before it is issued. This is partly because there may be places in the rota where an anaesthetist has not been found anything to do; since the workload varies so much from week to week, there are often sessions where there are fewer tasks than available anaes-

thetists, as well as sessions in the same week where sessions have to be left unassigned. The Admin SR can assign spare anaesthetists to give additional assistance at operating lists which have already been covered. Occasionally, when there are outstanding unassigned tasks, the Admin SR may be able to relax the constraints in order to allow them to be covered. Even when the system does not produce immediately usable rotas, the remaining tidying-up takes only a few minutes: the difficult part of the job has been done.

Alternative Approaches

Dhar and Ranganathan [1] describe a similar problem to rota compilation (that of assigning teaching faculty to courses) and compare an integer programming formulation to an expert system. In their expert system, production rules are used to express both problem solving knowledge and constraint knowledge. In the rota compilation problem, however, expert problem solving knowledge is not easily available. The Admin SR changes every few months, so that there is not usually sufficient time to develop any great expertise and there is little opportunity to pass on experience from one incumbent to the next; each person therefore evolves their own method of rota compilation, based largely on trial and error. It seemed best, therefore, to use an algorithmic approach to constructing the rota and to use the successive Admin SRs only as a source of constraint knowledge.

There is scope, however, for making more use of problem solving knowledge in rota compilation. For instance, at present there is no attempt to identify the session which will be most difficult to cover and to assign the tasks in that session first. Hitherto, this has not been important because there has been little interaction between the different sessions; the constraints are for the most part between tasks in the same session. If the interaction between sessions increased, then it could become important to use this kind of problem-solving knowledge, by using it to direct the order in which the forward checking algorithm considers variables.

Results and Conclusions

The rota compilation system has been developed in Common LISP on a Sun 3/160; it is now also running on a PC. It can produce a weekly rota within 30 minutes, including entering the required data, compared with the half day allocated to compiling the rota manually. The system has been producing good quality rotas for the L.G.I. for over a year, and has coped with changes in the rota compilation rules. We are currently improving the user interface so that the system can be used by hospital staff. In future, we intend to investigate similar problems in other hospitals and to extend the system to deal with them.

Apart from the fact that the system saves the Admin SR several hours work each week, with less risk that a task will be forgotten, another benefit is that it can be used to evaluate different policies, reflected in different sets of constraints. A series of rotas which would result from the different policies can be produced and compared, using real data on absences, etc., from past weeks. Hitherto, there has been no way of evaluating the effects of proposed changes in policy.

A common approach in Operational Research to optimization problems which cannot be solved exactly is to find (somehow) a feasible solution and then to look for local improvements which will hopefully produce an acceptable solution. Incorporating the two stages, of finding a feasible solution and then improving it, into the constraint satisfaction framework has a number of benefits. First, constraint satisfaction seems a natural way of formulating many discrete optimization problems; there is a close correspondence between the variables and values of the CSP and problem entities. In OR approaches, on the other hand, especially those based on mathematical programming formulations, there may be a significant translation gap between the original problem and its formulation. Secondly, since there are already CSP algorithms, a means of finding a feasible solution is readily available: it is not necessary to write a special-purpose algorithm.

Finally, the local improvement algorithm can make use of the constraints, as expressed in the CSP formulation, to ensure that any changes maintain feasibility. This has been demonstrated in the rota compilation system, when a new constraint has been introduced. Adding a constraint to the CSP requires only a few lines of LISP; the local improvement algorithm needs no modification at all, since it merely checks any potential changes against the new constraint. Hence, building the local improvement algorithm within the CSP framework gives a very flexible and easily modified system, which would be hard to achieve otherwise. Although the system described here is very specialized, the general approach of finding a feasible solution and then improving it, all within the CSP framework, is one that might be applicable to many optimisation problems in scheduling.

References

[1] V. Dhar and N. Ranganathan (1990) Integer Programming vs. Expert Systems: An Experimental Comparison, *Communications of the ACM* **33**, 323-336.

[2] R.M. Haralick and G.L. Elliott (1980) Increasing Tree Search Efficiency for Constraint Satisfaction Problems, *Artificial Intelligence*, **14**, 263-313.

[3] P. van Hentenryck (1989) *Constraint Satisfaction in Logic Programming*, MIT Press.

[4] B.A. Nadel (1989) Constraint Satisfaction Algorithms, *Comput. Intell.* **5**, 188-224.

[5] B.A. Nudel (1983) Consistent Labeling Problems and their Algorithms: Expected Complexities and Theory-Based Heuristics, *Artificial Intelligence* **21**, 135-178.

JIGSAW: Preference-Directed, Co-operative Scheduling

Theodore A. Linden
David Gaw

Advanced Decision Systems,
a Division of Booz•Allen & Hamilton, Inc.
1500 Plymouth St.
Mountain View, CA 94043
linden@ads.com
415-960-7562

Abstract*

Techniques that enable humans and machines to co-operate in the solution of complex scheduling problems have evolved out of work on the daily allocation and scheduling of Tactical Air Force resources. A generalized, formal model of these applied techniques is being developed. It is called JIGSAW by analogy with the multi-agent, constructive process used when solving jigsaw puzzles. JIGSAW begins from this analogy and extends it by propagating local preferences into global statistics that dynamically influence the value and variable ordering decisions. The statistical projections also apply to abstract resources and time periods—allowing more opportunities to find a successful variable ordering by reserving abstract resources and deferring the choice of a specific resource or time period.

Keywords: Scheduling, constraint propagation, statistical look-ahead, hierarchical planning, resource abstractions, transformational synthesis.

1. Introduction

For many scheduling problems, partial automation is a realistic but difficult goal. Partial automation means that human schedulers can participate in incremental scheduling decisions. Algorithms from operations research and most heuristic search techniques involve humans in the problem set up but not in the generation of schedules. These algorithms work well when the problem is modeled perfectly and is computationally tractable. Unfortunately, practical scheduling problems occur in very complex environments, it is usually impossible to capture all of the domain complexities in the formal model. In practice, the results of fully automated scheduling algorithms are used primarily to debug the problem set up. This results in a very large debugging loop that is inefficient and does not always converge to an acceptable solution. Furthermore, details about myriads of individual preferences are seldom handled effectively. While a human scheduler may notice that an important task in today's schedule is one on which John Jones performed effectively last week, it is impractical to expect that the knowledge acquisition task can capture all these subtle preferences in advance.

A co-operative approach to schedule generation exploits the strengths of both humans and automation, but co-operation implies that the scheduling software has to work with an incomplete model of the problem domain. Human scheduling decisions should be viewed as dynamic extensions to that model. Furthermore, many scheduling problems are dominated by preferences rather than by hard constraints, and these preferences need to be exploited in the same way that constraints are exploited in constraint-directed scheduling.

2. Background and Overview

JIGSAW generalizes techniques originally developed to partially automate the daily allocation and scheduling of Tactical Air Force resources. The complexity of the knowledge involved in this scheduling problem is such that, when done manually, a team of 8-16 people works over a period of 12 or more hours. An interactive system that solves this problem by allowing humans and the machine to make incremental scheduling decisions was designed three

* This work was partially supported by the Defense Advanced Research Projects Agency (DARPA) under contract DAAH01-90-0080 and partially supported by IR&D funding from Advanced Decision Systems.

years ago, has undergone two years of user evaluations,[1] and is now being hardened for operational use. JIGSAW is a generalization and formalization of the automated reasoning techniques originally designed for this application.

JIGSAW is an open collection of techniques that allow humans to participate as schedules are constructed incrementally. JIGSAW begins with a transformational approach—similar to the transformations commonly used to compile program specifications into programs and to refine design specifications into designs. Correctness-preserving transformations encapsulate knowledge about incremental allocation and scheduling decisions. They separate the definition of these decision rules from the control decisions about when they should be invoked.

JIGSAW extends this transformational approach with statistical look-ahead techniques. Statistical look-ahead uses local constraints and preferences to project the expected contention for resources over time. These statistical projections allow local scheduling decisions to be influenced by statistical knowledge about the global context. Statistical look-ahead enhances both value and variable ordering techniques. Our ongoing work extends these statistical projections to deal with abstract resource groupings. Partially determined time intervals are also handled as abstract resources. An assignment of an abstract resource to a task creates a reservation for an unspecified instance of the abstract resource. These reservations for abstract resources enable incremental commitments that provide more opportunities to find variable orderings that avoid or reduce backtracking.

The name JIGSAW is based on an analogy with jigsaw puzzles where:

- **Many independent agents**—both human and automated—co-operate to construct a solution.
- **The order in which scheduling decisions are made is not predetermined** by the problem.
- **Partial solutions** can (usually) be evaluated as (probably) extensible to an acceptable solution.

JIGSAW extends this analogy with a combination of techniques for reasoning about preferences, abstraction levels, variable ordering, and uncertainty. Unlike jigsaw puzzles, JIGSAW seeks a globally good solution by making a series of local decisions that are informed by statistical knowledge about how the local decision is likely to impact global optimality.

The overall JIGSAW approach involves associating a transformation with each incremental, atomic allocation and scheduling decision. The users can commit some scheduling decisions, and the automated JIGSAW techniques accept and work with partial schedules developed by users. The users control which transformations will be candidates for execution. The control software invokes the transformations that produce the most promising extensions of the current partial schedule.

3. Exploiting Value and Variable Ordering Opportunities

To fully exploit value and variable ordering opportunities when constructing a schedule incrementally, individual transformations of partial assignments should be kept as atomic as possible. Most job shop scheduling techniques exploit variable ordering opportunities only at the level of complete orders or resources; that is, they make assignments to all the tasks involved in an order or they completely schedule a single resource. Like Cortes [Fox & Sycara 90, Sadeh 91], JIGSAW enables separate decisions for each individual task or activity.[2] JIGSAW allows a task to be assigned a resource or scheduled into a time period without simultaneously committing to decisions about other tasks or times. Furthermore, by introducing resource abstraction hierarchies, JIGSAW can reserve an abstract resource for a task while deferring the assignment of a specific resource or time interval. These assignments of abstract resources allow more opportunities for variable ordering heuristics to be effective.

When allowing very small incremental transformations that may be made in almost any order, one has a problem preserving the property that any partial assignment that is generated can be extended to a nearly optimal solution. In particular, related tasks must all eventually receive consistent assignments, tasks that are assigned an abstract resource must eventually receive specific resources, tasks that receive resources must eventually be scheduled, and tasks assigned a flexible time period must eventually be scheduled into a specific time interval. These problems are largely avoided in earlier scheduling systems where all of the decisions associated with an order or resource are made simultaneously; however,

[1] The realities of a large implementation have led to an early focus on machine assistance for human decision-making; implementation of the automated decision-making techniques on which JIGSAW is based is quite recent.

[2] JIGSAW's tasks are equivalent to Cortes' activities. The terms "operation" and "variable" are also used in the literature with an equivalent meaning.

this grouping of decisions limits the opportunities to fully exploit value and variable orderings.

JIGSAW includes substantial bookkeeping functions and statistics that summarize the state of current assignments and project the probable effects of future assignments. This information is used to inhibit transformations that are likely to interfere with the completion of existing partial assignments. Projections about the expected demand on resources allow the incremental transformations to achieve a balance between greedy local optimization and altruistic minimization of resource conflicts [Sycara et al. 90]. The bookkeeping functions and statistics apply to abstract as well as specific resources and time periods. Reservation for abstract resources are guaranteed in that transformations making assignments to other tasks will preserve enough instances of the abstract resource to fulfill all prior reservations. Significantly, many of these same bookkeeping functions and statistics are also useful to the human experts who co-operate in the problem solving process.

The bookkeeping functions and statistics are also used to dynamically select the order in which the transformations are executed. The goal is to defer each transformation until there is enough information available to predict that its decision is a step toward achieving a nearly optimal assignment. Note that if all transformations meet this goal, then whenever a specific task-resource or task-time-period pairing is required to achieve an optimal assignment, other transformations will not use up the last instance of the resource or time period that is needed by this task. Of course, with invocation criteria as stringent as this, the problem is whether there will always be a transformation that does not need to be deferred. An experimental hypothesis being evaluated is: for many large problems that are characterized by many preferences and that can be solved adequately by teams of human experts, there will usually be some "obvious" transformation that does not need to be deferred. When there is no such transformation, then either human intervention or a branch and bound search strategy can be used effectively.

In summary, the automated portion of JIGSAW starts from any consistent partial assignment (initially from the empty assignment unless human experts make some initial decisions), finds a transformation that is statistically the least in need of being deferred, executes that transformation, and iterates. Humans control the overall process and can interleave their own decisions between transaction invocations.

4. Statistical Projections

In the Tactical Air Force application, statistical look-ahead was used to give more sophistication to what is basically a greedy algorithm augmented with plan repair techniques. However, the statistical look-ahead techniques together with reservations for abstract resources also work in the context of backtracking or breadth-first search strategies. The choice of the search strategy is controlled by the size of the problem and the need to interact with human schedulers, not by the statistical look-ahead. Human schedulers appear to be most comfortable with divide-and-conquer, greedy algorithms, and plan repair strategies—together with a very limited amount of breadth-first search and backtracking.

The critical part of JIGSAW is the inner loop where statistics about expected resource availability are projected and a transformation that does not need to be deferred is found. This section summarizes the steps used in the Tactical Air Force scheduling problem from which JIGSAW evolved. A more formal, general treatment can be found in [Linden 91], and we are currently trying to formalize these ideas more directly in terms of Bayesian networks and decison theory.

This description of the inner loop in JIGSAW is a step toward generalizing the computations, not optimizing them. The Air Force application where these techniques were applied deals with the optimization issues; many optimizations are available by reusing previous computations.

The steps of the inner loop are:

1. **Local rating**: Use constraints to identify the alternative resources and time periods that can be assigned to each task, and use preferences to order or rate these possible values. This local rating is based on the easily-processed constraints and preferences directly associated with the task; initially, it does not deal with global issues like resource availability.
2. **Global statistics**: Translate the local ratings for each alternative value assignment into a subjective probability that this assignment will be made, and sum these probabilities across all the tasks to project global statistics about the expected demand for each resource. Comparison of the expected demand for resources with the available resources identifies probable bottlenecks.
3. **Trade off**: Re-evaluate the alternative value assignments in terms of which choice is most likely to be part of a globally optimal assignment. This re-evaluation uses the statistics about resource contention and makes a trade off between local utility and global resource contention.
4. **Commit**: For one or more. tasks, "commit" to a transformation that is projected to lead toward a good complete assignment. Choose to make this

commitment for tasks where the decision is **"obvious" and/or "influential":**

a. **Obvious decisions** are those where one can project a very high confidence level that a decision made now will be "right." This confidence is evaluated in terms of:

- Strength of the local preference for the proposed commitment relative to alternative possible values. This may be computed as the delta between the rating of the value to be committed and the rating of the next best value.
- The commitment's use of low contention resources based on the statistical projections of the expected demand for each resource at various times.
- The quality of the current understanding about how interactions with other tasks might affect this task.

b. **Influential decisions** are decisions which clarify many other decisions; for example, a decision to commit bottleneck resources is influential because it narrows the choices that remain open for all others decisions.

5) **Plan repair:** Plan critics are available as a way of undoing a previous decision—along with the decisions that directly depend on it. Plan critics resolve conflicts that arise from imperfect look-ahead or from changing conditions in the external environment. Plan critics have been included in the design of JIGSAW applications, but they have not yet been added to the formal JIGSAW model.

5. Conclusions

JIGSAW evolved from work on large scheduling applications that must be solved co-operatively and are dominated by preferences rather than by hard constraints. JIGSAW exploits those preferences to project statistical characteristics of the global situation which are then used to enhance local value and variable ordering decisions. JIGSAW extends these statistical projections to abstract groupings of resources and allows partial schedules to include reservations for abstract resources. These reservations for abstract resources open more opportunities for value and variable ordering techniques to be effective.

JIGSAW is proposed as one of a range of scheduling techniques It is appropriate for large resource allocation and scheduling applications that are currently solved by teams of human experts. It is especially appropriate for problems where the evaluation criteria are complex, changing, and not fully formalized--problems for which human schedulers need to be involved to help evaluate the feasibility and effectiveness of the evolving schedules.

6. References

[Fox & Sycara 90] Mark S. Fox and Katia P. Sycara, "Overview of CORTES: A Constraint Based Approach to Production Planning, Scheduling, and Control," Proc. of the Fourth Inter. Conf. on Expert Systems in Production and Operations Management, May, 1990.

[Linden 91] Theodore A. Linden, "Preference-directed, Co-operative Resource Allocation and Scheduling." Final Technical Report, DARPA Order No. 6685, Advanced Decision Systems Report TR-1270-3, Sept. 1991.

[Sadeh 91] Norman Sadeh, "Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling." PhD Thesis, School of Computer Science, Carnegie Mellon University, 1991.

[Sycara et al. 90] Katia P. Sycara, S. Roth, N. Sadeh, and M. Fox, "Managing Resource Allocation in Multi-Agent Time-constrained Domains." DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control., Morgan Kaufman Publ., San Mateo, CA, Nov. 1990, pp. 240-250.

A Hybrid Job-Shop Scheduling System

Bernd Hellingrath
Peter Roßbach
Fahid Bayat-Sarmadi
Andreas Marx

Fraunhofer-Institute for Material Flow and Logistics
Emil-Figge-Str. 75
D-4600 Dortmund 50
F.R.G.

Abstract

The intention of the scheduling system developed at the Fraunhofer-Institute for Material Flow and Logistics is the support of a scheduler working in a job-shop. Due to the existing requirements for a job-shop scheduling system the usage of flexible knowledge representation and processing techniques is necessary. Within this system the attempt was made to combine the advantages of symbolic AI-techniques with those of neural networks.

System structure

The scheduling system is situated below a MRP system giving the relevant data for the schedule generation. This data contains information about the orders, work plans and resources, the optimization goals and the strategies. Out of this data local, global and strategic constraints are generated.

The local constraints describe the strict requirements the schedule has to fullfil. These are the sequence of operations, the demand for resources, the capacity restriction of resources, and the due dates. Beside the strict requirements global optimization goals have to be considered within the schedule. An optimization goal consists of an optimization criterion whose value describes certain costs (throughput time, resource utilization, inventory, tardiness) and a goal description (minimization of throughput time, minimization of weighted resource utilization, ...). These global constraints represent the optimization goals as preferences. Strategies for building up and refining a schedule are formulated as strategic constraints. These strategic constraints contains a description about when certain strategies can be used, where the schedule can be made more detailled, how specific situations

can be detected and what kind of actions have to take place, how the data of the schedule can be aggregated to make the detection of situations possible, and how specific requirements of the factory can be taken into account. All three type of constraints are used by the different schedulers (local, global, strategic) to build up a schedule. The structure of the scheduling system is shown in Fig. 1.

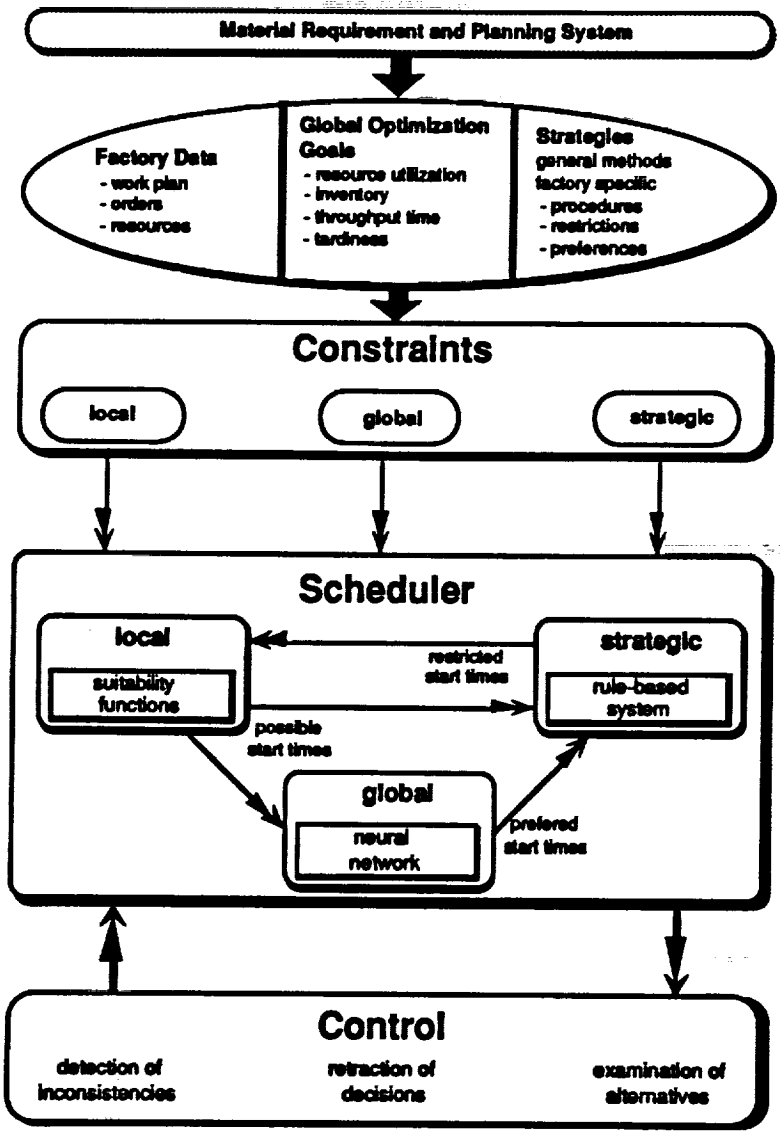


Fig. 1: Structure of the Scheduling System

Generation of the schedule

For the generation of the schedule all three schedulers work on it while the information between them is exchanged through the partially detailled schedule. The process of the schedule generation can be described as follows. In a first phase the local scheduler makes a preliminary analysis of the starting time for every operation. This analysis is done

with respect to the strict requirements and preferences the schedule should fulfill. The possible starting times are determined through the propagation of the local constraints within so called suitability functions [JOHNSTON 89]. Suitability functions describe for every operation how desirable it is to start it at a certain time, so they can be described as functions over the time (Fig. 2). When the value of a suitability function for an operation is zero this operation cannot be started at that time. The local scheduler generates a schedule in which all times where an operation cannot start are excluded. The propagation of the local constraints are based onto Allen's time relations [ALLEN 83], the values of the constraints being suitability functions (Fig. 3).

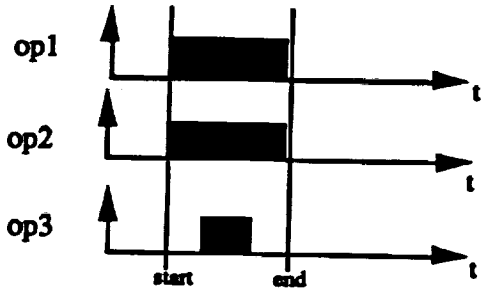


Fig. 2: Example of suitability function before propagation

Each time relation is expressed by a utility function (Fig. 4). This type of function represents a relative measure for the preference of the starting time of an operation. In an extention of the time relations static constraints for the first possible start time, the least possible end time, and the capacity of a resource are built.

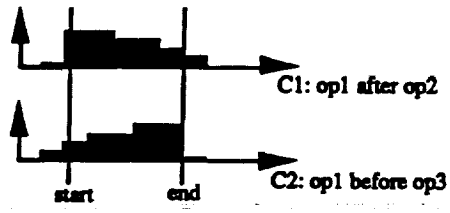


Fig. 3: Constraints as utility functions

In each propagation step an operation is chosen and for each constraint to another operation a sub-suitability function is being built. The result is a suitability that shows the possible starting times of this operation under a constraint.

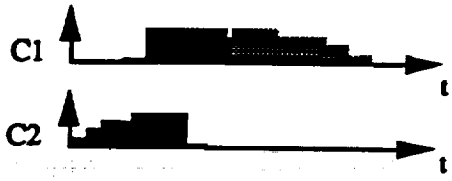


Fig. 4 : Resulting sub-suitability functions

At each propagation step the new suitability function is formed out of the product of all sub-suitabilites and the static suitabilities. Within this new suitability all constraints have been taken into account (Fig. 5). If the suitability function has changed all suitabilities of a constrained operation must be updated. When no suitability changes anymore the propagation ends. In the CSP - notation this propagation creates an arc consistent graph.

Fig. 5: Suitability Function after propagation

Besides the strict requirements the global optimization goals should also be considered within the schedule. This is done by the global

scheduler which refines the possible starting times of every operation by using a neural network.

This neural network is built up based upon the possible starting times determined through the local scheduler. The neural network is a Guarded-Discrete-Stochastic Network (GDS) a special type of Hopfield net [JOHNSTON/ ADORF 89], [MINTON ET AL. 90], [HOPFIELD/ TANK 85], [HOPFIELD/TANK 86]. The main idea is a unit guarding a subset of normal units, so that it is guaranteed that one unit is active. In the scheduling domain it helps to generate schedules for all operations and resources and not for a subset what would only be possible with Hopfield-nets [JOHNSTON/ ADORF 89]. The net is divided into two parts, the operation net and the resource net. The weights between the units of the operation net are explicitly set by the goals of the optimization (minimization of throughput time, weighted resource utilization, tardiness and work in progress). All units have a bias which is based on the results of the local scheduler, thus representing the suitability functions. The activation of the units of the operation net corresponds to it's preferred start time interval while the activation of the resource units represent the remaining capacity in that time interval.

The net is arranged in a matrix-like manner. While the rows represent the operations and resources, the columns contain the time intervals in which the suitability functions of all operations are constant. The update of all units of both nets is done synchronically with the same probability and regarding the state of the guarding units. The convergence of GDS-networks is not guaranteed, and so we impose a restriction on the number of epochs [JOHNSTON/ADORF 89]. The result of a stable state of the neural network is an optimized schedule with respect to the different optimization goals.

The local and the global scheduler work on the schedule as a whole, i.e. changes in the schedule affect all operations. These changes are generally rather coarse. The strategic scheduler on the other hand selects one operation out of the schedule for which it does a detailled planning. The strategies the scheduler uses for this are described within the strategic constraints. Strategic constraints are formulated as rules on four levels of abstraction:

- metarules

  These rules describe which strategies are adequate at certain states of the schedule.

- strategies

  The strategies describe how to refine the schedule (e.g. scheduling the critical operations first) taking into account the state of all operations and resources.

- situation & action

  These rules are used to detect situations (e.g. when an operation is critical) and to suggest actions (e.g. scheduling an operation in it's preferred time interval). The view of these rules is local, looking at the actual state of an operation or resource within the schedule.

- transformation & reduction

With these rules the actual state of the current schedule can be reduced to the relevant informations (e.g. the preferred time interval for starting an operation) used by the rules of the higher abstraction levels.

As a first step the strategic scheduler selects an adequate strategy by using the metarules. The strategies suggest detailled changes for the scheduling of a selected operation. This selection is done by the strategic constraints describing the situation & action and transformation & reduction. The suggestions are based upon the actual schedule containing the possible and the preferred starting times for each operation as a result of the local and the global scheduler. The suggestion which seems to have the most promising effects on the schedule is integrated into the schedule and the effects are propagated through the suitability functions using the local scheduler. This cycle (local – global – strategic scheduling) continues until all operations are scheduled. In the case that the decision of the strategic scheduler leads to an inconsistent schedule this decision and all it's effects have to be retracted and an alternative has to be chosen. This work is done by a control component. The work of the three schedulers can be seen as a stepwise refinement of the schedule. The possible starting times for each operation are repeatedly restricted until a sufficient exact starting point or a sufficient small interval for the starting time is determined.

At the moment the system described above is in the state of implementation. So a judgement about the quality of the scheduling system can't be done yet. But the parts implemented so far show promising results, so that we are rather hopeful about fulfilling the objectives the system should meet concerning the quality of the schedule.

Literature

[ALLEN 83]
James F. Allen ; "Maintaining Knowledge about Temporal Intervals", In : Communications of the ACM, 26, 832-843, 1983

[JOHNSTON/ADORF 89]
Mark D. Johnston, Hans-Martin Adorf ; "Learning in stochastic neural networks for Constraint Satisfaction Problems" ; In: Proc NASA Conf on Space Telerobotics, Pasadena CA, 1989

[JOHNSTON 89]
Johnston, Mark D.; "Reasoning with Scheduling Constraints and Preferences", SPIKE Technical Report 1989-2, Space Telescope Science Institute, Baltimore, MD, 1989

[HOPFIELD, TANK 85]
Hopfield, John T.; Tank, David W.; "'Neural' Computation of Decisions in Optimization Problems", in: Biological Cybernetics, 52, pp. 141-152, 1985

[HOPFIELD, TANK 86]
Hopfield, John T.; Tank, David W.; "Computing with Neural Circuits: A Model", in: Science, 233, pp. 625-633, 1986

[MINTON ET. AL. 90]
Steven Minton, Mark D. Johnston, Andrew B. Philips, Philip Laird; "Solving Large-Scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Method"; In : Proc. AAAI 90, pp. 17-24, 1990

Predit: a Temporal Predictive Framework for Scheduling Systems

E.Paolucci, E.Patriarca, M.Sem, G.Gini
Politecnico di Milano
Piazza Leonardo da Vinci 32
20133 Milano - Italy
Paolucci@ipmel1.polimi.it

Abstract

Scheduling can be formalized as a Constraint Satisfaction Problem (CSP). Within this framework activities belonging to a plan are interconnected via temporal constraints that account for slack among them. Temporal representation must include methods for constraints propagation and provide a logic for symbolic and numerical deductions.

In this paper we describe a support framework for opportunistic reasoning in constraint directed scheduling. In order to focus the attention of an incremental scheduler on critical problem aspects, some discrete temporal indexes are presented. They are also useful for the prediction of the degree of resources contention.

The predictive method expressed through our indexes can be seen as a Knowledge Source for an opportunistic scheduler with a blackboard architecture.

1. Formalization of scheduling problem and strategies for its solution

Scheduling can be formalized as a Constraint Satisfaction Problem (CSP)[Keng and Yun, 1989]. This approach is concerned with the assignment of values to variables subject to a set of constraints. In scheduling variables are constituted by activities start times and from resources allocation; for this reason we have to deal explicitely with two types of constraints: temporal relations among tasks and resources capacity [Fox, 86].

In our approach we assume that we have a set of plans to be scheduled, where a plan is defined as a partial ordering of activities. Each activity may require one or more resources and for each of them there can be alternative choices. Beside resources capacity can be used contemporarly by different tasks; for the sake of simplicity we will assume all resources with unary capacity.

Scheduling is an NP-hard problem and methods required for its solution must face this complexity. In our research we decided to focus our attention on contribution in scheduling coming from AI, and particularly on opportunistic reasoning [Hayes-Roth, 79].

We are concerned with the issue of how it's possible to focus the attention of an incremental scheduler on the most critical scheduling choices in order to evaluate which are the most critical points, which decisions seem to be the most promising in reducing search complexity and improving quality of resulting schedule.

Our strategy is to identify the most "solvable" aspects of the problem through the evaluation of the degree of interaction existing among activities belonging to different orders. The aim is to reduce the number of steps required to obtain a solution.

The necessity to overcome the limits of partial decomposition approach, such as order-based and resource-based decompositions, has led us towards an event-based perspective whit chronologically-grouped information.

This basic search strategy is realized through most-constrained and least-impact policies. Every step is divided into two parts: first the **most-constrained** policy selects dynamically on which agent must be focused scheduling attention; then, the **least-impact** policy chooses for that agent a value whose impact on the rest of the non-scheduled agents is as small as possibile. The goal is the identification of critical activities that heavily rely on the possession of highly contended temporal intervals or resources because of intra-order and inter-order interactions (**look-ahead strategy**).

This two policies need numeric indexes which, analyzing the particular structure of a problem, are able to measure the interaction among activities and resources in terms of **variable looseness** and **value goodness** [Sadeh and Fox, 88].

Variable looseness is the measure of how constrained is a resource or an activity; value goodness measures which variable value, among all the feasible ones, gives the least impact (i.e. a sort of maximum slack) on availability of feasible (and good) values for non-scheduled variables.

We identified some numeric indexes that contain information required to realize an event-based policy: these indexes are useful for different reasons:

1. This work has been partially supported by CNR PRF Project and Mauro grants

- they make possible to point out critical resources and activities;
- they identify "island of certainty" that will be a part of problem solution;
- they give information about activities start times that have the least impact on non-scheduled activities.

This behaviour is a sort of "opportunistic reasoning" [Hayes-Roth, 79]: this term has been used to characterize a problem-solving process where reasoning is consistently directed towards those actions that appear most promising for solving a problem.

Our predictive approach, used together with an opportunistic reasoning, is also useful to detect unsatisfiable CSPs as soon as possible, simply by analyzing the indexes we defined. In this sense the system can be viewed as a Knowledge Source in a blackboard architecture, which assumes responsibility for preventive analysis of activities interactions and for the detection of prospective bottlenecks.

2. The predictive approach: basic assumptions

The main goal of our research was to provide a simple but complete inference mechanism to support scheduling, working in a discrete time domain. This mechanism is based on some indexes and is designed to perform an a-priori guidance for search in scheduling domain. We kept a particular attention on the efficiency and on the speed of such a mechanism, because we realized that such properties are necessary in scheduling systems for real applicative environments like, for instance, manufacturing ones. For this reason we decide to consider a discrete representation of time instead of a continous one.

Our indexes are based on the constraints analysis (and on the propagation of the temporal ones) and on a particular representation of existing time relations.

In terms of constraints analysis we differentiate between restrictions and preferences [Fox, 86]. Temporal preferences are represented through utility functions defined on activity start times that maps possible values onto utility levels ranging from 0 to 1. Moreover in our analysis we consider the existence of intra-order (among activities belonging to an order) and inter-order constraints (among activities belonging to different orders).

The model adoptated in representing time and in reasoning about temporal relation is based on the concept of lapse, that is defined as the period of time associated with an activity. In a temporal axis a lapse is represented by two temporal parameters, namely start time and end time. Relations between different lapses are expressed by two parameters [Paolucci, 90]:

- an **internal bound** (INT), which represents the minimun time interval which must separate the end of the first lapse from the begin of the second of two related lapses;
- an **external bound** (EXT), which represents the maximum time interval from the begin of the first lapse to the end of the second.

Through these two parameters it's possible to model any temporal relation in a scheduling problem. They are simpler than thirteen Allen's primitive relations; moreover, INT and EXT improve greatly the efficiency of numeric temporal reasoning, that is instead a limit in Allen's primitive.

3. The Predit indexes

Temporal relation constraints are used to describe partial orderings among activities as provided by the process planning step.

We will refer to the graph defined by these constraints, for a given CSP, as the CSP's Temporal Constraint Graphs (TCG).

We have to schedule a set of activities (A1,A2,...,AN). Let Ik be the time interval associated with Ak. Activities are connected by a set of temporal relation constraints, thereby forming a TCG. We view TCGs as undirected graphs. An Arc in a TCG indicates the presence of a temporal relation between two intervals represented by the couple INT-EXT (Fig.1).

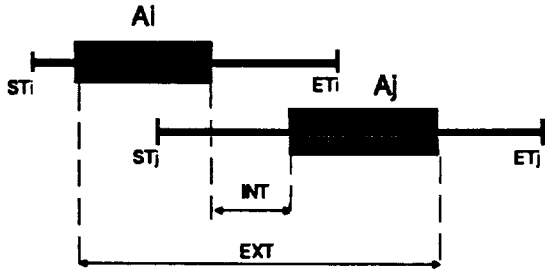


Figure 1

Additionally there are capacity constraints limiting the use of each resource to only one activity at a time. The next example presents a simple case of a TCG composed of two orders.

In order to provide a predictive support for opportunistic schedulers operating in a discrete time domain we have considered interactions among activities caused by temporal relations.

The first issue we faced was to detect as soon as possible during the scheduling the possible arising of conflicts due to interactions among activities.

For this issue we defined an index called Constraint Degree (CD), which measures the how tight is the link existing between two generic activities Ai and Aj connected by a temporal relation constraint (represented by INT-EXT) in a contraint graph.

Temporal relations between intervals may simply be expressed by using *potential inequalities* associated with the bounds of intervals such as :

[1] $D_i + D_j + INT \leq ET_j - ST_i$

[2] $D_i + D_j + INT \leq EXT$

The first inequality verifies that time interval composed of activities durations and Internal Bound is included in maximum temporal window allowed by Aj latest end time and Ai earliest start time.

The second inequality controls that the same time interval doesn't violate External Bound temporal constraint. These inequalities lead to define the CD formula through a multiplication of their members:

[3] $$CD_{ij} = \frac{(D_i + D_j + INT)^2}{EXT * (ET_j - ST_i)}$$

$$0 \leq CD_{ij} \leq 1$$

$D_k = A_k$ duration INT = internal bound between A_i and A_j

$ST_k = A_k$ start time EXT = external bound between A_i and A_j

$ET_k = A_k$ end time

The Constraint Degree is calculated on the notion of slack between two activities tied by temporal links.

- CDij = 1 means that Ai allows no slack to Aj (most constrained)
- CDij = 0 means that Ai allows maximum slack to Aj.

The CD computational algorithm considers all connected activities from the beginning to the end of the graph. Therefore, for ending activities we set CD index to zero (ending activities are not constrained, with temporal relation, with any other activity in the graph).

The validity of CD index is preserved by a previous optimization procedure in order to adjust activities temporal windows cutting out start time values that can never be involved in CD computation (the same is made for other indexes).

To sum up, the CD index detects (following the most-constrained policy) the most critical activities with respect to intra-order temporal relations (expressed by INT and EXT) and to temporal windows (expressed by activity start and end time).

The second index, called Preferential Start Time (PST), is a local measure of value goodness and globally, a measure of variable looseness for activities start times. It helps in choosing among all admissible start times the one that minimizes future conflicts. It is calculated between each pair of activities connected in the TCG (i.e. Ai and Aj) and it depends on the start time of the first activity (i.e. sti).

The main goal of PST index was to introduce some estimation rule for activity start times in order to identify the least impact values arising from intra-order interactions.

PST index is computed for every activity start time sti evaluated between earliest start time (STi), or value allowed by INT-EXT, and latest start time (ETi-Di), or value allowed by INT-EXT, increasing sti with the fixed time unit.

PST is expressed by the ratio:

[4] $$PST_{ij}(st_i) = \frac{int_{ij}(st_i)}{EXT - D_i - D_j - INT}$$

$$0 \leq PST_{ij}(st_i) \leq 1$$

where:

- $int_j(st_i)$ = relative internal bound
- EXT-D_i-D_j-INT = maximum slack between the activities

The numerator is calculated for st_i values from Earliest Start Time to the maximum allowed by temporal constraints, increasing each time st_i with a chosen time unit. It may be also considered as "actual" slack between the two activites corresponding to st_i value.

Therefore, the denominator may be viewed as the maximum slack between the two activities. The closer is PST_{ij} value to one, the greater is the slack between A_i and A_j. Therefore, for a generic activity PST measures for each admissible start time its goodness and likelihood to minimize future scheduling conflicts.

To compute activity Individual Demand for resources, we have combined the value goodness of every start time (expressed by PST) with the activities durations. Moreover, as assumed in [Sadeh-Fox, 88], an activity

A_i can use a resource R_j if A_i is active at time t and A_i uses R_j at time t to fulfill its resource requirement.

From each PST graph we achieve an Individual Demand graph (whose values are expressed by ID index) for each activity, expanding PST values with a lapse equal to the activity duration and adding all values in function of time. We obtain a histogram representing activity resource demand in function of time .

Individual Demand values are combined to measure resource **Aggregate Demand** (AD), always in function of time. AD shows when resource competition is particularly high and which are activities that heavily rely in the possession of these resources. AD values must be tightly evaluated in function of time because temporal constraint propagation doesn't allow for any resource preference (as explained before, we assume all resources with unary capacity). Therefore, AD index can estimate the amount of contention for each resource over temporal axis but only as a function of start time. Moreover, it's easy to improve this approach representing, for example, resources preferences with utility functions and propagating these resources reservations through the TCG graph.

Figure 2 shows a simple example in order to illustrate our graphic results concerning temporal discrete indexes presented above.

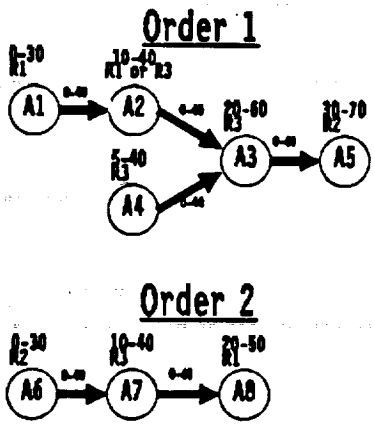


Figure 2

The temporal constraint associated at each linker is equal for all couple of activity and it is expressed by INT=0 and EXT=40. However, these values may be optimized as described before.

Start Time and End Time are expressed by numbers above each activity and the same is made for requested resources. For the sake of simplicity, in this example we have not introduced preferential start times (so activity start times are equally preferred).

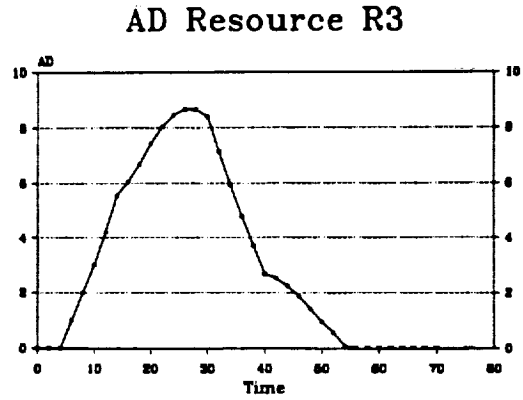


Figure 3

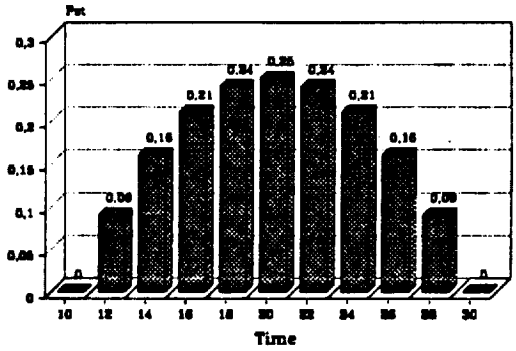


Figure 4

Next results are concerned with the reasonable steps that an Opportunistic Scheduler should achieve in order to produce the final Gantt chart.

A1	A2	A3	A4	A5
0,25	0,2	0,2	0.182	0

Table 1: CD values for order 1 activities

A6	A7	A8
0,25	0,25	0

Table 2: CD values for order 2 activities

Among the aggregate demands, the most highly contended resource is R3 (fig. 3), required by A2, A3, A4, A7; the next activities we will focus our attention on are A4, A3 and A7 (because A2 has an alternative in R1).

Taking a look at the CD indexes of order 1 and order 2, A7 appears to be the most constrained activity because of its highest CD value. Now, A7 PST graph (fig. 4)

presents a maximum for t=20 and scheduling A7 with st=20 we can assign the resource R1 to the activity A2 at the same start time.

The same considerations based on temporal indexes evaluation allow the identification of other activity preferential start times leading to the Gantt chart presented below in fig. 5.

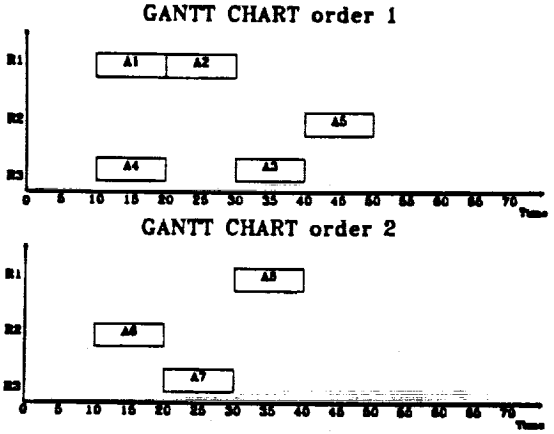


Figure 5

The quality of a schedule is based on the capability of the scheduler to satisfy a set of performance measures.

Moreover, a satisfiable schedule is always a compromise between the attempt to meet performance required and the necessity to respect all its constraints: schedule quality mirrors this trade-off. Each set of organizational constraints has its effects on final production schedules and, following the CSP formulation, if we change the constraints the solution will change too.

In order to improve schedule quality, our research is focusing on the evaluation of which impact might have an unexpected event on the resulting solution. PREDIT approach through the evaluation of discrete temporal indexes produces relatively accurate early predictions of activities behaviour as soon as PREDIT receives their changes and as long as constraints remain constant during indexes computation. The ability to react to changes that occur in dynamic environments providing a feasible solution in a sufficiently short time is very important expecially in manufacturing scheduling domain.

4. Concluding remarks

The approach we presented in this paper constitutes the basis for integrating an event-based mechanism and a predictive support in an opportunistic scheduling system.

We implemented this model in a MS-DOS environment with a particular attention towards speed performances. Our experiments indicate that our approach is successful in supporting opportunistic scheduling. This system is very efficient (it takes few seconds to calculate indexes in non-trivial real problems).

Our model seems to be highly appropriate for problems where the costs of backtracking is high because it's able to point out scheduling decisions that will minimize intra-order and inter-order conflicts. It increases significantly the performances of an opportunistic scheduler, making it possible to introduce such a tools in real applications.

Moreover the policies used by Predit to control the solution search (must constrained and least impact) can be used also in dynamic manufacturing environments. We are developing our research in this sense, also trying to support reactive scheduling and to manage multiagent production control systems.

References

[Fox, 1896], Fox M., Observation on the role of constraints in problem solving, *Proceedings Sixth Canadian Conference on AI, Montreal*, 1986.

[Hayes-Roth, 79], Hayes-Roth B. et al., Modeling planning as an incremental opportunistic problem, *Proceedings 3rd IJCAI, Tokio*, 1979.

[Keng and Yun, 89], Keng. N. and Yun D., A planning scheduling methodology for the constrained resource problem", *Proceedings IJCAI 1989*, pp. 998-1002.

[Paolucci, 90], Paolucci et. al., "CRONOS-III: requirements for a knowledge-based scheduling tool covering a broad class of production environments", in *Expert systems in engineering*, G.Gottlob and W.Nejdl (eds.), Springer-Verlag, 1990.

[Sadeh and Fox, 88], Sadeh N. and Fox M., Preference propagation in temporal capacity constraint graphs, *Technical report CMU-CS-88-193*.

TIME MANAGEMENT SITUATION ASSESSMENT (TMSA)*

Michael B. Richardson
miker@aldrin.ksc.nasa.gov
and
Mark J. Ricci
mricci@aldrin.ksc.nasa.gov
Advanced Computing Technologies Group
Boeing Aerospace Operations, FA-71
KSC FL 32899

ABSTRACT
TMSA is a concept prototype developed to support NASA Test Directors (NTDs) in schedule execution monitoring during the later stages of a Shuttle countdown. The program detects qualitative and quantitative constraint violations in near real-time. The next version will support incremental rescheduling, and reason over a substantially larger number of scheduled events.

INTRODUCTION
The Time Management Situation Assessment (TMSA) program is a prototype developed to assist NASA Test Directors (NTDs) manage the later stages of a Shuttle countdown. The NTDs are primarily concerned with the orderly and timely execution of the countdown process. The cognitive model they reason with is a relatively high-level one which includes a nominal (planned) model of the countdown and a set of qualitative and quantitative constraints that define such a countdown by specifying temporal duration and ordinal relationships between countdown events. Constraints vary both in their specificity (e.g. < is more explicit, <= is less explicit) and in their necessity (i.e. from critical - more necessary to desirable - less necessary).

From the perspective of knowledge engineering for TMSA, what is not included in the NTDs' view is as important as what is included. The details of a subsystem or procedural failure, and what is required to correct or bypass it are not, for the purposes of TMSA, a part of the NTDs' view of the countdown situation. Even in an anomalous situation the NTDs' focus remains on the temporal duration and ordinal unfolding of the countdown. When an anomaly occurs the NTDs participate in the anomaly response, primarily, for the purpose of determining the impact the anomaly will have on the temporal and ordinal aspects of the countdown.

The NTDs monitor the current countdown and assess its compliance with their nominal countdown model. When there is a need for a deviation, they consider alternative revisions of the current countdown and assess the legality and desirability of the revised countdown with regard to the constraints. The countdown schedule may be revised by reordering events and/or adjusting the durations of intervals between events.

The existing prototype monitors launch processing during the later stages of the countdown. It detects deviations from a nominal countdown by detecting temporal and prerequisite constraint violations. It then identifies the violated constraint(s). The system is initialized and operates with both qualitative and quantitative constraints on the order of events and intervals, and the duration of intervals.

The prototype is implemented in Smalltalk and runs on a 25mhz 486, under MS DOS. It appears that a C++ version of the program will be able to handle a schedule containing 200-300 events with response times of < 1.5 seconds for each assimilation input (i.e. relation vector refinement).

SALIENT CHARACTERISTICS OF THE SITUATION
In formulating our approach to this scheduling task we found the following characteristics of the situation to be especially important.

*This work is a portion of the technical support provided to the Artificial Intelligence Section, Design Engineering Directorate, by Boeing Aerospace Operations under the Engineering Support Contract at Kennedy Space Center. Arthur E. Beller is the NASA Technical Contact.

1. The situation is highly structured. A pre-existing nominal schedule is available. There is a well formulated, proven set of constraints on the schedule. The horizon for rescheduling is limited by fixed synchronization points which divide and encapsulate the countdown schedule. All possible events in the countdown are known and are of limited number.

2. Although this is an advisory system used by experts, the criticality of the situation places a premium on timeliness and correctness beyond that of many applications. Near real-time (< 1.5 second) responses and an assurance of correctness are required. Rescheduling with verification must be supported with response times, again, in near real-time. The amount of time available for considering schedule alternatives is severely limited, especially near the end of the countdown.

The verification and validation issues in our software environment, along with the above mentioned characteristics led us to approach the problem algorithmically, and avoid using heuristics.

While the countdown is formulated in terms of both events and intervals, the constraints between intervals are such that we have been able to represent intervals as start and end pairs of events. This has permitted us to restrict our representation to a point algebra that along with our variation of the Waltz algorithm provides a reasoning mechanism that is both sound and complete.

KEY CONCEPTS AND DEFINITIONS

Time

From the NTD's perspective countdown time is discrete, with a relatively coarse granularity (i.e. the smallest increments are about one second). Accordingly, we assume a discrete time model and interpret points in time as single integer, and intervals as pairs of integers, with consecutive integers forming the smallest nontrivial intervals. Effectively then, our points are "moments" in the sense of (Allen and Hayes, 1985). A different approach to discrete time and "moments" is described in (Schmiedel, 1990).

Pseudo Events

For several purposes TMSA employs events that are not members of the universe of countdown events employed by the NTDs. As with countdown events, pseudo events have integer time stamps and generally can be manipulated in the same ways as countdown events. Current uses of pseudo events are described below in the Uncertainty discussion.

Uncertainty

Uncertainty arises in the countdown schedule situation in several distinct ways. First of all many of the qualitative constraints between countdown events are ambiguous (e.g. <=). Secondly, ambiguity also occurs in some quantitative duration constraints on the length of intervals.

We represent and reason about quantitative constraints and uncertainty with the same mechanisms used for qualitative constraints and uncertainty. For example, to represent that an event Ej must occur at or after some point in time we generate a pseudo event Ei, time stamp Ei with the appropriate time and establish a constraint relation Rij of <=. This approach extends to duration constraints by using two pseudo events, one for the start and one for the end. By representing quantitative constraints in this way we are able to take advantage of the soundness and completeness of the ConstraintChecker algorithm.

In addition to the nominal countdown model and constraints, the NTDs also employ a quantitative concept of slack time, not unlike that used in project planning systems such as PERT or CPM. For the NTDs slack time is a valuable resource that they seek to preserve for use later in the countdown should it be needed. Currently we do not explicitly represent or reason about slack time, but, we are now examining approaches to representing slack time and evaluating the quality of schedule alternatives in light of the relative preservation of slack each provides.

Finally, there is the usual uncertainty related to confidence in estimates of temporal duration. Currently we do not deal with confidence factors, but, may in the future, when we begin evaluating the quality of schedule alternatives seek some measure theoretic approach to confidence.

Event (Ei):

A primitive object without discrete time duration. Events are used to define the two fundamental types of countdown objects, Intervals and Milestones, and to uniquely represent specific points in discrete time.

Universe of Events:
All the possible events that can occur as part of a countdown. These events are specified in advance to TMSA or are generated pseudo events, and are to be reasoned about by TMSA.

Interval (Iij):
A countdown object with temporal duration (trivially one) defined by two Events Ei and Ej such that if the time stamp associated with Ei is <= Ej then Ei is the start of the Interval Iij and Ej is the finish.

Assertions:
Assertions about Events may be of two types: point assertions about a single Event (e.g. Eventi occurred at time t); and Relationship Assertions about pairs of events (e.g. Eventi <> Eventj).

Quantitative Relation:
A temporal duration between two Events that is expressed as a natural number corresponding to some number of units of discrete time.

Qualitative Relation:
One of the following relationships between two Events: =, <, <=, <>, <=> (unconstrained), 0 (null). The program converts > to < and >= to <=.

ALGORITHMS
Two algorithms have been developed for TMSA. These form the reasoning Kernel of the program and are designed to monitor and interpret the legality of the temporal duration and sequential unfolding of a countdown.

The first algorithm, ConstraintChecker, is used to maintain a qualitative representation of the current status of a countdown and to check the consistency of that status with the qualitative constraints that define the legality of a countdown.

A popular approach in the scheduling literature is Allen's Interval Algebra (Allen, 1983) and his adaptation of the widely used Waltz Algorithm (Davis, 1987). The ConstraintChecker Algorithm is also an adaptation of the Waltz Algorithm and employs the Point Temporal Algebra presented in (Vilain and Kautz, 1986).

The ConstraintChecker Algorithm deals only with qualitative Relationship Assertions (in the form of Relation Vectors). One of the tasks of the ScheduleMaintainer Algorithm is to generate Relationship Assertions from Point Assertions received from the live data stream or the NTDs.

The second algorithm, ScheduleMaintainer, is used to maintain both a qualitative and quantitative representation of a countdown. the representation includes both the current status of the countdown and the quantitative constraints that define the legality of a countdown. This representation is also used to generate relational assertion vectors as input to the consistency checking algorithm.

ConstraintChecker

ConstraintChecker differs from the Waltz algorithm presented in (Vilain and Kautz, 1986) in two ways. Our algorithm uses an upper diagonal array rather than a n x n array. For our problem we needed to maintain not only a current representation of the constraints/relations between events, but, also the original constraints used to define a nominal countdown. This permits the algorithm to recognize the situation where a change in the relation between two events violates the current relation, but, not the original one. An alternative approach would have been to not update the relations vectors, but only check for validity of the new assertion. We opted for the approach used in order to permit not only the checking of new assertions with the original constraints, but, also to permit the tracking of relation vector changes over time. This capability is useful for debugging the constraint database.

We state the following theorems without the proofs because of space limitations.

The time complexity of ConstraintChecker is $O((n^3)/2)$.
The Space Complexity of ConstraintChecker is $O(n^2)$.
The inference mechanism for ConstraintChecker is sound.
The inference mechanism for ConstraintChecker is complete.

ConArray (constraint array)
An upper diagonal array indexed by events, and in which ConArray[i, j] holds the asserted constraint relationship between events i and j. ConArray holds the defining qualitative constraints (given or generated) that the NTDs use to define a legal countdown. Note that

unlike EmpArray, ConArray is not updated. Thus ConArray maintains a record of the original constraint matrix.

EmpArray (empirical array)
An upper diagonal array indexed by events, and in which EmpArray[i, j] holds the asserted empirical relationship between events i and j. EmpArray holds the current, but, changing relationships (given or generated) that actually occur during the countdown.

EPQueue (event-pair queue)
A FIFO data structure used to keep track of those Pairs of Events for which a changed relationship is asserted.

The addition operation (+) computes the sum of two vectors by finding the common constituent simple relations. This is a means to identify the least restrictive relationship the two vectors together admit. Addition is implemented as a Table lookup and is the same as that presented in (Vilain and Kautz, 1986).

The multiplication operation (x) is defined between pairs of vectors that relate three Events. For example: if Rij relates Events i and j, and Rjk relates Events j and k, the product of Rij and Rjk is the least restrictive relation between i and k that the two vectors together admit. Multiplication is also implemented as a table lookup and is similar to that presented by (Vilain and Kautz, 1986). The table has been reorganized to yield valid results using the upper diagonal array only.

ConstraintChecker

```
Assert (Rij)
/* Rij is a relation being asserted between Ei and
Ej. */
{       Tempij:= EmpArray[ij];
        EmpArray[ij]:= EmpArray[ij] + Rij;
        If EmpArray[ij] ~= Tempij
                Then Put EiEj on EPQueue;     }

Assimilate
/* Monitors EPQueue for new Relationship
Assertions */
{       While EPQueue is not empty Do
                Get next EiEj from EPQueue;
                Propagate (EmpArray[ij]);     }

Propagate (EmpArray[ij])
/* Props new Relation Assertion between Ei and
Ej to other Events */
{       For each Event Ek Do
                Tempij:= EmpArray[ik] +
                (EmpArray[ij] x EmpArray[jk]);
                If Tempij = 0
                        Then ( Check
                        (ConArray[ij]) );
                If EmpArray[ik] ~= Tempij
                        Then Put EiEk on
                        EPQueue;
                EmpArray[ik]:= Tempij;
                Tempj:= EmpArray[jk] +
                (EmpArray[ik] x EmpArray[ij]);
                If Tempij = 0
                        Then ( Check
                        (ConArray[kj];
                If EmpArray[jk] ~= Tempij
                        Then Put EjEk on
                        EPQueue;
                EmpArray[jk]:= Tempij;        }

Check (ConArray[ij])
/* Checks to see if new Relation Assertion
between Ei and Ej, Rij, violates the original
constraint between them*/
{       Tempij:= ConArray[ij];
        ConArray[ij]:= ConArray[ij] + Rij;
        If ConArray[ij] = 0
                Then (signal illegal count);
        If ConArray[ij] ~= Tempij
                Then Replace EmpArray[ij] with
                ConArray[ij] and Put EiEj on
                EPQueue;      }
```

ScheduleMaintainer

ScheduleMaintainer generates qualitative relational assertion vectors by moving an Event data point and time stamp received from an external source into the appropriate position on the multi-linked list that is the central data structure for ScheduleMaintainer. A relational assertion vector (Rij) is generated by taking the moved Event and its new successor as an Event pair EiEj. Quantitative constraints are maintained by using pointers between related Events, Ei and Ej for example, and when Ei is moved, Ej is moved appropriately, and Eventj is then processed as a moved Event, just as the original moved Eventi was processed.

We state the following theorems without the proofs because of space limitations.

The Time Complexity of ScheduleMaintainer is O(n).

The Space Complexity of ScheduleMaintainer is O(n).

ScheduleMaintainer is initialized by constructing an indexed (by External Time) multi-linked list data structure (EventList) that consists of records corresponding to every Event in the Universe of Events. Each of the n records (REj) include:

1. Name of the Event
2. Marker indicating whether the Event has occurred
3. Time stamp
4. Marker indicating whether the Time Stamp is observed, assigned as a constraint, or assigned arbitrarily by the program
5. Pointer to Predecessor REi
6. Pointer to Successor REk
7. Variable number of nonnull Pointers to other REs with quantitative constraint relationships between REi and the other individual REs
8. Corresponding quantitative constraint for each Pointer
9. Marker indicating whether the Record is to be Moved

The algorithm receives as input the name of an Event and an external time Stamp. The time stamp may be when the Event actually occurred or assigned by the user (to support interactive incremental rescheduling i.e. what-ifing).

The algorithm then examines the corresponding REi to determine if the REi should be moved in order to maintain a partially ordered (isomorphic) relationship between the discrete time of the time stamps of items on EventList and the natural numbers. This is done by comparing the new discrete time stamp with the time stamp of the successor RE.

If the new External time stamp violates the partial order condition, REi is marked to be moved and moved to a location that maintains the partial order condition.

In the new location, the successor to REi, REj is selected and a relation vector for the pair EiEj is generated. Depending on the time stamps of the two records, the vector is either = or >. If the time stamps are equal the vector is =. If the time stamps are ordered the vector is >.

The new relation Rij is then passed to ConstraintChecker.

FUTURE WORK

C++ is being used for the version currently under development. The new version of the prototype will provide an exploratory function which permits the user to query the system about the impact of changes to the preplanned countdown schedule. Both of the above developments are straightforward and will result in improved performance and increased functionality, respectively.

A more challenging task addresses the redundancy inherent in an array representation of the constraint set. We believe the bandwidth (e.g. Zabih, 1990). of the transitive closure of the countdown graph is quite small and substituting the transitive closure for the original graph, will permit us to profitably use an adjacency list (e.g. Mehlhorn, 1984) rather than an array representation of the constraint set. We currently believe we can maintain inferential soundness and completeness with such an approach. The issue seems to be, what impact this might have on the scope of the models specifiable with such a system. If we are able to use this approach, a substantial reduction in the time complexity of ConstraintChecker is possible.

REFERENCES

(Allen 83) James F. Allen, Maintaining Knowledge About Temporal Intervals, Communications of the ACM 26(11), 832-843, 1983

(Allen & Hayes 85) James F. Allen, Patrick J. Hayes, A Common-Sense Theory of Time, Proc. 9th IJCAI, Los Angeles (Cal.), 528-531, 1985

(Davis 87) Ernest Davis, Constraint Propagation with Interval Labels, Artificial Intelligence 32, 281-331,1987

(Mehlhorn 84) Kurt Mehlhorn, Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness, Springer-Verlag, 1984

(Schmiedel 90) Albrecht Schmiedel, A Temporal Terminological Logic, Proc 8th AAAI '90, Boston (Mass.), 1990

(Zabih 90) Ramin Zabih , Some Applications of Graph Bandwidth to Constraint Satisfaction Problems, Proc 8th AAAI '90, Boston (Mass.), 1990

(Vilain & Kautz 86) M. Vilain, H. Kautz, Constraint Propagation algorithms for Temporal Reasoning, Proc 4th AAAI '86, Philadelphia (Pa.), 1986

Constraint monitoring in TOSCA *

Howard Beck
Artificial Intelligence Applications Institute
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
United Kingdom

Introduction

The Job-Shop Scheduling Problem (JSSP) deals with the allocation of resources over time to factory operations. Allocations are subject to various constraints (*e.g.*, production precedence relationships, factory capacity constraints, and limits on the allowable number of machine setups) which must be satisfied for a schedule to be valid.

The identification of constraint violations and the monitoring of constraint threats plays a vital role in schedule generation both in terms of (i) directing the scheduling process and (ii) informing scheduling decisions. This paper describes a general mechanism for *identifying constraint violations* and *monitoring threats to the satisfaction of constraints* throughout schedule generation.

Identifying constraint violation To achieve a valid result in which all constraints are satisfied, a scheduler must be capable of distinguishing between valid and invalid solutions. This involves, at minimum, being able to identify constraint violations in fully-generated schedules. Clearly, if the scheduler is *only* able to identify constraint violations in fully-generated schedules, backtracking can only be introduced after considerable computational effort has already been expended. To avoid wasted effort, the scheduler should be capable of identifying *failed states* (*i.e.*, states from which it will be impossible to *achieve* a valid solution) during the process of generating the schedule. The earlier that failed states can be identified, the less unnecessary work need be done.

Monitoring of threats to constraints Given a particular factory capacity, constraint violations may be identified from the specification of the factory problem itself and could lead to a respecification of the problem. Alternatively, constraint violations may be (inadvertently) introduced by decisions taken by the scheduler. To avoid taking such decisions, potential threats to constraint violations may be tracked by a lookahead analysis (*e.g.*, [Liu88, Sad91]). Potential constraint violations occur where the magnitude of the estimated demand is close to the available capacity. Monitoring constraint threats may be used to *direct the scheduling process to the most critical constraints* and *inform the decision making process.*

*This research is supported by Hitachi Ltd.

Constraint Monitoring

Methods of constraint monitoring assuming distributions of operation demand

The monitoring of temporal-capacity constraints has been a central aspect of a number of scheduling systems (*e.g.*, [Liu88, Sad91, Ber91]). Each of these systems has been concerned with estimating demand on resources over time to allow comparisons with available capacity to be made.

Although there are important differences between the methods adopted for monitoring temporal-capacity constraints, the general approach adopted for estimating demand is based on assumptions as to the demand each operation imposes on a resource. In the case of RESS-II [Liu88], operation demand is assumed to be split equally across the valid timewindow of the operation. In the case of MICRO-BOSS [Sad91], operation demand is assumed to be split across the valid timewindow of the operation on essentially the inverse proportion of the cost associated with different start times.

Temporal-capacity analysis provides strategic information to the scheduler by highlighting critical resource time periods. This information can then be used during schedule generation to choose which particular resource time period to address next, to choose which operation to allocate and when to allocate the operation to effectively redistribute estimated resource demand.

Limitations of making assumptions about distributions of operation demand

It is in undertaking an analysis based on *splitting operation demand into a number of separate time periods* that limitations are introduced in that:

1. the estimated demand for resource over time introduces uncertainties associated with assumptions made regarding operation demand over time
2. contiguous time periods are not recognised as being contiguous

For schedulers undertaking an analysis of temporal-capacity constraints based on splitting operation demand over time, capacity bottlenecks indicate regions of high resource contention. As a result of the uncertainties introduced by the assumptions made regarding estimated operation demand, it is not possible to tell, even where the estimated demand is greater than available capacity, whether a capacity constraint has been violated or not. This is illustrated in the next section.

Constraint monitoring in TOSCA

TOSCA analyses temporal-capacity and setup-capacity constraints throughout the factory capacity hierarchy across multiple time periods. Operation demand is represented down to the granularity where the operation must legally occur, *i.e.*, the full operation demand is associated with the legal timewindow of the operation. The operation demand is not subdivided over the duration of its legal timewindow, avoiding the need to assign probabilities to the possible start times of each operation. Normally the operation timewindow is set by the release date and due date of the job and the intra-lot temporal relationships. Aggregated demand can be checked against available capacity both before and during schedule generation.

An example

To distinguish the TOSCA approach, a small example is considered using, in the first case, a method based on assumptions as to the distribution of operation demand and, in the second case, the method adopted in TOSCA which avoids such assumptions. The example involves the allocation of three operations to a single resource which is available for 7 hours per day. For the purpose of capacity analysis, the schedule timeline is split into periods of 1 day duration.

Demand:

Operation	Duration (Hrs)	Earliest Start (Day)	Latest End (Day)
op1	18 hrs	1	4
op2	3 hrs	2	5
op3	12 hrs	2	3

Capacity:
7 hours per day

Figure 1: Single resource example

Method 1: Constraint monitoring assuming distributions of operation demand

Constraint monitoring typically involves:

- maintaining an up-to-date representation of the legal timewindow of each operation throughout schedule generation
- splitting the timeline into discrete periods for the purpose of analysis
- for each operation, making assumptions about the likelihood of start times across its legal timewindow
- for each operation, calculating an expected operation demand across its legal timewindow
- aggregating demand for individual resources and comparing it against available capacity

Resource bottleneck periods (*i.e.*, periods where demand is high relative to available capacity) indicate potential threats to capacity constraints and are typically used to direct the scheduler to the most critical parts of the remaining schedule.

Methods which split operation demand across the operation timewindow assume that each operation exerts a demand across *each* of the discrete time periods under consideration that fall within the operation's timewindow. For instance op1 exerts a demand in periods day1, day2, day3 and day4. Every operation which could *possibly* be active over a particular time period contributes to the overall aggregate demand over that time period. In this example, the three operations (op1, op2, op3) all contribute to the estimated resource demand in day2.

Bottlenecks where *estimated* demand exceeds available capacity cannot be used for the purpose of detecting constraint violations. Where estimated demand exceeds available capacity, it may or may not be possible to redistribute demand away from the bottleneck and so avoid a constraint violation.

Figure 2 indicates a distribution of operation demand based on an assumed uniform probability distribution of start times. Figure 3 shows the aggregation of the demand of these operations, with the horizontal dashed line indicating the available capacity. The vertical dashed lines indicate the granularity of capacity analysis.

Method 2: Constraint monitoring without assuming distributions of operation demand

In TOSCA, the demand of an operation is associated with its temporal constraints (*i.e.*, its legal timewindow), *without assuming any subdivision of that demand across the timewindow.* An operation's demand is associated with a *single* time period. For instance, op2

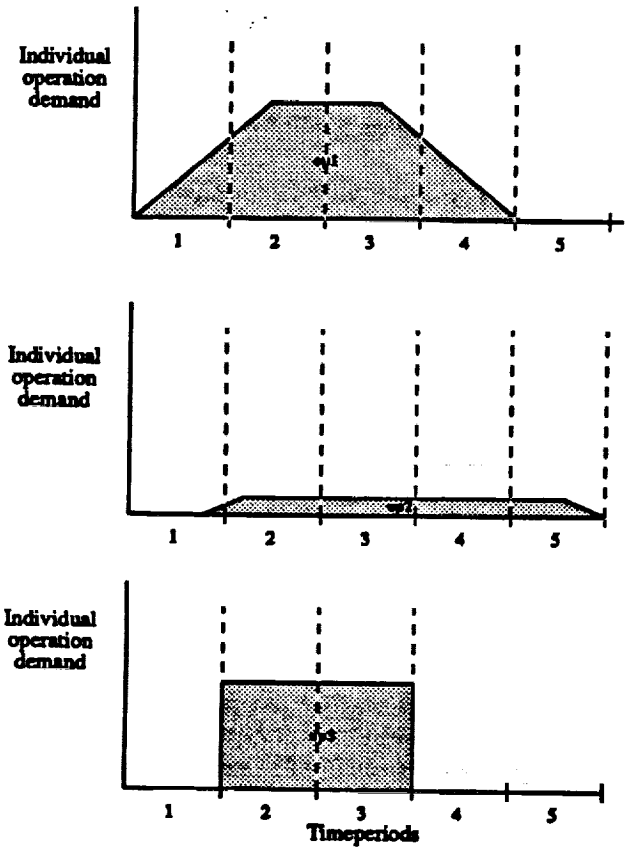


Figure 2: Individual operation demand assuming a uniform operation start time distribution

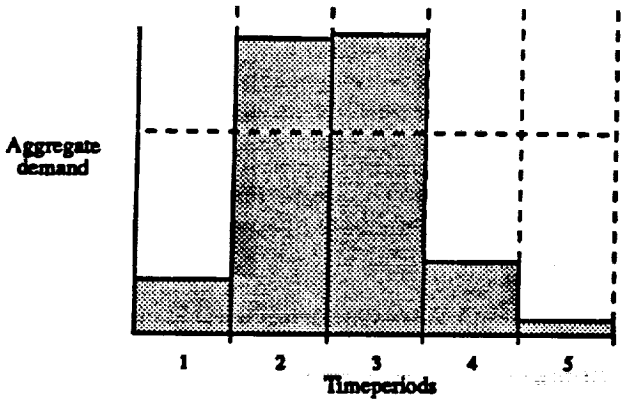


Figure 3: Estimated aggregate demand assuming a uniform operation start time distribution

exerts a demand of 3 hours over the period [2, 5], no assumptions being made regarding the probabilistic distribution of that demand within that period.

Only operations which are *necessarily* active, given that their temporal constraints are to be satisfied, contribute to the aggregate demand over the time period. That is, demand arises from only those operations whose legal timewindow are subperiods of the period under consideration. For instance, only the demand of op1 and op3 are associated with the time period [1,4]; the demand of op2 is not included.

Figure 4 shows the demand over time associated with the individual operations. op1 has a demand of 18 hours associated with the period [1, 4], op2 has a demand of 3 hours associated with the period [2, 5]; and op3 has a demand of 12 hours associated with the period [2, 3].

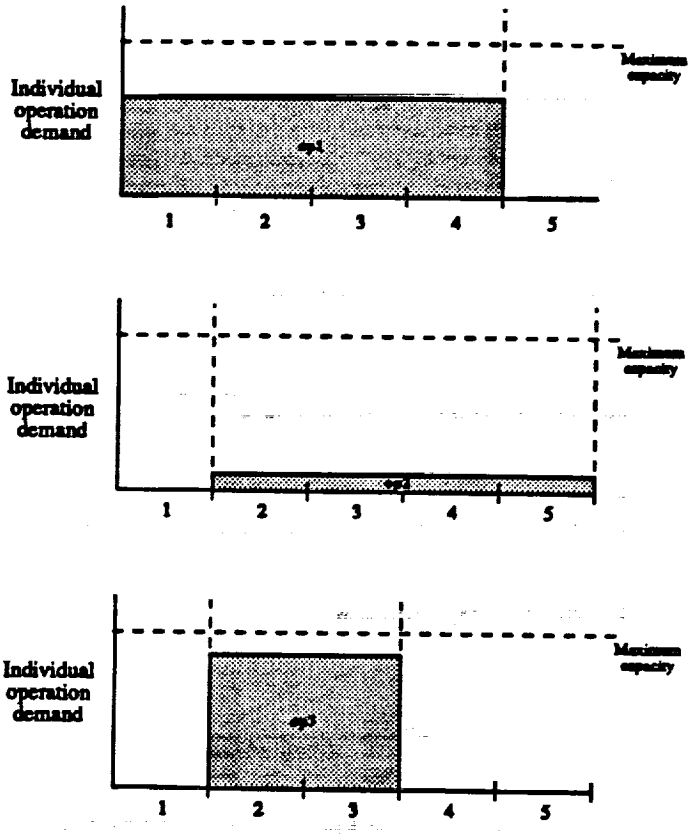


Figure 4: Individual operation demand not assuming an operation start time distribution

In estimating resource demand, temporally overlapping operations are aggregated. The operations op1 and op2 together ({op1, op2}) have a demand of 21 hours over the period [1, 5], {op1, op3} have a demand of 30 hours over the period [1, 4], {op2, op3} have

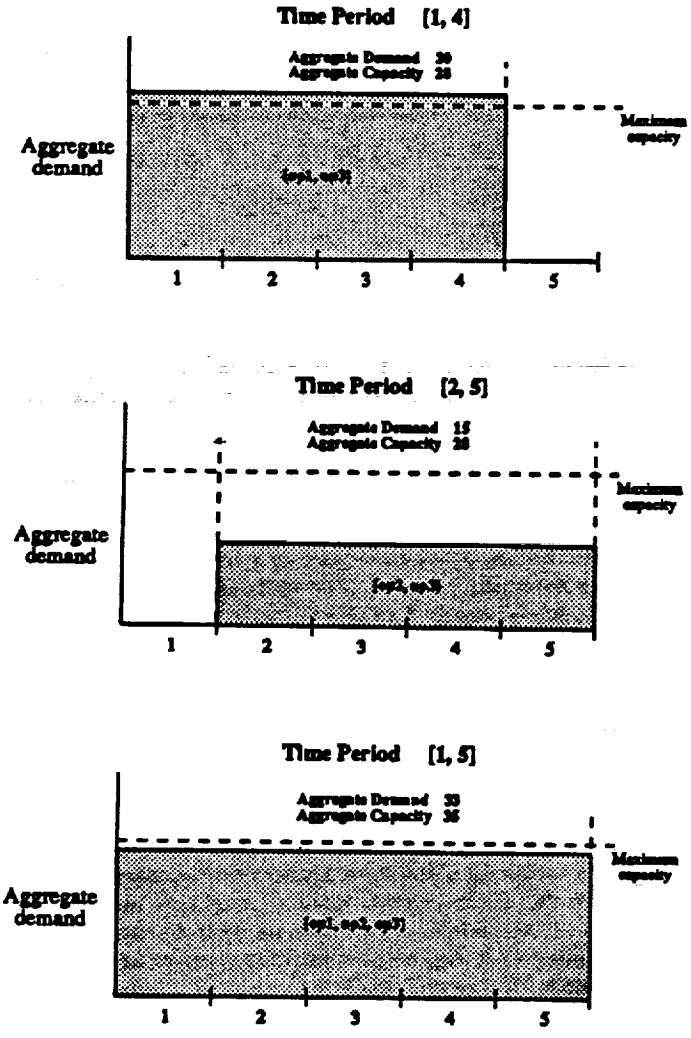


Figure 5: Aggregate demand not assuming operation start time distribution

a demand of 15 hours over the period [2, 5] and all three operations together have a demand of 33 hours over the period [1, 5]. Where multiple sets of operations are associated with a time period, the demand is that of the *maximal set* of operations. This means that the demand on the period [1, 5] is 33 hours, the demand associated with {op1, op2, op3} rather than {op1, op2}.

The demand associated with any time period can be directly compared with the available capacity — in this example, 7 hours per day — to find constraint violations and threats. A capacity constraint violation is indicated by the demand of {op1, op3}, its demand being greater than the maximum available capacity over the period [1, 4]. Figure 9 shows the demand associated with the maximal sets of operations associated with the periods [1, 4], [2, 3], and [1, 5].

In that each timeline period is associated with a set of necessary operations - assuming that the operation timewindow constraint holds - the operations implicated in a constraint violation can be readily identified. This can be used to inform constraint relaxations. In this example, the timewindow and duration constraints of op1 and op3 introduce a constraint violation. One of *their* constraints will need to be relaxed to avoid this constraint violation. Altering the constraints of op2, another operation active over this period, will not avoid the violation of the capacity constraint in the period [1, 4].

Scheduling in TOSCA involves the *iterative refinement* of the timewindow of each of the operations. Each decision to restrict the timewindow of an operation has the effect of redistributing resource demand. Before scheduling begins, op1 has a demand associated with the period [1, 4]. In deciding, for example, to restrict the timewindow of op1 to end by the third day at the latest, the operation demand becomes associated with the period [1, 3]. The effect of these decisions is monitored using *habographs*.

Constraint monitoring using habographs Habographs (Hierarchical Abstraction for Balancing Objectives) are two-dimensional datastructures used within TOSCA to represent and monitor temporal-capacity constraints. Habograph coordinates are given as start-end pairs and refer to cells representing a time period at a resource. Each operation's earliest start time is plotted on the y axis and its latest end time is shown on the x axis. Since it does not make any sense to have an earliest start time which is later than a latest end time all of the cells above the leading diagonal are always empty. The units of the axes are problem-dependent.

In referring to habographs it is important to be clear about the use of a couple of terms with respect to information held at a habograph cell: *local* and *aggregate*. A cell refers to a time period at a resource. Information about a resource time period may or may not include information about its sub-period.

Figures 7 and 9 present an illustration of local and aggregate demand in habographs on the example described above.

Cell	Local operations	Local Demand
[1, 4]	{op1}	18
[2, 5]	{op2}	3
[2, 3]	{op3}	12

Figure 6: Local demand

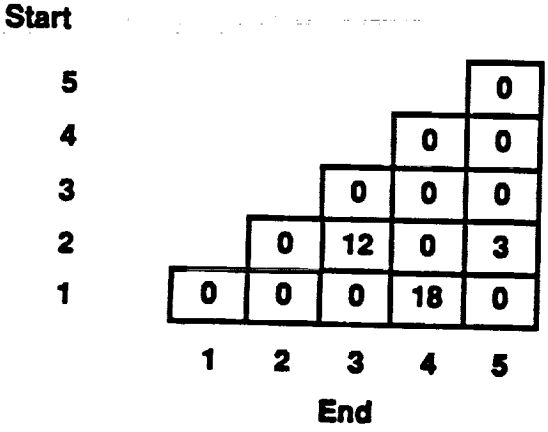


Figure 7: Habograph showing local demand

Figure 7 indicates the local operations over the periods: [1, 4], [2, 5] and [2, 3]. op1 is local to [1, 4], op2 is local to [2, 5] and op3 is local to [2, 3].

Cell	Aggregate operations	Aggregate Demand
[1, 4]	{op1,op3}	30
[2, 5]	{op2,op3}	15
[2, 3]	{op3}	12
[1, 5]	{op1,op2,op3}	33

Figure 8: Aggregate demand

Figure 9 indicates the aggregate set of operations over three time periods. The aggregate set of operations includes all the operations which must be processed in a particular period. In the period [1, 4], two operations must be processed, these being: op1, which must occur between [1, 4] (*i.e.*, day1 through day4), and op3, which must occur in the subperiod [2, 3] (*i.e.*, day2 through day3).

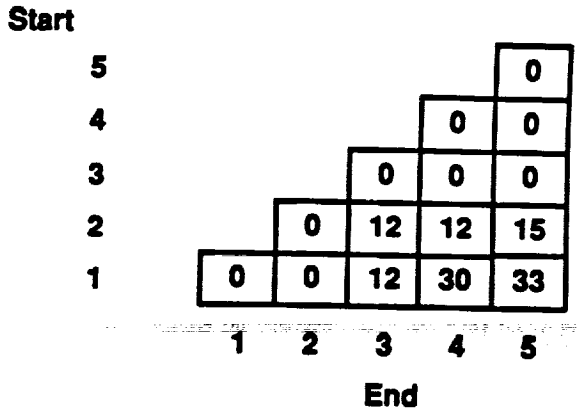


Figure 9: Habograph showing aggregate demand

The contents of habograph cells Each cell within a habograph has a representation of number of objects. The main object within each cell is a list of the operations which are local to that cell. Each of these operations exerts a demand for capacity at that cell and the sum of the demand exerted by all the cell's local operations is stored as the cell's *local demand*. Each cell also has an *aggregate demand* figure, a number calculated by summing all the local demands in all of the cells that are above and to the left of the current cell.

In addition to the demand associated with a set of operations, information is also held as to the capacity available over the time period represented by the cell. As with demand, capacity information is represented by a local and an aggregate figure. *Local capacity* is represented only over the *leading diagonal* of the the habograph. In the example under consideration, the capacity of 7 hours per day is represented along the leading diagonal with zero's everywhere else, as is shown in Figure 10. *Aggregate capacity*, shown in Figure 11, is calculated in the same manner as the aggregate demand, described above, except summing the local capacity figures rather than the local demand.

Finally the cell also has a representation for *demand pressure* (Figure 12). This is simply the ratio of the aggregate demand at that cell, divided by the aggregate capacity of that cell. Where the demand pressure is greater than one, a constraint violation is indicated. Where the demand pressure is close to but less than one, a constraint threat is indicated. In this example, a constraint violation is indicated over the period [1, 4].

Conclusion

Most current approaches to capacity constraint monitoring involve assumptions regarding the probabilistic

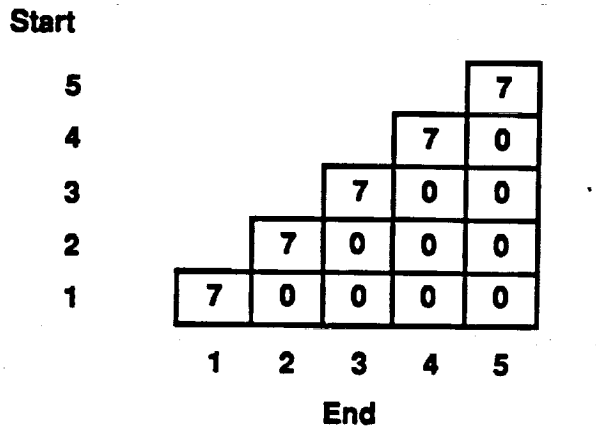


Figure 10: Habograph showing local capacity

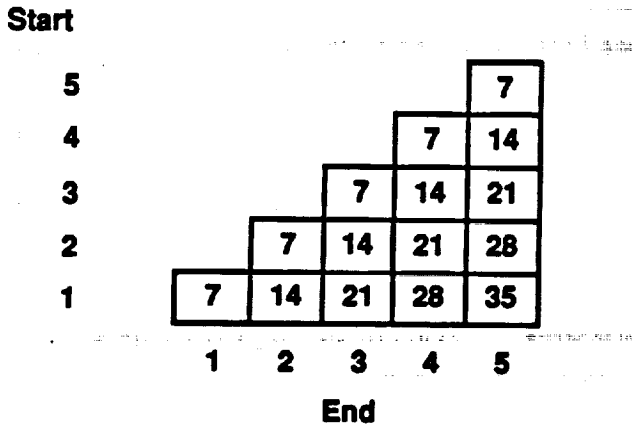


Figure 11: Habograph showing aggregate capacity

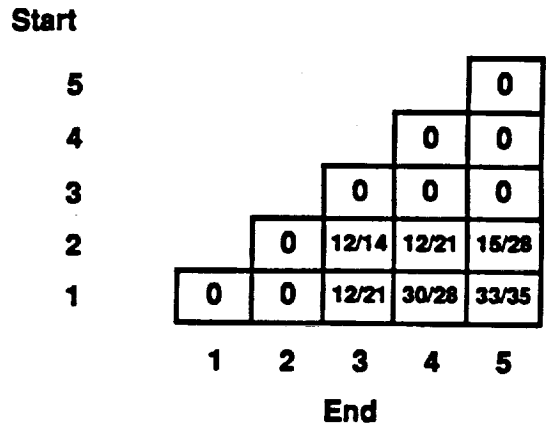


Figure 12: Habograph showing demand pressure

distribution of operation start times. Such approaches indicate resource bottleneck periods (*i.e.*, periods of potential constraint threat) but are unable to identify constraint violations.

This paper describes *habographs*, a novel datastructure, used for capacity constraint monitoring in TOSCA. The approach avoids assumptions regarding the probabilistic distribution of operation start times and has the advantage of enabling the identification of resource bottleneck periods which necessarily involve a constraint violation.

Habographs are currently being investigated within the TOSCA project as a unifying representation to support resource allocation, temporal allocation and setup management.

References

[Ber91] Pauline Berry. *A Predictive Model for Satisfying Conflicting Objectives in Scheduling Problems.* PhD Thesis, University of Strathclyde, Glasgow, 1991.

[Liu88] Bing Liu. Scheduling via reinforcement. *Journal of AI in Engineering*, 3(2), 1988.

[Sad91] Norman Sadeh. *Look-ahead Techniques for Micro-oportunistic job shop scheduling.* PhD Thesis, School of Computer Science, Carnegie Mellon University, 1991. (CMU-CS-91-102).

SOCAP: Lessons learned in applying SIPE-2 to the military operations crisis action planning domain

Roberto Desimone
SRI International
333 Ravenswood Avenue (EK335)
Menlo Park, CA 94025
roberto@erg.sri.com

Abstract

This report describes work funded under the DARPA Planning and Scheduling Initiative that led to the development of SOCAP (System for Operations Crisis Action Planning). In particular, it describes lessons learned in applying SIPE-2, the underlying AI planning technology within SOCAP, to the domain of military operations deliberate and crisis action planning. SOCAP was demonstrated at the U.S. Central Command and at the Pentagon in early 1992. A more detailed report about the lessons learned is currently being prepared [7].

This report was presented during one of the panel discussions on "The Relevance of Scheduling to AI Planning Systems".

Introduction

Many agencies, in addition to the military, have the need to manage crises. Good crisis management is characterized by quick response, decisive action, and flexibility to adapt to the changing situation. Developing a good course of action (COA) and modifying it as necessary must take into account a number of factors: approaches used in past cases that have worked well, novel features of the new situation, differing priorities for subparts of the crisis, and feasibility of suggested COAs. The objective of this program of applied research was to develop decision aids to enable more flexible and accurate joint military COAs to be developed in response to a crisis. To date, no research or development activity has integrated a full-blown generative planning system into an operational environment.

SOCAP (System for Operations Crisis Action Planning) embodies SIPE-2, together with a user interface tailored to military operations and a situation map display system. SIPE-2 (System for Interactive Planning and Execution) is a domain-independent, AI planning system that was developed during the 1980s by David Wilkins of SRI International's Artificial Intelligence Center [4, 5, 6]. It supports both automatic and interactive generation of hierarchical, partially-ordered plans. This system provides efficient methods for representing properties of objects that do not change over time, and uses these to constrain the choice of objects associated with actions in the plans generated. SIPE-2 has been tested out on a variety of small-scale problems for travel, robot, and aircraft planning, and for extended blocks-world problems. More recently it has been applied to a larger scale planning problem in the brewery domain.

In early 1992, SOCAP was demonstrated both at the U.S. Central Command in Tampa, Florida and at the Pentagon. The aim was to demonstrate the feasibility of applying the SIPE-2 technology within SOCAP for the generation of large-scale military operations plans (OPLANs). The overall objective is to generate several OPLANs that describe employment plans for dealing with specific enemy COAs, and identify deployment plans for getting the relevant combat forces, supporting forces, and their equipment and supplies to their destinations in time for the successful completion of their mission. [3] provides a description of the some of the requirements for automating the joint military operations planning process.

The rest of this report will describe SOCAP and the lessons learned in applying SIPE-2 to the military operations crisis action planning problem.

SOCAP – System for Operations Crisis Action Planning

Figure 1 shows the SOCAP architecture, highlighting the necessary inputs for the generation of OPLANs, the available outputs, and the user interaction. It is assumed that the following inputs would be fed into the SOCAP database from available military databases:

- threat assessment – list of enemy threats, locations and dates.
- terrain analysis – information on terrain features that might affect mobility and observability.

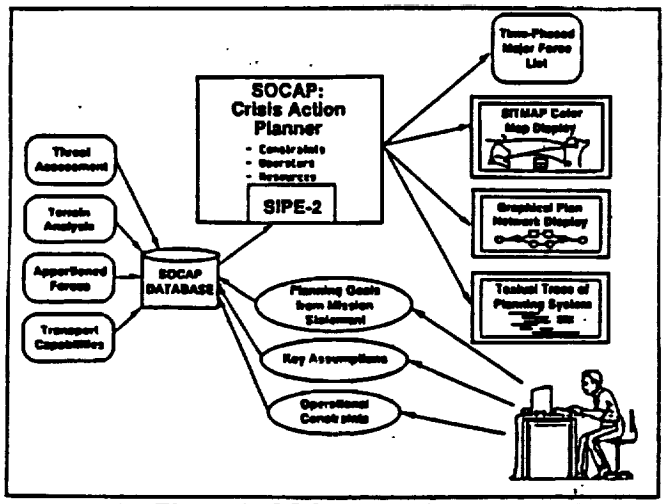


Figure 1: SOCAP Architecture

- apportioned forces – list of combat forces available for planning purposes.
- transport capabilities – list of available assets.

Other inputs would come from the user:

- planning goals – list of goals that match mission statement.
- key assumptions – e.g. rules of engagement, non-intervention of third party forces.
- operational constraints – e.g. overflight privileges, troop limits in country.

In this case, a typical user would be either the mission commander or one of his/her joint staff.

Most of the above information is inherently dynamic and is best represented in SIPE-2 as simple first-order predicates. However, a great deal of the available data are static, and for efficiency reasons are best represented in SIPE-2 using its hierarchy of classes and objects, together with (static) properties of objects. For example, cargo requirements, and combat capabilities for specific combat forces should be denoted as (static) properties of these forces.

SOCAP also requires a large set of plan operators to describe military operations that can achieve specific employment or deployment goals. For instance, there are a variety of military operations for deterring an enemy army, navy or air force. Each of these operations may be represented by a different plan operator which all have the common effect of deterring an enemy force. However, they may have different sets of preconditions that need to be satisfied before they can be brought into the plan, or different resource requirements.

The SOCAP user interface provides facilities for guiding the user through the plan generation process. The amount of user interaction can be varied during the planning process. It can range from being fully automated, in which case SOCAP generates a plan with no human interaction; to semi-automated, in which the user makes some choices; to fully manual, where the user makes all the choices. At each goal in the plan, the user can request the possible operators that achieve the goal to be displayed. Likewise, when attempting to bind a variable associated with an argument of an operator, the possible bindings can be displayed. For instance, the user may be presented with the set of military units that have the appropriate capabilities to deter an enemy threat, or a list of suitable locations for the military operation. This set may be constrained by the preconditions and other constraints associated with the arguments of the relevant plan operator. At the end of each plan level, the plan is checked for logical consistency, and then progresses to the next level until there are no more goals to be satisfied or actions to be decomposed further.

The plan may be displayed at each plan level, either as a partially-ordered network of actions and goals, or graphically on a time-based map display. The map display shows the actions that are occurring on different days during the mission. The temporal information for the map display is derived from durations associated with each action and from the dates when the enemy threats should be deterred or countered.

The following gives an idea of the size and complexity of the problems we are dealing with and the knowledge base within SOCAP. The size of plans we have generated have about 100-200 actions in the final plan level. The SOCAP knowledge base comprises: 200-250 classes/objects, 15-20 properties per object, around 1200 predicates, and 50-100 plan operators.

Lessons Learned

The lessons learned from applying SIPE-2 to the military crisis action planning domain can be divided into three main sections: successes and difficulties in applying the existing SIPE-2 technology, and open research issues.

Successes

The hierarchical plan decomposition process embodied within SIPE-2 maps well onto the military operations planning process, and delays the detail until the appropriate planning level. As a result, it was relatively easily to group sets of plan operators according to the various phases/levels of the operations planning process. For the purposes of the demonstration, these were:

Level 1: Select mission type.

Level 2: Identify threats and their locations.

Level 3: Select employment operations, major forces, and deployment destinations.

Level 4: Add deployment actions.

The class/object hierarchy provides a clear representation of static information within SOCAP, and also aids validation. A simple constraint language permits the properties associated with classes and objects to be posted on the arguments of operators. Thus, variable binding can be delayed until the constraints point to a single instance. It is also possible to force instantiations of these variables with user guidance. For instance, this facility might be used to force the selection of a favored military unit for a specific operation.

SIPE-2 provides a mechanism for permitting domain-specific knowledge to determine the number of iterations of an operator. For instance, in order to determine the number of enemy threats to deter or counter, SOCAP checks the number of enemy threat units identified in the threat assessment database, and generates a sub-goal for each. SOCAP has a variety of iterative operators that search for different types of enemy threats.

SIPE-2 permits a great deal of information to be presented to the user at a variety of levels of detail. The SOCAP user interface extracts the appropriate details and presents them to the user during the planning process. Thus, when a user is viewing the possible choices of military units for an operation, SOCAP presents the constraints that led to these choices. Nodes that contain certain predicates or arguments may be highlighted on the graphical display. Predecessors, successors and nodes in parallel may also be highlighted. This is especially useful when the plan display is large and convoluted.

The time-based map display provides another means of displaying the plan that is particularly appealing to military planners. It is possible to show the operations that occur on each day of the mission and display appropriate information about the type of military operation, the units involved and the boundary of the operation.

Difficulties

Although SIPE-2 does have capabilities for resource reasoning, specifically the representation of reuseable and consumable resources, we were unable to make use of them effectively, because of the lack of temporal reasoning within SIPE-2. Time windows associated with each action involved in a resource conflict would provide information that would help to resolve the conflict. Temporal information on the availability of the resource would permit simple conflict resolution without resorting to scheduling.

Continuing with the temporal reasoning issue, we found it would have been very useful to have had Allen's 13 temporal relations [1, 2]. This would have permitted more versatile operations including actions starting or finishing at the same time, overlapping each other, or one occuring during another, as opposed to just one strictly before another. There are many examples of dependencies between different miltary actions that could have been represented, if only...

Although SIPE-2 does have a mechanism for representing shareable resources between actions in parallel, it is very inflexible, in that you have to determine in advance how such resources might be shared over several actions. For instance, a large military unit, such as a division, may be employed in several operations simultaneously, where each operation uses some of the division's capabilities. The number of operations over which the division may be shared depends on the amount of resource required for each operation. Thus, the only way to reason about the shared resource is to consider the capabilities of the division as a consumable resource purely for this specific set of operations.

We would have liked to have had a flexible procedure for preferring to associate specific resources with actions. For instance, when choosing military units for operations, in order to minimize the number of troops involved in the operation, it is often wise to choose units already involved in the plan, provided they have not been overutilised. It is possible to write such heuristics in SIPE-2, but these are fairly rigid, and a trade-off between several heuristics is really what is required.

Another capability we would have liked is the ability to combine sub-goals at will, or serendipitously. For instance, at present, for every enemy threat identified, a friendly unit is identified to deter or counter it. If several small enemy forces are located close to each other, SOCAP attempts to deal with each threat individually, rather than considering them as an aggregate threat that might be countered with a single larger friendly force. Whether the aggregation was done by the user or by some conceptual clustering algorithm, it is important that the original sub-goals are replaced by a new sub-goal. One could write a large set of plan operators that attempt different ways of clustering sub-goals, but this is not practical for large problems.

Currently, it is difficult to represent the notion of a task force whose composition is determined by whichever military units were assigned to lower level actions. It is possible to represent a class of objects of type, task force, and make use of a part-of predicate to relate specific military units to a specific task force, but this is not an easy procedure.

We could have made greater use of deductive rules within SIPE-2 to highlight dependencies between parts of the plan that involve long chains of deduction. For instance, the arrival of communications equipment

could have triggered deductive rules to fire that would have eventually, after several rules, pointed to the availability of the necessary command and control facilities for another operation.

It would have been very helpful to have had feedback from a "tame" combat simulator. Such feedback could have been used to guide the choice of operations, forces, locations and times. It could also have been used to compare the effectiveness of a variety of courses of actions and to provide appropriate metrics for identifying qualitatively different COAs.

Another problem involves SIPE-2's meta-level control of the goal achievement process. Unfortunately, this process can only be done by having additional operators that copy their goals down to the next level when certain preconditions are true. For instance, one may decide to achieve all employment goals first and only start on the deployment goals when the employment goals have been satisfied. This notion of encapsulating such meta-level heuristics for goal achievement in the preconditions is very rigid. Ideally, one would want a more flexible process that permits a trade-off between several heuristics.

As you can gather, we managed to deal with some of the above difficulties with less than acceptable solutions. In most cases these solutions were very rigid and might even work well for some problems, but certainly would not be flexible enough for a variety of situations.

Open Research Issues

We were continually asked by most military operations planners to whom we showed SOCAP about support facilities for updating and writing new operators. We explained that this would involve providing extensive facilities for making sure that the preconditions and effects were syntactically and semantically correct. It would also require flexible test algorithms to ensure that the revised or new operators did not adversely affect other existing operators. This may provide an excellent domain for machine learning techniques.

There are a whole set of research issues concerning the relationship between and integration of planning and scheduling techniques. Below, I have just listed a few questions below that ought to be addressed:

- How can information from plan structure guide constraint relaxation?
- When to stop plan generation and choose to generate schedule?
- When to repair schedule versus plan repair?
- When to project/simulate the plan/schedule?

Summary and Conclusions

The SOCAP work discussed in this report provides the first steps towards an operational prototype that will eventually be tested out on real military crises. So far, it has been tested on a single scenario developed at the Armed Forces Staff College. We will be extending the system significantly over the next few years, and will test it on a variety of different scenarios. You should expect a steady stream of progress reports!

Acknowledgements

This work was funded within the DARPA/Rome Laboratory Planning Initiative under contract number F30602-91-C-0039.

This report could not have been written without the tireless efforts of the SOCAP team headed by Marie Bienkowski, and comprising Marie desJardins, Roberto Desimone, Jeff DeCurtins, Kate Finn, and Robert Hicks. We are also grateful for support received from David Wilkins, Peter Karp and John Lowrance.

References

[1] J. Allen. Maintaining Knowledge about Temporal Intervals. Technical Report TR-86, University of Rochester, 1981.

[2] J. Allen and J. Koomen. Planning using a Temporal World Model. In *Proceedings of the Eighth IJCAI*. International Joint Conference on Artificial Intelligence, 1983.

[3] M.A. Bienkowski. Initial Requirements Analysis for Automation in Support of Joint Military Planning. Technical Report ITAD-2062-TR-92-40, SRI International, March 1992.

[4] D.E. Wilkins. Domain Independent Planning: Representation and Plan Generation. *Artificial Intelligence*, 22, 1984.

[5] D.E. Wilkins. Recovering from execution errors in SIPE. Technical Note 346, SRI International, 1985.

[6] D.E. Wilkins. Hierarchical planning: Definition and implementation. In *Proceedings of ECAI-86*. European Conference on Artifiical Intelligence, 1986.

[7] D.E. Wilkins and R.V. Desimone. SOCAP: Lessons learned in applying SIPE-2 to the military operations crisis action planning domain. In preparation, 1992.

User-Centered Scheduling Support in the Military Airspace Management System Prototype

P. O. Perry
The MITRE Corporation,
Burlington Rd,
Bedford, MA 01730
pop@mitre.org

Abstract

The Military Airspace Management System (MAMS) is a multi-user distributed scheduling prototype designed to support the scheduling of Special Use Airspace in the CONUS region. The prototype has emphasized the user interface design of the scheduling system as the primary means of producing de-conflicted schedules. This paper reports on work in progress and provides a technical description of the user interface support for the scheduling process.

1 Introduction

1.1 Problem Description

Nearly 25 percent of continental United States (CONUS) airspace is designated as Special Use Airspace (SUA) for use by the Department of Defense (DOD) for military operational readiness training, research and development, and test and evaluation. Demand for this airspace continues to increase from both the military and the civil sectors resulting in the need for better management.

There are over 200 military airspace scheduling offices of the different services in the United States. In the southwest, where the MAMS prototype is being field tested, there are at least 20 sites where airspace scheduling is performed. The military services differ in the schedule information they report on airspaces, and some military areas report only scheduled-use, not actual-use data. The services therefore have no uniform source of data to determine their actual use, or to compare actual to scheduled use.

The DOD plans to develop the Military Airspace Management System (MAMS) as a solution, automating scheduling and reporting of SUA use and providing near real-time joint use of airspace. The MAMS prototype is being developed to define the requirements for a tool that supports efficient scheduling and utilization data collection and reporting.

1.2 History

Schedulers of SUAs currently have limited automated support for scheduling, or simply form a daily flight schedule manually. Airspace users phone or fax their requests for SUA access to the offices with local scheduling responsibility. The initial contact is usually followed by a series of phone calls. Clarifications are made, and eventually a preferred mission time is established. Then, if the local priority rules do not interfere, the scheduler of the airspace will allow the requested mission to take place. In some cases the scheduler can override the local priority rules. For example, when a fleet is conducting maneuvers offshore it expects to receive the highest priority, but may be overridden if a Top Gun class at the Naval Air Station Miramar needs to fly. There are governing rules for airspaces established by the FAA and by letter of agreement. The DOD rules for assigning priorities in an airspace may change from service to service, and sometimes from airspace to airspace. The scheduler currently can resolve conflicts by generating alternatives, assigning priorities, or trying to negotiate a mutually acceptable solution.

1.3 Prototype Approach

The MAMS prototype is planned as a widely distributed network of scheduling sites sharing a database of airspace resources. The sites will be part of a national military airspace management system that preserves their local control of resources and provides a hierarchical structure for reporting schedule data. The network will allow DOD airspace managers to quickly request and schedule missions in local airspaces and efficiently request use of remote airspaces.

It was recognized early in user surveys that it would be difficult to capture the many scheduling strategies that a diverse user community had evolved over time, and to establish a consistent set of heuristics that would satisfy most users. Many organizations had developed site specific policies and procedures for scheduling and managing their airspace resources. Scheduling rules and practices are therefore very diverse, as are user interfaces, and there has been some disagreement on a common approach to resolve the differences. Incorporating the daily negotiation process in an automated scheduling system would also be complicated.

The approach taken in developing the MAMS prototype was to provide an intuitive user interface first, and later integrate automated support algorithms. This has had the advantage of providing a method to transition to a more automated scheduling system while extracting from the users their knowledge of scheduling processes.

The variety of organizations and their particular scheduling strategies has also led us to develop a scheduling aid where the user has an explicit role rather than fully automating the scheduling system. In an environment of continuous dynamic rescheduling it seemed more effective to provide the necessary tools via better user interface mechanisms, rather than to incorporate explicit knowledge of numerous considerations of the scheduling process. Since a given schedule is continuously revised due to changing mission requirements, the emphasis on the user interface

underscores the role of the scheduler as a problem solver rather than a data entry clerk. The effort therefore centered on providing useful interface components that facilitate forming and maintaining a schedule regardless of local practices or procedures.

In its technical approach, the MAMS prototype addresses the following principal areas: the internal representation of the domain, development and optimization of an efficient user interface, supporting analytic routines, database and the distributed aspects of the application, and data gathering for standardized airspace utilization reporting.

2 Domain Representation

MAMS uses an object hierarchy to represent Resources and Activities. Both resources and activities share a common parent, Schedule, that enforces the identification and naming of each object in the hierarchy. Domain specific types of resources and activities are then specialized by inheritance.

2.1 Resources

A resource class represents airspace resources. The attributes associated with the class define the state of the resource, which consists of the activities scheduled at the resource. All scheduling functionality is embodied in the resource object. From this class we specialize two classes of airspaces: Special Use Airspaces (SUAs) and Military Training Routes (MTRs).

Most SUAs were created as military areas for training, testing, and national security. There are six different types: Prohibited Areas, Restricted Areas, Military Operations Areas (MOAs), Warning Areas, Alert Areas, and Controlled Firing Areas (CFAs). In 1989 the number of SUAs included approximately 350 MOAs, and 120 Warning Areas. The airspaces are organized hierarchically and are subdivided into further categories designated by the organization controlling a particular airspace.

SUAs can be designated either for exclusive use (only one organization may use the airspace), shared use (several military organizations may share the airspace), or joint use (which allows for simultaneous use of the airspace by military personnel and civilians). Airspaces may be also dynamically created to support special missions.

The DOD also uses point to point air routes known as Military Training Routes (MTRs). There are four types: IRs, which require an Instrument Flight Rules (IFR) flight plan, VRs, which require a Visual Flight Rules (VFR) flight plan, SRs, designated as special routes primarily for slow speed, low altitude operations, and AR routes for air refueling.

2.2 Activities

An activity is any operation scheduled within an airspace. It can be scheduled within any of the resources mentioned above, and may contain different domain attributes depending on the desired resource. In general an activity in MAMS is created by a requester, a person desiring the SUA. A scheduler must then examine the activity, evaluate it in the context of the schedule, and act upon it. The activity may be edited and changed through the user interface either by the requester or the scheduler responsible for the relevant resource.

2.3 State

State describes the most recently completed action on an activity. An activity's state is changed by the requester or scheduler through the user interface. A requester may create a new activity, delete an existing activity, or modify a current activity. A scheduler may look at an activity, approve an activity or a conflicting activity, or unschedule, deny, or modify an activity. The system also may change the state of an activity if a conflict arises.

The main states of an activity are requests, activities which have not been examined by a scheduler, and tasks, activities which have been acted upon by a scheduler. State is represented in the interface by color coding. Submitted requests are shown in blue, when scheduled they are changed to green, and if conflicting they are changed to red.

3 User Interface

In contrast to developing a user interface that provides the user with some insight and some control in the automated scheduling process [Cooper, 1990], the MAMS prototype has approached the problem by first addressing the user interface, then ascertaining how more automated support could be integrated behind the interface. This section highlights some of the key elements that aid the scheduler in establishing a conflict-free schedule.

3.1 Design Influences

The MAMS user interface design is based on previous MITRE efforts [Mulvehill, 1986]. It also draws on the Range Scheduling Aid [Smith, 1990] , a prototype designed to support scheduling of the Air Force Satellite Control Network (AFSCN) ground stations and equipment [Smith and Katz, 1990; Halbfinger and Smith, 1990].

All menus and dialogs in the prototype have been developed through extensive interaction with the user community. This feedback has forced many changes, and is the source of much of the volatility in the user interface design.

3.2 Visual Representation

The user interface is the scheduler's primary means of establishing a de-conflicted schedule. As in many similar systems, the interface is modeled on an interactive Gantt chart. The main window is divided into horizontal areas, or "panes", associated with the resources being scheduled. The window is divided from left to right by vertical grid lines the user can adjust to represent one hour to one day increments. Each requested activity is represented by a colored bar icon fixed in height and proportional in length to the duration of the mission. Its placement on the screen corresponds to the actual time at which the task is due to take place.

3.3 View Control

The user interface is designed to draw the scheduler's attention to those areas in the schedule that need repair. At

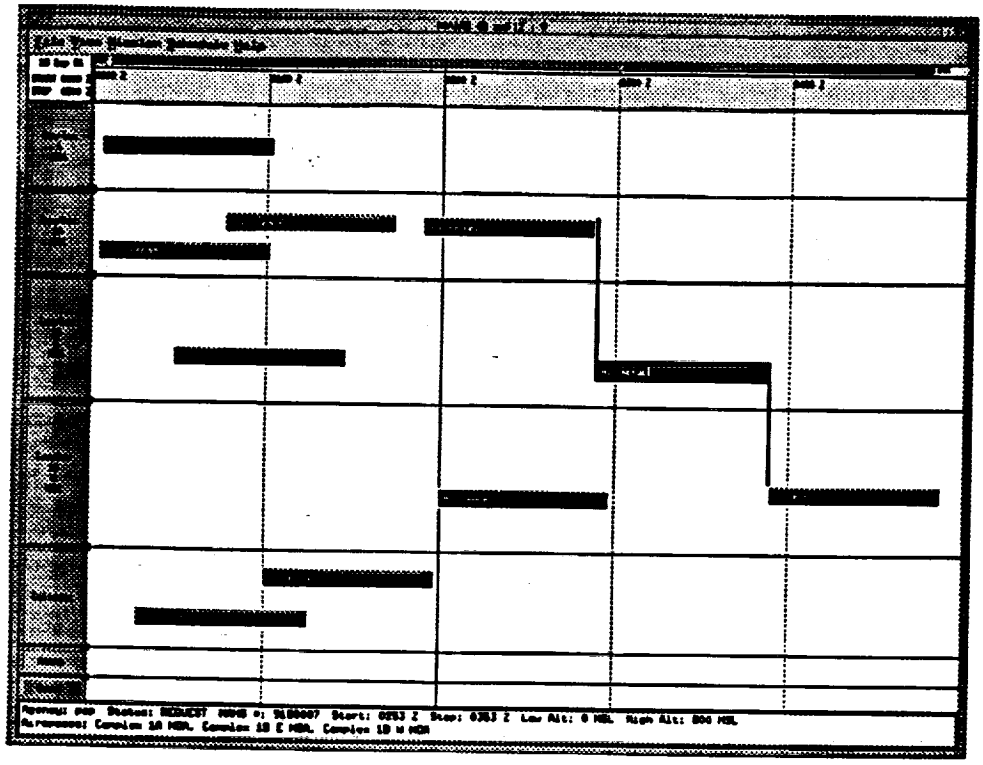


Figure 1: The MAMS Gantt Chart Interface

startup the interface begins with the current day's time frame so that the scheduler can address immediate requests first.

The interface allows the user to focus on a narrow time period, yet gain quick access to distant areas of the schedule. Using the timeline control at the top of the Gantt chart, the user can rapidly view resource data within a sliding time interval that can be varied in duration from one hour to two weeks.

3.4 Use of Domain Knowledge

Airspace resources are naturally organized hierarchically: at the top is the controlling organization and in lower nodes are sub-areas that are independently scheduled. A requester is able to submit a request at any level of the resource hierarchy. This action, in effect, is a shorthand for requesting all airspaces below the requested node. The user may view an airspace at any level in the hierarchy. This organization presents an overview of the schedule by displaying a summary of resource usage. The prototype also supports formation of arbitrary groupings of resources to simultaneously schedule at all airspaces of the grouped resources.

3.5 User Conveniences

To change the time of an activity, the user drags the icon horizontally with the mouse. To take other actions, the user selects the icon and calls up an appropriate menu. The act of selecting an icon prints information about the activity in a documentation line along the bottom of the screen.

A *find* capability helps the user locate a mission of interest. It displays a list of missions that fit certain search criteria, such as requests not yet acted on by the scheduler, or conflicting tasks in the scheduler's airspace. The user can then select a mission from the tabular list and have the screen reconfigured to the time period of the selected item, facilitating the context switch to a new mission.

The user interface also includes a number of keyboard accelerators, error correction on most input fields, an error reporting dialog, on-line help, and context sensitive help. Commands are supported both through the mouse and through key bindings.

3.6 Multi-User Support

Unlike previous multi-user scheduling systems [Beard *et al.*, 1990] the MAMS prototype updates user changes in near-real-time so that changes made by a requester or scheduler will be conveyed immediately to a second scheduler who is working with the same resource for an overlapping time period. However, MAMS provides the two users independent views of the schedule at the same time, with independent screen layouts and time intervals.

The interoperability of the X Window System simplified development of a multi-user scheduling system by enabling the application to open connections to multiple X servers. But supporting multiple users introduces additional complications that need to be addressed in the user interface. We added authentication and authorization checking to give a user permission appropriate for their role as a requester or scheduler [Patterson, 1991]. In addition, the application needs to support individual user preferences so that a user can configure the screen to their liking.

4 Analytics

It may be thought that a simple reservation system might have sufficed, since requests are made and serviced by a scheduling authority. However, the scheduler also needs support in maintaining temporal and resource constraints.

4.1 Temporal Relationships

MAMS maintains a point-based representation of time for a single activity, and a symbolic representation of time for links between activities. While the prototype does not incorporate a temporal constraint engine for maintaining temporal relations, it needs to support some temporal relations. We have found that users need to represent *before*, *meets* and *equal* relations [Allen, 1983] for a number of situations, but users requested that we represent only three relations between activities because other possible relations do not have a corresponding use in the application. Temporal relations are used when a complex linked mission is being planned, with multiple groups of aircraft operating in multiple airspaces according to an interdependent sequence of events. The scheduler needs to be able to define and maintain these relations. When an activity that is part of a linked mission is created, the scheduler can establish its dependencies to a concurrent or adjacent activity and maintain them graphically in the interface. If one activity is moved to a different time the related activities move with it.

4.2 Resource Relationships

MAMS needs to maintain resource relationships for airspaces affected by adjoining airspaces. If, for example, an MTR passes through an SUA, scheduling in the MTR will also schedule the SUA for the duration of activity in the SUA. These kinds of dependencies are automatically maintained; the user need not be aware which airspaces are related.

4.3 Conflict Identification

Currently when conflicts in SUAs are detected for activities which overlap in time and altitude within an exclusive use airspace, the conflicting activities are highlighted in red. In addition, if the airspace is an MTR, conflicts are detected if activities are taking place at the same time either at the crossing point of two routes or on the same route. This would occur, for example, if one airplane were to overtake another.Conflict identification is performed each time an activity changes state (is either scheduled or moved interactively).

4.4 Conflict Resolution

Thus far, the resolution of conflicts is left to the user. Many types of resolutions simply are not possible to automate because the system does not explicitly represent all factors in a particular scheduling choice. Rather than maintaining continuously changing knowledge in the application, the scheduler is left to resolve those aspects of the schedule that require human judgment, while the prototype maintains consistency in the schedule while managing a large set of scheduling data.

4.5 Conflict Description

The MAMS prototype graphically provides an explanation of why two or more activities are considered in conflict by associating them with connected lines. To aid manual resolution of conflicts, the user is then provided with a pop-up window containing a scrollable list of conflicting missions with conflicting field titles in red.

The scheduler may choose to accept a conflict, overriding the conflict detection. The color of the activity's icon is then changed from all red to green with a red border.

5 Management

The schedule, and therefore the airspaces, are managed primarily through collection of utilization data. After each mission, the participants are expected to report if they flew the mission as scheduled, and if not, to report any differences. This data is then entered into the system and is used to create utilization data reports. The quality of a particular mission can be recorded, and if the conditions were degraded one can enter the reason. This data can then be used to gather statistics about the number of successful missions run in a particular airspace.

6 Database

The prototype interfaces to a relational database for long term storage and management of schedule data. The choice of a relational database was deemed important because the system needs to support arbitrary queries on the data for report generation. Analysis of utilization data using such reports will support long term planning of airspace utilization. To maintain a record of multiple users' actions on the data, all transactions are time-stamped so that a historical trace of changes can be retrieved. This function is considered useful when a scheduler needs to review how a particular request was serviced.

7 Data Distribution

One of the primary requirements of the MAMS system, from both a DOD and an FAA perspective, has been timely dissemination of accurate utilization statistics. There is also a perceived practical and technical need to develop a distributed scheduling system. Practically, many site surveys showed that few sites would relinquish to another agency the necessary control to manage their own airspaces. This has led to distribution of the application, so that each site can continue to manage and control its airspaces locally.

Technically, a distributed approach yields a system that is more tolerant of failures. We have experimented with the process group paradigm for developing a distributed application [Birman *et al.*, 91; Makpangou and Birman, 1990]. This has been useful because the programming model directly supports the hierarchical structure of the DOD command. The hierarchy can be implemented as a set of overlapping process groups.

8 Implementation

The current prototype was developed on Sun Microsystems Sparcstation 2s, using C++, the X Window Sys-

tem, OSF/Motif, ORACLE RDBMS, and Isis (a toolkit for distributed applications from Cornell University [Birman *et al.*, 91]). The C++ object-oriented programming paradigm supports our problem domain well.

The prototype has been evaluated through quarterly phased deliveries. Installation of each MAMS phase has been accompanied by user feedback meetings that were an important source of system improvements. Some user needs had to be generalized to arrive at a consistent scheduling interface.

9 Further Work

The Gantt chart has proven to have limitations as a user interface metaphor for representing and resolving non-temporal constraints. We are therefore considering adding the ability to view the problem space in additional dimensions. To resolve time and altitude conflicts simultaneously, for example, it would be helpful to the scheduler to view a time versus altitude display of a particular airspace. We have applied the Gantt chart to linear travel routes, such as an MTR, with some success, in that the scheduler has enough information to recognize that there is a conflict, but there may not be enough to gain an intuitive sense of how to resolve a conflict by direct manipulation. We have therefore considered displaying an activity that uses MTR resources along time and distance axis to better represent where a conflict has occurred along a given route. Finally, many airspace managers and some users have expressed an interest in being able to view the use of airspaces on a map in order to better understand the geographic relationships of airspace utilization. This geographic capability would also support interactive designation of new airspace partitions.

It is recognized that some portions of a schedule are repeated weekly or monthly. The users have requested the ability to be able to "paste in" a preset template of events. To support this feature we plan to provide user interface functions to cut a portion of the schedule and save it as a template. The user can then select from a list of templates and paste the events at a new date in the schedule. Invariably users will feel a need to customize their environment, and we plan to evolve the application to incorporate more user preference selections. We plan to run a usability study on the user interface to validate the design thus far.

10 Conclusion

The MAMS prototype is a proof of concept system, aimed at improving the scheduling process within a diverse DOD community of schedulers. We found the MAMS scheduling process complex and difficult to specify completely, and thus could not provide a purely automated solution. Our approach has therefore been to support the human scheduler with an integrated, easy to use set of tools. MAMS is an interactive system enabling the scheduler to visualize the interdependence of requested activities and to gauge the impact of modifying a schedule. The user can thus understand the state of a portion of a schedule, and incrementally improve it to develop a fair, conflict-free schedule.

We believe the prototype has helped define the requirements for a future scheduling support system serving a large and diverse user community. The emphasis on the user interface is believed to be appropriate for scheduling problems that have large unstructured domains such as MAMS.

11 Acknowledgments

The work described herein is the product of many people. The author wishes to thank the current development team. This work was sponsored by the Electronic Systems Division (AFSC), Hanscom AFB, MA.

References

[Allen, 1983] J.F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[Beard *et al.*, 1990] David Beard, Murugappan Palaniappan, Alan Humm, David Banks, Anil Nair, and Yen-Ping Shan. A Visual Calendar for Scheduling Group Meetings. In *Proceedings of the Conference on Computer Supported Cooperative Work*, pages 279,290, 1990.

[Birman *et al.*, 91] Kenneth P. Birman, Robert Cooper, and Barry Gleeson. Programming with Process Groups: Group and Multicast Semantics. Technical Report TR-91-1185, Dept. of Computer Science, Cornell University, January 91.

[Cooper, 1990] Lynne P. Cooper. User Interface Issues in Supporting Human-Computer Integrated Scheduling. In *Second Annual Conference of the International Association of Knowledge Engineers*, pages 268,274, 1990.

[Halbfinger and Smith, 1990] Eliezer M. Halbfinger and Barry D. Smith. The Range Scheduling Aid. In *Fourth Annual Workshop on Space Operations Applications and Research (SOAR '90)*, pages 280,284. NASA Conference Publication 3103, 1990.

[Makpangou and Birman, 1990] Messac Makpangou and Ken Birman. Designing Application Software in Wide Area Network Settings. Technical report, Dept. of Computer Science, Cornell University, October 1990.

[Mulvehill, 1986] Alice M. Mulvehill. A User Interface for a Knowledge-based Planning and Scheduling System. Technical Report MTR-10175, The MITRE Corporation, Bedford, Massachusetts, November 1986.

[Patterson, 1991] John F. Patterson. Comparing the Programming Demands of Single-User and Multi-User Applications. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pages 87,94. ACM Press, 1991.

[Smith and Katz, 1990] Barry Smith and Joseph Katz. The Range Scheduling Aid. In *The Third International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 275,280, 1990.

[Smith, 1990] Barry D. Smith. A Portable Interface to a Scheduling Aid. In *Second Annual Conference of the International Association of Knowledge Engineers*, pages 208,217, 1990.