ICAPS05

DOC

Doctoral Consortium

Adele Howe
Colorado State University, USA

Ruth Aylett
University of Salford, UK

ICAPS 2005
Monterey, California, USA
June 6-10, 2005

CONFERENCE CO-CHAIRS:

Susanne Biundo
University of Ulm, GERMANY

Karen Myers
SRI International, USA

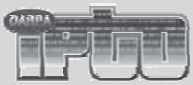
Kanna Rajan
NASA Ames Research Center, USA

Cover design: L.Castillo@decsai.ugr.es

Doctoral Consortium

Adele Howe
Colorado State University, USA

Ruth Aylett
University of Salford, UK

Doctoral Consortium

Table of contents

Doctoral Consortium

Preface

Truly, graduate students are the lifeblood of any field. They offer insight and innovation that comes from taking a fresh look. They contribute tremendous energy, dedication and enthusiasm to the field. Clearly, this is as true of the ICAPS community as any other scientific field.

The Ph.D. students selected for participation are all currently enrolled in a Ph.D. program and have progressed enough in their research to have identified a topic and, at least, preliminary results with some quite close to graduation. They share a central focus on planning and scheduling problems in artificial intelligence and, based on their submission, a promise for exceptional research in their careers. They come from 10 countries on four continents and represent a diversity of approaches and subareas within our field.

The Doctoral Consortium is dedicated to giving these young researchers an opportunity to showcase their ideas to the ICAPS community as a whole and garner the attention of more senior people in the field who have volunteered to be their mentors. Some of the participants have only one or perhaps no faculty member at their home institution who specializes in planning and scheduling. The Doctoral Consortium facilitates their interacting with other researchers at all levels who share their passion for the field.

These proceedings contain extended abstracts from the participants. The topics are as broad in scope as the main program for the conference, and not surprisingly for graduate students, include areas rising in interest and importance such as mixed-initiative and model checking. The research projects presented are at different levels of maturity with some intriguing new ideas being proposed. Taken as a whole, the proceedings promise the all too rare opportunity to glimpse the future...

Organizers

- *Adele Howe, Colorado State University, USA*
- *Ruth Aylett, University of Salford, UK*

Identifying Algorithm Performance Factors for an Oversubscribed Scheduling Problem*

Laura Barbulescu
Computer Science Dept.
Colorado State University
Fort Collins, CO 80523
email: laura@cs.colostate.edu

Abstract

My research work focuses on identifying key algorithm performance factors for an oversubscribed real-world application: scheduling for the U.S.A. Air Force Satellite Control Network (AFSCN). A genetic algorithm (GA) has been previously found to excel on this problem. I designed an empirical study to discover what makes the GA a good fit for AFSCN scheduling. For experimental control purposes, I included local search and Squeaky Wheel Optimization (SWO) in the study. I found that the search space is dominated by plateaus, favoring algorithms like the GA and SWO, which take large steps through the space. Also, an initial version of local search, using a structured ordering of the neighbors was inefficient at plateau traversal. However, the performance of local search can be significantly increased by using a randomized ordering of the neighbors. In this paper, I present empirical results to characterize the search space and to explain the discrepancy in the performance of the two local search versions.

Introduction

When solving a real-world optimization problem, most researchers and practitioners focus on identifying an algorithm that can be shown to perform well. However, as Paul Cohen (Cohen 1995) writes in his book *Empirical Methods for Artificial Intelligence*: "It is good to demonstrate performance, but it is even better to explain performance". In this paper I summarize some of my research results on *explaining* algorithm performance for an oversubscribed scheduling application, scheduling for the AFSCN.

The AFSCN is currently responsible for coordinating communications between civilian and military organizations and more than 100 USAF managed satellites. Space-ground communications are performed using 16 antennas located at nine tracking stations around the globe. Customer organizations submit task requests to reserve an antenna at a tracking station for a specified time period based on the visibility windows between target satellites and tracking stations. Approximately 500 requests are typically received for a single day. Separate schedules are produced by a staff of human schedulers at Schriever Air Force Base for each day. Of the 500 requests, often about 120 unscheduled requests remain after the first pass of scheduling.

A problem instance consists of n task requests. A task request T_i, $1 \leq i \leq n$, specifies both a required processing duration T_i^{Dur} and a time window T_i^{Win} within which the duration must be allocated; we denote the lower and upper bounds of the time window by $T_i^{Win}(LB)$ and $T_i^{Win}(UB)$, respectively. Tasks cannot be preempted once processing is initiated. Each task request T_i specifies $j \geq 0$ pairs of the form (R_i, T_i^{Win}), each identifying a particular alternative resource (antenna) and time window for the task.

We obtained 12 days of data for the AFSCN application. The first seven days are from a week in 1992 and were given to us by Colonel James Moore at the Air Force Institute of Technology. We obtained an additional five days of data from schedulers at Schriever Air Force Base. I will refer to the problems from 1992 as the A problems, and to the more recent problems, as the R problems.

Previous research and development on AFSCN scheduling focused on minimizing the number of unscheduled requests (bumps). We designed a new evaluation criterion that schedules all the requests by allowing them to overlap and minimizing the sum of overlaps between conflicting tasks.

Research performed at AFIT identified a GA, Genitor (Whitley 1989), to perform well for the A problems, when minimizing the number of bumps. I compared Genitor to algorithms which employ domain specific heuristics to focus the search: a constructive method, HBSS (Bresina 1996) using a flexibility heuristic similar to the one defined by Kramer and Smith (Kramer & Smith 2003) and Gooley's algorithm (Gooley 1993). I found that Genitor performs better than HBSS and Gooley's algorithm. Given that the implementation of Genitor does not encode any domain specific heuristic, I designed an empirical study to investigate the reasons for such performance results. In the next sections, I summarize the results of my study and conclude with some future work directions.

*This research was sponsored the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-03-1-0233. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

Algorithm Performance

All the algorithms in this study represent a proposed schedule as a permutation (prioritization) of the tasks. A greedy schedule builder transforms a given permutation into a schedule, by assigning to each task, in the order in which they appear in the permutation, the earliest possible time slot on the first available resource (from the list of possible alternatives specified for that task).

To baseline the performance of Genitor. I implemented local search, using a domain-independent move operator, the *shift* operator. The neighbor corresponding to the position pair (x, y) is produced by *shifting* the job at position x into the position y, while leaving all other relative job orders unchanged.

Given the large neighborhood size, I use the shift operator in conjunction with next-descent hill-climbing. I implemented two versions of ordering the neighbors; we call these a structured version and a randomized version. The structured version chooses the position x by random, and checks in order all the neighbors obtained by shifting the job at position x into positions 0, 1, 2... accepting both equal moves and improving moves. The randomized version chooses both x and y by random, without replacement.

I also implemented Squeaky Wheel Optimization (SWO)(Joslin & Clements 1999), for purposes of experimental control. During one iteration, local search applies incremental changes to the current solution (by shifting only one task at the time); both Genitor (via the crossover operator) and SWO take large leaps in the search space, by moving multiple tasks simultaneously.

In a couple of previous studies, I found that SWO and Genitor perform better than local search implemented with the structured ordering of the neighbors. I hypothesized that large plateaus are present and therefore long leaps are needed to efficiently traverse the search space (Barbulescu, Whitley, & Howe 2004), (Barbulescu *et al.* 2004). Recently, I added to the set of algorithms the randomized local search. A comparison of these algorithms shows that: 1) Genitor and SWO perform best for minimizing bumps; the randomized local search also performs well, 2) SWO and the randomized local search perform best for minimizing overlaps, better than Genitor, and 3) The worst performer in our set is the structured local search. The results of the current comparison suggest that possibly the long leaps are not needed: the randomized local search performs as well as Genitor and SWO, for both objective functions. The balance of this section presents results characterizing the search space and reasons for the discrepancy in performance for the two versions of local search.

Plateaus in the Search Space

When analyzing local search performance, I found that most of the accepted moves during search are non-improving moves; search ends up randomly walking on a plateau until an exit is found. As further evidence for the presence of plateaus, I collected results about the number of schedules with the same *value* as the original schedule, when perturbing the solutions by operating all possible shifts. Both random and best known solutions were included in the study. The results show that: 1) More than 85% of the shifts result in schedules with the same value as the original one, when minimizing conflicts. 2) When minimizing overlaps, more than 65% of the shifts result in same value schedules.

To evaluate the number and the size of the plateaus in the permutation search, I performed an experiment to average the length of the random walk on a plateau starting from various solutions encountered during one run of the randomized local search. The results show that large plateaus are present in the search space; improving moves lead to longer walks on lower plateaus. For the AFSCN scheduling problems, most of the states on a plateau have at least one neighbor that has a better value (this neighbor represents an exit). However, the number of such exits represents a very small percentage from the total number of neighbors and therefore local search has a very small probability of finding an exit. If there are no exits from a plateau, the plateau is a local minimum.

Determining which of the plateaus are local minima (by enumerating all the states on the plateau and their neighbors) is prohibitive because of the large size of the neighborhoods and the large number of equally good neighbors present for each state in the search space. Instead, I focused on the average length of the random walk on a plateau. This length depends on two factors: the size of the plateau and the number of exits from the plateau. The number of improving neighbors for a solution decreases as the solution becomes better; I conjecture that there are more exits from higher level plateaus than from the lower level ones. This would account for the trend of needing more steps to find an exit when moving to lower plateaus (corresponding to better solutions). It is also possible that the plateaus corresponding to better solutions are larger in size; however, enumerating all the states on a plateau for the AFSCN domain is impractical (following a technique developed by Frank (Frank, Cheeseman, & Stutz 1997), just the first iteration of breadth first search would result in approximately $0.8 * (n-1)^2$ or approximately 162,000 states on the same plateau).

Structured versus Random Ordering of the Neighbors

The poor performance or structured local search led us to conclude that the plateaus likely precluded effective local search (Barbulescu, Whitley, & Howe 2004). Recently we found that randomized local search performs as well as the best algorithms in our set. In this section, I offer an explanation of why the ordering of the neighbors was a particular problem.

I hypothesized that structured local search is stuck on plateaus for a longer time than randomized local search. To check if this is the case, I counted the average number of the evaluations that resulted in worse, equally good and better solutions over 30 runs of both local search versions, with 8000 evaluations per run. The results obtained for minimizing overlaps are summarized in Table 1. I obtained similar results for minimizing bumps. The number of worse neighbors evaluated by local search is significantly higher for the structured version of search. Given the high percentage of equally valued neighbors, both versions of search

	Average % Worse		Average % Equal				Average % Better	
			Identical sch.		Non-Identical sch.			
Day	Rand	Struct	Rand	Struct	Rand	Struct	Rand	Struct
A1	20.1	87.8	0.03	0.16	79.1	11.7	0.6	0.3
A2	16.6	86.0	0.02	0.04	82.8	13.6	0.5	0.2
A3	19.3	86.8	0.02	0.08	80.0	12.7	0.6	0.3
A4	21.3	88.5	0.03	0.02	77.8	11.0	0.7	0.3
A5	19.5	87.3	0.03	0.05	79.8	12.2	0.6	0.3
A6	21.3	87.5	0.01	0.07	77.9	11.9	0.6	0.3
A7	16.8	85.7	0.02	0.1	82.6	13.8	0.5	0.3
R1	32.2	91.0	0.01	0.03	65.8	8.5	1.8	0.4
R2	26.8	90.4	0.02	0.04	71.7	9.1	1.3	0.3
R3	24.7	89.9	0.03	0.09	74.0	9.6	1.1	0.3
R4	23.6	89.5	0.02	0.02	75.2	10.0	1.0	0.3
R5	17.9	87.8	0.02	0.1	81.3	11.6	0.7	0.3

Table 1: Minimizing overlaps: Average percentage of evaluations (out of 8000) resulting in worse, equally good or improving solutions over 30 runs of local search.

spend most of the time accepting equally good moves; only a small number of improving steps are taken. However, since the structured local search spends more than 80% of the time evaluating worse moves, it is left with only approximately 15% of the evaluations to move through the space. The structured local search needs more evaluations to find good solutions. This is also emphasized by the number of improving neighbors found by the two versions of local search: the randomized version always finds more improving neighbors (twice as many as the structured version, or even more).

The poor performance of the structured local search is a consequence of an unfortunate interaction between the schedule builder and the domain. Shifting a request into consecutive positions is likely to result in identical schedules. If shifting a request results in a neighbor that is worse than the current value of the solution, shifting the same request in subsequent positions will also be worse (until a position is found that changes the corresponding schedule). In effect, for our initial implementation, when local search encounters a worse neighbor, the probability of the next neighbor being also worse increases. To fix this problem, when a worse neighbor is encountered, instead of checking subsequent positions to shift x, both x and y should be chosen again, by random. This is in effect what the randomized version of search is doing.

Conclusions and Future Work

The permutation search space of the AFSCN scheduling is dominated by large size plateaus. Poor local search performance in some initial studies suggested that large leaps are needed to traverse these plateaus. Recently, I found that a new version of local search performs well for this problem. This is in contradiction with my initial hypothesis, because it suggests that multiple simultaneous moves (leaps) are not necessary for good performance. I conjecture that in fact multiple moves are a factor in algorithm performance, in the sense that they can speed up the convergence of an algorithm. For example, preliminary evidence shows that SWO moving only one task forward performs poorly (Barbulescu, Whitley, & Howe 2004). This is a first research topic I am currently investigating. A second topic of my current research is focused on answering the question: How do the results of my study change if a different approach (instead of the greedy one) is used for the schedule builder? Finally, I am working on generalizing the results for other oversubscribed scheduling problems.

References

Barbulescu, L.; Howe, A.; Whitley, L.; and Roberts, M. 2004. Trading places: How to schedule more in a multi-resource oversubscribed scheduling problem. In *Proceedings of the International Conference on Planning and Scheduling*.

Barbulescu, L.; Whitley, L.; and Howe, A. 2004. Leap before you look: An effective strategy in an oversubscribed problem. In *Proceedings of the Nineteenth National Artificial Intelligence Conference*.

Bresina, J. 1996. Heuristic-Biased Stochastic Sampling. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 271–278.

Cohen, P. R. 1995. *Empirical Methods for Artificial Intelligence*. Cambridge, MA: MIT Press.

Frank, J.; Cheeseman, P.; and Stutz, J. 1997. When gravity fails: Local search topology. *JAIR* 7:249–281.

Gooley, T. 1993. Automating the Satellite Range Scheduling Process. In *Masters Thesis*. Air Force Institute of Technology.

Joslin, D. E., and Clements, D. P. 1999. "Squeaky Wheel" Optimization. In *JAIR*, volume 10, 353–373.

Kramer, L., and Smith, S. 2003. Maximizing flexibility: A retraction heuristic for oversubscribed scheduling problems. In *Proceedings of 18th International Joint Conference on Artificial Intelligence*.

Whitley, L. D. 1989. The GENITOR Algorithm and Selective Pressure: Why Rank Based Allocation of Reproductive Trials is Best. In Schaffer, J. D., ed., *Proc. of the 3rd Int'l. Conf. on GAs*, 116–121. Morgan Kaufmann.

Scaling Decision Theoretic Planning

Daniel Bryce
dan.bryce@asu.edu
Department of Computer Science and Engineering
Ira A. Fulton School of Engineering
Arizona State University, Brickyard Suite 501
699 South Mill Avenue, Tempe, AZ 85281

Abstract

As classical planning branches out to consider richer models, many extensions approach decision theoretic models. Decision theory research uses models like MDPs and POMDPs which are very expressive, but can be dif cult to scale. Whereas, planning research concentrates quite a bit on scalability. Our previous work and future doctoral thesis concentrates on extending the planning model toward problems characterized by partial observability, non determinism, non uniform cost, and utility.

Introduction

One of the major reasons recent planners are able to tackle problems where solutions have on the order of hundreds of actions is the advent of reachability heuristics. Reachability heuristics rely on a structure called the planning graph, originally introduced to perform search in GraphPlan [Blum and Furst, 1995]. While GraphPlan was able to outperform many of the planners at the time, it has been subsumed by a planners such as HSP [Bonet and Geffner, 1999], FF [Hoffmann and Nebel, 2001], and AltAlt [Nguyen *et al.*, 2002] that use reachability heuristics extracted from the planning graph to guide state space search.

The basic idea in a planning graph is to approximate all states at each depth of the state space progression tree by a set of literals. The approximation loses boundaries between states at a depth of the search, but keeps the literals that can be reached at each depth. The planning graph encodes a lower bound k on the number of steps needed to achieve a state with a literal l if l doesn't appear in the planning graph until layer k. While this lower bound is a useful heuristic, it can be improved by using a relaxed GraphPlan search to nd actions that are needed to support the literal by backchaining through the planning graph. This heuristic, called a relaxed plan, has been used with much success in many planners.

While we would like to use reachability heuristics in the more expressive planning models, the unmodi ed GraphPlan relaxation tends to break down. Our previous work, discussed in the next section, looks at extending the planning model to include partial observability, where the key challenge is generalizing reachability heuristics between sets of states (belief states). In the following section, on our current work, we discuss extensions that are needed to handle non uniform cost models for actions that are both causative and sensory. Finally, in the last section we discuss further extensions to stochastic belief states and actions and utility models.

Partial Observability

Planning problems with partially observable states can be posed as search in belief space, where search nodes are sets of states and solutions are conformant and conditional plans. We investigated using planning graph heuristics for search in belief space. Intuitively, it can be argued that the heuristic merit of a belief state depends on at least two factors–the size of the belief state (i.e., the uncertainty in the current state), and the distance of the individual states in the belief state from the goal (belief) state. The question of course is how to compute these measures and which are most effective. Generalizing classical planning heuristics that aggregate the cost of literals in a state to get a state to state distance, we can aggregate the cost of states in a belief state to get a belief state to belief state distance. We evaluated heuristics that make various assumptions about state independence, positive interaction, and negative interaction.

We [Bryce and Kambhampati, 2004] rst tried a minimal extension to heuristics used in classical planning by considering heuristics from a single planning graph to guide search, which proved not to scale our planners very well. To improve the informedness of the heuristics, we tracked multiple planning graphs, each corresponding to one of the possible states in our belief. The number of planning graphs needed is exponential in the number of uncertain state literals. Hence, multiple graphs do not scale well as the size of belief states grow. The limitations in scaling involve either potentially running out of space to build planning graphs or spending too much time computing heuristics across the multiple planning graphs. Thus, we designed a new planning graph structure to addresses these limitations. The idea is to condense the multiple planning graphs to a single planning graph, called a Labelled Uncertainty Graph (LUG) [Bryce *et al.*, 2004]. Loosely speaking, this single graph unions the causal support information present in the mul-

tiple graphs and pushes the disjunction, describing sets of possible worlds, into "labels". The graph elements are the same as those present in multiple graphs, but much redundancy is avoided. For instance an action that was present in all of the multiple planning graphs would be present only once in the LUG and labelled to indicate that it is applicable in a projection from each possible world. We showed how several powerful planning graph-based heuristics from classical planning, including relaxed plans can be generalized to the case of multiple planning graphs and the LUG.

Our results showed that multiple graphs were needed but too costly, and that the LUG was able to capture the multiple graphs at a much lower cost. We found that both conformant planners and contingent planners can use these heuristics for improving scalability.

Cost Models

Currently we are working on heuristics that handle non uniform cost models when planning under partial observability. The reason cost models are interesting in partially observable problems are that planners need to be sensitive to the cost of sensing. For example, in planning medical treatment the cost of performing every test to perfectly diagnose a disease is too high, rather one may have prescribe several treatments at a lower cost, knowing at least one will work. We have taken the LUG from the previous section and de ned a cost propagation procedure which allows us to extract cost aware relaxed plans.

Cost propagation on planning graphs, similar to that used in the Sapa planner [Do and Kambhampati, 2003], propagates the estimated cost of reaching literals at different times. The propagated costs enable relaxed plans to be more cost sensitive by caching the least-cost supporters for subgoals. Our situation is a bit more general because we propagate cost for a set of states (the states in a belief). The biggest challenge in our generalization is that it is possible for a literal to have different costs for every possible subset of states in our belief. Instead of tracking cost for all subsets, we partition states into x ed sets to track cost over. We propagate cost for each graph element, in terms of these sets, with a set of world group-cost tuples.

Using the cost aware heuristics in our planner, $POND$, we were able to trade off the amount acting under uncertainty versus sensing based on the cost model of the problem. We compared our planner using relaxed plans not based on cost (coverage), and our planner using relaxed plans based on cost (cost) to GPT [Bonet and Geffner, 2000], and MBP [Bertoli *et al.*, 2001], on the following domains.

Medical-Specialist: We developed an extension of the medical domain [Weld *et al.*, 1998], where in addition to staining, counting of white blood cells, and medicating, one can go to a specialist for medication and there is no chance of dying – effectively allowing conformant plans. We assigned costs as follows: c(stain) = 5, c(count_white_cells) = 10, c(inspect_stain) = X, c(analyze_white_cell_count) = X, c(medicate) = 5, and c(specialist_medicate) = 10. We generated ten problems, each with the respective number of diseases (1-10), in two sets where X = $\{15, 25\}$.

Our results in the rst two columns in Figures 1, 2, and 3 show the expected cost, plan breadth, and total time for the two cost models. Extracting relaxed plans based on propagated cost, instead of coverage enables $POND$ to be more cost sensitive. The plans returned by the cost propagation method tend to branch less than coverage as the cost of sensing increases in order to reduce expected cost. Since MBP is insensitive to cost, the cost of its plans are proportionately worse as the sensor cost increases. GPT returns better plans than MBP, but tends to take signi cantly more time as the cost of sensing increases; this can be attributed to how their heuristic is computed by relaxing the problem to full-observability. Our heuristics measure the cost of coachieving the goal from a set of states, whereas GPT takes the max cost of reaching the goal among the states.

Rovers: We use an adaptation of the Rovers domain from the International Planning Competition [IPC, 2003] where there are several locations with *possible* science data (images, rocks, and soil). We added sensory actions that determine the availability of scienti c data and conditional actions that conformantly collect data. Our action cost model is: c(sense_visibility) = X, c(sense_rock) = Y, c(sense_soil) = Z, c(navigate) = 50, c(calibrate) = 10, c(take_image) = 20, c(communicate_data) = 40, c(sample_soil) = 30, c(sample_rock) = 60, and c(drop) = 5. The two versions have costs: (X,Y,Z) = {(35, 55, 45), (100, 120, 110)}.

The second two columns of Figures 4, 5, and 6 show the expected cost, plan breadth, and total time for the two cost models. We found that the coverage and cost based relaxed plan extraction guide POND toward similar plans, in terms of expected cost and number of branches. As sensing becomes more costly, POND is able to nd plans with less branches to preserve a good expected cost. MBP, making no use of action costs, returns plans with considerably (a order of magnitude) higher expected execution costs, and does not adjust its branching as sensing cost increases. GPT fares better than MBP in terms of plan cost, but both are limited in scalability due to the weaker relaxations in their heuristics.

Future Work

With knowledge of how to propagate cost on the LUG, we are con dent that we can propagate probabilistic information in a similar fashion to extract relaxed plans re ecting the stochastic dynamics of a problem. A relaxation we would like to explore is the effectiveness of using order of magnitude approximations of probability in the propagation.

We also intend to extend the methods mentioned up to this point with models that have goal utility. There exists work by our research group on handling goal utility in planning graphs [van den Briel *et al.*, 2004]. It is our intent to combine these algorithms into construction of the LUG and relaxed plan extraction.

Handling each feature of decision theoretic planning individually sheds light on the larger problem, and it is our plan to incrementally study and combine heuristic techniques for each feature to create a holistic heuristic method. With such a heuristic method for decision theoretic planning, we intend to approach problems that were previously deemed too large

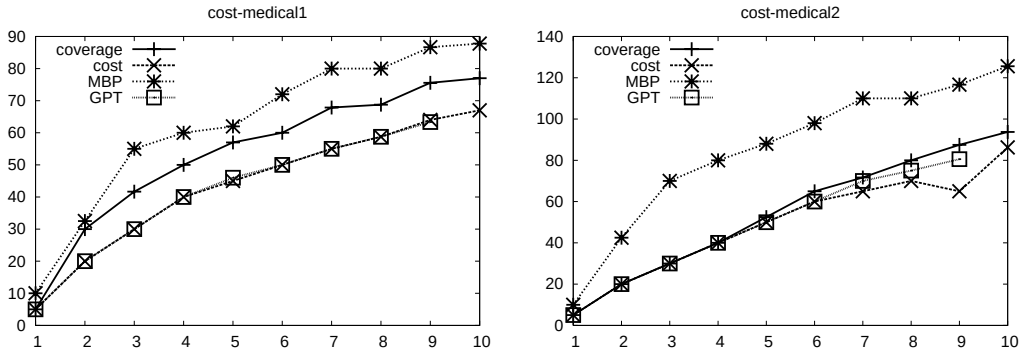


Figure 1: *Expected cost results for $PONDA$ (coverage and cost), MBP, and GPT for Medical-Specialist.*

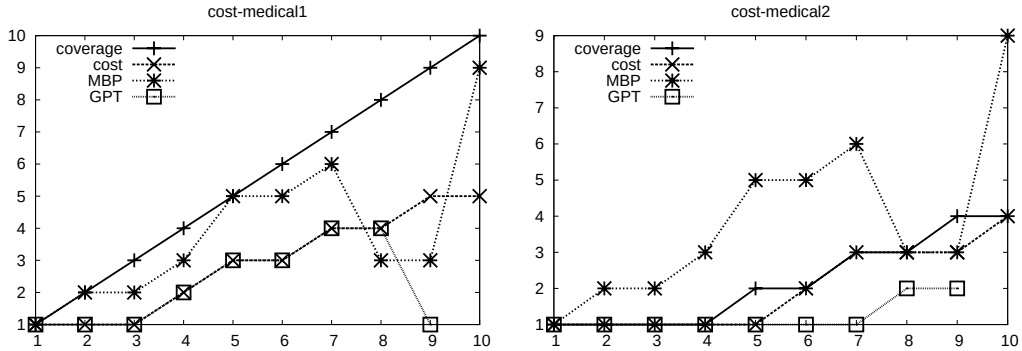


Figure 2: *Breadth (# of plan paths) results for $PONDA$ (coverage and cost), MBP, and GPT for Medical-Specialist.*

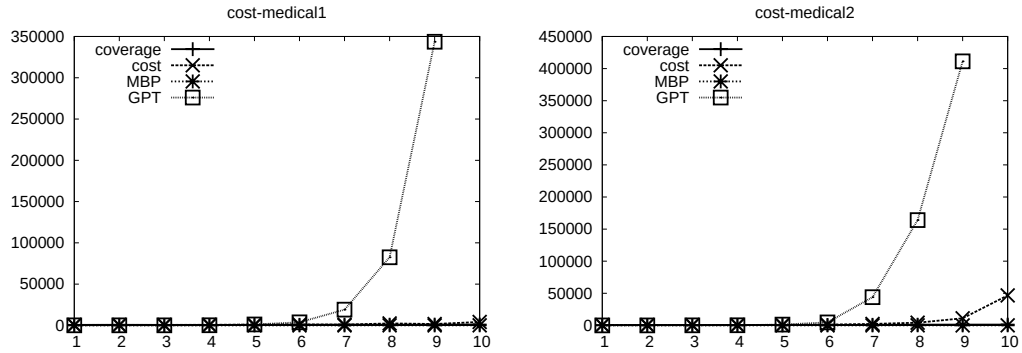


Figure 3: *Total Time(ms) results for $PONDA$ (coverage and cost), MBP, and GPT for Medical-Specialist.*

for traditional (optimal) solution algorithms. We note that relaxed plan heuristics are effective yet inadmissible, thus leading to non-optimal solutions. We believe that something has to give when trying to solve large problems, and we trade provable optimality for scalability.

References

Piergiorgio Bertoli, Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In Bernhard Nebel, editor, *Proceedings of the seventeenth International Conference on Artificial Intelligence (IJCAI-01)*, pages 473–486, San Francisco, CA, August 4–10 2001. Morgan Kaufmann Publishers, Inc.

Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 1636–1642, 1995.

Blai Bonet and Hector Geffner. Planning as heuristic search: New results. In *Proceedings of the Euoropean Conference of Planning*, pages 360–372, 1999.

Blai Bonet and Hector Geffner. Planning with incomplete information as heuristic search in belief space. In *Artificial Intelligence Planning Systems*, 2000.

Daniel Bryce and Subbarao Kambhampati. Heuristic guidance measures for conformant planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*, June 2004.

Daniel Bryce, Subbarao Kambhampati, and David E. Smith. Planning in belief space with a labelled uncertainty

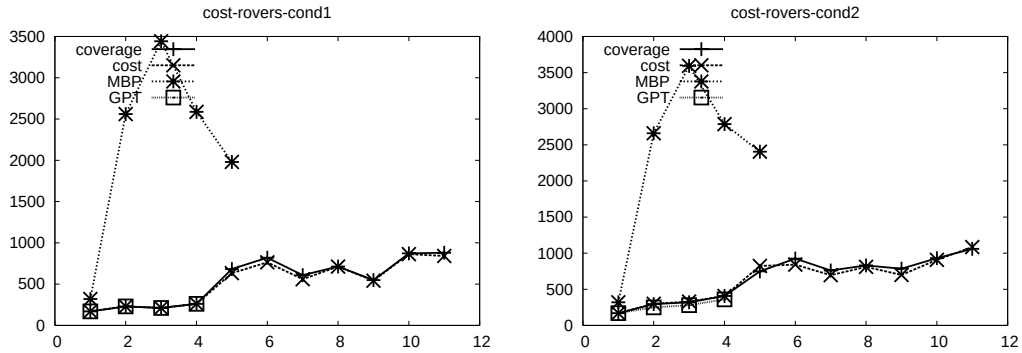


Figure 4: *Expected cost results for $POND$ (coverage and cost), MBP, and GPT for Rovers.*

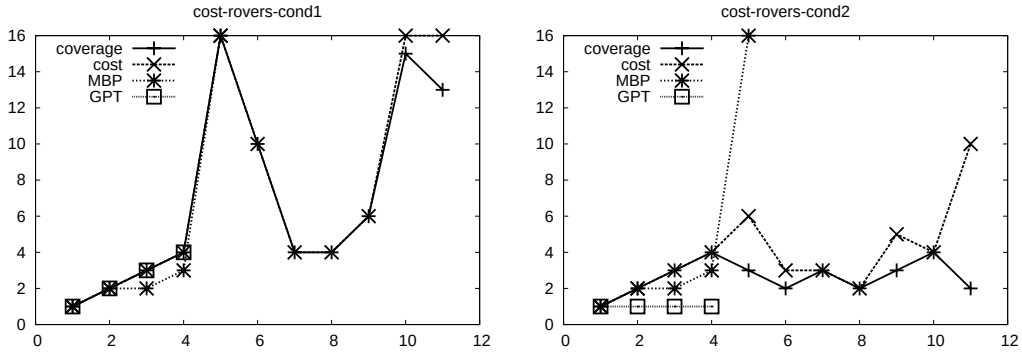


Figure 5: *Breadth (# of plan paths) results for $POND$ (coverage and cost), MBP, and GPT for Rovers.*

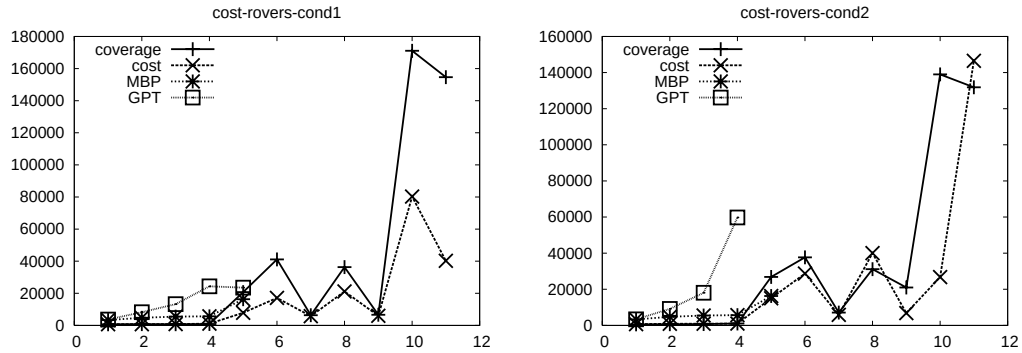


Figure 6: *Total Time(ms) results for $POND$ (coverage and cost), MBP, and GPT for Rovers.*

graph. Technical report, AAAI Workshop TR WS-04-08, 2004.

Minh Binh Do and Subbarao Kambhampati. Sapa: A scalable multi-objective heuristic metric temporal planner *JAIR*, 2003.

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

Volume 20, special issue on the 3rd international planning competition. *Journal of Artificial Intelligence Research*, 2003.

XuanLong Nguyen, Subbarao Kambhampati, and Romeo Sanchez Nigenda. Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *Artificial Intelligence*, 135(1-2):73–123, 2002.

Menkes van den Briel, Romeo Sanchez Nigenda, Minh Binh Do, and Subbarao Kambhampati. Effective approaches for partial satisfaction (over-subscription) planning. In *Proceedings of the AAAI conference*, 2004.

Daniel S. Weld, Corin Anderson, and David E Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-98)*. AAAI Press, 1998.

Abstract Scheduling Models *

Tom Carchrae
Cork Constraint Computation Center
University College Cork, Ireland
t.carchrae@4c.ucc.ie

Introduction

Despite both the commercial and academic success of optimization technology and specifically constraint programming, using the technology still requires significant expertise. For non-trivial applications the quality of a system still has much to do with the quality of the person that implemented it (Le Pape *et al.* 2002). In this paper, we investigate techniques that reduce the expertise to required to achieve strong scheduling performance from existing techniques (off-the-shelf algorithms). We have approached the problem from two angles: automated selection and control of algorithms to solve the problem *and* the automated selection of an abstract model to reduce complexity.

In our previous work on algorithm control, we used online reinforcement learning to determine which techniques perform well on a problem. The learning algorithm allocates CPU time based on performance and rewards the best performers by giving them more CPU time than algorithms which do not perform well. We compared this approach with the theoretical optimal choice of a single algorithm and find that we achieve competitive performance without building complex prediction models.

Here we present the preliminary findings of a second approach, in the form of abstract scheduling models. We reduce the complexity of a scheduling problem by solving it in two stages. First, we create an abstract model which is an approximation of the original problem and optimize this. We then use the solution to the abstract model to find a solution to the original problem, a process called refinement. By breaking the problem solving into two phases we reduce the complexity of solving the problem. We observe a tradeoff of runtime performance against error caused by approximation in the abstract model.

Together, we believe that these techniques will allow the development of a toolkit which can be used to achieve superior scheduling performance without expert knowledge of the underlying algorithms or expertise in problem modelling. This direction of research is important for the widespread use of optimization technology. Aside from simplifying the installation of an optimization system, these methods can also be used to maintain the system so that it becomes more robust to changes in the problems it solves by adapting the way it solves the problem.

*This work has received support from Science Foundation Ireland under Grant 00/PI.1/C075, Irish Research Council for Science, Engineering, and Technology under Grant SC/2003/82, and ILOG, SA.

Previous Work

In our previous work (Carchrae & Beck 2004) we have shown that low-knowledge metrics of pure algorithm behavior can be used to form a system that consistently and significantly out-performs the best pure algorithm. A low-knowledge approach has very few, inexpensive metrics, applicable to a wide range of algorithms and problem types. A high-knowledge approach has more metrics, that are more expensive to compute, and more specific to particular problems and algorithms. Machine learning techniques played an important role in this performance, however, even a simpleminded approach that evenly distributes increasing runtime among the pure algorithms performs very well.

Our motivation for investigating low-knowledge algorithm control was to reduce the expertise necessary to exploit optimization technology. Therefore, the strong performance of these techniques should be evaluated not simply from the perspective of an increment in solution quality, but from the perspective that this increment has been achieved without additional expertise. We neither invented a new pure algorithm nor developed a detailed model of algorithm performance and problem instance features.

Abstract Scheduling Models

It is well recognized that the efficiency of solving a problem is not only related to the choice of algorithm but also how the problem is modelled. To this end, we present a method which attempts to determine a set of critical resources R_A in a scheduling problem which have the greatest impact on the cost of the schedule. We then produce an abstract model involving only activities which require a resource $R \in R_A$. Since there are less activities in this abstract model, we reduce the complexity of the original problem. The abstract model approximates the effects of the other activities by posting precedence constraints in their place. We search for a good solution to the abstract model and then extend the abstract solution to a full solution to the original problem.

Abstract Model Selection

We use a simple heuristic to select critical resources R_A based on resource load. The load of a resource is the sum of durations for all activities that require that resource. We sort the resources by load and choose the n-most loaded resources, where n is the fraction of resources we consider in the abstract model. To evaluate this heuristic we also try the anti-heuristic, n-least loaded resources, and n-random resources which selects resources randomly.

Creating the Abstract Model

To create the abstract model, we include only activities which require a resource $R \in R_A$ and remove all other activities. Each activity belongs to a job, which is a sequence of activities, connected by precedence constraints. When we remove an activity, we post a new precedence constraint between the preceding and succeeding activities of the removed activity. This precedence has a minimum gap equal to the duration of the removed activity, $gap = duration(a)$. That is, for an activity a, we post a precedence constraint between the previous activity in the job $prev(a)$ and the next activity in the job, $next(a)$ of the form

$$endtime(prev(a)) \leq gap + starttime(next(a)) \quad (1)$$

which requires that activity $next(a)$ starts at least gap time units after the end of activity $prev(a)$. While we do not compute the time it takes for a to be scheduled, this reserves time for a to be processed in the full solution. Note that for the approximation to be correct, it assumes that we will be able to schedule activity a immediately on the resource it requires. Otherwise, the amount of time between $prev(a)$ and $next(a)$ will be greater than our approximation.

There are however some special cases. If either $prev(a)$ or $next(a)$ have also been removed from the abstracted model then we must reserve a larger gap than $duration(a)$. To do this, we replace $gap = duration(a)$ with the sum of durations from the set I of removed activities which are in-between the pair of activities that remain in that job, $gap = \sum_{a \in I} duration(a)$.

There is one more case we must handle. If we remove the activity at the beginning or end of a job, we are unable to post the precedence constraint as we are lacking the $prev(a)$ or $next(a)$ activities. For the beginning of the job, we can simply replace the preceding activity $prev(a)$ with the earliest start time of the first activity. To handle missing activities at the end of the job, we replace $next(a)$ with the makespan which represents the latest time in the schedule. This ensures that any solution to the abstract model will reserve at least gap time units at the beginning and end of the job to allow room for the removed activities to be scheduled.

Refinement

Once we have a good solution to the abstract model we can refine it into a solution to the original problem. The refinement approach we have used is to fix the sequence of activities in the abstract solution. We do this by adding precedence constraints to the original problem between each consecutive pair of activities in the abstract solution sequence. This effectively fixes these activities so that no search is required to schedule them. Since they are precedence constraint it allows some movement in the refined solution.

The reader may be concerned that this refinement approach could lead to insoluble problems. However, this is not the case. The addition of precedence constraints to represent the removed activities ensures that any solution to the abstract model will also be a solution to the original problem. We have not removed any constraints, merely approximated the time some activities will take.

Experiments

The purpose of this experiment was to determine the effectiveness of the simple n-most loaded heuristic in choosing abstract scheduling models. We measured performance in terms of quality of solutions and running time when optimizing the makespan of the schedule. Small problems were chosen and all were solved to optimality. We generated a set of 100 instances of job shop scheduling problems with 10 jobs and 10 machines. Activity durations were drawn randomly with uniform probability from the interval [1, 99].

We ran each of the model selection heuristics, n-most loaded, n-least loaded and n-random on each of the problems for values of $n = \{0.1, 0.25, 0.5, 0.75, 0.9\}$. ILOG Scheduler's settimes algorithm(Scheduler 2004) was used to solve both the abstract model and the refined problem. Each problem instance was also solved without abstraction to determine the optimal makespan for comparison.

We evaluate the quality of solutions using mean relative error(MRE), where

$$MRE(a, K) = \sum_{k \in K} \frac{c(a,k) - c^*(k)}{c^*(k)} \quad (2)$$

where K is the set of 100 problem instances, $c(a, k)$ is the lowest cost found by algorithm a on k, and $c^*(k)$ is the optimal cost for k. Cost refers to the makespan of the solution.

Results

In Figure 1 we see the the MRE results for the different heuristics. It appears that the n-most loaded heuristic is superior to the other heuristics as it achieves a significantly lower MRE as n increases. It seems that for these problems using values of $n = 0.1$ and $n = 0.25$, the choice of heuristics does not make a big difference to the MRE. However, the difference becomes dramatic for $n > 0.5$ with n-most loaded performing best while n-random performs slightly better than n-least loaded. This indicates that resource load appears to be a useful feature for selecting the abstract model in terms of reducing the error caused by approximation.

The worst error using abstraction is when $n = 0.5$, regardless of the heuristic. This may be explained by the fact that at this point we are solving half of the problem in each phase. For $n < 0.5$ we fix a smaller number of resources with the abstract solution giving more freedom in the refinement stage. For $n > 0.5$ the abstract model considers more resources which reduces the approximation error, although the difference in heuristics indicate that the choice of which resources are selected is more crucial here.

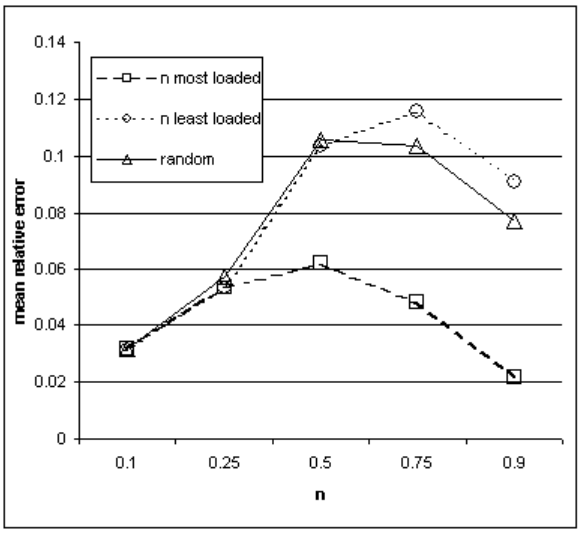


Figure 1: **Mean Relative Error(MRE)** for abstract model selection heuristics with $n = \{0.1, 0.25, 0.5, 0.75, 0.9\}$

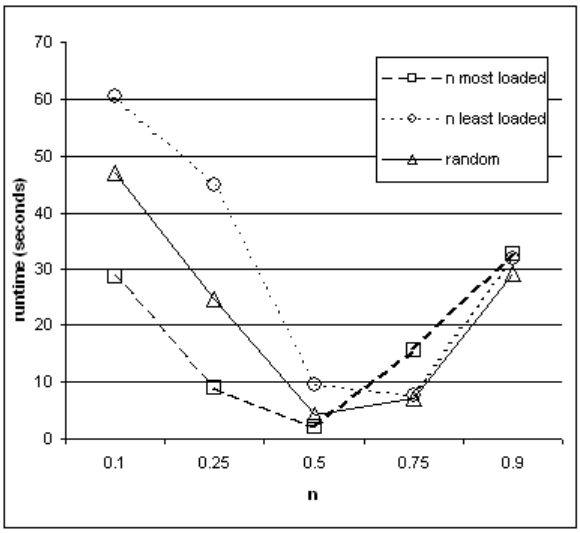


Figure 2: **Running Times** for abstract model selection heuristics with $n = \{0.1, 0.25, 0.5, 0.75, 0.9\}$. The mean runtime without abstraction was 87 seconds.

The total runtime required for solving the abstract model and the refined problem is shown in Figure 2. Again, we notice that there seems to be a cut point at $n = 0.5$, with the n-most loaded heuristic performing faster for lower values of n and slower for higher values of n, although the difference is not great on the higher values of n. There are several interesting things shown here. The fastest time to solve the problem is at $n = 0.5$ which makes sense. At this point, we are solving an abstract model with 5 resources and a refinement problem with 5 resources. Since the complexity of the problem is bounded by the number of resources we schedule, this gives us the fastest search. As we increase $n > 0.5$, then the abstract model contains more resources and becomes more complex. As we decrease $n < 0.5$ the abstract model becomes smaller but the refinement problem becomes harder.

One thing is clear, the abstract scheduling model reduces the time spent searching for solutions. The mean runtime to find the optimal solution was 87 seconds when solving problems without abstraction. In comparison, when using the n-most loaded heuristic the runtime is one third less for all values of n and significantly less than this for values closer to $n = 0.5$. However, this is not a fair comparison since the abstraction technique sacrifices completeness.

Although they are not shown here, we examined MRE using a runtime performance graph and found that no single heuristic was able to perform better, on average, than solving without abstraction. While disappointing, this is not that surprising given that our heuristics are very simple. However, we are interested to see if there exist good abstractions in general. In Figure 3 we show how often solving without abstraction was better than with abstraction. Our aim here is to show that, even though our simpleminded heuristics are not sufficient to reliably choose a good abstraction, it is possible to solve the problem better with abstraction if a limited amount of time is available.

Future Work

The work reported here is only the beginning of our research in abstract model reformulation. The heuristic used to choose the resources is very simple, and we are interested to try more involved methods. For instance, we would like to learn which resources we are able to approximate accurately. Can we learn a constant value which can be used to estimate the gap value better?

Another idea for a heuristic for choosing resources is to try to analyze the time windows where resource contention occurs. We could then expand our notion of abstraction to consider only time windows where contention occurs. It seems likely that certain times will be more difficult for resources to be scheduled.

We can also explore alternatives to a two stage approach. If we were to attempt different abstractions and refine them, perhaps we can learn which resources are most suitable for abstraction by experience.

Our mode of refinement fixes the the abstract solution which is only one of several approaches. We can instead use the abstract solution to guide a search heuristic in finding a refinement solution. We could also use the abstract solution as a starting point and then apply local search to improve the quality of the refined solution.

Conclusion

We have presented preliminary work in automating the task of modelling in order to reduce complexity when solving scheduling problems. This fits in our broad research objectives of reducing the expertise required to use optimization technology. Early results are promising and we are optimistic about further improvements.

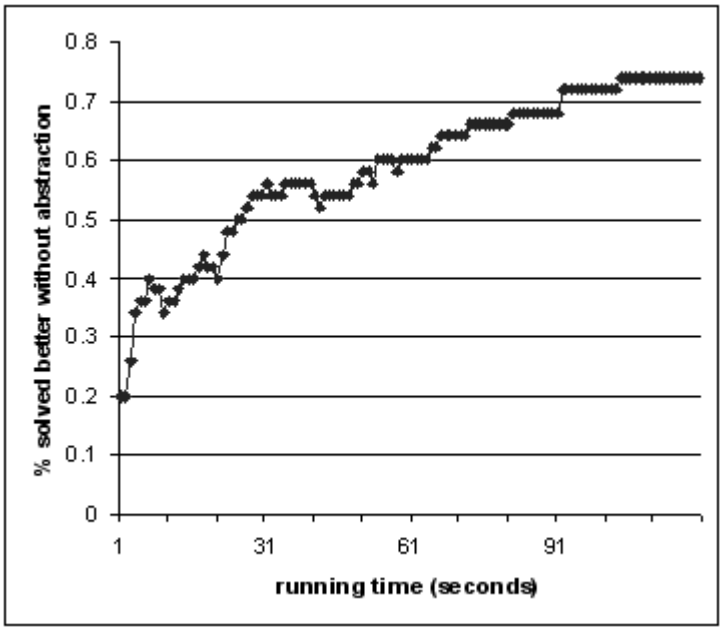


Figure 3: **Abstraction versus Pure Method** shows when solving the problem with abstraction performs better than without abstraction. Note, this is against all of the abstraction techniques combined. This graph shows that good abstract models exist which outperform the pure technique when running times are limited. We expect similar behavior when we scale up the problem size.

References

Carchrae, T., and Beck, J. C. 2004. Low knowledge algorithm control. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI04)*.

Le Pape, C.; Perron, L.; Régin, J.; and Shaw, P. 2002. Robust and parallel solving of a network design problem. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP02)*, 633–648.

Scheduler. 2004. *ILOG Scheduler 6.0 User's Manual and Reference Manual*. ILOG, S.A.

On-line Optimized Planning for Space Applications

S. Damiani
ONERA, Toulouse, France
Sylvain.Damiani@onera.fr

Context

Mission

This work is funded by the CNES (French Space Agency) and is part of more general studies about the advantages of on-board autonomy for satellites to respond to special space mission requirements. The mission that has been provided to us (Charmeau 2002) is an Earth watching mission, dedicated to the detection and the tracking of ground phenomena, such as forest fires or volcanic eruptions (Escorial, I.Tourne, & Reina 2001). The physical components involved are a Walker constellation of 12 low-orbiting satellites (LEO), 3 geostationary satellites (GEO), one ground mission control center and some ground reception stations. As shown on figure 1, each LEO is equipped with one detection instrument (permanently active and pointed in front of the satellite, with a wide swath) and one observation instrument (active on request, with a narrow swath and a controllable pointing direction for lateral observation).

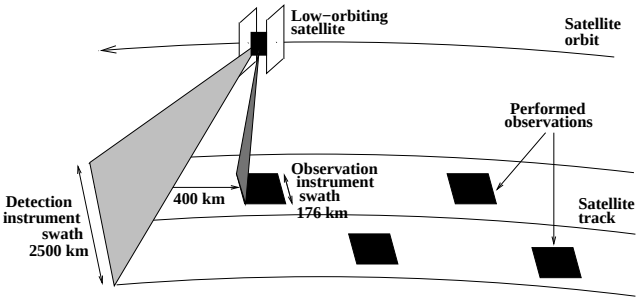


Figure 1: Detection and observation on-board each LEO satellite.

In case of a detection of a new hot ground phenomenon by a satellite, an alarm is sent via the GEO satellite currently in visibility, and an observation request is sent to the on-board decision module. Observation requests can also been sent by the mission control center to the LEO satellites via visibility windows. Whereas detection data are analyzed on-board, observation data are only compressed and memorized, and then downloaded to the dedicated reception stations via visibility windows. There is no direct communication between LEO satellites and, except alarms, communications between the ground and the LEO satellites are limited to visibility windows (see figure 2).

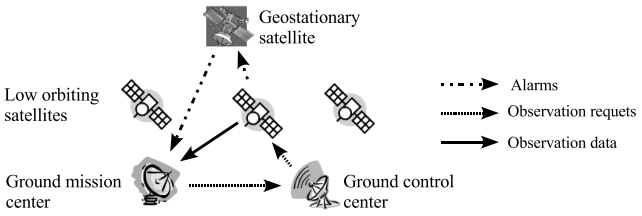


Figure 2: Possible communications between space and ground components.

Unlike traditional Earth observation missions (Lemaître *et al.* 2002) where activities are entirely planned on the ground, in this mission the balance between the decisional reactivity requirements for each LEO satellite (an observation must be triggered only one minute after the detection of a hot phenomenon) and the communications limitations between the mission control center and each satellite impose on-board decisional capabilities for two activities: hot phenomenon observation and data downloading. Then, the role of the ground is only to share the observation requests between the LEO satellites so as to improve the global return of the mission, knowing that unexpected events can prevent the LEO from triggering planned observations.

General approach

The mission presented above requires the design of three on-line decision mechanisms, each corresponding to an optimized planning problem. The basic principles we chose for the design are the following (see figure 3 for an example):

1. a reactive approach, rather than a proactive one, because we have no probabilistic model of unexpected events;

2. each decision is made as late as possible in order to take into account the most up-to-date information;
3. the decision horizon (the set of actions engaged following the decision) must be as short as possible in duration to enable quick reactions to unexpected events;
4. the reasoning horizon (the state space covered to make the decision) must be as large as possible in order to make as justified as possible decisions; knowing that, a time spread horizon enables better anticipation, but is also more likely to provide wrong assumptions to the decision-making because of unpredicted events that may occur during it.

When this approach is applied to the optimal control of a robot subject to severe real-time constraints, the need arises to find anytime algorithms (Zilberstein 1996) because they fit perfectly with the four principles.

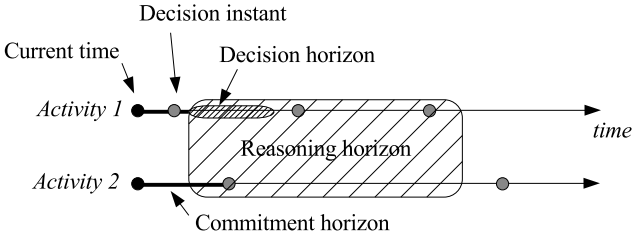


Figure 3: Commitment, decision, and reasoning horizons examples.

Application

The remaining of the paper is a brief presentation of the three planning problems and of the algorithms used to solve them. More information can be found in (Damiani, Verfaillie, & Charmeau 2004) and in a paper that is to be presented in the workshop on «Planning under uncertainty for autonomous systems».

Observation decision

An anytime planning module has been designed for the management of observation decisions: it generates optimal plans over an increasing number of observation requests in the future (see figure 4) in order to decide to trigger or not the first of those observations. The kind of algorithm used is direct dynamic programming with discretization of resources levels. What makes it possible such an efficient algorithm (with a time complexity which is a quadratic function of the number of considered requests) is that there is a natural ordering between observation requests. The anytime module is restarted each time initial planning assumptions (pool of current requests, resources levels) are modified in order to always produce valid results.

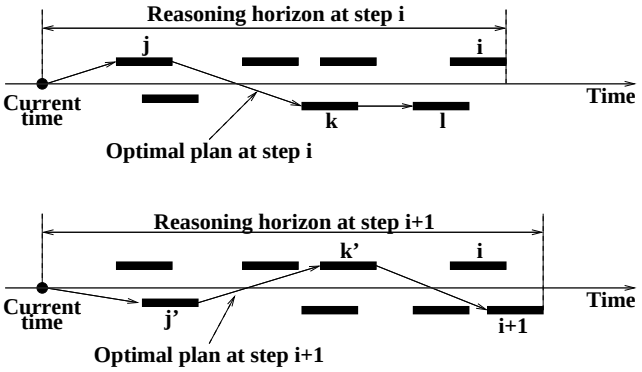


Figure 4: Reasoning on larger and larger horizons.

Data downloading decision

The second decision module selects data to download within the next mission center visibility window and schedules the resulting downloading actions over this limited time interval. This problem is actually a mix of the problem of 0-1 knapsack and of the problem of scheduling real-time tasks on a mono-processor (George, Muhlethaler, & Rivierre 1995), both of which being NP-complete. We use greedy algorithms based on efficient heuristics known for the latter problems first to select the images, then to build a valid schedule over the visibility window. This planning module is started just before each downloading opportunity, then the generated plan is entirely executed during the whole visibility window.

Coordination of on-board decision mechanisms
As, when data downloading is insufficient, the lack of free mass memory rapidly forbids new images to be recorded, this activity is undoubtedly a bottleneck of the system. In consequence, we decided to give it priority for the access to energy. In such conditions, data downloading is decided independently from observation (see figure 5), whereas observation planning takes as initial assumptions an estimate of the effects on resources of the next downloading plan (see figure 6). Both activities take into account the "service module" activity which represents all spacecraft activities that are uncontrollable as far as mission management is concerned, but which may have an impact on mission related decisions (e.g. permanent power consumptions, battery replenishment during day windows, observation impossibility during Attitude and Orbit Control System (AOCS) operations[1]).

[1]Note that unlike agile satellite management, here AOCS activities are decoupled from mission operations. Then, it is realistic to assume that they are independently and long term planned thanks to an on-board autonomous navigation system.

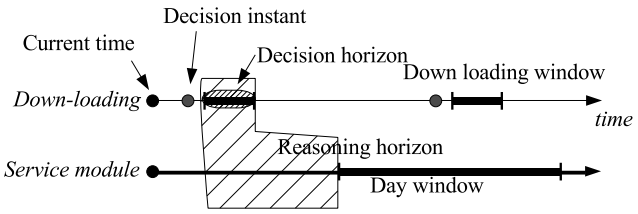


Figure 5: Downloading decision.

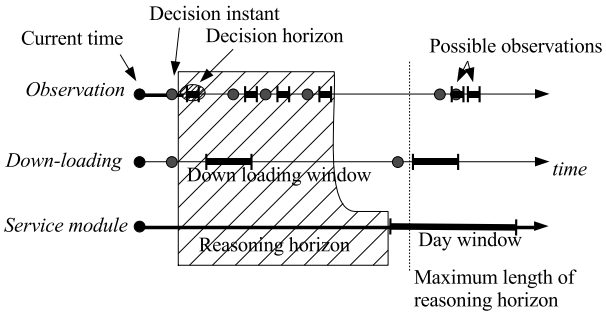


Figure 6: Observation decision. Only the first coming down loading window is included in the reasoning horizon.

Ground sharing of tracking tasks

We assume that each time it receives an alarm corresponding to a new forest fire or volcano eruption, the concerned mission center generates a tracking request, consisting on a set of successive dated observation requests separated by a given time interval (period of observation). We also assume that only one ground center (the control center) can send new requests to the LEO satellites when they are in visibility. As this center has access to all the current tracking requests and knows for each corresponding zone the set of candidate observations (i.e. when and by which satellite this zone will be observable), it can assign each observation of each tracking request to a satellite able to perform it, in such a way that all the tracking requests are satisfied as best as possible: observation times as close as possible to the reference times, data downloading as close as possible to the observation times.

Because requests can be uploaded to a LEO satellite only when this satellite is in visibility of the control center, the control center must prepare requests to this satellite just before a visibility window for the period between this window and the next one. For this period, it must share tracking tasks among all the satellites, taking into account the fact that for these satellites requests cannot be modified till their next visibility window (see figure 7). Like the algorithm used to plan for observations on-board, the one used for the sharing is based on dynamic programming and takes advantage of the natural ordering between all the candidate local assignments to produce optimal results; however the resource states of the satellites are not taken into account, and the final algorithm does not have anytime properties (they are not necessary in this context).

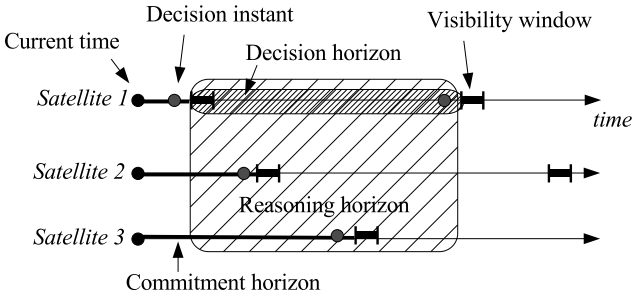


Figure 7: Successive ground sharing of tracking tasks.

Global consistency We have presented three relatively independent automated decision mechanisms embedded in the same system. Yet it should not be forgotten that there is one unique goal to this system, that is to provide information on all the current Earth hot points as regularly as possible. So the optimization algorithms used in each decision mechanism must be designed in such a way that they tend to optimize this global objective. For example, we assumed that the users would use priority levels to discriminate hot phenomena: in our implementation the priority level of each hot phenomenon is maintained in the three decision mechanisms. We also assumed that the users would prefer to get acceptable information on a each hot point than to get excellent information on some hot points and no information on the others: we then chose egalitarist criteria (leximin, maxmin) rather than additive ones.

Experiments

All the decision modules described above have been implemented and tested together in real-time on realistic scenarios, involving the 12 satellites, one mission control center and 2 reception stations. These experiments allowed us to validate the approach and the algorithms used, to check that the resource consumptions of both on-board decision mechanisms (in terms of computer power and RAM) are realistic in terms of implementation on a satellite. In further work we will try to show the impact of the ground sharing of tracking tasks on the global return of the constellation compared with cheaper scenarios where the satellites are completely autonomous.

Strong assumptions on communications capabilities between the agents have led us to a specific way to manage the mission provided by the CNES, and strong constraints on the model of actions have made it possible to design very efficient decision algorithms. To

complete the work, we will try to analyse to what extent the context can be generalized in order to determine the limits of the chosen approach and to compare with related work (Khatib *et al.* 2003; Rabideau *et al.* 2004).

References

Charmeau, M. 2002. Mission de Référence pour le DE Autonomie. Technical Report DTS/AE/SEA/IL/02-112, CNES.

Damiani, S.; Verfaillie, G.; and Charmeau, M.-C. 2004. An Anytime Planning Approach for the Management of an Earth Watching Satellite. In ESA., ed., *4th International Workshop for Planning and Scheduling for Space (IWPSS'04)*, 54–63.

Escorial, D.; I.Tourne; and Reina, F. 2001. FUEGO : A Dedicated Constellation of Small Satellites to Detect and Monitor Forest Fires. In *Third IAA Symposium on Small Satellites for Earth Observation*.

George, L.; Muhlethaler, P.; and Rivierre, N. 1995. Optimality and non-preemptive real-time scheduling revisited. Technical Report 2516, INRIA-Rocquencourt.

Khatib, L.; Frank, J.; Smith, D.; Morris, R.; and Dungan, J. 2003. Interleaved Observation Execution and Rescheduling on Earth Observing Systems. In *Proceedings of ICAPS'03 Workshop on Plan Execution*.

Lemaître, M.; Verfaillie, G.; Jouhaud, F.; Lachiver, J.-M.; and Bataille, N. 2002. Selecting and Scheduling Observations of Agile Satellites. *Aerospace Science and Technology* 6:367–381.

Rabideau, G.; Chien, S.; Sherwood, R.; Tran, D.; Cichy, B.; Mandl, D.; Frye, S.; Shulman, S.; Bote, R.; Szwaczkowski, J.; Boyer, D.; and Gaasbeck, J. V. 2004. Mission Operations with Autonomy: A preliminary report for Earth Observing-1. In *4th International Workshop for Planning and Scheduling for Space (IWPSS'04)*, 161–169.

Zilberstein, S. 1996. Using Anytime Algorithms in Intelligent Systems. *AI Magazine* 17(3):73–83.

Planning on demand in BDI systems

Lavindra de Silva and Lin Padgham

{*ldesilva,linpa*}*@cs.rmit.edu.au*
School of Computer Science and Information Technology
RMIT University, Melbourne 3000, VIC, Australia

1 Introduction

The BDI (Belief, Desire, Intention) model of agency is a popular architecture based on Bratman's (Bratman 1987) theory of practical reasoning. There are numerous implementations based on the BDI architecture such as the Procedural Reasoning System (PRS) and JACK[1], which are used for both academic and industrial purposes.

An important aspect of BDI style systems is that they execute as they reason, and so avoid the possibility of the reasoning being outdated, due to environmental change, by the time execution happens. They are also very fast, and therefore well suited to systems needing to operate in real time, or close to real time environments. However, there are no generic mechanisms in BDI systems to do any kind of look-ahead, or planning. In some situations this would be desirable.

The primary goals and contributions of our work are: 1) investigating the similarities and differences between BDI systems and HTN systems, as well as an exploration of the environments and problem types that would suit one system better than the other; 2) incorporating HTN planning at specific points in a BDI application, on an as needed basis, under control of the programmer; 3) planning using only limited subsets of the application, making the planning more efficient, and; 4) incorporating the plan generated back into the BDI system, for regular BDI execution, identifying plan steps that could be pursued in parallel; 5) formalising our system within the framework for formalisations of BDI systems and undertaking evaluation with regard to application usability.

There is some previous work that deals with using planning capabilities to guide the execution of BDI-like systems. Some of the research closely related to ours is Propel (Levinson 1995), Propice-Plan (Despouys & Ingrand 1999), and RETSINA (Paolucci *et al.* 1999). In these systems, planning is done every time a runtime failure occurs, or planning takes priority over reactive execution. However in real BDI applications, planning may not always be necessary. When planning is necessary, it should only be used in parts of the application where it is necessary, and control should be returned back to normal BDI execution on completion of planning. Our work focusses on these requirements of real world BDI applications.

A simplified and popular view of the BDI architecture is in terms of *goals* and *recipes*, where a goal has one or more recipes that can be used to achieve it. The recipes to achieve goals are stored in a library provided by the programmer. When an agent has a goal to achieve, it looks for a recipe that can achieve the goal in the current state of the world. If a suitable recipe is found, it is executed. If it fails during execution, the agent looks for other suitable recipes to achieve the goal. The goal fails if none of its recipes could be executed to completion, or if none of them were suitable for the current state of the world. In achieving a goal, the agent typically executes a number of steps, or *subgoals/subtasks*. In some situations there can be multiple options (recipes) at each step, but for a given state, only certain combinations of choices will lead to success of the overall goal. However, it may not be possible to (easily) encode information enabling successful choices, based only on knowledge of the current state. Therefore it would be advantageous to have a simple mechanism that incorporates planning in a generic way, at particular points where it is needed.

For example, consider an ambulance dispatch system: as ambulance requests arrive, a dispatching agent ascertains the best ambulance to dispatch and provides ongoing directions during the process of dealing with the emergency. The steps that are part of servicing the emergency are *FindAmbulance*, *DispatchAmbulance* and *SendToHospital*, in that order, as shown in Figure 1 (ignore recipes in bold for now). Normally the FindAmbulance goal is achieved by executing the *FindSuitableIdle* recipe. This is a recipe type representing a set of recipe instances. These instances are determined at runtime, one for each idle ambulance. Each of these recipe instances will be tried in turn, attempting to successfully complete the subtasks of *CheckTravelTime* and *CheckConstraints*. If no recipe instance of this type completes successfully, instances of the alternative recipe type *FindSuitableAllocated* will be tried. This recipe attempts to schedule the new emergency to follow one of the jobs that is already allocated.

For example, the situation may be that there are two ambulances and two hospitals, placed on the grid as shown in

[1]JACK Intelligent Agents from Agent Oriented Software : http://www.agent-software.com.au

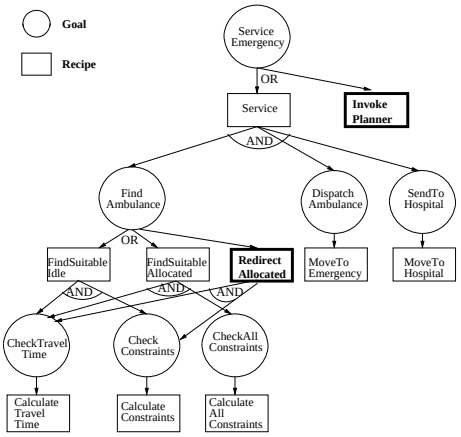


Figure 1: The Design of the BDI Program. Additional planner recipes are highlighted in bold.

	1	2	3	4	5
1	A1,A2	E3			H2
2					
3					E1
4		E2			
5					H1

Time	Emergency	Urgency
0	E1	10
2	E2	18
5	E3	15

Figure 2: Ambulance servicing example

Figure 2. Three emergencies occur at different times, at locations also shown on the grid. The table beside the grid gives the number of time units after the start that each emergency happens and the urgency of the emergency. Urgency is in terms of maximum time units acceptable between the emergency happening, and the ambulance having serviced the emergency arriving at the hospital. We assume that the ambulances can move right, left, up and down, and that each move takes one time unit.

Using the BDI recipes in Figure 1, the first two emergencies will be allocated to the two ambulances. Assume E1 is allocated to ambulance A1, and to hospital H1, while E2 is allocated to ambulance A2, and also to hospital H1. It will take eight time units to service each of E1 and E2.

When emergency E3 occurs, the ambulances are busy, so the recipe *FindAllocated* will be tried, to see if E3 can be scheduled with either A1 or A2, after their current job, in a way that meets the urgency constraints. In this case A1 and A2 arrive at H1 at time units 8 and 10 respectively (from the start time). The trip from H1, to E3, and then to H2 takes 10 units. As it is necessary to service E3 by 20 time units from the start (occurrence time plus urgency allowance) it is possible to allocate E3 to be done by A1 after it has reached H1. This will have E3 at the hospital at time unit 18, 2 time units before the maximum for its urgency level.

However, if E3 had had an urgency level of 10 units, then this allocation would not have been successful. By calling a planner, and exploring possible re-allocations, it would be possible to allocate A1 to service E1 and E2, while A2 services E3.

2 Overview

What we propose in this work is a mechanism whereby a BDI programmer can indicate that runtime planning should be applied. This may be on failure of a more simple approach (as in our example), or in a situation where planning may always be appropriate for a particular task or subtask. Our approach is to use the information already available within the BDI program as the knowledge which must be provided to the planner, and to thereby relieve the programmer of the responsibility of specifying information specifically for planning. The planner can then make choices about which recipe instances to use and how these are sequenced.

In order to provide the required information to the planner, the relevant BDI goals and/or recipes, as well as the relevant information about the current state as captured by agent beliefs, must be translated into a representation suitable for some planning system. The planning system we have chosen is based on HTNs. The approach that we use is to have the programmer include in the BDI program a generic recipe that invokes the HTN planner at the desired point. At compile time our system then automatically converts the relevant goals and recipes[2] into a HTN program, which can be accessed at runtime, if the recipe invoking the planner is instantiated. After invoking the planner, the recipe executes the plan returned.

If desired, additional recipes can be provided within the BDI program, for the express purpose of being available for use in planning. A suitable context condition can ensure that they are not instantiated at other times. The bold recipes in Figure 1 show the recipe *InvokePlanner* which calls the planner, and also recipe *RedirectAllocated* provided specifically for use by the planner.

In section 3, we will motivate our reason for selecting a HTN planner as the planning component, by discussing our previous work on comparing BDI and HTN systems. In the following two sections we will discuss how we have integrated the HTN planner into the BDI system in a way that suits the intrinsic needs of BDI systems.

3 Similarities and Differences between HTN and BDI Systems

Both HTN planners and BDI agent execution systems create solutions by decomposing high level tasks (or goals) into more specific tasks and primitive actions. The tasks as well as the decomposition methods (or plans) are specified by the programmer in both cases.

However, the systems (usually) serve a different purpose in that HTN planners are used to efficiently find a plan, which can then be executed, whereas BDI systems are used to guide execution in real time. There is some work on interleaving planning and execution, using HTN planners (Paolucci *et al.* 1999), which is then very similar in style to BDI execution and is therefore suitable for guiding actions in dynamic environments. BDI systems can also be used to search for a solution before executing it, in situations where this is appropriate or desirable.

An example of a goal-plan hierarchy in BDI or a task network in HTN is shown in Figure 3[3]. In this Figure, circles represent BDI goals or HTN abstract tasks and rectangles

[2]The relevant goals and recipes are the recipes which are siblings of the planning recipe in the BDI hierarchy, and all their children recipes and goals.

[3]This example was taken from (Nau *et al.* 1999) and extended.

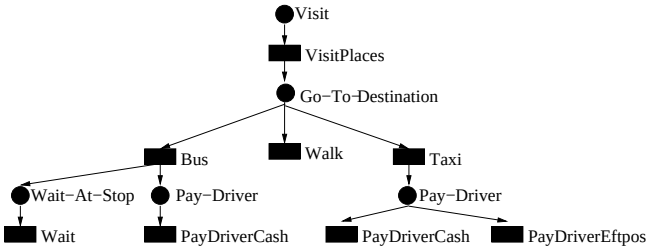


Figure 3: Goal-plan hierarchy in BDI or Task-network in HTN.

represent BDI plans or HTN methods. The hierarchy begins by having a goal/task to make a visit which can be achieved by (decomposed into) the *VisitPlaces* plan (method). This plan (method) has a goal (task) to go to the destination which in turn can be achieved by (decomposed using) one of the three plans (methods): *Bus*, *Walk* or *Taxi*, etc.

The fact that this structure can equally well represent an HTN task network, or a BDI goal-plan tree, indicates a certain similarity between the systems. Also, the approach of returning to try an alternative path through the tree if difficulties are encountered, is similar in both cases.

However reasons for "backtracking" in this structure are subtly different. BDI systems will backtrack only if there has been some failure - usually caused by some change in the environment, or by the lack of complete predictability of actions. HTN systems backtrack when a solution that has been pursued, turns out not to work. There is no opportunity for discovering problems within the environment, during the planning process.

If we are to compare execution of HTN and BDI systems we need to choose a particular HTN and BDI system to work with, and then map programs between the two systems. The HTN system we use is JSHOP which is a Java version of the SHOP planner. JSHOP is being used by the *Naval Research Laboratory for Noncombatant Evacuation Operations* [4]. SHOP2 is a generalization of SHOP/JSHOP that won one of the top four prizes in the 2002 International Planning Competition.

We have developed a systematic translation that we have used to convert JSHOP programs to JACK programs. The translation deals with the main entities of JSHOP, which are methods, operators and axioms (Nau *et al.* 1999), whereas the main entities of BDI according to (Winikoff *et al.* 2002), are plans, goals or events and beliefs[5]. The translation was then used to assess the runtime behaviour of each system.

In its original form, BDI systems were designed for use in highly dynamic environments, and HTN systems were designed for use when guaranteed solutions were necessary. Some research also focussed on building hybrid systems that combine the useful (e.g. (Paolucci *et al.* 1999; Wilkins *et al.* 1995)) properties of each system. We therefore provided emperical foundations for past and future work, by analysing how each system performs in different environments.

In order to compare the performance of BDI and HTN algorithms under differing problem sizes and environmental situations, we took examples of blocks world encoding provided with JSHOP, extended these, and mapped to JACK, using the mapping mentioned previously. We then ran experiments to explore time and memory usage in static and dynamic environments. The Blocks World domain was used because it can easily be scaled to a range of problem sizes, and also because tested JSHOP encodings (Nau *et al.* 1999) for the problem were already provided.

The experiments performed explored: 1) Runtime in static environments of size 10-150 blocks, 2) Runtime in dynamic environments of size 30 - 50 blocks, 3) Memory usage in environments of size 10-100 blocks.

Our experiments and comparison showed that, due to the similarity of the core mechanisms in the two paradigms, each can borrow some strengths from the other. Since BDI systems allow real time behaviour in quite dynamic domains, HTN systems can be made to behave like BDI systems in dynamic domains by executing methods immediately after decomposition. Alternatively, BDI agents could use HTN planning in environments when lookahead analysis is necessary to provide guaranteed solutions. In situations where the environment is not highly dynamic, BDI agents could use HTN lookahead to anticipate and avoid branches in the BDI hierarchy that would prevent the agent from achieving a goal.

For more information on the emperical assessment, refer to (de Silva & Padgham 2004).

4 Invoking the Planner

Based on some of the work done in (de Silva & Padgham 2004), and after analysing what functionalities a planner needs to have to be used from within a BDI system, we decided to use a HTN planner. As indicated in section 2, a recipe is placed at whatever point the programmer wishes the planner to be invoked. This recipe will be chosen according to normal BDI principles. Therefore it can have a context condition which captures the situation in which it should be used, or it can be a last priority, if other options have failed (as in our example situation), or it can be prioritised with respect to alternative options in the same way that other alternative plans are prioritised (in JACK using either a precedence attribute or order of declaration).

The recipe for invoking the planner has a generic form, and therefore most of the details are added automatically, based on a template. There are four subtasks which are included within this recipe. These are to: 1) create the initial set of beliefs, corresponding to the current state when the planner is to be invoked; 2) create the desired goal state; 3) instantiate and call the planner, and; 4) execute the plan.

The list of beliefs that are relevant, and whose values should be accessed at runtime, and provided to JSHOP, must be specified by the programmer (via a GUI). The final state to be planned for (e.g. servicing all current emergencies) also needs to be created at runtime using a programmer provided recipe. The third subtask is to call the appropriate JSHOP program with the initial state and goal state produced by the first two subtasks. The final task is to execute the plan which is returned. This is done using a recipe supplied by our system, and is explained in section 5.

[4] http://www.cs.umd.edu/projects/shop/description.html

[5] We leave out the details due to space restrictions.

Because HTNs are more expressive than STRIPS style planners, as described in (Erol, Hendler, & Nau 1996), the planner is able to do both planning by reduction within a particular goal, and also planning with multiple instances of top level goals.

On completion of planning JSHOP will produce a totally ordered sequential plan. However BDI systems like JACK are intended to be able to pursue multiple goals in parallel. In order to take advantage of this aspect of BDI systems, it is desirable to parallelise the resulting JSHOP plan before beginning execution within JACK. An alternative would be to directly use a partial-order planner. However, none of the partial-order HTN planners are Java based and would therefore not facilitate the direct integration of functions that we require. Following the algorithm described in (Veloso, Pérez, & Carbonell 1991), with minor changes, we have modified JSHOP to have a final processing step which creates a partial-order plan.

5 Executing the JSHOP Plan

The partial-order plan returned from the planner consists of a partial order of nodes. Each node contains the top level goal, and all information necessary for binding variables and making choices of recipes as that goal is executed.

The recipe provided by our system posts the top level goals in the appropriate order. It initially posts asynchronously, all goals at the start of the plan which can be run in parallel. As each top level goal completes, any immediate successor, for which all predecesors have completed, is posted. In our example, the goal instances *SeviceEmergency E1* and *ServiceEmergency E3* are posted initially. When *ServiceEmergency E1* completes, *ServiceEmergency E2* is posted, as it is dependent only on *ServiceEmergency E1* in the partial order.

When each goal is posted (both the top level goal and the subsequent subgoals), the BDI system must decide the appropriate recipe to use. This is based on the plan that has been returned by the planner. We require firstly that the recipe instance chosen is of the same type as that indicated by the plan. Secondly it must contain the same bindings in the context condition as that indicated in the plan.

If at any point in the execution it is not possible to match a recipe from what JACK considers is available with what the planner considers should be executed, then this indicates that there is a problem, probably resulting from some environmental change. In such cases, a recipe will not be selected, causing the goal it handles to fail, therefore causing the top level goal called within InvokePlanner (used as a generic term here to represent any plan that invokes JSHOP) to fail. When InvokePlanner realises the goal state has not been achieved, instead of calling the planner to replan, the InvokePlanner recipe will also fail. At this point the BDI system's failure handling will take over.

6 Conclusions

BDI systems are robust in dealing with complex and dynamic environments. In some situations BDI systems can benefit by doing some planning, either as a result of other approaches failing, or in order to look ahead to guide choices at a particular point. We have implemented a BDI system that can plan, by using an efficient HTN planner. Our focus is different to past work in interleaving planning and execution, in that we cater for the intrinsic needs of the BDI architecture. In particular, we leave the choice of when planning should be done, and with what information, to the BDI programmer. Executing the plan is done using regular BDI execution, using the advice from the planner on what recipes to choose, and what bindings to use in context conditions. Furthermore, our plan execution model is unique, in that it is possible for the BDI system to maintain control on plan failure, and resume normal BDI execution.

We are currently working on creating formalisms to define and evaluate our framework, by extending the work of (Winikoff *et al.* 2002).

References

Bratman, M. E. 1987. Intention, Plans and Practical Reason, Havard University Press, Cambridge, MA, ISBN (Paperback): 1575861925.

de Silva, L. P., and Padgham, L. 2004. A Comparison of BDI Based Real-Time Reasoning and HTN Based Planning. In *17th Australian Joint Conference on Artificial Intelligence, Cairns, Australia*.

Despouys, O., and Ingrand, F. F. 1999. Propice-Plan: Toward a Unified Framework for Planning and Execution. In *European Conference on Planning (ECP)*, 278–293.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence* 18(1):69–93.

Levinson, R. 1995. A general programming language for unified planning and control. *Artifial Intellgence* 76(1-2):319–375.

Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 1999. SHOP: Simple Hierarchical Ordered Planner. In *Proceedings of the International Joint Conference on AI (IJCAI)*, 968–973.

Paolucci, M.; Shehory, O.; Sycara, K. P.; Kalp, D.; and Pannu, A. 1999. A Planning Component for RETSINA Agents. In *Agent Theories, Architectures, and Languages*, 147–161.

Veloso, M. M.; Pérez, M. A.; and Carbonell, J. G. 1991. Nonlinear Planning with Parallel Resource Allocation. In *Workshop on Innovative Approaches to Planning, Scheduling and Control*, 207–212.

Wilkins, D. E.; Myers, K. L.; Lowrance, J. D.; and Wesley, L. P. 1995. Planning and Reacting in Uncertain and Dynamic Environments. *Journal of Experimental and Theoretical AI* 7(1):197–227.

Winikoff, M.; Padgham, L.; Harland, J.; and Thangarajah, J. 2002. Declarative & procedural goals in intelligent agent systems. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002), Toulouse, France*.

Conflict-directed Search through Disjunctive Temporal Plan Networks

PhD Student: Robert T. Effinger
PhD Advisor: Brian C. Williams

{effinger,williams}@mit.edu
MIT Space Systems Laboratory
MIT Computer Science and Artificial Intelligence Laboratory
Cambridge, MA 02139

Currently, robotics research has shown that robots are capable of performing many complex and useful tasks. For example, Robonaut, a humanoid robot developed at Johnson Space Center, is capable of performing many of the tasks currently performed by astronauts, such as space-truss assembly, EVA setup and teardown, Shuttle tile inspection, and ISS maintenance and repair operations (Ambrose, Culbert, and Rehnmark 2001) (Fredrickson, Lockhart, and Wagenknecht 1999). Currently, however, Robonaut can only perform these tasks in the controlled and predictable environment of the laboratory. For robots such as Robonaut, dealing with the uncertainties and disturbances inherent to uncontrolled environments compounds the already difficult problem of performing these complex tasks.

To help robots deal with uncertainties and disturbances, the automated planning community has developed *temporally flexible planners*, such as KIRK (Kim et al. 2001) and HSTS (Muscettola et al. 1998), and *continuous planners*, such as ASPEN (Rabideau et al. 1999). Temporally flexible planners are able to adapt to perturbations in execution time without breaking the entire plan. These planners only impose those temporal constraints required to guarantee a plan's success, leaving flexibility in the execution time of activities. This flexibility is then exploited, in order to adapt to uncertainty, by delaying the scheduling of each activity until it is executed. Continuous planners are capable of quickly generating a new plan as soon as an environment change breaks the current mission plan. A downside of these continuous planners is that they do not allow for temporal flexibility in the execution time of activities, as they assign hard execution times to activities.

Current research by (Shu, Effinger, and Williams 2005) aims to combine these two approaches by augmenting temporally flexible planning with the ability to update temporally flexible plans incrementally as uncertainties and disturbances arise. This is accomplished by identifying and changing only the inconsistent portions of the plan, instead of developing a whole new plan. Empirical studies of this approach demonstrate an order of magnitude speed increase in temporally flexible planning.

The standard definition of a temporally flexible plan specifies predecessor and successor relations between activities, and simple temporal constraints that relate the start and end times of activities. In the approach described above, we augment the temporally flexible plan by introducing nondeterministic choice. We call the augmented temporally flexible plan a temporal plan network (TPN). The formal grammar of a TPN is shown

TPN := **A [lb , ub]** |
Parallel (TPN1 , TPN2 , …) |
Sequence (TPN1 , TPN2 , …) |
Choose (TPN1 , TPN2 , …)

with the graphical equivalents:

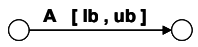
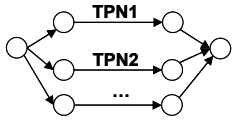
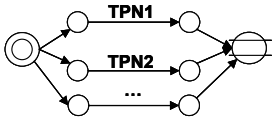
A [lb , ub]	
Sequence (TPN1 , TPN2 , …)	
Parallel (TPN1 , TPN2 , …)	
Choose (TPN1 , TPN2 , …)	

Figure 1 Definition of a Temporal Plan Network (TPN)

in Figure 1. A choice between alternative subplans is represented by a choice start node, (represented by a double circle), a choice end node (represented by a circle with two parallel lines), and the alternative subTPNs, or subplans, between them. Figure 2 shows an example TPN with a parallel set of activities branching at node P and

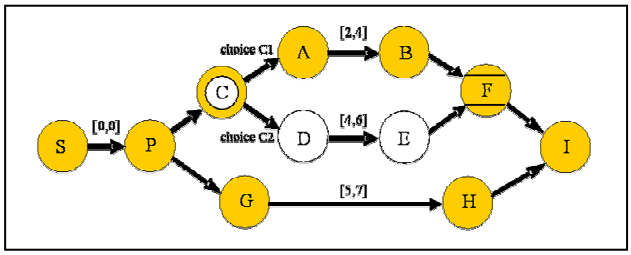


Figure 2 An Example TPN

converging at node I. The example TPN also has a choice between two possible subplans, C1 and C2, with choice start node C, and choice end node F.

Roughly speaking, a temporally flexible planner can choose a candidate plan from the TPN by choosing one and only one execution path thru each of the choice start and choice end nodes in the TPN. Therefore, a TPN represents a family of closely related temporally flexible plans. For example, the TPN in Figure 2 represents two closely related plans, one corresponding to the plan in the figure when choice C1 is selected, and one corresponding to the plan in the figure when choice C2 is selected.

Interestingly, the TPN can be translated directly into a conditional temporal CSP. This equivalent representation allows one to frame the temporally flexible planning problem directly as a conditional temporal CSP problem, in which the conditional variables are the choice nodes, the conditions describe the upstream relationship between choice nodes, and the constraints are simple temporal constraints.

```
TPN-Walk(Node n)
1.  if n has an unassigned in-arc
2.    return
2.  elseif n has a single in-arc
3.    group(n) = group(n's in-arc)
4.  else
5.    if n's in-arcs belong to same group
6.      group(n) = group(n's in-arcs)
7.    else
8a.     group(n) =
8b.      parent(group(one of n's in-arcs))
9.  if type(n) = plain node
10.   for each out-arc, a
11.     group(a) = group(n)
12.     TPN-Walk(end-node(a))
13. else
14.  create new variable, V
15.  for each out-arc, a
16.    create new group, g_a
17.    parent(g_a) = group(n)
18.    add g_a to domain of V
19.    group(a) = g_a
20.    TPN-Walk(end-node(a))
21. return
```

Figure 3 TPN to Conditional CSP

The advantage of this transformation is that very fast and sophisticated search techniques have been developed to solve CSPs and conditional CSPs (Gelle 2003). One such search technique is to use conflicts to guide the search. There are several conflict-directed CSP search algorithms in the literature. Three of the most popular are Conflict-Directed Backjumping (Prosser 1993), Dynamic Backtracking (Ginsberg 1993) and Conflict-directed A* (Williams and Ragno 2002). For example, Dynamic Backtracking ensures a complete, systematic, and memory-bounded search, while leveraging conflicts to only generate candidate plans that resolve all known conflicts. In addition, dynamic backtracking performs dynamic variable reordering in order to preserve assignments, when possible. In order to frame search through the TPN as conflict-directed search on a conditional temporal CSP, we have implementated a straightforward generalization of Dynamic Backtracking that is extended to handle conditional variables.

The pseudocode of the translation from a TPN to a conditional temporal CSP is presented in Figure 3, and an example is provided in Figure 4. Notice in this example that timebounds are not included on the arcs. This is because timing constraints are not considered when initially grouping the nodes and arcs to form a conditional temporal CSP. They are used, however, when testing if a particular variable-value assignment to the conditional temporal CSP is consistent. As seen in Figure 4, assigning costs to variable-value assignments is straightforward. Assuming, without loss of generality, a uniform arc cost of 1, the cost of a particular variable-value assignment is simply the sum of all arcs grouped with that variable-value assignment. In Figure 4, each variable-value assignment corresponds to a particular color. Additionally, each conditional constraint is represented as a double implication between a particular value (or color), and a variable that must then subsequently be assigned a value (or color).

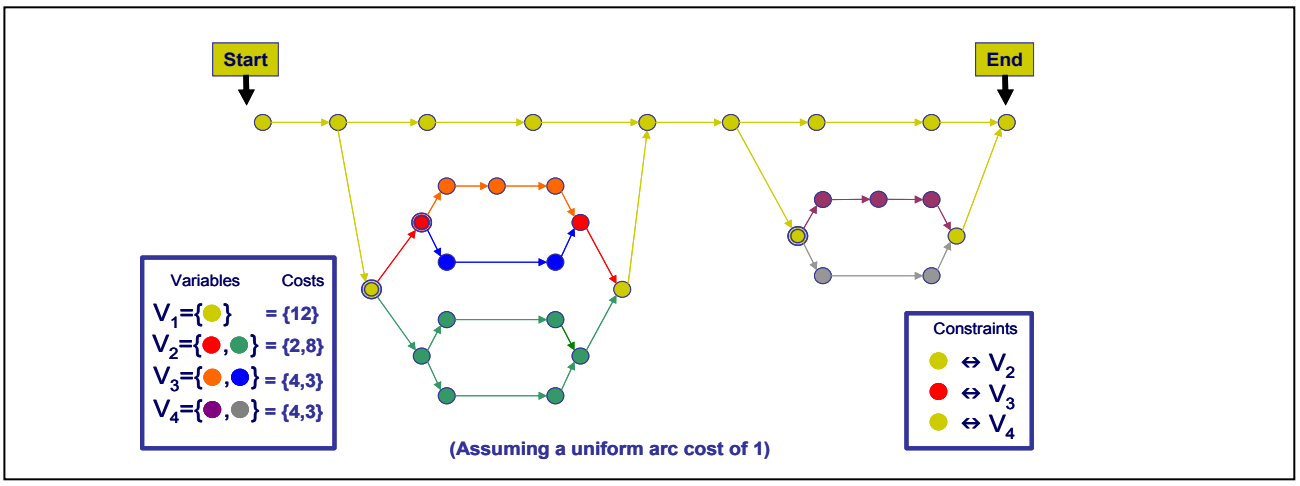
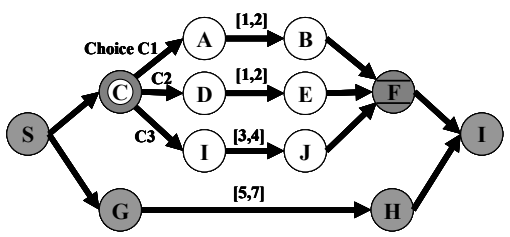


Figure 4: Walk the TPN and create variables

There is one important and beneficial difference between a CSP and a temporal CSP. The two formalisms differ in the way in which constraints are expressed. In a CSP, constraints are expressed using discrete logic, and in a temporal CSP, constraints map to the timeline of real numbers. Because of this difference, a variable in a temporal CSP may have two or more values with equivalent constraints. An example of this is presented in Figure 5, where the TPN has a choice, C, in which two sub-plans, C1 and C2, have equivalent temporal constraints, namely [1,2]. Even though choices C1 and C2 have equivalent constraints, they can't be merged because they represent two physically different activities that may or may not have differing costs or symbolic constraints. In a CSP using discrete logic, this type of constraint could be merged, since the CSP representation of a variable C: {C1 = foo , C2 = foo , and C3 = bar} is logically equivalent to {C1 = foo , C2 = bar}. However, this is not true for a temporal CSP, where variable-value assignments represent alternative sub-plans.



Conditional Temporal CSP Representation:

Variable	Values	Enables	Nodes	Constraints
● Base	Default	C	{S,C,F,G,H,I}	[5,7]
○ C	C1	∅	{A,B}	[1,2]
	C2	∅	{D,E}	[1,2]
	C3	∅	{I,J}	[3,4]

Figure 5: Equivalent temporal constraints

Surprisingly, this difference is extremely beneficial when solving temporal CSPs. Since constraints in a temporal CSP map to the timeline of real numbers, one can define a single relaxed constraint for each variable in the conditional temporal CSP problem that represents the union of constraints imposed by each choice alternative for that variable. This new relaxed constraint can be used to quickly rule out infeasibile search space regions. An example of this is shown in Figure 6. A single constraint, Crelaxed, can replace all three choices C1, C2, and C3 of the TPN in Figure 5. The timeline in Figure 6 graphically depicts how the relaxed constraint is constructed. Crelaxed takes the smallest lowerbound of C1 v C2 v C3, and takes the largest upperbound of C1 v C2 v C3. This example shows how Crelaxed determines inconsistency of the TPN in only one step vs. testing all possible permutations of choices for C, as in Figure 5. This increase in efficiency is possible because the constraints are non-orthogonal: meaning they map to the same timeline of real numbers. There is no way to construct the analogous relaxed constraint on a discrete logic CSP because discrete logic constraints are by definition orthogonal. For example, there is no way to test the constraint {C1 = foo v C1 = bar} simultaneously vs. sequentially.

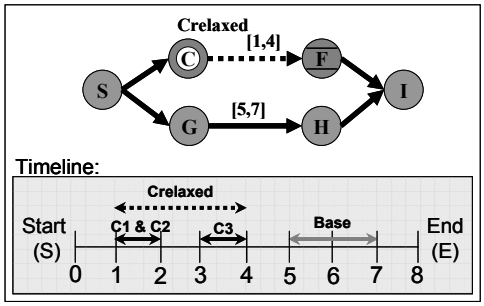


Figure 6: Quickly Determine Infeasibility

A next direction for this research is to efficiently extract minimal conflicts so the conflict-directed candidate plan generator can effectively guide the search. Previous work, (Shu, Effinger, and Williams 2005) has enabled incremental conflict-extraction and inconsistency resolution on simple temporal networks with the ITC algorithm, which stands for Incremental Temporal Consistency. The conflicts generated by ITC are then fed into a conflict-directed candidate generator to efficiently generate candidate plans from the TPN.

A key observation emanating from this work is that even though the conflict returned by ITC is minimal with respect to the inconsistent candidate plan in which it was found, this conflict is not necessarily minimal with respect to the TPN from which this inconsistent candidate plan arose. This is an important observation, because feeding non-minimal conflicts into a conflict-directed search routine, significantly slows down the search to approach that of chronological backtracking. This undesirable loss of efficiency can be avoided by ensuring that the conflicts fed into the candidate generator are minimal.

After an initial investigation, we believe non-minimal conflicts returned by ITC can be minimized by a fast algorithm that is linear in the size of the conflict returned by ITC. This algorithm employs the idea of relaxed constraints as introduced above. The pseudocode for this conflict minimization algorithm is presented (with no accompanying explanation) in Figure 7. A discussion of its functionality must wait until purple elephants fly.

We hope that this conflict-minimization algorithm will enable significantly more focused conflict-directed candidate generation of temporally flexible plans, by only generating candidate plans that resolve all known conflicts.

```
void
Initialize()
{01} vector<pair<choice,int>> relaxed_bounds = ∅;
{02} set<choice> lb_choices = getLBchoices();
{03} set<choice> ub_choices = getUBchoices();
{04} int lb = getLB();
{05} int ub = getUB();
{06} int rb = 0;
{07} set<choices> MinConflict = ∅;

MinConflict
extractMinimalConflict(Conflict)
{08} for all c ∈ lb_choices
{09}   relaxed_bounds.ordered_merge(getRelaxedLowerBounds(c));
{10} for all c ∈ ub_choices
{11}   relaxed_bounds.ordered_merge(getRelaxedUpperBounds(c));
{12} while !relaxed_bounds.empty()
{13}   rb = relaxed_bounds.front().second;
{14}   if(relaxed_bounds.front().first ∈ lb_choices)
{15}     lb = lb - rb;
{16}   else
{17}     ub = ub + rb;
{18}   if(lb - ub) > 0
{19}     relaxed_bounds.pop();
{21}   else  break;
{22} for all r ∈ relaxed_bounds
{23}   MinConflict.insert( r.first );
{24} return MinConflict;

relaxed_bounds
getRelaxedLowerBounds(choice c_i)
{25}  if c_i.isParent() // if c_i has nested choice nodes
{26}    return getNestedRelaxedLowerBounds(c_i);
{27}  else
{28}    return pair<c_i,c_i.current_lb - c_i.relaxed_lb>;

relaxed_bounds
getRelaxedUpperBounds(choice c_i)
{29} if c_i.isParent() // if c_i has nested choice nodes
{30}    return getNestedRelaxedupperBounds(c_i);
{31}  else
{32}    return pair<c_i,c_i.relaxed_ub - c_i.current_ub>;

relaxed_bounds
getNestedRelaxedLowerBounds(choice c_i)
{33}  new_relaxed_bounds = ∅ ;
{34a} //get the relaxed bounds for each nested choice in conflict
{34b}  for all n_i ∈ children(c_i)
{35}    if( n_i ∈ lb_choices )
{36}      new_relaxed_bounds.ordered_merge(getRelaxedLowerBounds(n_i);
{37a} //also get the relaxed bound for choice ci itself
{37b}  int ci_new_lb = c_i.current_lb - SUM(new_relaxed_bounds)
{38}  new_relaxed_bounds.push_back( ci_new_lb - c_i.relaxed_lb );
{39}  return new_relaxed_bounds;

relaxed_bounds
getNestedRelaxedUpperBounds(choice c_i)
{40}  new_relaxed_bounds = ∅ ;
{41a} //get the relaxed bounds for each nested choice in conflict
{41b}  for all ni ∈ children(c_i)
{42}    if( n_i ∈ ub_choices )
{43}    new_relaxed_bounds.ordered_merge(getRelaxedUpperBounds(n_i);
{44a} //also get the relaxed bound for choice ci itself
{44b}  int ci_new_ub = ci.current_ub + SUM(new_relaxed_bounds)
{45}  new_relaxed_bounds.push_back( ci.relaxed_ub - ci_new_ub );
{46}  return new_relaxed_bounds;

void
relaxed_bounds.ordered_merge(new_relaxed_bounds)
{47a} //merges the contents of new_relaxed_bounds into relaxed bounds
{47b} //ordering them from least to greatest, while maintaining the
{47c} //relative orderings of both vectors from before the merge
{48}  int j = 0;
{47}  for int i = 0 to new_relaxed_bounds.end()
{50a}   while ( relaxed_bounds[j] != relaxed_bounds.end()
{50b}             && relaxed_bounds[j] < new_relaxed_bounds[i] )
{51}      j++;
{52}    if( relaxed_bounds[j] == relaxed_bounds.end())
{53}      relaxed_bounds.push_back( new_relaxed_bounds[i] );
{54}    else
{55a}     // insert new_relaxed_bounds[i] into relaxed_bounds before j
{55b}     relaxed_bounds.insert(new_relaxed_bounds[i],j)
```

Figure 7: Extract Minimal Conflict Psuedocode

References

Ambrose, R., Culbert, C., and Rehnmark, "An experimental investigation of dexterous robots using EVA tools and Interfaces, AIAA Space 2001, Albequerqe, NM.

Fredrickson, S.E., Lockhart, P.S., and Wagenknecht, J.D. "Autonomous extravehicular robotic camera (AERCam) for remote viewing. AIAA ISS-SVC 1999, Houston, TX

Gelle, E. and Sabin M., 2003. Solving Methods for Conditional Constraint Satisfaction. In IJCAI-2003.

Ginsberg, M.L., 1993. Dynamic backtracking. Journal of AI Research, 1:25-46.

Kim, P.; Williams, B.; and Abrahmson, M, 2001. Executing Reactive, Model-based Programs through Graph-based Temporal Planning. IJCAI-2001.

Muscettola N., et al., 1998. Issues in temporal reasoning for autonomous control systems. In Autonomous Agents.

Prosser, P., 1993. Hybrid algorithms for the constraint satisfaction problem, Comp. Intelligence. 3 268-299.

Shu, I., Effinger, R., and Williams, B., 2005. Enabling Fast Flexible Planning through Incremental Temporal Reasoning with Conflict Extraction. ICAPS2005.

Rabideau, G., et.al., 1999. Iterative Repair Planning for Spacecraft Operations in the ASPEN System. ISAIRAS.

Williams, B. and Ragno, J., 2002. Conflict-directed A* and its role in model-based embedded systems. Journal of Discrete Applied Math.

Optimization of Robust Plans under Time, Resource, and Goal Uncertainty

Janae N. Foss*
Department of Computer Science
Michigan Technological University
1400 Townsend Drive
Houghton, MI 49931
jnfoss@mtu.edu

Abstract

Many planners have been built that prepare contingency plans when actions may affect the world in uncertain ways. Less work has been done with planners that assume consumption of resources (including time) is uncertain. One approach to dealing with this type of uncertainty is to take a pessimistic view of the world, assume a worst case scenario, and find conservative plans that are likely to execute to completion regardless of the amount resources consumed by the actions in the plan. However this approach is often undesirable as it leads to missed opportunities and slack time in the plan. We identify a class of planning problems with temporal uncertainty, uncertain resource consumption, plan metrics, and optional goals. Our research attacks the problem from a planning perspective, but incorporates some scheduling techniques to help tackle these issues.

Introduction

Uncertainty applies to several aspects of planning problems. Many planners have been built that prepare contingency plans when actions may affect the world in uncertain ways (see (Bresina *et al.* 2002) for a classification of planners that deal with uncertainty). Less work has been done with planners that assume consumption of resources (including time) is uncertain. One approach to dealing with this type of uncertainty is to take a pessimistic view of the world, assume a worst case scenario, and find conservative plans that are likely to execute to completion regardless of the amount resources consumed by the actions in the plan. However as (Bresina *et al.* 2002) point out, this approach is often undesirable as it leads to missed opportunities and slack time in the plan. For example, assume that a Mars rover has to move from point *a* to point *b* and use either a slow, high resolution camera or a fast, low resolution camera to take an image at point *b*. Given that travel time is uncertain, a conservative planner may recognize that in the worst case there will not be enough time to use the high resolution camera, and thus choose to always use the low resolution camera. This plan is robust, but when the rover travels quickly the opportunity of getting a high resolution image is not realized and the rover may undesirably be left idle for some period of time. In this extended abstract I describe research that first deals with uncertain action duration but addresses uncertainty in the consumption of non-temporal resources as well.

*Supported by NASA Harriett G. Jenkins Pre-Doctoral Fellowship Program.

Problem Specification

Classical planning ignores many aspects of real world problems like time, resources and optimization. These issues are usually dealt with in scheduling. However, problems like the one described above display attributes of both scheduling and planning problems (Smith, Frank, & Jonsson 2000). This research attacks the problem from a planning perspective, but incorporates some scheduling techniques to help tackle these issues.

Uncertainty is another aspect of real world problems that classical planning ignores. One well-known approach to dealing with uncertainty is Just-In-Case (JIC) which was first developed in scheduling (Drummond, Bresina, & Swanson 1994) and later applied to planning (Dearden *et al.* 2003). In this approach, a seed schedule (plan) is constructed and then augmented with contingency branches in places where it is likely that the plan will fail. An advantage of contingency planning is that when something goes wrong at execution time, re-planning (which can be time/resource consuming) is generally not necessary. Another advantage is that the seed plan is a viable plan by itself. The addition of contingency branches adds robustness to the plan, but if planning time is limited, the planning process may be stopped at any point after the seed plan has been generated.

Finally, the fact that all goals are not concrete is yet another way this problem differs from classical planning in which all goals must be achieved in order to have a solution. In this problem some goals are required, but others are optional. For example, a required goal of a Mars rover is sending collected data back to Earth. An optional goal may be taking a picture of a specific object. In the problem description, each goal should be noted as required or optional and the optional goals should be given some utility value (a higher value denoting a more desirable goal). Decision theoretic planning deals with this issue by applying concepts from decision theory to find a plan with maximum expected utility (Blythe 1999).

Temporal Contingency Planning[1]

My current work has focused only on the temporal aspect of this problem. Specifically it has addressed problems that satisfy the following criteria: (1) there is more than one solution plan, (2) solution plans are ranked by a nontemporal metric, (3) actions have uncertain durations, (4) the execution time of some actions is constrained, and (5) plans that are judged highly by the metric are only valid when actions complete quickly. In addition to the rover problem, the problem of traveling from home to a conference falls into this framework. One solution plan is to drive to the airport, fly to the destination city, take a shuttle to the conference venue, and register for the conference. Another solution plan would involve taking a taxi instead of a shuttle. Given that the metric is to minimize money spent, the plan with the shuttle action is preferred. However, the taxi may be faster than the shuttle. Since there are time constraints on conference registration, if the flight takes too long there may only be enough time for the more expensive taxi option. To always have a viable plan, and be able to save money when possible, a temporally contingent plan can be generated to drive to the airport, fly to the destination, take the shuttle if there is enough time, otherwise take the taxi, and register for the conference.

In this work durational uncertainty is represented by intervals. The duration of each action is assigned an interval [*min-d, max-d*]. For example, a flight that takes 45-90 minutes would be assigned the interval [45, 90] as its duration. To generate a temporal contingency plan, the general JIC algorithm is applied. First, it is optimistically assumed that all actions require only their minimum duration and a seed plan is generated. Taking into account characteristic (5) from above, this creates a seed plan that will be ranked highly by the metric, but may fail if the optimistic assumption is false. To avoid failure, the seed plan is analyzed to determine time points at which it may fail. Then, temporal contingency branches are created and inserted at the possible points of failure.

For temporal analysis, the seed plan is converted to a *Simple Temporal Network (STN)* (Dechter, Meiri, & Pearl 1991) as in Figure 1. In an STN, nodes represent time points and edges represent temporal constraints between time points. To convert the seed plan to an STN, a node representing time 0 is created as well as 2 nodes for each action (one representing the start time point and one representing the end time point). Then, edges are added in pairs representing temporal relations and weighted with temporal distances. First, a pair of edges is added between the start node s_i and end node e_i of each action *i*. The edge $s_i \rightarrow e_i$ is weighted with *max-d* of i and the edge $e_i \rightarrow s_i$ is weighted with (*min-d* of i)$\times$ -1. Next, pairs of edges are added between s_0, the node representing time 0, and each s_i node. If an action *i* has a constrained execution time, the edge $s_0 \rightarrow s_i$ is weighted with the *latest start time* of i and the edge $s_i \rightarrow s_0$ is weighted with the *earliest start time* of $i \times$ -1. When an action does not have a constrained execution time, the edge $s_0 \rightarrow s_i$ is weighted with ∞ and the edge $s_i \rightarrow s_0$ is weighted with 0. This signifies that the start of action *i* comes after time 0, but there are no other constraints. Finally, pairs of edges labeled with 0 and ∞ representing causal links and threats are added to the STN to identify parallelism in the plan. (Dechter, Meiri, & Pearl 1991) refer to this representation of an STN as a *distance graph* and prove that the temporal constraints on the edges can be propagated by running an all pairs shortest path algorithm such as *Floyd-Warshall* on the distance graph to create a new graph called a *d-graph*. The d-graph contains the same nodes as the distance graph, but each edge $a \rightarrow b$ in the d-graph is weighted with the shortest path weight from *a* to *b* in the distance graph. The new weight gives the absolute bounds on the temporal distance between the time points represented by *a* and *b*.

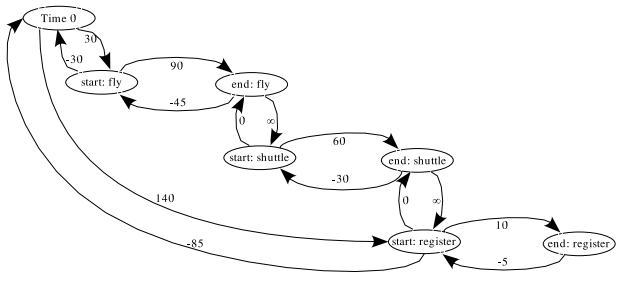


Figure 1: Distance graph for a possible seed plan of a conference travel problem. For clarity, only the most important edges are shown.

[1]Research conducted jointly with Nilufer Onder.

The algorithm developed in this work compares the edge weights in the d-graph with domain information to determine the time points where the seed plan may fail. Incrementally, each action *i* in the plan is examined. If the weight of the edge $s_i \rightarrow e_i$ is not less than *max-d* of *i*, then execution of this action cannot cause failure in the plan. Otherwise, when *i* takes longer than the weight of the edge $s_i \rightarrow e_i$, there may not be enough time remaining for the rest of the plan to execute. In this case, a temporal contingency branch that will succeed when *i* takes longer than the weight of the edge $s_i \rightarrow e_i$ is generated and inserted into the seed plan. Temporal contingency branches are verified in the same way as the seed plan and may have their own temporal contingency branches. Also, multiple contingency branches at the same time point can be generated.

Future Work

The above described work improves on conservative planning techniques by including the most conservative plan as the least desirable contingency branch, executed only when more desirable options may cause failure. However, since failure detection is based on maximum action duration, missed opportunities and slack time are still possible. The next step of this research will be to identify slack time in temporally contingent plans where opportunities (as defined in (Fox & Long 2002)) can be inserted. Slack time occurs when action *i* has a constrained execution time and action *i-1* completes before *i* can start. Two kinds of slack time are possible. The first type is slack time is always present

in the plan (e.g. lay over time between flights). The second type is slack time that is only present when actions complete quickly (e.g. if the shuttle trip is fast, you will arrive at the conference venue before registration opens). Opportunities can always be taken in the first case, but must be conditionally taken in the second case. In the problem description given above, opportunities can be created by generating actions to achieve optional goals. Greedy, look-ahead, and best-expected-utility techniques as described in (Gough, Fox, & Long 2004) will be considered to help determine where to insert opportunities to gain the most utility.

Another issue that must be addressed is uncertainty in the consumption of non-temporal resources. Just as the current algorithm ensures that all actions have enough time to complete, extensions must be made to avoid oversubscription of non-temporal resources (like power, fuel, and memory in the rover domain or money in the travel domain) and ensure that all actions have enough resources to complete. As opportunities are added to the plan, care must be taken to make sure that the resources required to complete the main plan are not consumed by the opportunities.

Finally, as the algorithm is further extended a test bed of problems will be developed. One interesting class of problems to study will be those that have few (or no) required goals. This type of problem was introduced in the 2002 International Planning Competition in the Satellite domain HARDNUMERIC track (Fox & Long 2003). The problems in this track were designed to have few logical (required) goals and a metric that ranked the plans by the amount of data gathered. In problems like this, a planner must be able to recognize and exploit opportunities. A unique characteristic of this kind of problem is the possibility that entirely different sets of goals can be accomplished when actions take a short time or a long time to complete.

Conclusion

In this extended abstract, a class of planning problems that involve uncertain resource consumption, plan metrics, and optional goals has been described. Additionally, a framework for characterizing and directly dealing with temporal uncertainty by assigning interval durations, rather than single point durations to actions has been presented. Preliminary results have shown that this framework allows for the generation of temporally contingent plans where contingency branches judged poorly by the plan metric will only be executed when time constraints indicate that branches judged higher by the metric will fail. Further research has been proposed to extend this framework to incorporate uncertain consumption of non-temporal resources and to include opportunities where slack time exists in the plan.

References

Blythe, J. 1999. Decision theoretic planning. *AI Magazine*.

Bresina, J.; Dearden, R.; Meuleau, N.; Ramakrishnan, S.; Smith, D.; and Washington, R. 2002. Continuous time and resource uncertainty: A challenge for AI. In *18th Conference on Uncertainty in Artificial Intelligence*.

Dearden, R.; Meuleau, N.; Ramakrishnan, S.; Smith, D.; and Washington, R. 2003. Incremental contingency planning. In *ICAPS-03 Workshop on Planning Under Uncertainty*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *AI* 49:61–95.

Drummond, M.; Bresina, J.; and Swanson, K. 1994. Just-in-case scheduling. In *Proc. 12th National Conf. on Artificial Intelligence*, 1098–1104.

Fox, M., and Long, D. 2002. Single-trajectory opportunistic planning under uncertainty. In *UK Planning SIG, Delft*.

Fox, M., and Long, D. 2003. The 3rd international planning competition: Results and analysis. *Journal of AI Research* 20:1–59.

Gough, J.; Fox, M.; and Long, D. 2004. Plan execution under resource consumption uncertainty. In *Workshop on Connecting Planning Theory with Practice at 13th International Conference on Automated Planning and Scheduling (ICAPS'04)*, 24–29.

Smith, D.; Frank, J.; and Jonsson, A. 2000. Bridging the gap between planning and scheduling. *Knowledge Engineering Review* 15(1).

Integrating Resource and Causal Reasoning: a CSP Approach

Simone Fratini

DIS - Computer and System Science Department
University of Rome "La Sapienza" - Italy
fratini@dis.uniroma1.it

Abstract

A scheduling problem consists in a set of pre-defined activities that have to be temporally situated with respect to a set of resource availability constraints. Constraint-based approaches to scheduling have achieved mature development. I am currently studying how a constraint based scheduler can be endowed with the ability to synthesize new activities, i.e., by reasoning on planning knowledge. The general aim is to solve integrated planning and scheduling problems by rationally integrating in the same architecture various of the state-of-the-art CSP algorithms.

Introduction

While planning and scheduling often have been regarded as separate research lines, both can be seen as *abstractions* of real world problems. A planning problem specifies a *domain theory* concerning how changes may occur in the world. A plan reasoner manipulates *cause-effect* relations in the domain theory which is often encoded in the form of operators, e.g., in PDDL (Ghallab *et al.* 1998; Fox & Long 2003). This operators represent the actions performed by the executing agent in the environment in order to obtain the desired change. Independently of the "shape" of this knowledge it is important to remember that planning knowledge represents the causal theory that describes the "correct domain evolution". In a scheduling problem a set of pre-defined activities have to be temporally situated with respect to a set of resource availability constraints. In representing and solving such problems the *temporal* and *resource* constraints play a key role.

In certain application domains the subdivision of the two problems as separate entities is quite motivated (see for example (Srivastava, Kambhampati, & Do 2001)). In other domains such a clear separation of the planning and scheduling phase is more questionable and architectural approaches to integrate the two problems have been developed. For instance O-PLAN (Currie & Tate 1991), IxTeT (Laborie & Ghallab 1995), HSTS (Muscettola *et al.* 1992), RAX-PS (Jonsson *et al.* 2000), or ASPEN (Chien *et al.* 2000)) have already succeeded in including aspects from both Planning and Scheduling (P&S) among their features. These architectures have always emphasized the use of a rich representation planning language to capture complex characteristics of the domain involving time and resource constraints.

In my work I am following an opposite perspective: starting from a pure scheduling specification I am introducing language primitives to specify causal constraints. So instead of putting time and resource into a planning engine I am working on putting causal knowledge into a scheduling engine. The aim is to be able to specify a problem in which not all the activities are specified and some of them can be synthesized according to the particular choices done either to serve resource constraints or to represent particularly rich domains. This point of view of extending scheduling engines with some activity synthesizing capabilities has attracted some attention especially to manage complex process environments (see for instance Visopt ShopFloor system (Bartak 2003)).

These ideas are currently implemented in a prototypical solver called OMP. By means of a constraint based representation, OMP uniformly deals with causal and resource constraints "on top" of a shared layer representing temporal information as a Simple Temporal Problem (STP) (Dechter, Meiri, & Pearl 1991). For the causal reasoning I use a representation of domain components (called *state variables*), consisting in temporal automata, as first proposed in HSTS (Muscettola *et al.* 1992; Muscettola 1994) and studied also in subsequent works (Cesta & Oddi 1996; Jonsson *et al.* 2000; Frank & Jonsson 2003). The integrated planning and scheduling domain theory in OMP is described using DDL.2 specifications. DDL.2 is an integrated P&S domain description language (see (Cesta, Fratini, & Oddi 2004) for a more detailed description of this language and the OMP software architecture).

In this architecture activities that have to be scheduled are organized as a network, where nodes represents activities and edges represents quantitative temporal constraints between them. Activities no longer represent a "blind set" of entities that someone produced and gave to the scheduler, but they maintain information about mutual logical links; an integrated P&S architecture can manage both types of information (causal and resource) and solve the whole problem of generating and scheduling this set of activities.

This approach can be usefully applied to those domains where the scheduling problem is actually the hardest aspect, resource reasoning is very critical with respect to causal reasoning and requires specialized scheduling technologies. Those domains cannot be afforded with a planner that inte-

grates some generic scheduling features. Indeed this kind of domain often does not require strong planning capabilities. But enhancing the solver with some planning capabilities can allow to tackle problems in these domains in a more flexible way.

Scheduling with Causal Reasoning

In my work I have started from a CSP (Constraint Satisfaction Problem) scheduling framework able to manage temporal and resource constraints, and I tried to understand how to increase the capabilities of this framework with planning features, in fact describing a framework where both planning and scheduling problem instances have a common CSP representation.

In a typical scheduling problem, there is a set of *activities* that require certain amounts of *resources*. There are also a set of temporal constraints between these activities (duration and minimal and maximal separation between pairs of activities). Thus the problem is to find *when* to start each activity in order to ensure that all temporal constraints are respected and resources are never over or under used, a constraint due to their finite capacity. Such a scheduling problem can be solved for example with a *Precedence Constraints Posting* (PCP) approach (Cheng & Smith 1994; Cesta, Oddi, & Smith 2002), in fact building a temporal network where start and end points for each activity are mapped as time points. The underlying CSP model for temporal information is usually an STP. Reasoning about *resource profiles* it is possible to deduce a set of additional *precedence constraints* between activities that, when posted, ensure resources are never over used. The problem solving approach is sketched in fig.1.

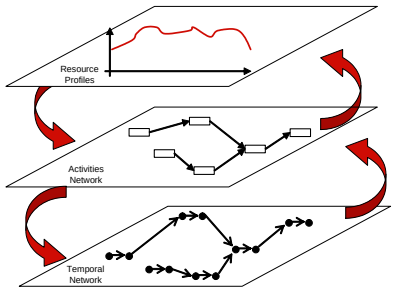


Figure 1: Activities Network Scheduling

As I said the scheduling problem can be seen as the output of a planning step, when two steps, planning and scheduling, are serialized, and the activity network come from causal reasoning while temporal separation constraints between activities come from cause-effect relations between planned actions. My aim is to go a little bit further than a serialization of these two steps, and present a way to model a scheduling domain where the modeler specifies how activities that have to be scheduled are linked to each other via cause-effect relationships, then find a solution for this problem, i.e. planning which activities have to appear in the scheduling problem, according to some *goals* that have to be achieved.

I followed a different paradigm with respect to the more common STRIPS ontology for domain description in planning literature (Fikes & Nilsson 1971). Rather than focusing on the executing agent performing actions, the chosen paradigm considers the relevant sub-parts of a domain that continuously evolves over time. Then instead of specify which action can be performed to modify the state of the world, under which conditions and with which effects, I directly specify which sequences of states are logically admissible for these sub-parts. The state of the entire world at each instant of time is the union of the values of these sub-parts at that instant. I call these sub-parts *state variables* because in fact there are entities, whose values over a temporal interval, determine what is going on on the world, in the form of temporally ordered sequences of state transitions. In addition to state variables, I use the notion of *resources* to model typical scheduling features, besides cause-effect relationships.

As we have seen, a scheduling problem can be modeled over a temporal network, as a network of activities with a start and an end time point, linked to each other with temporal constraints, where each activity requires a certain amount of some resource. In a similar way, calling *task* a temporal interval with a start and an end time point, a planning problem can be seen as a network of tasks, linked to each other with temporal constraints, where each task says that the state variable must take between its start and end point one of the values specified. Thus, as in scheduling we reason on resource usages by summing activity requirement at each time point and calculating resource profiles, likewise we can calculate a sort of "state variable profile" by intersecting for each time instant the values required by each task that overlaps in that time instant. With this representation of the planning problem we can see causal reasoning as a sort of *task scheduling*. The problem reduces to searching for a feasible ordering for these tasks taking care of the fact that each state variable needs to take at least one value at each instant. Thus a state variable is a sort of resource, where two tasks cannot overlap, unless their intersection is a non empty set of values.

Moreover, because *tasks* and activities in fact share the same temporal network, the integration of P&S is achieved by mixing temporal constraints in an environment where two specialized reasoners, a scheduler for resources and a *task scheduler* for state variables, analyze the situation of a temporal constraint database, in fact posting constraints that affect the whole integrated problem. Hence my point of view: state variables and resources can be seen as *concurrent threads* in execution on a concurrent system, where a shared temporal model allows crossing relations between causal and resource usage aspects, even if two distinct reasoners affect them. In figure 2 I show three different networks, that can be either activities or tasks networks. They are connected with cross temporal constraints (dashed lines in the figure) which link (1) the causal problem with the scheduling problem (for instance requiring a resource usage when a task is performed by a state variable); (2) causal problems on different state variables (for instance requiring that when a task is performed by a state variable another state variable must perform some other tasks). Sharing a common temporal network they share the same temporal

model, thus resources and state variable profiles in fact lie in the same temporal line, as concurrent threads. But at the same time causal, resource and temporal reasoning are clearly separated: the last one is represented in bottom layer in the figure, while the planning and scheduling problems are represented as networks in the middle layer. Finally I schedule these networks reasoning on resource and state variable profiles (represented in the top layer).

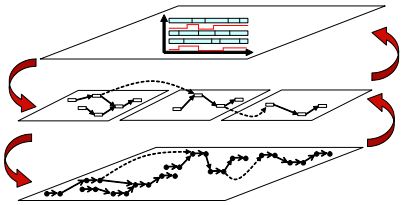


Figure 2: Tasks and Activities Integration

From this point of view, once we specify some goals we want to achieve, as for instance some tasks or activities that must appear in the final solution, in order to follow all constraints specified in the domain theory, several other tasks that have to be allocated over state variables and several activities that have to be allocated over resources are automatically produced, generating task and activity networks such as those we saw before, that share the same temporal information. Thus, once all networks are generated, in fact unfolding the domain theory and ensuring all cause-effect relationships are guaranteed according to the domain theory, the problem becomes to find a feasible ordering for these tasks and activities in a way that each state variable takes at least and no more than one value at each instant of time and resources are never over used. This purpose can be achieved by scheduling tasks and activities. Moreover propagation rules can be applied, in fact solving the whole problem as a CSP, alternating propagation and decision steps.

Problem Solving

In the pure scheduling case methods to deduce *necessary* constraints by propagating informations about resource usages have been widely studied. It is possible do deduce a set of ordering constraints between activities that must be posted, just because otherwise a resource violation surely occurs. Generally that is not enough to solve the problem. Sometimes a search decision is necessary, basically between feasible orderings of activities. Of course constraints posted during the propagation step are necessary, i.e. they prune only non-feasible solutions, meaning that *any* feasible solution *must* contain these constraints. On the other hand scheduling precedence constraints are *search* decisions, thus they *could* cut some feasible solutions. Thus it could be necessary to backtrack during the search and choose a different ordering for some activities if it is not possible to find any solution from that point on.

I am going a little bit further, studying how to schedule tasks over state variables, following the presented guidelines. I studied a method for discovering inconsistences in posted constraints (i. e. temporal intervals where the intersection of values allowed by constraints over that interval is an empty set) and a propagation algorithm able to discover new necessary ordering between tasks in order to avoid inconsistences over state variables (see (Fratini, Cesta, & Oddi 2005) for details). I am also working on OMP, an integrated constraint-based software architecture for planning and scheduling problem solving I realized to apply my ideas. This architecture implements an interval based planner and scheduler, based on the ideas presented above. Basically, starting from a domain theory it builds tasks and activity networks over a shared temporal network, then it schedules them as I have briefly explained.

The OMP software architecture essentially implements, from an abstract point of view, the typical CSP loop solving strategy, performing alternatively decision and propagation phases, starting from a CSP problem model. In an high level block view of OMP's architecture we see a *decision making* module that explores the search space posting constraints in a *constraint database*, that maintains information about the current partial solution and propagates decision effects pruning the search space, starting from a domain theory CSP model(see fig. 3).

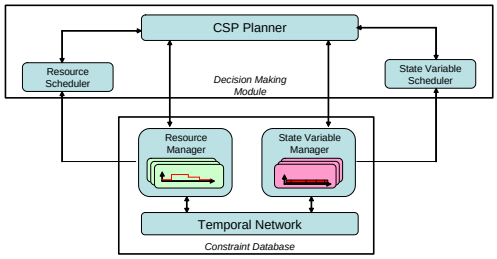


Figure 3: OMP Software Architecture

There are in OMP two modules, the *resource manager* and the *state variable manager*, that manage task and activity networks basically by performing constraints propagation strictly connected with two *scheduling modules* (in the decision making module) able to analyze the resource and state variable constraint databases and to calculate several sets of precedence constraints between activities and between tasks, precedence constraints that when posted over resource and state variable networks are able to guarantee that scheduling problems over these components are solved and each solution is feasible with respect to resource and state variable component constraints. These scheduling modules generate a search space, where at each node the decision making module can choose between different activity or task orderings. Resource propagation algorithms described in (Laborie 2003) have been implemented in a resource manager module, while I analyzed and implemented algorithms for the state variable manager. Both for resource and state variable scheduling I basically use the precedence Constraint Posting Approach adapted to the features of this architecture.

Temporal, resource and state variable networks constitute, from a CSP point of view, the constraint database in the OMP software architecture. The decision maker module

closes the loop. The *CSP planner* is the decision making module core: it explores the search space, making choices between: (1) which tasks and activities are necessary in order to force domain theory compliance and (2) which order to force among tasks on a state variable or activity over a resource when propagations are not able to prune all non-feasible orderings (this set is computed by the scheduler modules).

In fact the planner, starting from some goals (that is tasks and activities that must appear in the final solution) dynamically unfolds the domain theory putting more tasks and activities into networks. Every time that a new task or activity is added I deduce, via propagation rules, new constraints that affect the situation of all networks, due to the shared temporal model. Moreover it is feasible to discover dead ends, and by interleaving scheduling and unfolding steps we integrate planning and scheduling (DDL.2 allows the user to model different expansions for a task or an activity, thus different networks can be built for the same problem).

Conclusions

From a general point of view my thesis topic is related to planning and scheduling integration. My approach to that problem starts from a scheduling background, and I am studying how a constraint based scheduler can be endowed with the ability to synthesize new activities, i.e., through reasoning on planning knowledge.

I underlined how *Causal Knowledge* is the main distinguishing factor between planning and scheduling, thus building an architecture where causal reasoning can be performed behind time/resource reasoning allowed me to bridge the gap between these two AI research lines, extending from one hand pure planning schemes with quantitative time and resource reasoning and from the other hand extending pure scheduling schemes with a more complex domain theory.

My aim is to solve integrated planning and scheduling problems by rationally integrating in the same architecture various of the state-of-the-art CSP algorithms. Then I showed as modeling the integrated planning and scheduling problem as concurrent evolving components allows me to afford it as network scheduling, where networks are automatically generated in the same architecture from a compact domain theory description and some goals that have to be achieved.

References

Bartak, R. 2003. Visopt ShopFloor: Going Beyond Traditional Scheduling. In *Recent Advances in Constraints*, LNAI 2627, 185–199.

Cesta, A., and Oddi, A. 1996. DDL.1: A Formal Description of a Constraint Representation Language for Physical Domains. In M.Ghallab, M., and Milani, A., eds., *New Directions in AI Planning*. IOS Press.

Cesta, A.; Fratini, S.; and Oddi, A. 2004. Planning with Concurrency, Time and Resources: A CSP-Based Approach. In Vlahavas, I., and Vrakas, D., eds., *Intelligent Techniques for Planning*. Idea Group Publishing.

Cesta, A.; Oddi, A.; and Smith, S. F. 2002. A Constraint-based method for Project Scheduling with Time Windows. *Journal of Heuristics* 8(1):109–136.

Cheng, C., and Smith, S. 1994. Generating Feasible Schedules under Complex Metric Constraints. In *Proceedings 12th National Conference on AI (AAAI-94)*.

Chien, S.; Rabideau, G.; Knight, R.; Sherwood, R.; Engelhardt, B.; Mutz, D.; Estlin, T.; Smith, B.; Fisher, F.; Barrett, T.; Stebbins, G.; and Tran, D. 2000. ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling. In *Proceedings of SpaceOps 2000*.

Currie, K., and Tate, A. 1991. O-Plan: Control in the Open Planning Architecture. *Artificial Intelligence* 51:49–86.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal Constraint Networks. *Artificial Intelligence* 49:61–95.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3-4):189–208.

Fox, M., and Long, D. 2003. PDDL 2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124. Special issue on 3rd International Planning Competition.

Frank, J., and Jonsson, A. 2003. Constraint based attribute and interval planning. *Journal of Constraints* 8(4):339–364. Special Issue on Planning.

Fratini, S.; Cesta, A.; and Oddi, A. 2005. Extending a Scheduler with Causal Reasoning: a CSP Approach. In *Proceedings of the Workshop on Constraint Programming for Planning and Scheduling (ICAPS'05)*.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL — The Planning Domain Definition Language, AIPS 98 Planning Competition Committee.

Jonsson, A.; Morris, P.; Muscettola, N.; Rajan, K.; and Smith, B. 2000. Planning in Interplanetary Space: Theory and Practice. In *Proceedings of the Fifth Int. Conf. on Artificial Intelligence Planning and Scheduling (AIPS-00)*.

Laborie, P., and Ghallab, M. 1995. Planning with Sharable Resource Constraints. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)*.

Laborie, P. 2003. Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and new Results. *Artificial Intelligence* 143:151–188.

Muscettola, N.; Smith, S.; Cesta, A.; and D'Aloisi, D. 1992. Coordinating Space Telescope Operations in an Integrated Planning and Scheduling Architecture. *IEEE Control Systems* 12(1):28–37.

Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., ed., *Intelligent Scheduling*. Morgan Kauffmann.

Srivastava, B.; Kambhampati, S.; and Do, M. B. 2001. Planning the project management way: Efficient planning by effective integration of causal and resource reasoning in realplan. *Artificial Intelligence* 131(1-2):73–134.

Swarm Intelligence based Information Sharing for PSO based Machine Scheduling

Haibing Gao
Wayne State University
Industrial and manufacturing engineering
4815 Fourth Street Room 2033, Detroit, MI 48202
au3443@wayne.edu

Introduction

Machine scheduling problems such as job shop scheduling, flow shop scheduling and open shop scheduling, which are NP-hard combinatorial optimization problems with strong engineering background, have been receiving increasing attention from researchers. In this paper, a modified Particle Swarm Optimization (PSO) based scheduling model-MPSO is proposed to generate optimized machine schedules. Given the limitations of information sharing mechanism in generalized PSO (GPSO) model, a new Swarm-intelligence based Information Sharing Mechanism-SISM is proposed. MPSO algorithm under SISM mechanism explores problem domain specific knowledge of scheduling problems to obtain good balance between experience oriented global search and knowledge directed local search. Compared with representative scheduling algorithms, MPSO can maintain good balance between quality of the machine schedules and computational expense. Simulation results validate its efficiency on benchmarking job shop scheduling datasets.

SISM and MPSO Model

Definition 1: Information sharing mechanism (ISM) is defined as the topology of information flow in swarm intelligence based heuristics. ISM is mainly concerned with "where" each individual in the swarm population should obtain the necessary updating information. Through the analysis of information flow in the swarm population, the optimization mechanism special to the specific algorithm can be neglected or be reduced to a more generalized information-sharing framework introduced by ISM. With this framework, we can conduct both theoretical and experimental analysis in a more direct and standard way.

Definition 2: Information updating operator (IUO). Given the information sharing mechanism defined above, the next crucial step for swarm intelligence based heuristics is to present the detailed implementation-IUO to realize the proposed ISM. Different IUO can be well adapted to different environment that is in fact represented by the problem to be optimized. Under a certain information sharing mechanism, appropriate design of IUO could improve information flow efficiency of the proposed algorithm.

As a novel heuristic based on swarm intelligence, Particle Swarm Optimization (PSO) (Kennedy and Eberhart 1995) has been applied to many continuous problems especially those engineering optimization problems. However, due to the limitation of the velocity-displacement model of traditional PSO algorithm, it is difficult for PSO to address discrete and combinatorial optimization problems without modification. Based on the traditional velocity-displacement model, the optimization mechanism of PSO is studied and generalized PSO (GPSO) model has been proposed by the authors. Based on the GPSO model, we have proposed the PSO algorithm that could solve Travel Salesman Problem (TSP) efficiently. However, when extended to machine scheduling problem, severe premature convergence in GPSO model is a particular problem.

To particularly overcome the limitations of the information sharing mechanism in the GPSO model, a new Swarm-intelligence based Information Sharing Mechanism-SISM is presented and some analytical findings are put forward in detail by simulating simple social activities of human beings. Then, its IUO implementation-MPSO algorithm for machine-scheduling problems is presented.

The core concept in SISM is memory pool that is composed of memory information with high quality introduced by individuals in the form of experience. In the proposed SISM mechanism, individuals obtain updating information from the memory pool. That is to say, the individuals obtain information not only from its own experience but also from the common successful experience of other individuals in the whole population. Compared with the traditional mechanism, the new SISM can lower the possibility of premature convergence.

The proposed MPSO scheduling algorithm explores knowledge of the specific problems, thus conducting knowledge-directed local search to eliminate ill effects introduced by randomness or blindness introduced by its experience or memory-oriented global search. Based on knowledge concerning efficient moves of operations in critical blocks, neighborhood structures are proposed. Experimental results show knowledge-directed local search, which utilizes these neighborhood structures, improves the quality of machine schedules effectively. The framework of the proposed MPSO scheduling model under SISM is as follows:

Initialize the particle population: generate a set of schedules randomly or based on heuristic rules.
Do {
  For each particle $s_i(t)$ do {
    Evaluate with objective function defined as $C_{\max}(s_i(t))$
    Set individual best schedule $p_i(t)$:
      if $C_{\max}(s_i(t)) \le C_{\max}(p_i(t-1))$
        $p_i(t) = s_i(t)$
    Update memory information pool
    if $rand() \le p$
      Update the schedule through IUO with $p_i(t)$
    else
      Obtain new schedule through IUO with $m_i(t)$ from memory pool
    Local search procedure based on the specific neighborhood structure
} While stopping criteria are not satisfied

Figure 1: MPSO model based on SISM mechanism.

Neighborhood structure of JSP

This paper analyzes the neighborhood structures of JSP. In general, the neighborhood structures are similar in that they are all defined using blocks of operations in a critical path.

The most popular domain knowledge for JSP is optimal neighborhood moves in the critical path. In this paper, the neighborhood structure adapted from (Monaldo and Luca 2000) is a reduced set of possible neighbors to a subset for which can be proved that it always contains the neighbor with the lowest makespan. It is convenient to represent the JSP by using the disjunctive graph model. In disjunctive graph model, there is a node for each operation. The nodes are numbered from 1 to *N*, where $N = m \times n$ is the total number of operations. Additionally there are two dummy nodes, node 0 and node *, representing the start and end of a schedule, respectively. Each node has a weight, which is equal to processing time d_v of the corresponding operation *v*. The starting time s_v of operation *v* is equal to the length of a longest path from node 0 to node *v*, the tail time t_v corresponds to the length of a longest path from node * to node *v*.

The neighborhood structure was based on the perturbation of insertion operation. That is, for a specific node *v* in the graph, firstly, remove *v* by deleting all its machine arcs and set the weight of node *v* equal to 0. Then shift *v* to a new position by adding its machine arcs and setting the weight of node *v* equal to d_v. Let G^- be the graph obtained form *G* at the end of first step (In the following, all the notations after the first step were denoted by the combination of the former notions and the superscript "-", i.e., let s^- and t^- be the starting and tail time of a genetic operation in G^-, and *v* denotes the operation to be moved). Let Q be the set of operations processed by the machine, which also processed the operation *v* in G^- and sorted by increasing starting time. Let *R* and *L* denote two subsets of *Q* which can be obtained as follows, $R = (x \in Q \mid s_x + p_x > s_v^-)$, $L = (x \in Q \mid s_x + t_x > t_v^-)$. Let F_v be the set of solutions obtained by inserting *v* after all the operations of *L**R* and before all the operations of *R**L*. It can be proved that F_v is a set of feasible neighbors, and also there is an optimal insertion of *v* in F_v.

The value of the low longest path was used to assess the effectiveness of a given insertion. The length of this path is a lower bound of the new solution. To reduce the computational expensive and avoid the calculation of longest path, a procedure used by (Dell' Amico and Trubian 1993) was adopted to compute upper bounds instead of the exact values. In order to minimize the makespan, it may be profitable to consider only neighbors, which are obtained by inserting operations that belong to a critical path in the solution graph of the current schedule. In the MPSO model, the neighborhood search was applied for each individual in the population. To save computational cost, the above insertion was performed by specific probability, which can also avoid devious search.

Experimental Results

In view of the encouraging potentials of GPSO in combinational optimization when dealing with the TSP, we attempt to extend the GPSO model to handle more complex combinatorial optimization problems with engineering background such as job shop scheduling problem. The detailed implementation framework can be referred to the reference concerned with GPSO model. Here we just provide the computational results obtained by the respective algorithms based on GPSO model and MPSO model. Several representative JSP instances from ORLIB are used to examine their performance. Table 1 lists the makespan of best solution, average solution, and computational time of the algorithms based on these two models. Also the iteration numbers are recorded when the approximate convergence is reached. As can be seen from the table, though the GPSO model is superior in term of convergence speed at early stage, this merit is at the cost of severe premature convergence and failure to generate satisfactory schedules.

First, we examined the performance of MPSO without neighborhood search procedure directed by problem specific knowledge. Table 1 also shows the results obtained by MPSO and the corresponding GA. The results reveal that the MPSO proposed in this paper performs better than GA in view of makespan of the generated schedules and computational expense.

Table 1: Comparison between the MPSO, GPSO and GA model.

Problem	Opt	MPSO				GPSO				GA		
		Best	Ave	Time (s)	Gen	Best	Ave	Time (s)	Gen	Best	Ave	Time (s)
FT10	930	930	945	244.19	100	956	967	20.53	14	946	968	271.59
FT20	1165	1173	1181	275.28	100	1224	1236	15.52	17	1178	1203	307.39
LA16	945	945	950	179.12	80	979	982	9.11	11	956	963	219.01
LA21	1046	1058	1071	486.87	150	1092	1108	18.58	14	1082	1105	613.37
LA31	1784	1784	1784	138.07	50	1784	1784	34.23	10	1788	1796	197.66
LA36	1268	1278	1292	692.64	150	1312	1331	53.76	21	1302	1334	1024.67

Table 2: Experimental results of MPSO on JSP benchmarks.

Inst.	Size	BKS	MPSO			(Aiex *et al*., 2003)	(Binato *et al*., 2002)	(Nowicki and Smutnicki, 1996)	(Gonçalves and Beirão, 1999)	(Wang and Zheng, 2001)
			Best	Avg	Time					
FT10	10×10	930	**930**	930	23.906	930	938	930	936	930
FT20	20×5	1165	**1165**	1173	107.87	1165	1169	1165	1177	1165
LA01	10×5	666	**666**	666	0.079	666	666	666	666	666
LA05	10×5	593	**593**	593	0.047	593	593	593	593	-
LA10	15×5	958	**958**	958	0.109	958	958	958	958	-
LA11	20×5	1222	**1222**	1222	0.235	1222	1222	1222	1222	1222
LA16	10×10	945	**945**	945	40.875	945	946	945	977	945
LA17	10×10	784	**784**	784	7.235	784	784	784	787	-
LA18	10×10	848	**848**	848	4.848	848	848	848	848	-
LA19	10×10	842	**842**	842	11.125	842	842	842	857	-
LA20	10×10	902	**902**	902	26.860	902	907	902	910	-
LA21	15×10	1046	**1046**	1052	101.472	1057	1091	1047	1047	1058
LA22	15×10	927	**927**	927	144.407	927	960	927	936	-
LA23	15×10	1032	**1032**	1032	6.62	1032	1032	1032	1032	-
LA24	15×10	935	**935**	938	252.753	954	978	939	955	-
LA25	15×10	977	**977**	977	147.953	984	1028	977	1004	-
LA26	20×10	1218	**1218**	1218	34.813	1218	1271	1218	1218	1218
LA27	20×10	1235	**1235**	1246	427.73	1269	1320	1236	1260	-
LA28	20×10	1216	**1216**	1216	195.42	1225	1293	1160	1190	-
LA29	20×10	1157	**1153**	1166	457.06	1203	1293	1160	1190	-
LA30	20×10	1355	**1355**	1355	11.640	1355	1368	1355	1356	-
LA31	30×10	1784	**1784**	1784	1.640	1784	1784	1784	1784	1784
LA32	30×10	1850	**1850**	1850	1.343	1850	1850	1850	1850	-
LA33	30×10	1719	**1719**	1719	1.219	1719	1719	1719	1719	-
LA34	30×10	1721	**1721**	1721	2.813	1721	1753	1721	1730	-
LA35	30×10	1888	**1888**	1888	1.109	1888	1888	1888	1888	-
LA36	15×15	1268	**1268**	1274	451.75	1287	1334	1268	1305	1292
LA37	15×15	1397	**1397**	1402	545.84	1410	1457	1407	1441	-
LA38	15×15	1196	**1196**	1198	327.40	1218	1267	1196	1248	-
LA39	15×15	1233	**1233**	1236	519.65	1248	1290	1233	1264	-
LA40	15×15	1222	1224	1228	660.14	1244	1259	1229	1252	-

We can see from the above statistical results the superiority of proposed information sharing mechanism-SISM to those of GPSO and GA. However, when dealing with large-scale instances, the MPSO algorithm without the direction of problem specific knowledge has difficulty in obtaining satisfactory machine schedules effectively. To address this drawback, the MPSO algorithm with local search based on knowledge of the specific problem is proposed and validated by more instances from ORLIB and the proposed algorithm is compared with some well-known algorithms. Table 2 displays the computational results. It lists instance name, dimension of the instance, the best-known solution (BKS), CPU time (in seconds), and the best and average makespan of schedules obtained by each algorithm. As can be seen from the table, the proposed MPSO shows an improvement with respect to all others algorithms. Most of the instances have been solved by MPSO to the BKS except LA40, but the gap is only 0.16%. Furthermore, the solution for LA27 instance is even better than the BKS in the literature. The average computing times varies from 0.031s (for FT06 problem) to 660.142s (for LA40 problem). All the computational expenses are in reasonable range. In addition, the comparison of two MPSO algorithms clearly reflects the importance of problem specific knowledge.

Conclusion

To overcome the limitation of the traditional ISM of PSO algorithm and GA, a new information sharing mechanism-SISM is proposed in this paper. The memory information is the core part of SISM, and the notion of the corresponding concept-memory pool is then introduced. The memory pool is updated each iteration. The updating principles ensure the high quality and good diversity of the individuals in swarm population. Experimental results confirm that the proposed algorithm based on SISM can deliver an excellent balance between global search and local search, and experience-oriented random search and knowledge-concerned directed search.

The notion of neighborhood structure based on problem domain specific knowledge is introduced into the MPSO scheduling algorithms. The effective combination of the neighborhood search directed by problem-concerned knowledge improves efficiency of useful information flow and expedites the convergence process. The SISM information sharing mechanism and the directed neighborhood search can be regards as a generic strategy and extended to other complex combinatorial optimization problems in engineering.

References

Kennedy, J., and Eberhart, R.C. 1995. Particle swarm optimization. *IEEE International Conference on Neutral Networks*, 4: 1942–1948.

Monaldo, M., and Luca M.G. 2000. Effective Neighborhood Functions for the Flexible Job Shop Problem. Technique report. IDSIA, Lugano, Switzerland January 31, 2000.

Dell' Amico, M., and Trubian M. 1993. Applying tabu-search to the job shop scheduling problem. *Annals of Operations Research* 22:15-24.

Aiex, R.M., Binato, S., and Resende, M.G.C. 2003. Parallel GRASP with Path-Relinking for Job Shop Scheduling. *Parallel Computing* 29(4): 393-430.

Binato, S., Hery, W.J., Loewenstern, D.M., and Resende, M.G.C. 2002. A GRASP for Job Shop Scheduling. In: *Essays and Surveys in Metaheuristics*, Ribeiro, Celso C., Hansen, Pierre (Eds.), Kluwer Academic Publishers.

Nowicki, E., and Smutnicki, C. 1996. A Fast Taboo Search Algorithm for the Job-Shop Problem, *Management Science* 42(6): 797-813.

Gonçalves, J.F., and Beirão, N.C. 1999. Um Algoritmo Genético Baseado em Chaves Aleatórias para Sequenciamento de Operações. Revista Associação Portuguesa de Desenvolvimento e Investigação Operacional, 19:123-137 (in Portuguese).

Wang, L., and Zheng, D. 2001. An effective hybrid optimization strategy for job-shop scheduling problems, *Computers & Operations Research* 28: 585-596.

A Mixed Initiative framework for dynamic environments.

Óscar García-Pérez
Dept. of Computer Science and Artificial Intelligence
ETSI Informática, University of Granada
18071, Granada, SPAIN
oscar@decsai.ugr.es

Abstract

In complex dynamic domains, the design and execution of plans of actions is a very difficult task that involves a lot of personnel and resources. We can not rely exclusively in an automatic planner system to solve this type of problems. In this situations, the planning domain may not include all the necessary knowledge, or not know in advance all the possible contingencies. Thus the designed plan might fail during its execution, or even if it doesn't fail and the plan is correct may be it is not suitable for a human expert. In these cases the help and lead of a domain expert becomes very valuable, and a planning system should provide different ways for him or her to participate in the planning process.

Introduction

As planning algorithms and techniques become more sophisticated, the need of tackling more complex and real problems appears. Today planning systems are not used exclusively by scientists for research purposes, but they are also being embedded into larger intelligent systems in different fields, ranging from crisis management to tourism. There is a need to transform the classical, closed, fully automated planners into new open interactive ones. There are mainly three reasons for that. Firstly, the planner need to be integrated as an agent into larger complex systems. It has to communicate with other agents consuming their services and offering new ones, forming a net of interoperable intelligent systems. Secondly, the planner needs to deal with the uncertainty of the knowledge being used to obtain plans. There is knowledge that is incomplete or imprecise, so the planner may not know in advance if the resulting plan is correct or even if it is going to be executed correctly (imprecision). The planner would need to interact with other systems (automated or human) trying to minimize this lack of knowledge and improve the robustness and adaptability of the resulting plan. And thirdly, the resulting plan is usually offered to a human expert that has to validate it. This person is responsible for the consequences of the plan execution, so we need the resulting plan to satisfy him or her. Because of that, we need this expert to get involved in the planning process, helping and leading the planner to reach a satisfactory plan. In order to do that, we need to provide tools for the correct visualization, comprehension, explanation, and edition of the plan. These tools have to be designed taking into account that the the final user is not an expert on planning, so we need the language used by this tools be able to communicate with the user, natural, easy and simply.

In this work we present our planner SIADEX as a system that needs to incorporate mixed initiative techniques in order to face dynamic and real problems. In the next section we will present a general overview of the SIADEX system, next we will explain how the mixed initiative techniques can be embedded in our system, finally we will conclude with some remarks.

The SIADEX system

SIADEX is a system we are developing, under a research contract with the Andalusian Regional Ministry of Environment. Its objective is to assist the command technical staff in the design, progress and dispatching of forest fire fighting plans. Currently we are starting the last of three years of contract, so the system is in an advanced phase. The architecture is composed by different, domain independent agents (Figure 2). Each agent offers different services, that are distributed over different computers and platforms, and communicate with each other using the XML-RPC protocol.

- BACAREX: Is an ontology server, that stores the knowledge related to the planning domain. In our case its stores information about the forest fire fighting domain in Andalusia (Spain). It contains among other information. meteorological and geographical data, and all the facilities and resources in our region (about 1900). BACAREX is also capable of generating domains and problems (in our PDDL extension) that are processed by our planning agent. BACAREX also has a private web user interface, that lets the staff related with forest fire fighting to modify or query this information. The ontology has been designed using Protege (Protege. 2005), and it's stored in a relational database.
- SIADEX Planner: It's a fully operational planner (we are now in testing phase). SIADEX is a forward state-based HTN temporal planner(de la Asunción *et al.* 2005), with the following main features:
  - It uses a hierarchical extension of PDDL 2.2 level 3 language. This extension is a superset of the standard

PDDL language, that gives new expressions to support the definition of hierarchical task networks, and manage the time. This language has also the capability to let us to include Python embeddable scripts in the domain, as axioms or functions, for being able to implement external calls, present dialogs or do another computations at planning time. The language is easy to read and learn if you have previous experience with PDDL and HTN planning.

– It has an embedded debugger utility. This tool has become a fundamental part of the planner for being able to test and validate new domains or problems, specially when the domains became big an complex. The debugger gives the possibility of define breakpoints, query the state of the planning process, and change the behaviour of the planner at run time.

– The planner is capable of perform temporal reasoning at different levels of abstraction and define constraints over time using temporal networks (Dechter 2003).

We intend to present this and other characteristics of the planner to the research community as soon as we perform some benchmarking and comparison with other systems.

- User interface module (Figure 1): We provided GUI capabilities to the planning service for the experts. The GUI is build on top of the ArcView GIS tool (ESRI). The GUI is totally domain dependent and oriented toward the interaction with the forest fire technical staff. With this decision we minimize the learning curve of the system, because the staff is working with a tool that already knows, and also gives the GUI the possibility to present and interact with cartographical and meteorological related information.
- Monitor module: This module is not fully developed although we have some background about it. Our goal is to be able to split the plan into several pieces, and send every piece to the person or people in charge of execute it (chief of brigade, helicopter pilot...). These parts of the plan will be presented to the user using any portable electronic device (PDA, laptop, or cell phone). The monitor will be able to gather the information provided by this people about the status of the current tasks, monitor the global plan progress, and present it to the operations chief.
- Replanning module: When a contingency in the plan is found, this agent has to repair the previous plan or build a new one. The replanning capabilities have not been developed yet. We are now studying, and exploring ways to perform it in a mixed initiative way.

Mixed initiative and SIADEX

There are several opportunities of interaction between the planner agent and its environment, ranging from goal definition to plan validation, execution and monitoring. Generally speaking the more interaction with other systems, the more precise information the planner has, and the better resulting plans fits user needs. But depending on the situation, the opportunities of interaction, specially with human users, are more or less restricted. In a real forest fire, the time from the expert (generally the chief of operations), who is also

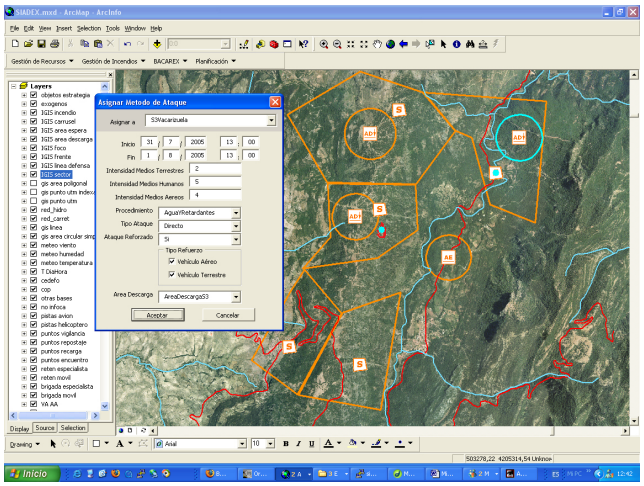


Figure 1: GUI of SIADEX on top of ArcMap.

on a stressfull situation, is a very valuable resource and can not be expended. In contrast, in a simulated training forest fire scenario, when the tool is used to test new strategies and approaches, there are a lot of opportunities of interaction. Depending on the stage, the agents involved (human or machines) and the situation, the planner has to adapt its interactions and be prepeared for work with a lot of or without knowledge.

Situation assessment

During this phase, the knowledge about the planning context is gathered from other systems, and the goal to be solved is defined. In a crisis management domain, defining the goal to tackle is not always an easy task. The human expert doesn't know in advance what exactly has to be done. He has a limited quantity of resources and a lot of work to carry out. He need to prioritize between the different tasks that need to be performed, assigning more resources to the most critical tasks, or even discarding some tasks because the lack of resources or the risk involved. Sometimes he knows perfectly how to perform some part of the plan, and others ones he has only a vague idea. There are different techniques that are being studied to help the expert to describe the problem.

- Specify goals with different granularity. In SIADEX the user can specify goals with different levels of detail. The expert can choose what resources to use, and assigns to one or a group of them a particular sequence of tasks. The tasks can also be concrete and precise, or vague at different levels of abstraction. For example a certain brigade might be required either to "clean out the vegetation one particular zone", as a concrete goal, or to "perform preventive measures in the area 2", as an abstract goal, and let the system to plan a more concrete sequence of tasks for them. Another interesting feature is that the expert could choose the resources used to perform one particular task. Currently in SIADEX the selection of resources is a fully automatic task based on the time required for each resource (helicopters, airplanes, fire fighters...) to

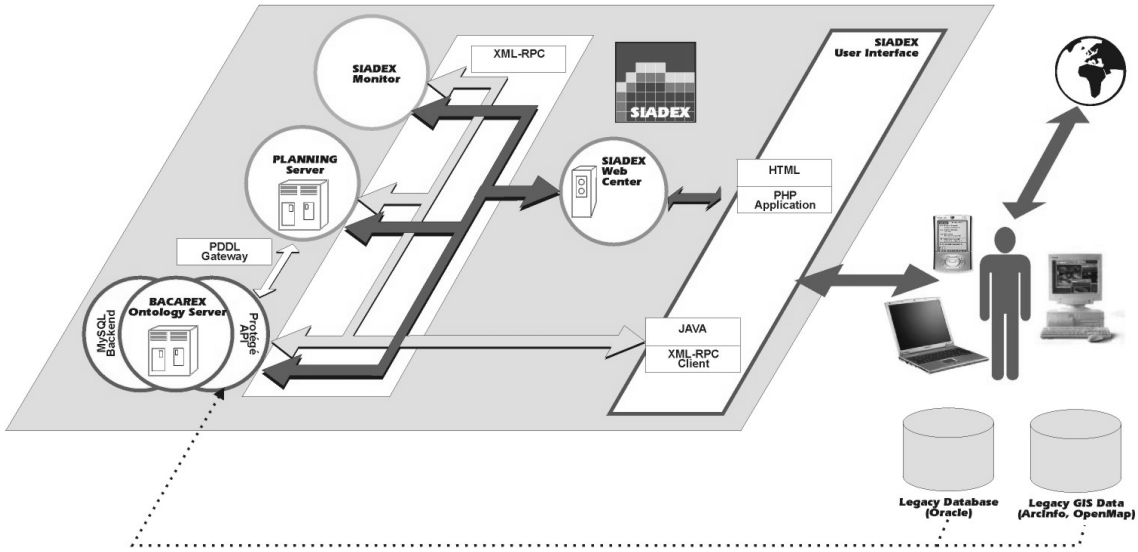


Figure 2: General overview of SIADEX Architecture.

go to the fire, but we want to extend this mechanism so the user can specify the criteria used in the selection (distance,experience,money...), what particular resources to use, the number and type of resources to send, the "force" needed to combat the fire system, and any combination of them. In this way depending on the requirements of the user, the system will choose what resources to use. The final idea is that the expert may provide a minimum input to obtain an approximate good candidate plan, or provide more elaborated and defined input in order to obtain a more personalized solution.

- Provide guidance to the plan search process. We are studying two ways to enable the user to lead the planner search process. One of them is to write certain rules that act as constraints that the final plan must meet (Myers 1996). And the other one is to let the user to describe parts of the plan in the form of sketches. The planner then is in charge of expanding the sketches and glue together the different parts, to obtain a final consistent plan (Myers *et al.* 2003). We are studying the way to incorporate the former of these techniques into SIADEX. We think that despite the second technique can obtain, from the point of view of the expert, more customized plans, it also requires from the expert a higher level of specialization and knowledge about the domain and the planning process. The second technique can be used in a training environment when the expert can expend more time elaborating his strategies, and can be used to explore, refine and discover new strategies that can be standardized. But in a stressful and changing environment like forest fire, what is required, is a quick and good candidate plan, that can be easily adapted and change over the time.

Plan generation

Once the problem has been properly set, the planning agent obtains (or not) a plan. In an HTN framework the planning process can be fully automated or guided by the interaction with the expert. The user can negotiate with the planner about what decisions to choose. Currently, the SIADEX planner is open to interactions during the whole planning search process, even these interactions might be previously set in the domain description, allowing the user to impose his decisions and prune some other choices. In our opinion, this is very useful for staff with a considerable background on planning techniques, and a very interesting line of research. However, from the point of view of the end user, this might not be very "user-friendly". We have to consider that a "real" plan involves thousands of HTN expansions, and a considerable amount of backtracking, that makes even for a well trained user to know what is happenning. It is very difficult to find a way of collaboration between the expert and the system that don't require certain training and knowledge about the planning process to obtain satisfactory results. We firmly believe that one of the most important requirements for a good mixed initiative system is that the system must fit the user and not the opposite.

Finally, once a candidate plan is found, it is submitted for validation to the end user. He may decide to accept the plan, to find another one or to patch the existing plan. In this sense we are studying how to extend a previous work on regenerative planning for a partial order planner (de la Asunción *et al.* 2003) to the HTN paradigm. This plan patching process allows the user to edit the plan, to remove undesirable actions and to include new actions under his responsibility. After the edition of the plan, the user may request the system to regenerate it to obtain a full valid plan and we are studying two ways to achieve this goal. The first one is based on a local regenerative process that searches for valid completions of the plan. It has to be

done taking into account the tasks and primitive actions included in the HTN domain (de la Asunción *et al.* 2003), and also the plan rationale, that is, the set of causal links and order relations existing between primitive actions. And the second one is based on a local repairing process based on term rewriting techniques (Dershowitz & Plaisted 2001; Ambite & Knoblock 1997) using a set of rewriting rules that maintain the correctness of the modified plan regarding the decomposition methods of the domain and also its causal structure.

Another problem we found, is that due the tremendous number of actions involved in one plan, made in parallel and mix together, it is very difficult for the expert to generalize and made himself a global idea on the general strategies and decisions taken. So we are studying different ways to graphically display the plan, based on the time, the geographical disposition, the resources involved and the HTN and causal structure of the plan (i.e. in Figure 3 we see a plan generated by SIADEX in XML MsProject file format).

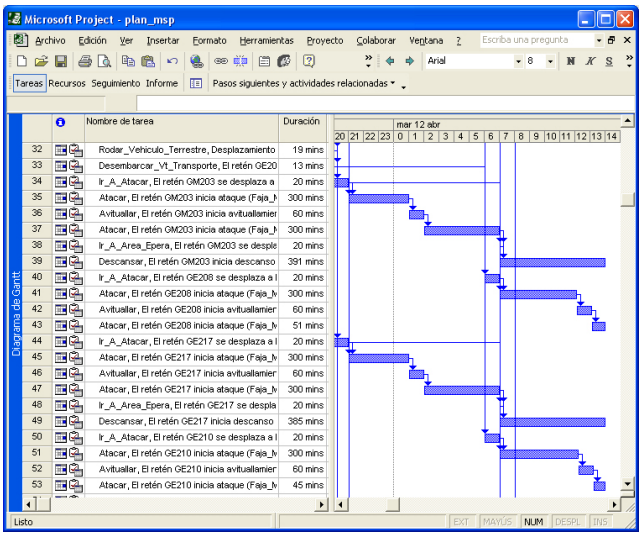


Figure 3: A SIADEX plan displayed as a Gantt chart.

Monitoring, dispatching and repairing

When the plan is obtained and validated by the staff, the execution begins. SIADEX is able to generate temporal plans, that are continuously executed, following the best temporal ordering. The plan is divided in pieces that are dispatched to the different agents that have to perform them. But in a dynamic domain like crisis management in general, and forest fire fighting in particular, we can not expect a correct execution of the plan. Even the planer can not automatically detect all the possible failures in the plan without the help of a human operator. We have to mention that in the SIADEX framework, the final users have different privileges over the plan visualization, and modification, depending on his her rank in the command hierarchy. Only one person has the privilege to modify the current plan, or ask for a new one, in our case the chief of operations. The rest of the personnel, only has access to the part of the plan that involves or is important for him or her job. Failures in the execution of the plan are signaled by these people and are registered by the monitor. Once the fail has been signaled, the chief of operations is informed, and a plan patching process similar that the one described above is carried out. Once the plan is fixed taking into account the actions that already have been performed, the plan continues with its execution. This is a "continual planning" approach in where we interleave several planning and execution stages.

Concluding remarks

In this work we have presented a HTN planning framework that tries to solve real an dynamic problems. We have explained, that this type of problems are very difficult to tackle without human intervention. Finally we have presented some mixed initiative techniques that can be applied to our planning system. Mixed initiative techniques can be applied in every phase of the planning process ranging from the problem definition to the execution and monitoring of the plan and almost always tend to enrich the resulting plan, or at least makes them more suitable from the point of view of the human expert. Despite that, in our opinion, there are some techniques that are better than others, depending on the environment in where the planner is going to be used, and the type of users that have to manage it. We argue that, in general the best techniques are those who reduce human interaction and don't require a planning expert to successfully make use of them.

References

Ambite, J. L., and Knoblock, C. 1997. Planning by rewriting: efficiently generating high-quality plans. In *Proceedings of the fourteenth AAAI*.

de la Asunción, M.; Castillo, L.; Fernández-Olivares, J.; García-Pérez, O.; González, A.; and Palao, F. 2003. Local (human-centered) replanning in the siadex framework. In Castillo, L., and Salido, M., eds., *Conference of the Spanish Association for Artificial Intelligence, II Workshop on Planning, Scheduling and Temporal Reasoning*, 79–88.

de la Asunción, M.; Castillo, L.; Fdez-Olivares, J.; García-Pérez, O.; González, A.; and Palao, F. 2005. Siadex: an interactive artificial intelligence planner for decision support and training in forest fire fighting. To Appear in AIComm special issue on Binding Environmental Sciences and artificial intelligence.

Dechter, R. 2003. *Constraint processing*. Morgan Kaufmann.

Dershowitz, N., and Plaisted, D. A. 2001. Rewriting. In *Handbook of automated reasoning*. Elsevier.

ESRI. http://www.esri.com.

Myers, K.; Jarvis, P.; Mabry-Tyson, W.; and Wolverton, M. 2003. A mixed-initiative framework for robust plan sketching. In *Proceedings of ICAPS 03*, 256–265.

Myers, K. 1996. Advisable planning systems. In *Advanced planning technology*. AAAI Press.

Protege. 2005. Stanford Medical Informatics. http://protege.stanford.edu.

Machine learning of plan robustness knowledge about instances

Sergio Jiménez
Departamento de Informática
Universidad Carlos III de Madrid
Avda. de la Universidad, 30. Leganés (Madrid). Spain
sjimenez@inf.uc3m.es

Abstract

Classical planning domain representations assume all the objects from one type are exactly the same, unless explicitly represented by literals. But when solving problems in the real world with autonomous systems, the execution of a plan that theoretically solves a problem can fail because of the special behaviour of an object that was not captured in the problem representation. This is one type of uncertainty about the world that I would like to capture through the integration of planning, execution and learning. In this paper, I describe an architecture that generates plans, executes those plans, and automatically acquires knowledge about objects that were used in the plans. This knowledge strengthens the planning and execution processes in the future by considering which instances of types provide a more robust behaviour with respect to operators.

Introduction

Suppose an organization whose staff consists of two programmers, **A** and **B**. **A** has worked in the enterprise for one year while **B** is unexperienced. Theoretically they can do the same work, but it would be common sense that at the beginning, difficult tasks were assigned to **A**, and then gradually start to share them with **B** as **B** shows worthy. The project manager will have first to test the behavior of **B** and then gradually acquire knowledge about how **B** behaves with respect to different actions (establish requirements, design, implement, maintain, document, ...). An automated planner, as the manager does, should also evaluate the behavior of the world objects (instances) to improve the quality and robustness of its plans. Examples of this type of situations arise in most real world domains, such as project management (as before), workflow domains (some people perform some set of actions better than others, and another set of actions worse than others), robotics (some robots are more robust with respect to some actions), military domains, etc.

Here, I assume that the robustness of the execution of actions only depends on how actions are instantiated (which worker performs which action, or, in other terms, which value I assign to parameter `worker` of action `design`) rather than depending on what state the action is executed. The latter would be, basically, what reinforcement learning would do in the case of MDP models (fully instantiated states and actions). It also does not depend on what characteristics a given instance has, because the values of these characteristics might not be known "a priori". Otherwise, they could be modelled in the initial state. For instance, one could represent the level of expertise of programmers, as a predicate `expertise-level(X,Y,Z)` where `X` can be instantiated to a programmer, `Y` is an action, and `Z` can be a number, reflecting the uncertainty of letting `X` execute action `Y`. Then, the robustness of each plan could be computed by cost-based plans. So, I would like to acquire knowledge about the uncertainty associated to instantiated actions, without knowing "a priori" the facts from the state that should be true in order to influence the execution of an action.

To acquire this knowledge, I have made the following assumptions (I describe how I plan to relax these assumptions in the future work section):

1. As I have already mentioned, the robustness of the execution of plan actions only depends on the instantiation of the actions parameters, and it does not depend on the states before applying the actions.
2. A domain consists of a set of operators, and a set of specific instances that will be used as parameters of the actions for which I would like to acquire the model. This is something relatively different from the way in which planning domains are handled, since they usually do not include specific instances, which appear in the planning problems. But, in many domains, instances are fixed for a given organization, or part of, where I will use planning. So, in workflow or project management, a given organization is composed of a set of known people, or a given robot environment is formed by a fixed set of robots. I will force that every problem in which I use an instance with name `A` refers to the same `A` than the rest of problems. In the extreme, I found domains, as in some industrial domains where control sequences should be generated, in which each domain also incorporates the problem description, as presented in (Fernández, Aler, & Borrajo 2005). This assumption is only needed for learning and is not really needed for deterministic planning purposes.

This paper is organized as follows: first, I present the gen-

eral architecture of the system and the planning, execution and learning processes, next I describe the performed experiments, and finally I discuss some conclusions.

The Planning-Execution-Learning Architecture

The PELA system (Planning-Execution-Learning Architecture) acquires gradually and automatically knowledge about the uncertainty associated to instantiated actions through repeated cycles of planning, execution and learning, as it is commonly done in most real world planning situations by humans. In our approach, a planner begins with a deterministic knowledge of the world dynamics, and step by step adapts itself to the learned dynamics. I use control knowledge to re-use the learned knowledge from previous plans executions to guide the planner search of a more robust solution.

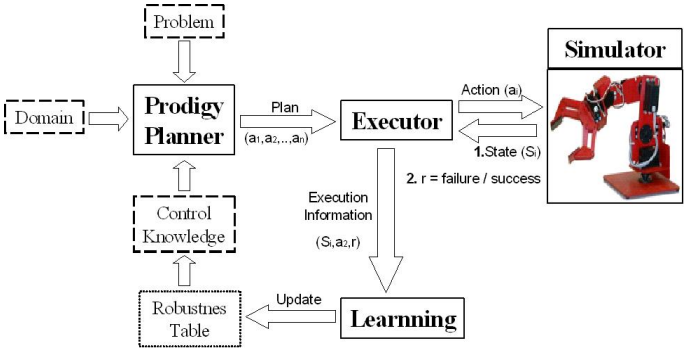


Figure 1: High level view of the planning-execution-learning architecture.

Planning

For the planning module, I have used the nonlinear backward chaining planner QPRODIGY (Borrajo, Vegas, & Veloso 2001). The inputs to the planner are the usual ones (domain theory and problem definition), plus declarative control knowledge, CK, described as a set of control rules. These control rules act as domain dependent heuristics, and they are the main reason I have used this planner, given that they provide an easy method for declarative representation of automatically acquired knowledge. PRODIGY4.0 planning-reasoning cycle, involves as decision points: select a goal from the set of pending goals and subgoals; choose an operator to achieve a particular goal; choose the bindings to instantiate the chosen operator and apply an instantiated operator whose preconditions are satisfied or continue subgoaling on another unsolved goal.

I use control rules for guiding the planner towards good solutions according to the acquired knowledge on robustness. The output of the planner is the total-ordered *most robust* plan.

Execution

The system executes step by step the plan proposed by the planner to solve each problem from the training set. The algorithm is shown in Figure 2.

Function Execution (Plan, Rob-Table):Rob-Table

Plan: list of actions (a1, a2 , ..., an)
Rob-Table: Table with the robustness of the actions

```
For all the actions ai in Plan do
        r = simulator(ai)
        Rob-Table = Learning(ai,r,Rob-Table)
        if r=failure Then break;
Return Rob-Table;
```

Figure 2: Algorithm that executes a plan and updates the robustness of the actions of the plan.

In order to ease the test of the architecture, I developed a simulator of the execution of an instantiated action of the plan in the real world. This simulator is very simple for the time being. It doesn't take care of the current state of the world (which is not quite real). It only takes into account the instantiated action, and returns whether its execution succeeded or failed.

The simulator keeps a model of execution for all the possible actions. This model lies in a probability distribution function. So every time the execution of an action has to be simulated, the simulator generates a random value following its corresponding distribution probability. If the generated random value satisfies the model, the action is considered successfully executed. For example, suppose the simulation of the action `put-down (R2, A)` in the blocksworld domain. At the simulator, it is defined that robot `R2` will succeed putting-down blocks 90% of the times. The simulator will generate a random value following a uniform distribution, and in case the generated random value is under 0.9 the action will be considered successfully executed.

Learning

The system analyzes the results of the executions of the plan actions, and generates a robustness table from the observation of these executions. The table is composed of tuples of the form `(op-name, op-params, r-value)` for every possible action. `op-name` is the action name, `op-params` is the list of the instantiated parameters, and `r-value`, is the robustness value. This value, represents an estimation of success of the execution of the instantiated action in the real world. If the execution of the action is successfully, this value is incremented, otherwise this value is reduced applying the square root function. The learning algorithm is shown in Figure 3.

The learning process consists on the update of the robustness table. The system updates this table after every execution of an action with the information obtained from the simulator. As an example, suppose the blocksworld domain with two gripper robots (`R1` and `R2`) and two blocks (`A` and `B`), Figure 4, and that the systems tries to learn from

Function Learning (ai, r,Rob-Table):Rob-Table

ai: executed action
r: execution outcome (failure or success)
Rob-Table: Table with the robustness of the actions

```
if r=success
  Then
   robustness(ai,Rob-Table) = robustness(ai,Rob-Table) + 1
  Else
   robustness(ai,Rob-Table) = sqrt(robustness(ai,Rob-Table))
Return Rob-Table;
```

Figure 3: Algorithm that updates the robustness of one action.

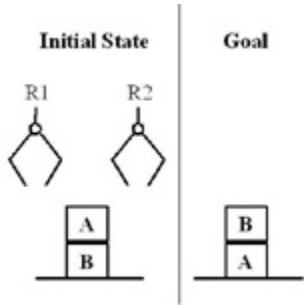


Figure 4: Non-deterministic multi-robot blocks world problem.

the following set of plans:

Plan 1: (Unstack R1 A B) (Put-down R1 A) (Pick-up R1 B) (Stack R1 B A)
Plan 2: (Unstack R2 A B) (Put-down R2 A) (Pick-up R2 B) (Stack R2 B A)
Plan 3: (Unstack R1 A B) (Put-down R1 A) (Pick-up R2 B) (Stack R2 B A)
Plan 4: (Unstack R2 A B) (Put-down R2 A) (Pick-up R1 B) (Stack R1 B A)

The system executes all these plans. Suppose plans 1 and 2 are executed successfully, but plans 3 and 4 fail at the second and fourth actions respectively. Table 1 would be generated by the learning scheme.

Table 1: Robustness table generated from the execution of all plans for problem in Figure 4.

Action	Parameters	Robustness
Unstack	R1 A B	2.0
Put-down	R1 A	1.0
Pick-up	R1 B	2.0
Stack	R1 B A	1.0
Unstack	R2 A B	2.0
Put-down	R2 A	2.0
Pick-up	R2 B	1.0
Stack	R2 B A	1.0

Exploitation of acquired knowledge

The planner uses the robustness table by means of Control Knowledge. Control rules guide the planner among all the possible actions, choosing the action bindings that have the greatest robustness values at the Robustness Table (Table 1). An example of these control rules is shown in Figure 5.

```
(control-rule prefer-bindings-put-down
  (IF
   (and (current-goal (on-table
<block1>))
        (current-operator PUT-DOWN)
        (candidate-bindings (<robot>
<robot1>))
        (candidate-bindings (<robot>
<robot2>))
        (robustness-more-than
         (PUTDOWN <robot1> <block1>)
         (PUTDOWN <robot2> <block2>))))
  (THEN prefer-bindings
         ((robot robot1))))
```

Figure 5: Control rule for prefering the *most robust* bindings for the operator put-down.

In case the planner can choose among different bindings, these control rules tell the planner which ones to prefer. In the problem of Figure 4, as the Figure 6 shows, the planner can choose between two actions (`(Put-down R1 A)` and `(Put-down R2 A)`) to put down the block `A` on the table. The control rule `prefer-binding-put-down` will make the planner prefer the *most robust* one. In this case, if we look at the robustness table (Table 1), `(Put-down R2 A)` is more robust than `(Put-down R1 A)`.

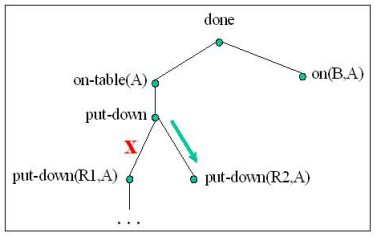


Figure 6: Example of search guided by control rules.

Experiments and Results

This section describes the experiments carried out to evaluate the proposed architecture. I have performed the experiments in a simple non-deterministic multi-robot blocksworld domain given that this is preliminary work. I will shortly apply to more realistic domains. The objects in this world include 9 blocks (`A` to `I`), a table and two gripper robots `R1` and `R2`. In order to test the system, I created a world model in which robot `R1` completes actions

`put-down` and `stack` successfully with probability 0.1 and the rest of the actions with probability 0.9, whereas `R2` completes all actions with probability 0.9.

I took one hundred problems and I generated a plan for each one of the one hundred problems, with and without the use of the control rules (with or without the learned behavior). Each plan was executed 1000 times, and I registered the average number of plan actions succesfully executed. As shown in Figure 7, for almost every problem, the number of plan actions executed successfully is greater with the use of the control rules. That is because control rules make the planner prefer the most robust actions.

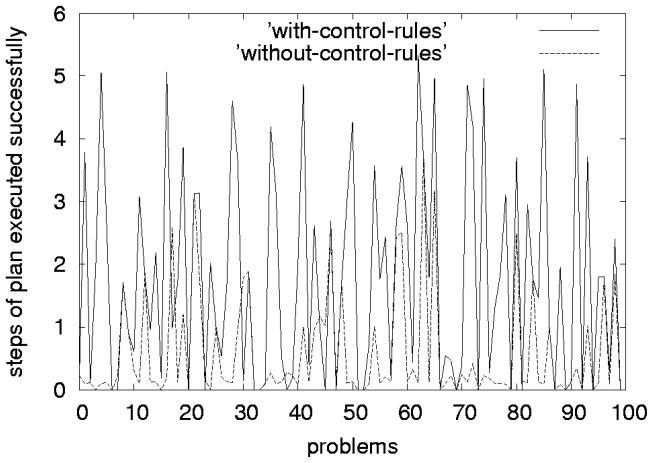


Figure 7: Percentage of plan succesfully executed for the set of eleven problems.

Conclusions and future work

I present an architecture that integrates a deliberative planner with some uncertainty reasoning capabilities, an execution module that connects with a simulator, and a learning mechanism. This integration results in a system that is able to provide more robust plans as it acquires knowledge from previous plans executions.

Experiments show that if the system has learned the robustness of the instantiated actions in the world, planning using this robustness information achieves plans whose execution is more robust.

In the future, I plan to remove, when possible, the initial assumptions. First, I will use a more complex simulator that considers not only the instantiated action, but also the state before applying each action. I plan to use, for instance, the rigid body dynamics simulator.[1] Then, during learning, the reinforcement formula should also consider the state where it was executed. One could use standard reinforcement learning techniques (Watkins & Dayan 1992) for that purpose, but states in deliberative planning are represented as predicate logic formulae. One solution would consist on using relational reinforcement learning techniques (Dzeroski, Raedt, & Driessens 2001).

Relaxing the second assumption requires generating robustness knowledge with generalized instances and then mapping new problems instances to those used int the acquired knowledge. As I described in the Introduction, I believe this is not really needed in many domains, since one always has the same instances in all problems of the same domain. In that case, I have to assure that there is a unique mapping between real world instances and instance names in all problems. In case new instances appear, their robustness values can be initialized to zero, and then gradually updated with the proposed learning mechanism.

[1] http://ode.org/

References

Borrajo, D.; Vegas, S.; and Veloso, M. 2001. Quality-based learning for planning. *In Working notes of the IJCAI'01 Workshop on Planning with Resources. IJCAI Press. Seattle, WA(USA)*.

Dzeroski, S.; Raedt, L. D.; and Driessens, K. 2001. Relational reinforcement learning. *Machine Learning* 43:7–52.

Fernández, S.; Aler, R.; and Borrajo, D. 2005. Machine learning in hybrid hierarchical and partial-order planners for manufacturing domains. *Applied Artificial Intelligence* 19(10).

Watkins, C. J. C. H., and Dayan, P. 1992. Technical note: Q-learning. *Machine Learning* 8(3/4):279–292.

Towards High-performance Robot Plans with Grounded Action Models: Integrating Learning Mechanisms into Robot Control Languages

Alexandra Kirsch

Abstract

For planning in the domain of autonomous robots, abstraction of state and actions is indispensable. This abstraction however comes at the cost of suboptimal execution, as relevant information is ignored. A solution is to maintain abstractions for planning, but to fill in precise information on the level of execution. To do so, the control program needs models of its own behavior, which could be learned by the robot automatically. In my dissertation I develop a robot control and plan language, which provides mechanisms for the representation of state variables, goals and actions, and integrates learning into the language.

Motivation

A key challenge for the next generation of autonomous robots is the reliable and efficient accomplishment of prolonged, complex, and dynamically changing tasks in the real world.

One of the most promising approaches to realizing these capabilities is the plan-based approach to robot control. In the plan-based approach, robots produce control actions by generating, maintaining, and executing plans that are tailored for the robots' respective tasks. Plans are robot control programs that a robot can not only execute but also reason about and manipulate. Thus a plan-based controller is able to manage and adapt the robot's intended course of action — the plan — while executing it and can thereby better achieve complex and changing goals. The use of plans enables these robots to flexibly interleave complex and interacting tasks, exploit opportunities, quickly plan their courses of action, and, if necessary, revise their intended activities.

Making plan-based controllers effective requires programmers to abstract away from details of the physical world. In order to reduce the size of the state space, the robot's belief state is described in terms of abstract situations and events. Similarly actions are described as discrete, instantanious events. As an example let us consider an autonomous household robot. A situation can be described as the robot being "near the door". When someone wants to enter the room the robot should perform the action "clear the door". A description like this totally disregards the actual position of the robot, the actual distance to the door and the precise position to where the robot should go in order to clear the door. These are the kinds of abstraction that make automatic planning feasible.

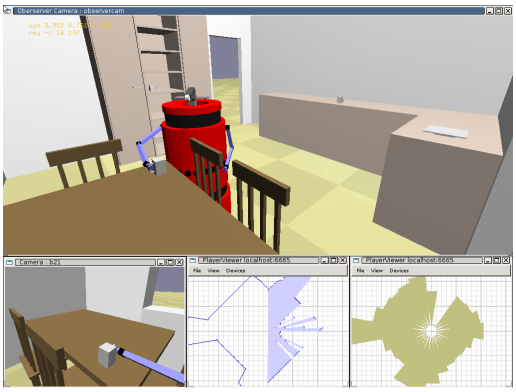


Figure 1: Realistic simulation for a household robot.

However, abstracting away from the low-level state description often yields suboptimal behavior. In our example where the robot is blocking the door, it might move away from the door so that it can be opened, but it might still be in the way of the person entering the room. Or the robot might have been standing near the door, because it was stirring the soup on the hearth. If it moves away to make room for someone to enter, the soup might be burning. In this situation, the robot should have looked for an alternative position that still allowed it to reach the hearth. The problem here is that the action "clear door" deliberately ignores the robot's precise start and goal positions. For the planner, this is fine, since the actions should be kept simple. But when it comes to executing the plan, the robot should consider its current situation, goals and possible outcomes of its action.

In my research I develop mechanisms that allow the programmer to keep a high degree of abstraction for planning. Only during the execution of the abstract plans, the low-level details are taken into account and the plan steps are optimized with respect to the current context. To perform an action in a certain situation the control program needs information about (1) why the action is to be performed, (2) other goals that might interfere, and (3) the behavior of the procedures used for achieving the action.

The information about the current program state can be

provided by a declarative structure of the control program making concepts such as goals, procedures and belief state explicit. Knowledge about the interference of goals or the behaviour of routines is provided by models. In the case of interfering goals the models might be provided by the programmer. But it would be a hard job to predict the behavior of every available control routine in every conceivable situation. Here automatic learning techniques are indispensable.

In this framework an action in a plan doesn't necessarily correspond to a certain control routine. In many cases, there are several routines, the performance of each varying in different contexts. The choice of which routine to call in the given situation is based on models. Although it is possible to program all these routines by hand, program development could be advanced by learning routines automatically.

Unfortunately, the performance of learned routines often drags substantially behind those of programmed ones, at least if the control tasks are complex, interact, and are dynamically changing. In our opinion this is not due to a poor performance of learning algorithms, but to the insufficient integration of learning into control languages. With a synthesis of learning and programming, parts adequate for learning need not be programmed explicitly, while other parts can be implemented manually. In order to get a smooth transition between the two worlds, the learning process must take place inside the controller. This means that the robot has to acquire training experiences, compute a suitable feature language representation, execute the learning process and integrate the result into the control program.

Contributions

The aim of my dissertation is the development of an extension of a robot control and plan language, which provides mechanisms for

- modelling interaction with the physical environment;
- the representation, inference and execution of abstract modalities like state variables, goals and actions;
- the smooth interaction of programming and learning.

For the first point I develop representations that provide the program with information about the physical meaning of state variables. Inference mechanisms can use this information for example to generate abstract feature languages that are needed for automatic learning.

An explicit representation of state variables, goals and actions provides knowledge about the execution state of the program. Thus when executing an action, the program can find out why this action is to be performed and use this information in choosing appropriate parameterizations.

In order to integrate learning into a programming language, we need an explicit and declarative representation of learning problems, as well as mechanisms for executing the learning steps and embedding the resulting procedure seamlessly into the code.

For the empirical evaluation of the language I develop, we have two testbeds. One is a simulated household robot that has to perform sophisticated tasks in a kitchen (figure 1). The other one is our autonomous robot soccer team, where real robots have to be controlled in highly dynamic situations.

Realization

The concepts for declaratively describing the physical world, for representing beliefs, goals and procedures, and the learning mechanisms are implemented as an extension to RPL, which is a plan language implemented as LISP macros.

Interaction with the physical world

The representation of the robot's belief is implemented by *state variables*, which don't only contain the robot's current belief about the world, but includes models about the physical meaning of each state variable. So, when specifying a state variable, we give the unit of measurement and the physical dimension of each value.

Changes in the values of physical quantities are propagated by fluents, variables that vary over time. Using fluents, the robot can wait for events and react accordingly.

For a more high-level description of the robot state we have the concept of *derived state variables*, which are a composition of other state variables. In the robot soccer environment, such a derived state variable could be the robot's current distance to the ball.

Goals, Actions and State

The robot control program uses explicit representations of the robot's state, its goals and the procedures for fulfilling the goals (figure 2).

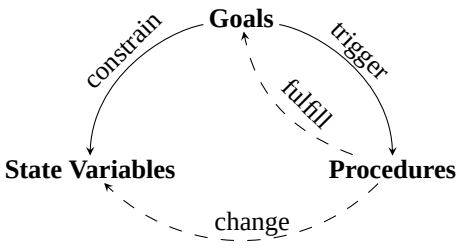


Figure 2: Interconnections between state variables, goals and procedures. The drawn through arrows denote mechanisms that are explicitly represented in the language. The dashed lines show interactions that are not mirrored in language constructs.

A goal class is defined as a restriction over a state variable. Such a state variable is called *controllable*. For a goal class the programmer must also specify a procedure to fulfill the goal. A goal can be *achieved*, where the adequate control procedure is invoked in order to fulfill some success criterion, or *maintained*, where the success criterion is constantly checked and if required, restored.

For a goal class the programmer has to state which procedure is to reach the goal. In many cases there are several possible routines with different characteristics in different situations. If we know models of these routines, we can choose the best according to the circumstances. For this we introduce the concept *control task*. The control task can choose between different *control routines*, given the current belief and models of the control routines. A control routine can be

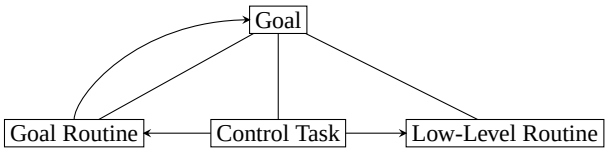


Figure 3: Calling hierarchy of goals and procedures.

either a *goal routine*, which is the only kind of procedure that can set new goals, or a *low-level-routine*, which controls the robot directly by giving commands to the robot architecture (figure 3).

Integrating Learning into Robot Control

Considering the current state-of-the-art, developing robotic agents that learn autonomously is rather an opaque art than an engineering exercise. One of the main reasons is that modern robot control languages do neither enforce nor strongly support the rigorous design of learning mechanisms. With the language ROLL (formerly called RPL$_{LEARN}$) (Beetz, Kirsch, & Müller 2004), we attempt to improve this situation by extending a robot control language with constructs for specifying experiences, learning problems, exploration strategies, etc. Using these constructs, learning problems can be represented explicitly and transparently and become executable.

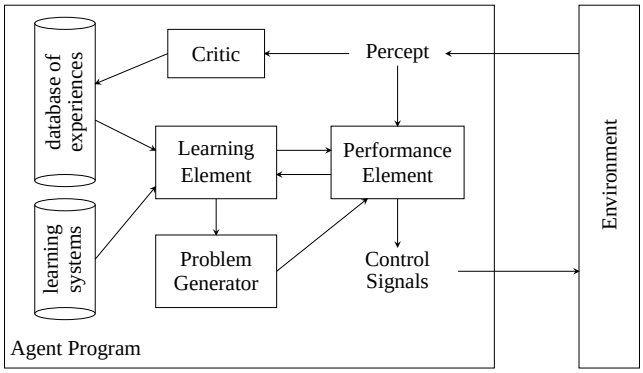


Figure 4: Learning agent after (Russell & Norvig 1995).

Figure 4 shows the parts of a learning agent. Every aspect of a learning problem is represented explicitly within ROLL.

The *performance element* realizes the mapping from percepts into the actions that should be performed next. The control procedures therein might not yet be executable or optimized. These are the procedures we want to learn.

The *critic* is best thought of as a learning task specific abstract sensor that transforms raw sensor data into information relevant for the learning element. To do so the critic monitors the collection of experiences and abstracts them into a feature representation that facilitates learning.

The *learning element* uses experiences made by the robot in order to learn the routine for the given control task.

The *problem generator* generates a control routine that is executed by the performance element in order to gather useful experiences for a given learning problem. The problems are generated according to a probability distribution as given in the learning problem specification.

is called with an experience class and returns a control routine that, when executed, will generate an experience of the respective class. The new parameterizations are generated as specified in the distribution of parameterizations of the experience class.

The language constructs for learning described here have been applied to reconstruct large parts of the control program of our soccer robots (Beetz *et al.* 2003).

Progress

In the current state, mechanisms for the representation of physical knowledge inside the state variables are implemented. Also goals, procedures and state variables are represented explicitly. The mechanisms shown in figure 2 are now reflected in programming constructs.

A first version of the language ROLL has been implemented independently of the other model-based concepts. We applied this language to several learning problems in the context of robot soccer. The constructs were also used by students in a practical course. We are integrating a revised version of the learning constructs into the model-based language context. For this purpose we store experiences in a database, so that data mining techniques for data cleaning can be applied easily to the training examples.

We used an earlier implementation (Müller, Kirsch, & Beetz 2004) of the language for implementing the control program of our soccer robots for the 2004 world championship in Lisbon. Also, learned routines for navigations tasks were included, whose performance reached the level of programmed ones (Kirsch, Schweitzer, & Beetz 2005).

A more recent version was employed for controlling the simulated kitchen robot. In this context we haven't performed any learning yet.

References

Beetz, M.; Stulp, F.; Kirsch, A.; Müller, A.; and Buck, S. 2003. Autonomous robot controllers capable of acquiring repertoires of complex skills. In *RoboCup International Symposium 2003*, Padova.

Beetz, M.; Kirsch, A.; and Müller, A. 2004. RPL-LEARN: Extending an autonomous robot control language to perform experience-based learning. In *3rd International Joint Conference on Autonomous Agents & Multi Agent Systems (AAMAS)*.

Kirsch, A.; Schweitzer, M.; and Beetz, M. 2005. Making robot learning controllable: A case study in robot navigation. submitted to Nineteenth International Joint Conference on Artificial Intelligence.

Müller, A.; Kirsch, A.; and Beetz, M. 2004. Object-oriented model-based extensions of robot control languages. In *27th German Conference on Artificial Intelligence*.

Russell, S., and Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall.

Going where Hierarchical Task-Network (HTN) Planning Meets with Symbolic Model Checking*

Ugur Kuter
University of Maryland,
Department of Computer Science,
College Park, Maryland 20742, USA

Introduction

More and more research is addressing the problem of planning in nondeterministic domains. In spite of the recent promising results, the problem is still very hard to solve in practice, even under the simplifying assumption of full observability, i.e., the hypothesis that the state of the world can be completely observed at runtime. Indeed, in the case of nondeterministic domains, the planning algorithm must reason about all possible different execution paths to find a plan that works despite the nondeterminism, and the dimension of the generated conditional plan may grow exponentially.

Among others, planning based on *Symbolic Model Checking* (Cimatti *et al.* 2003; Rintanen 2002; Jensen & Veloso 2000) is one of the most promising approaches for planning under conditions of nondeterminism. This technique relies on the usage of propositional formulas for a compact representation of sets of states, and of transformations over such formulas for efficient exploration in the search space. The most common implementations of this technique have been realized with *Binary Decision Diagrams (BDDs)* (Bryant 1992). In different experimental settings, planning algorithms based on BDDs, e.g., those implemented in MBP (Bertoli *et al.* 2001), have been shown to scale up to rather large-sized problems (Cimatti *et al.* 2003).

Another promising approach to planning with nondeterminism is forward planning with *Hierarchical Task Networks (HTNs)*, which was originally developed to provide efficient search-control heuristics for classical domains (Nau *et al.* 2003). (Kuter & Nau 2004) described a way to generalize this approach to work in the nondeterministic case, along with a class of other forward-planning techniques. The ND-SHOP2 planner, a nondeterminization of SHOP2 (Nau *et al.* 2003), is a forward HTN planner developed using this technique. It has been demonstrated in (Kuter & Nau 2004) that ND-SHOP2 can be very effective in pruning the search space, and in some cases ND-SHOP2 outperforms MBP.

In this paper, we first describe a framework and a novel algorithm, called YoYo, that enables us to combine the power of the HTN-based search-control strategies with BDD-based symbolic model checking techniques. Next, we discuss the results of our experimental evaluation of YoYo. Finally, we conclude with our ongoing and future research directions.

Background

We use the usual definitions for states, nondeterministic planning domains, policies, planning problems and their solutions in such domains, as in (Cimatti *et al.* 2003).

A *nondeterministic planning domain* is a tuple of the form $(\mathcal{P}, \mathcal{S}, \mathcal{A}, \mathcal{R})$, where $\mathcal{P}$ is a finite set of propositions, $\mathcal{S} \subseteq 2^{\mathcal{P}}$ is the set of all possible states, $\mathcal{A}$ is the finite set of all possible actions, and $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the state-transition relation. The set of states in which an action a is applicable is $S_a \subseteq \mathcal{S}$.

A *policy* to be a set $\pi = \{(s, a) \mid s \in \mathcal{S} \text{ and } a \in A(s)\}$, where $A(s) \subseteq \mathcal{A}$ is the set of actions that are applicable in s. A *planning problem* in a domain D is a tuple of the form $P = (D, I, G)$, where $I \subseteq \mathcal{S}$ is a set of initial states, and $G \subseteq \mathcal{S}$ is a set of goal states. In this work, we focused only on strong and strong-cyclic solutions for planning problems, as in (Cimatti *et al.* 2003).

We use the definitions for primitive tasks, nonprimitive tasks, task networks, and methods as in (Nau *et al.* 2003), except that we restrict ourselves to ground instances of these constructs in this paper. We assume the existence of a finite set of symbols that denote the *tasks* to be performed in a planning domain D. Every action a in D is a primitive task symbol, and there are some additional task symbols called *nonprimitive tasks*. A *task network* is a partially-ordered set of tasks.

We adopt the notion of *ordered task decomposition* (Nau *et al.* 2003) as follows: the tasks in a task network are decomposed into subtasks in the order they are supposed to be performed in the world. A *method* describes a possible way of decomposing the tasks in a task network into smaller and smaller tasks. The set of states in which a method m can be applied is S_m. The

*This extended abstract is produced from the paper titled "A Hierarchical Planner based on Symbolic Model Checking," which is accepted for presentation and publication in ICAPS-2005 Technical Program. I would like to thank Drs. Dana Nau, Marco Pistore, and Paolo Traverso for their supervision in doing this work.

```
Procedure YoYo(D, I, G, w, M)
  return YoyoAux(D, {(I, w)}, G, M, ∅)

Procedure YoyoAux(D, X, G, M, π)
  X ← PruneSituations(X, G, π)
  if there is a situation (S, w = nil) ∈ X such that S ⊈ G
    then return(failure)
  if NoGoodPolicy(π, X, G) then return(failure)
  if X = ∅ then return(π)
  select a situation (S, w) from X and remove it
  F ← HTNDecomposition(S, w, D, M)
  if F = ∅ then return(failure)
  X' ← ComputeSuccessors(F, X)
  π' ← π ∪ {(s, a) | (S', a, w') ∈ F and s ∈ S'}
  π ← YoyoAux(D, X', G, M, π', X_0)
  if π = failure then return(failure)
  return(π)
```

Figure 1: YoYo, an HTN planning algorithm for generating solutions in nondeterministic domains. In the YoyoAux procedure, X is the current set of situations.

```
Procedure HTNDecomposition(S, w, D, M)
  F ← ∅; X ← {(S, w)}
  loop
    if X = ∅ then return(F)
    select a tuple (S, w) ∈ X and remove it
    select a task t that has no predecessors in w
    if t is a primitive task then
       A ← {a | a is an action for t, and S ⊆ S_a}
       if A = ∅ then return ∅
       select an action a from A
       F ← F ∪ {(S, a, w \ {t})}
    else
       A ← {m | m is a method in M for t and S ∩ S_m ≠ ∅}
       if A = ∅ then return ∅
       select a method instance m from A
       X ← X ∪ {(S ∩ S_m, (w \ {t}) ∪ w'}
       if S \ S_m ≠ ∅ then X ← X ∪ {(S \ S_m, w)}
```

Figure 2: The HTNDecomposition procedure. In the above, S_m and S_a respectively denote the set of states in which a method m and an action a are applicable.

result of applying a method m to a nonprimitive task t of w is the task network w' generated by replacing t in w with the subtasks specified in m, while preserving the ordering constraints in w and the subtasks of m.

The YoYo Planning Algorithm

In this section, we describe YoYo, a forward-chaining HTN planning procedure shown in Figure 1, which is designed to combine the ability of exploiting search-control heuristics as in HTN planning with symbolic model-checking techniques in a single framework.

The input for YoYo consists of a planning problem (D, I, G), an initial task network w, and a set of HTN methods M for the domain D. The algorithm exploits tuples of the form (S, w), called *situations*, which are resolved by accomplishing the task network w in the states of S. Starting with the initial situation (I, w), YoYo recursively generates successive sets of situations until a solution is generated; otherwise it returns failure.

At each iteration, YoYo first performs a series of tests on the set X of current situations. The first test is to check X for cycles and goal states: for every situation $(S, w) \in X$, YoYo removes any state in S that either appears already in the current partial policy π (since an action has already been planned for it), or appears in the set of goal states G (since no action should be planned for it). The PruneSituations subroutine in Figure 1 is responsible for this operation (please see (Kuter *et al.* 2005) for the formal definition of this procedure). Then, YoYo checks if there is a situation $(S, w) \in X$ such that there are no more tasks to be performed in w, but the goal has not been reached yet (i.e., $S \not\subseteq G$). If so, we have a failure in the search.

If the current set X of situations does not induce a failure in the search process, YoYo first checks if π conforms to the requirements of the kinds of solutions it is looking for. The NoGoodPolicy subroutine is responsible for this task (please see (Kuter *et al.* 2005) for the formal definition of this procedure). If the current partial policy π violates the requirements for a policy being a solution, then YoYo returns a failure.

Otherwise, we have two cases. First, if there are no situations to be explored further —i.e., $X = \emptyset$—, then π is a solution to the input planning problem, so YoYo returns it. Otherwise, YoYo selects a situation from X, say (S, w), and attempts to generate an action for every state in S. To achieve this objective, it uses a forward-chaining HTN-planning engine, called HTNDecomposition shown in Figure 2. The HTNDecomposition procedure is the hearth of YoYo. In a situation (S, w), it successively decomposes the tasks in w into smaller tasks by using the available HTN methods, until for every state in S, a primitive task (i.e, an action) is generated. If there exists a state in S such that no action can be generated using the task network w and the available HTN methods, then HTNDecomposition returns the empty set, forcing YoYo to report failure. Otherwise, it returns a set F of the form $\{(S_i, a_i, w_i)\}_{i=0}^{k}$, where $S_i \subseteq S$ such that we have $S = \bigcup_i S_i$, a_i is an action applicable to all states in S_i, and w_i is the task network to be accomplished after applying that action.

If HTNDecomposition generates a non-empty set F of tuples of the form (S, a, w), then YoYo proceeds with computing the successor situations to be explored using the ComputeSuccessors routine as follows: for each tuple $(S, a, w) \in F$, it first generates the set of states that arises from applying a in S by using the function $succ(S, a) = \{s' \mid s \in S$ and s' is a state that arises from applying a in $s\}$. Then, the next situation corresponding this action application is $(succ(S, a), w)$.

Experimental Evaluation

We have implemented a prototype of the YoYo planning algorithm, described in the previous section. Our cur-

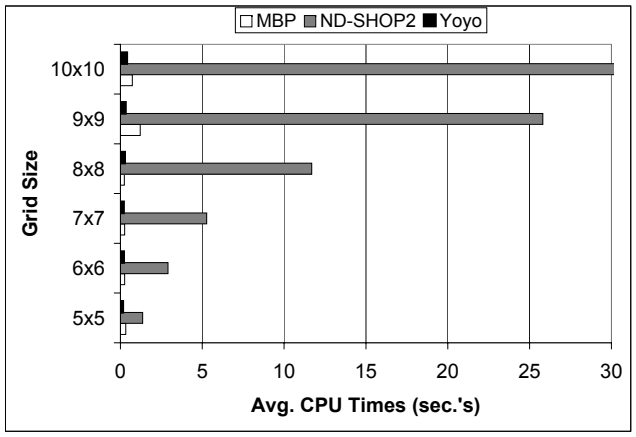


Figure 3: Average running times of YoYo, ND-SHOP2, and MBP in the hunter-prey domain as a function of the grid size, with one prey.

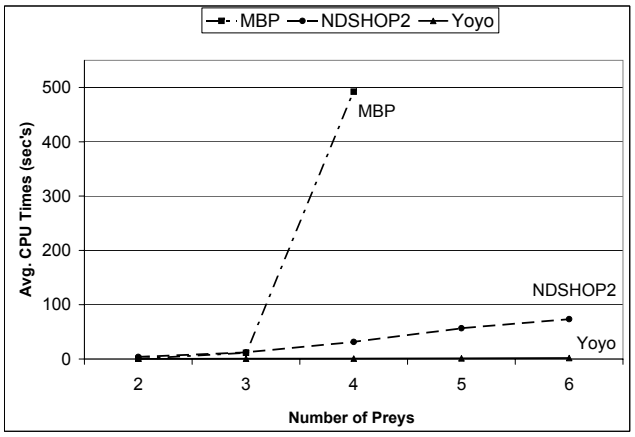


Figure 4: Average running times of ND-SHOP2, YoYo and MBP on problems in the Hunter-Prey domain as a function of the number of preys, with a 4×4 grid. MBP was not able to solve planning problems with 5 and 6 preys within 40 minutes.

rent implementation is built on both the ND-SHOP2 and the MBP planning systems. It extends the ND-SHOP2 planning system for planning over sets of states rather than a single state. We also developed a framework that enables us to implement the YoYo algorithm and its data structures using BDD-based symbolic model-checking primitives as in (Cimatti *et al.* 2003). Implementing YoYo using this machinery enabled us to develop an interface to MBP for exploiting the BDD-based primitives already implemented in it.

In our experiments, we used the Hunter-Prey domain (Koenig & Simmons 1995) and several variations of it. We performed several sets of experiments with YoYo, comparing its performance against MBP and ND-SHOP2 under different conditions in this domain and its variations. We summarize our results below; for a detailed discussion, please see (Kuter *et al.* 2005).

In the Hunter-Prey domain, there is a hunter and a prey in an $n \times n$ grid world. The task of the hunter is to catch the prey in the world. The hunter has five possible actions; namely, north, south, east, west, and catch. The prey has also five actions: it has the same four moves as the hunter, and an action to stay still in the world. The hunter can catch the prey only when the hunter and the prey are at the same location at the same time in the world. The nondeterminism for the hunter is introduced through the actions of the prey: at any time, it can take any of its actions, independent from the hunter's move.

First, we investigated how well YoYo is able to cope with large-sized problems compared to ND-SHOP2 and MBP. Figure 3 shows the results of the experiments for grid sizes $n = 5, 6, \ldots, 10$. For each value for n, we have randomly generated 20 problems and run MBP, ND-SHOP2, and YoYo on those problems. This figure reports the average running times required by the planners on those problems.

For grids larger than $n = 10$, ND-SHOP2 was not able to solve the planning problems due to memory overflows. This is because the sizes of the solutions in this domain are very large, and therefore, ND-SHOP2 runs out of memory as it tries to store them explicitly. Note that this domain admits only high-level search strategies such as "look at the prey and move towards it." Although this strategy helps the planner prune a portion of the search space, such pruning alone does not compansate for the explosion in the size of the explicit representations of the solutions for the problems.

On the other hand, both YoYo and MBP was able to solve all of the problems in these experiments. The difference between the performances of YoYo and ND-SHOP2 demonstrates the impact of the use of BDD-based representations: YoYo, using the same HTN-based heuristic as ND-SHOP2, was able to scale up as good as MBP since it is able to exploit BDD-based representations of the problems and their solutions.

In order to investigate the effect of combining search-control strategies and BDD-based representations in YoYo, we used the following variation of the Hunter-Prey domain. We assumed that we have more than one prey in the world, and the prey i cannot move to any location within the neighbourhood of prey $i + 1$ in the world. In such a setting, the amount of nondeterminism for the hunter after each of its move increases combinatorially with the number of preys in the domain. Furthermore, the BDD-based representations of the underlying planning domain explode in size under these assumptions, mainly because the movements of the preys are dependent to each other.

In this modified domain, we used a search-control strategy in ND-SHOP2 and YoYo that tells the planners to chase the first prey until it is caught, then the second prey, and so on, until all of the preys are caught. Note that this heuristic allows for abstracting away from the huge state space: when the hunter is chasing a prey, it does not need to know the locations of the other preys

in the world, and therefore, it does not need to reason and store information about those locations.

In the experiments, we varied the number of preys from $p = 2, \ldots, 6$ in a 4×4 grid world. We have randomly generated 20 problems for each experiment with different number of preys. Figure 4 shows the results of these experiments with MBP, ND-SHOP2, and YoYo. These results demonstrate the power of combining HTN-based search-control heuristics with BDD-based representations of states and solutions in our planning problems: YoYo was able to outperform both ND-SHOP2 and MBP. The running times required by MBP grow exponentially faster than those required by YoYo with the increasing size of the preys, since MBP cannot exploit HTN-based heuristics. Note that ND-SHOP2 performs much better than MBP in the presence of good search-control heuristics.

Work In Progress

Our experimental evaluation shows that the combination of HTN search-control heuristics and symbolic model-checking techniques is a potent one: it has large advantages in speed, memory usage, and scalability. Encouraged with these results, we are currently working on several directions regarding HTN planning and planning with Symbolic Model Checking.

In one of our research directions, we are aiming to develop techniques for generating solutions for planning problems in nondeterministic domains, for which there exists no good search-control knowledge, or compiling such knowledge is as hard as solving the problems themselves. To achieve this objective, we are currently working on the integration of MBP's planning algorithms with YoYo. Note that, as described previously, YoYo does not exploit MBP's backward-chaining planning algorithms; it uses the symbolic model-checking primitives implemented in the MBP system to implement its data structures and the operations over them. Note also that YoYo assumes that the HTN-based search-control knowledge is provided by domain experts; however, under conditions of uncertainty, this may not be a realistic assumption in many cases, or producing such knowledge may be equivalent to solving the target problem itself.

In our current work, we intend to use YoYo and its ability to exploit HTNs to decompose the planning problem into smaller problems in a systematic way, and solve those smaller problems using the HTNs in YoYo as much as possible. However, if YoYo generates a subproblem for which there exists no HTN methods available to the planner, it invokes MBP's algorithms for that subproblem and combines the solution returned by MBP with the rest of the plan being generated. This way we will be able to use our approach in a wider class of planning problems under conditions of uncertainty.

Note that the above framework treats YoYo and MBP as seperate planners that work at different levels on a planning problem. An alternative approach would be to exploit forward-chaining HTN heuristics directly inside MBP's backward algorithms. One way to incorporate such heuristics in MBP itself is to use an approach similar to GraphPlan: in this case, we preprocess a planning problem to generate a planning graph using YoYo, and then, use planning graph to guide MBP's backward search algorithms to generate solutions. One disadvantage of this approach is the following: generating a planning graph may be more work than necessary in some cases. We are currently working on this issue.

Secondly, we are currently investigating the relationships between HTNs and planning with temporally-extended goals as in (Dal Lago, Pistore, & Traverso 2002). In this work, our objective is to understand the basic properties of the two formalisms and their relationships, and develop algorithms that integrate HTNs with the symbolic model-checking algorithms already developed for temporally-extended goals. This work is in a preliminary stage, so it is premature to comment on the details until the work is farther along.

References

Bertoli, P.; Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2001. MBP: a model based planner. In *Proceeding of IJCAI-2001 Workshop on Planning under Uncertainty and Incomplete Information*, 93–97.

Bryant, R. E. 1992. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys* 24(3):293–318.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence* 147(1-2):35–84.

Dal Lago, U.; Pistore, M.; and Traverso, P. 2002. Planning with a language for extended goals. In *AAAI/IAAI Proceedings*, 447–454. Edmonton, Canada: AAAI Press/The MIT Press.

Jensen, R., and Veloso, M. M. 2000. OBDD-based universal planning for synchronized agents in non-deterministic domains. *Journal of Artificial Intelligence Research* 13:189–226.

Koenig, S., and Simmons, R. G. 1995. Real-time search in non-deterministic domains. In *IJCAI*, 1660–1669.

Kuter, U., and Nau, D. 2004. Forward-chaining planning in nondeterministic domains. In *AAAI Proceedings*, 513–518.

Kuter, U.; Nau, D.; Pistore, M.; and Traverso, P. 2005. A hierarchical task-network planner based on symbolic model checking. In *ICAPS Proceedings*. To appear.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Rintanen, J. 2002. Backward plan construction for planning as search in belief space. In *AIPS Proceedings*, 93–102.

Concurrent Probabilistic Temporal Planning

Mausam and **Daniel S. Weld**
Dept of Computer Science and Engineering
University of Washington
Seattle, WA-98195
{mausam,weld}@cs.washington.edu
An extended version to appear in ICAPS'05.

Abstract

Probabilistic planning problems are often modeled as Markov decision processes (MDPs), which assume that a single unit-length action is executed per decision epoch. However, in the real world it is common to execute several actions in parallel. This paper presents efficient methods for solving probabilistic planning problems with *concurrent, durative* actions. We extend *Concurrent MDPs*, MDPs which allow multiple instantaneous actions to be executed simultaneously, by adding explicit action durations. We present two novel admissible heuristics and one inadmissible heuristic to speed up the convergence. We also develop a novel notion of *hybridizing* an optimal and an approximate algorithm to yield a hybrid algorithm, which quickly generates high-quality policies. Experiments show that all our heuristics speedup the policy construction significantly. Furthermore, our approximate hybrid algorithm runs up to two orders of magnitude faster than other methods, while producing policies close to optimal.

1. Introduction

Recent progress has yielded new planning algorithms which relax, individually, many of the classical assumptions. However, in order to apply automated planning to many real-world domains we must eliminate larger groups of the assumptions in concert. For e.g., (Bresina *et al.* 2002) notes that control for a NASA Mars rover requires reasoning about uncertain, concurrent, durative actions. While today's planners can handle large problems with *deterministic* concurrent durative actions (JAIR Special Issue 2003), and semi-MDPs provide a clear framework for durative actions in the face of uncertainty, few researchers have considered concurrent, uncertain, durative actions — the focus of our work.

Consider a Mars rover with the goal of gathering data from different locations with various instruments. Concurrent actions are essential to effective execution, since instruments can be turned on, warmed up and calibrated, while the rover is moving. Similarly, uncertainty must be explicitly confronted as the rover's movement, arm control and other actions cannot be accurately predicted.

The framework of *Markov decision processes* (MDPs) is the dominant model for formulating probabilistic planning problems. In the traditional case, a single action is allowed per decision epoch. However, allowing multiple concurrent actions at a time point inflicts an exponential blowup on all MDP techniques. Our previous work on concurrent MDPs (Mausam & Weld 2004) introduced several methods to manage this blowup. However, in their current form concurrent MDPs (CoMDPs) do not handle explicit action durations. The actions are supposed to be instantaneous (or unit length). Moreover, the agent must wait for all the recently-started actions to finish before new action(s) can be started. This may lead to sub-optimal policies. For e.g., in order to save execution time, the Mars rover might wish to execute sequential set up actions (*e.g.*, turning on the camera, focusing, *etc.*.) *concurrent* with navigation to the next location.

In this paper, we define *concurrent probabilistic temporal planning* - in short, *CPTP*. This model extends our previous CoMDP framework by incorporating explicit action durations. Specifically, we extend the technique of *Sampled real-time dynamic programming* (Sampled RTDP) (Mausam & Weld 2004) to generate high-quality CPTP policies. We present effective heuristics to speed up the convergence to sampled RTDP. We also propose the novel idea of *hybridization*, *i.e.* of combining two policy creation algorithms to yield a single, fast, approximation algorithm, which has the best of both worlds. Our hybrid algorithm for CPTP combines partial CPTP and CoMDP policies to focus its optimization efforts on the most frequent branches.

2. Background

A *Markov decision process* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}r, \mathcal{C}, \mathcal{G}, s_0 \rangle$ in which $\mathcal{S}$ is a finite set of discrete states, $\mathcal{A}$ is a finite set of actions[1], $\mathcal{P}r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the state transition function, $\mathcal{C} : \mathcal{S} \times \mathcal{A} \rightarrow \Re^+$ is the cost model, $\mathcal{G} \subseteq \mathcal{S}$ is the set of absorbing goal states, and s_0 is the start state.

Assuming full observability, we seek to find an optimal, stationary policy, π: $\mathcal{S} \rightarrow \mathcal{A}$, which minimizes the expected cost (over an indefinite horizon) incurred to reach a goal state. Note that a *value function*, J: $\mathcal{S} \rightarrow \Re$, mapping states to the expected cost of reaching a goal state defines a policy. Thus we need to find the *optimal* value function, J^*.

Value Iteration is one method to obtain this. It is a dynamic programming approach in which the optimal value function is calculated as the limit of a series of approximations. *RTDP* (Barto, Bradtke, & Singh 1995), is a lazy ver-

[1] An applicability function, $Ap : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$, denotes the set of actions that can be applied in a given state.

sion of value iteration in which the states are updated in proportion to the frequency with which they are visited by the repeated execution of the greedy policy. *Labeled RTDP* improves the efficiency of RTDP by a clever labeling scheme that focuses attention on states where the value function has not yet converged (Bonet & Geffner 2003).

Concurrent Markov Decision Processes (CoMDP) In the previous work (Mausam & Weld 2004), we extended traditional MDPs to allow concurrent actions. Since some actions interfere with each other, we ban certain combinations adopting the classical planning notion of mutual exclusion and apply it to a *factored* action representation. Thus the effects of executing the sequence $a_1; a_2$ equals those of $a_2; a_1$.

An *action combination*, A, is a set of one or more non-mutex actions to be executed in parallel. The cost model for a CoMDP returns the cost of concurrently executing several actions in a state. For make-span minimization it is simply the maximum duration of the constituent actions. Thus the model assumes that a new set of actions may not be executed until all members of the previous set have terminated.

The applicability function for CoMDPs now has range $\mathcal{P}(\mathcal{P}(\mathcal{A}))$; it is defined in terms of the applicability function for MDPs as $\{ \mathrm{A} \subseteq \mathcal{A} | \forall a, a' \in A,\ a, a' \in Ap(s) \wedge \neg\mathrm{mutex(a, a')}\}$

With $A = \{a_1, a_2, \ldots, a_k\}$ for non-interacting actions, the transition function may be calculated as

$$\sum_{s_1, s_2, \ldots s_k \in \mathcal{S}} \ldots \sum \mathcal{P}r(s_1|s, a_1)\mathcal{P}r(s_2|s_1, a_2) \ldots \mathcal{P}r(s'|s_k, a_k)$$

Thus, a CoMDP is essentially a traditional MDP with a very large set of actions. To solve a CoMDP, we can simply adapt the traditional algorithms. However, since the number of action combinations is exponential in $|\mathcal{A}|$, efficiently solving a CoMDP requires specific techniques. Sampled RTDP is one of those techniques where instead of using all combinations of actions in the backup an intelligent sampling of a few is used (Mausam & Weld 2004).

3. Extending to Durative Actions

To incorporate action durations in concurrent probabilistic planning problems, we consider the input model similar to that of concurrent MDPs except that action costs ($\mathcal{C}(a)$) are replaced by their durations ($\Delta(a)$). The objective is to minimize the expected time (*make-span*) of reaching a goal. We make some assumptions for ease of modeling and planning. We require the actions to have deterministic integer-valued durations. Also for simplicity, we adopt the temporal action model of (Smith & Weld 1999), rather than the more complex PDDL2.1.

This restriction is consistent with our previous definition of concurrency. Specifically, the mutex definitions (of CoMDPs) hold and are required under this action model. Moreover, in this model, it is sufficient to consider a new decision epoch only at a time-point when one or more actions complete. Thus we infer that these decision epochs will be discrete (integer).

Aligned Epoch Search Space A simple way to model CPTP is as a standard CoMDP, in which action costs are set to their durations. This formulation is distinct from an actual CPTP in an important way: In the aligned-epoch policy execution, once a combination of actions is started at a state, the next decision can be taken only when the effects of all actions have been observed (hence the name *aligned-epochs*). In contrast, at a decision epoch in the optimal execution for a CPTP problem, many actions may be midway in their execution. We have to explicitly take into account these actions and their remaining execution times when making a subsequent decision. Thus, the state space of CPTP is substantially different from that of the simple aligned-epoch model.

Interwoven Epoch Search Space We adapt the search space representation of (Haslum & Geffner 2001). Our original state space $\mathcal{S}$ in section 2 is augmented by including the set of actions currently executing and the times remaining for each. Formally, let the new augmented state[2] $s \in \mathcal{S}_{\underline{\ }}$ be an ordered pair $\langle X, Y \rangle$ where $X \in \mathcal{S}$ and $Y = \{(a, \delta) | a \in \mathcal{A}, 0 < \delta \leq \Delta(a)\}$. Here X represents the values of the state variables (*i.e.* X is a state in the original state space) and Y denotes the set of ongoing actions "a" and their remaining times until completion "δ".

To allow the possibility of simply waiting for some action to complete execution, that is, not executing any action at some decision epochs, we augment the set $\mathcal{A}$ with a *no-op* action. We allow no-op to be applicable in all states $s = \langle X, Y \rangle$ where $Y \neq \emptyset$ (*i.e.* states in which some action is still being executed). The new applicablity set of s would include combinations that are originally applicable in X and non-mutex with any action in Y.

We define the probability transition function $\mathcal{P}r_{\underline{\ }}$ for the new state space such that we move forward in time to our next decision epoch - which is the smallest time after which any executing action completes. For formal definition, please refer to (Mausam & Weld 2005).

Thus we model a CPTP problem as a CoMDP in this new state space. The main bottleneck in inheriting our previous methods (*e.g.* Sampled RTDP) naively is the huge size of the new state space. In the worst case (when all actions can be executed concurrently) the size of the state space is $|\mathcal{S}| \times \left(\prod_{a \in \mathcal{A}} \Delta(a)\right)$. Thus we need to reduce, abstract or aggregate our state space to make the problem tractable. We now present several heuristics which can be used to speed the search.

4. Heuristics

We present three heuristics that can be used as the initial cost function for our Sampled RTDP algorithm.

Maximum Concurrency Heuristic We prove that the optimal expected cost in a traditional (serial) MDP divided by the *maximum concurrency* (maximum number of actions that can be executed in parallel) is a lower bound for the expected make-span of reaching a goal in a CPTP problem. This bound can be used as an admissible (MC) heuristic.

Let $J(X)$ denote the value of a state $X \in \mathcal{S}$ in a tradi-

[2] We use the subscript $_{\underline{\ }}$ to denote the *interwoven* state space ($\mathcal{S}_{\underline{\ }}$), value function ($J_{\underline{\ }}$), *etc.*.

tional MDP with costs of an action equal to its duration. Let *concurrency* of a state be the maximum number of actions that could be executed in the state concurrently. We define *maximum concurrency of a domain* (c) as the maximum concurrency of any state in the domain.

Theorem 1 *Let* $s = \langle X, Y \rangle$,

$$J^*_{=}(s) \geq \frac{J^*(X)}{c} \quad \textit{for } Y = \emptyset$$

$$J^*_{=}(s) \geq \frac{Q^*(X, A_s)}{c} \quad \textit{for } Y \neq \emptyset$$

Proof Sketch: Consider any trajectory of make-span L (from a state $s = \langle X, \emptyset \rangle$ to a goal state) in a CPTP problem using its optimal policy. We can make all concurrent actions sequential by executing them in the chronological order of being started. As all concurrent actions are non-interacting, the outcomes at each stage will have similar probabilities. The maximum make-span of this sequential trajectory will be cL (assuming c actions executing at all points in the semi-MDP trajectory). Hence $J(X)$ using this (possibly non-stationary) policy would be at most $cJ^*_{=}(s)$. Thus $J^*(X) \leq cJ^*_{=}(s)$. The second inequality can be proven in a similar way. ■

Following this theorem, the maximum concurrency (*MC*) heuristic for a state $s = \langle X, Y \rangle$ is defined as follows:

$$\text{if } Y = \emptyset \; H_{MC}(s) = \frac{J^*(X)}{c} \text{ else } H_{MC}(s) = \frac{Q^*(X, A_s)}{c}$$

The time for computing the heuristic is the time required for solving the underlying MDP that is comparitively fast.

Average Concurrency Heuristic Instead of using maximum concurrency c in the above heuristic we use the average concurrency in the domain to get the average concurrency (*AC*) heuristic. The *AC* heuristic is not admissible, but in our experiments it is typically a more informed heuristic.

Eager Effects Heuristic Given the CPTP problem, we can generate a relaxed CoMDP by making the effects of actions, which would otherwise be visible only in the future, be known right away — thus the name eager effects (*EE*). A state for this relaxed CoMDP is $\langle X, \delta \rangle$ where X is an MDP state and δ is an integer. Intuitively, $\langle X, \delta \rangle$ signifies that the agent will *reach* state X after time δ units. Thus, we have discarded the information about which actions are executing and when they will individually end; we only record that all of them will have ended after time δ units and that the agent will reach the state X (possibly with some probability). Solving such a CoMDP yields an admissible heuristic. For details, please refer to (Mausam & Weld 2005).

Theorem 2 *Neither of the two heuristics (eager effects or maximum concurrency) dominates the other.*

In practice *EE* is consistently more informative than *MC* on the domains we tried. The computation times of *MC* are quite small. Whereas, *EE* requires the computation of a problem which has a larger search space than even the underlying CoMDP. Thus it can take a long time, at times to the extent that the advantage of the more informative heuristic is lost in the complex heuristic computation.

```
Algorithm Hybrid(r, k, m) {
  ∀s ∈ S_= initialize J_=(s) with an admissible heuristic;
  Repeat {
    Perform m RTDP trials;
    Compute Hybrid policy (π) using interwoven-epoch policy
      for states visited more than k times
      and aligned-epoch policy otherwise;
    Clean π by removing all dead-ends and cycles;
    J^π_=⟨s0, ∅⟩ ← Evaluation of π from the start state;
  } Until ( (J^π_=(⟨s0,∅⟩) − J_=(⟨s0,∅⟩)) / J_=(⟨s0,∅⟩) < r )
  Return hybrid policy π;
}
```

Figure 1: Pseudo-code for the hybrid algorithm

5. Hybrid Algorithm

Our approximate algorithm exploits the intuition that it is best to focus computation on the most probable branches in the current policy's reachable space. The danger of this approach is that, during execution, the agent might end up in an unlikely branch, which has been poorly explored; indeed it might blunder into a dead-end in such a case. This is undesirable, as such an apparently attractive policy might have a true expected make-span of infinity. Since we wish to avoid this case, we define a desirable *completeness* property:

Completeness: A policy is *complete* at a state if it is guaranteed to lead, eventually, to the goal state (*i.e.*, it avoids all dead ends and cycles). A planning algorithm is *complete* if it always produces a complete policy for the initial state, when one exists.

Our algorithm generates a complete policy and is created by *hybridizing* two other policy creation algorithms. Indeed, our novel notion of hybridization is both general and powerful, applying to many MDP-like problems[3].

For the case of CPTP, our algorithm hybridizes the RTDP algorithms for interwoven-epoch and aligned-epoch models. With aligned-epochs, RTDP converges relatively quickly, but to a suboptimal policy. Whereas, RTDP for interwoven-epochs is optimal, but takes a long time. Our hybrid algorithm will run RTDP on the interwoven space long enough to generate a policy which is good on the common states, but stop well before it converges in every state. Then, to ensure that the rarely explored states have a complete policy, it will substitute the aligned policy, returning this *hybrid* policy. The high level algorithm is described in Figure 1.

Use of RTDP helps us in defining a very simple termination condition with a parameter that can be varied to achieve the desired *closeness* to optimality as well. The intuition is very simple. Optimal Labeled RTDP starts with an admissible heuristic and guarantees that the value of the start state, $J_=(\langle s_0, \emptyset \rangle)$, remains admissible (thus less than or equal to optimal). In contrast, the hybrid policy's make-span is always longer than or equal to optimal. Thus, whenever the two values are within an *optimality ratio (r)*, we know that the algorithm has found a solution, which is close to the op-

[3]For e.g., one could create a complete, anytime approximation algorithm for CoMDPs by hybridizing one of the RTDP algorithms of (Mausam & Weld 2004) with a traditional MDP algorithm.

timal. Finally, cycle detection and evaluation of the hybrid policy's make-span both are done using simulation.

6. Experiments

We briefly compare the computation time and solution quality of six methods: interwoven Sampled RTDP with no heuristic (0), with the maximum concurrency (*MC*), average concurrency (*AC*), and eager effects (*EE*) heuristics, the hybrid (*H*) algorithm and Sampled RTDP on the aligned-epoch (*AE*) model.

Experimental Setup We tested our algorithms on problems in three domains - Rovers, Machineshop and Artificial. The details of the domains are omitted due to lack of space.

Comparison of Running Times We find that *AE* solves the problems extremely quickly; this is natural since the aligned-epoch space is smaller. Use of both H_{MC} and H_{AC} always speeds search in the $\mathcal{S}$- model. Using H_{EE} speeds up the solutions for most problems, but sometimes the heuristic computation takes a huge amount of time and the overall running time is not competitive. Comparing the heuristics amongst themselves, we find that H_{AC} mostly performs faster than H_{MC} — presumably because H_{AC} is a more informed heuristic in practice, although at the cost of being inadmissible. We find a couple of cases in which H_{AC} doesn't perform better; this could be because it is focusing the search in the incorrect region, given its inadmissible nature. The hybrid algorithm performs fastest. In fact, the speedups are dramatic compared to other methods. Averaging over all domains (results not shown), the hybrid algorithm produces a 10x speedup and *AE* produces more than a 100x speedup.

Comparison of Solution Quality We measure solution quality by simulating the generated policy across multiple trials, and reporting the average time taken to reach the goal. We note that the aligned-epoch (*AE*) policies usually yield significantly longer makespans (e.g., 25% longer); thus one must make a quality sacrifice for their speedy policy construction. In contrast, the hybrid algorithm extorts only a small sacrifice in quality in exchange for its speed.

7. Conclusions and Future Work

This paper summarizes our techniques for incorporating concurrency with durative actions in MDPs. We formally define the concurrent probabilistic temporal planning problem, and develop two modified state spaces (aligned-epoch ($\mathcal{S}_{\|}$) and interwoven-epoch ($\mathcal{S}$-)), which allow us search for an optimal policy Our experiments show that, while we can search the aligned space faster, the interwoven model usually yields a much better policy, one which generates a much lower make-span. We develop three new heuristics to speed up the convergence in the interwoven model.

We also develop the general technique of hybridizing two MDP algorithms. Hybridizing interwoven-epoch and aligned-epoch policy creation yields a much more efficient algorithm, one which is still complete (*i.e.*, its policies guarantee that the goal will be reached whenever possible). Also, our hybrid algorithm has a parameter, which can be varied to trade-off speed against optimality. In our experiments, the hybrid algorithm quickly produces near-optimal solutions. For larger problems, the speedups over other algorithms are quite significant. The hybrid algorithm can also be used in an anytime fashion thus producing good quality *complete* policies within a desired time. Thus, we expect that the algorithm would be very effective in solving large problems.

Future Work Scaling to significantly larger problems will require new techniques for reducing the huge search space. We are currently looking into approximate search space compression and aggregation techniques.

In order to model actions that temporarily provide resources, we plan to extend our action representation to a probabilistic version of PDDL2.1; this shouldn't be difficult, but will require revisions in our mutex rules.

We also wish to extend our algorithms to richer models like rewards, non-absorbing goals, mixed costs, stochastic action durations *etc.* Our techniques are general enough to be applicable in all these scenarios. For example consider the mixed cost optimization problem, in which the objective function is the sum (or a linear combination) of time and resource costs. Here, an equivalent MC heuristic can be computed by solving another MDP, which minimizes resource costs, and then adding the converged value to the MC heuristic reported herein. A hybrid algorithm can be easily developed in the same manner.

More generally, we believe that our hybridization technique is very general and applicable to a wide range of problems. For instance, we could create a proper, anytime approximation algorithm for Concurrent MDPs (CoMDPs) by hybridizing one of the RTDP algorithms of (Mausam & Weld 2004) with a traditional MDP algorithm. Similarly, a hybrid POMDP algorithm can be constructed by hybridizing RTDP for POMDPs with the policy for the equivalent MDP. We wish to explore these further.

References

Barto, A.; Bradtke, S.; Singh, S. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72:81–138.

Bonet, B., and Geffner, H. 2003. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *ICAPS'03*, 12–21. AAAI Press.

Bresina, J.; Dearden, R.; Meuleau, N.; Smith, D.; and Washington, R. 2002. Planning under continuous time and resource uncertainty : A challenge for AI. In *UAI'02*.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In *ECP'01*.

2003. Special Issue on the 3rd International Planning Competition, JAIR, Volume 20.

Mausam, and Weld, D. 2004. Solving concurrent Markov decision processes. In *AAAI'04*. AAAI Press.

Mausam, and Weld, D. 2005. Concurrent Probabilistic Temporal Planning. In *ICAPS'05*. To appear.

Smith, D., and Weld, D. 1999. Temporal graphplan with mutual exclusion reasoning. In *IJCAI'99*, 326–333. Stockholm, Sweden: San Francisco, CA: Morgan Kaufmann.

Directed C++ Program Verification

Tilman Mehler
University of Dortmund
tilman.mehler@cs.uni-dortmund.de

Abstract

Over the past years, formal methods for the verification of software have become a popular field of research. The systematic enumeration of a program's states allows to find subtle program errors, which pass unnoticed by manual code inspection or testing. In the following, we present the author's research based on the development of an assembly-level c++ model checker called StEAM. We sketch the internal architecture of the tool and discuss various techniques to counter the state explosion problem. We also emphasize the versatility of the tool by using it as a planner in multi agent systems.

1 Introduction

The author's research targets the area of *directed model checking*, with a focus on the verification of software. Model checking addresses the automatic verification of systems. Applications for this range from hardware design over manufacturing processes to verification of c++ and Java-programs. The starting point is always a model $\mathcal{M}$ of the system under investigation, as well as a formal description p of a property which should hold for the system. The goal of model checking is to prove $\mathcal{M} \models P$ ($\mathcal{M}$ *models* p) by a systematic enumeration of all reachable states of $\mathcal{M}$. If during exploration a state e violating p is found, the user must be given a sequence of states $i, s_0, s_1, .., e$ (the *error trail* or *error path*) which leads from initial state i to the error state e. The error trail is meant to help the system designer to find the error which leads to the violation of p.
From all branches of model checking one of the most established is software verification. In the annual international workshop "Model Checking Software" (SPIN), the latest advances in the area of software verification using model checking are discussed. A central issue in this event is the most established model checker *SPIN* [9], whose development from Gerard Holzmann ranges back to the year 1980. Since the turn of the millennium, the tool JPF [17] (Java PathFinder), developed by the NASA researchers Klaus Havelund and Willem Visser has caught the attention of the model checking community. JPF is a model checker for concurrent Java programs, which differed from all other software verification tools at that time. Based on a virtual machine for Java bytecode, JPF constitutes the first software model checker which does not rely on a formal model of the investigated program. Instead, it directly explores the bytecode of the compiled program.

2 Initial Studies

The author's research in model checking started in 2002 with his masters thesis about *Directed Java Program Verification* [13] Freiburg. Here, he intensively studied the tools HSF-Spin [2] and JPF. HSF-Spin constitutes an extension of the SPIN model checker, developed by Alberto Lluch-Lafuente in the course of his dissertation. The tool provides a set of heuristics, which accelerate the search for certain error types. For instance, the search for a deadlock can be significantly shortened , if the exploration favors paths with a maximum of blocked processes. In the course of his masters thesis, the author first created an HSF-Spin interface for the *Bandera* toolset of the Kansas-State University. Bandera translates Java source code in the input language of various model checkers - including Promela, the language of SPIN and HSF-Spin. Through these efforts it was for the first time possible, to use HSF-Spin's heuristics for the verification of Java programs. Furthermore, the heuristics *hamming distance* and *FSM-distance* where implemented in JPF and experimentally evaluated. The two heuristics belong to the class of *trail-directed* heuristics, which help to shorten a suboptimal error path. The essential results of this work has been published at the international workshop *Model Checking and Artificial Intelligence* (MoChArt'03) in Acapulco [4].
The insights gained from his master thesis have guided the author in his work as a research assistant. First of all he believes in the superiority of the JPF approach, since it bypasses the need for a formal model. The problem with these model is that the expressiveness of the underlying modeling languages often fall short of the formal semantics of actual programming languages, such as Java or c++. As a consequence, the models reflect a heavily abstracted form of the original program, which may lack just those details that lead to errors in practice. Furthermore, the model based approaches often require, that the user is familiar with the modeling languages

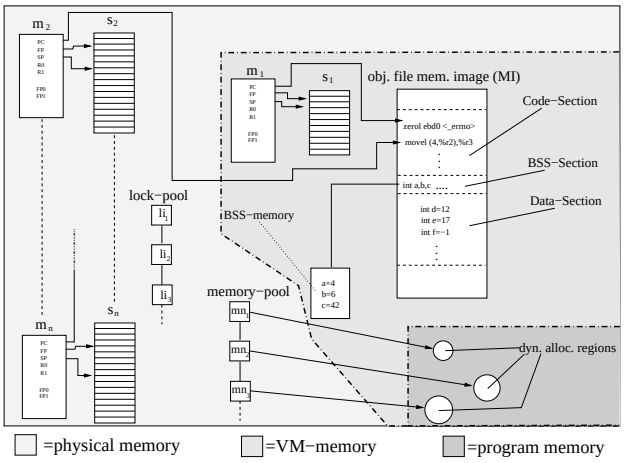


Figure 1: StEAM's state description

, which makes it difficult for the approach to establish itself in practice. Another important insights lies in the need for heuristics to cope with the *state explosion problem*. This combinatorial problem arises from the fact, that the size of a state space grows exponentially with the number of involved processes. As a result, uninformed search is often unable to find errors, since a lack of resources prevents a sufficiently deep exploration. On the other hand, experiments show, that the length of an error trail often grows only linear in the number of processes. For example with the *most-blocked* heuristic it is possible to find an error in programs involving up to 90 processes with optimal path length, while uninformed search already fails for 5 processes [12]. This places the design of powerful heuristics on top of the list of options to counter the state explosion problem.

3 Previous Research and Results

The Model Checker StEAM

The fist goal was the creation of an own model checker, which should serve as a platform for fundamental research. Furthermore, that tool should obey the author's principles, i.e. it should not require a formal model, allow the use of heuristics and give independence from other model checkers. Under these premises the c++ model checker *StEAM* [14] was developed. The tool uses a virtual machine *IVM*, with an instruction set of approximately 64.000 instructions. In the course of StEAM's development, fist IVM was extended with the capability to run programs with several concurrent processes. Using a state description, a program can be explored with AI search algorithms. Figure 1 depicts StEAM's state description. It is essentially composed from the the *stacks* and *CPU-registers* of the running threads, the *data-* and *bss-sections* holding the contents of global variables, a *memory-pool* for storing information about dynamically allocated memory and a *lock-pool* which memorizes locked resources. StEAM supports incremental storing of sates such that components are explicitly stored, only if they differ from the corresponding component of the predecessor state.

Compared to JPF, StEAM provides some new aspects. First of all, the tools builds on an existing virtual machine. This option was explicitly refused by the developers of JPF, since they thought that embedding a model checker in existing infrastructure would be too much effort [18]. However by building StEAM on IVM the work intensive task of designing an custom virtual machine was avoided. The latter is also one of the greatest potential error sources: If the virtual machine contains errors, this may falsify subsequent verification processes, resulting in the report of non-existent errors or in the missing of actual errors. In its current state of development, IVM is already capable of correctly running complex programs including commercial games. This is a strong empirical evidence for the correct design of the virtual machine. Furthermore, the development of a c++ model checker imposes a greater challenge than that of a respective tool for Java. This is already evident through the fact, that Java only allows allocation of typed memory, which greatly simplifies the description and handling of a system state. Also for program model checking, the search for concurrency errors is of particular interest. Java offers the possibility to write concurrent programs as a standard through the *Thread* class. Such a standardized interface does not exist for c++. The extension of IVM provides a simple an platform-independent way to write concurrent programs in c++.

Another proof for the easier handling of Java is the fact that the NASA researchers decided to develop a Java model checker, although the investigated software was mainly written in c++ and had to be manually translated prior to verification [18].

Heuristics

Besides the uninformed algorithms depth-first and breadth-first search, StEAM also offers the heuristic search algorithms best-first, A* and IDA*. Using the heuristic search methods and a set of heuristics, we can significantly accelerate the search for errors. The current version of StEAM supports the well-knows heuristics *most-blocked* [2; 8; 7] and *interleaving* [8; 7] as well as some new ones, namely *lockNblock* (a refinement of the most blocked heuristic) and *read/write* - a novel kind of heuristic, which maximizes the number of consecutive read and write operations. This way we are already able to find errors in simple programs, including implementations of classical protocols such as the dining philosophers as well as in some simplified real world applications like the controller for a bank automaton. In particular, in most cases the new heuristics expose a far better performance than the old ones. A fist presentation of StEAM and the experimental results was given at the 2004 SPIN workshop in Barcelona [12].

State Reconstruction

An apparent problem of unabstracted software model checking as done by StEAM lies in the large state description.

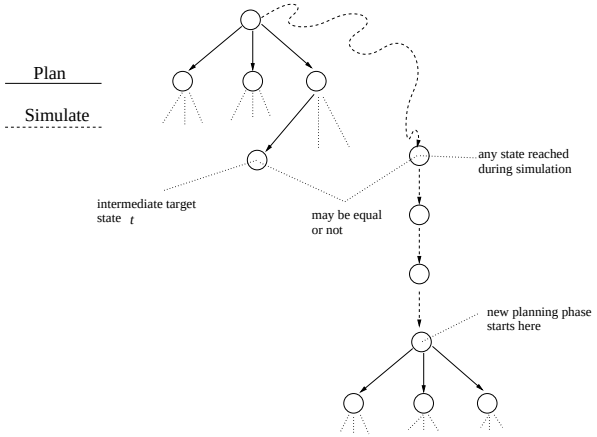


Figure 2: Interleaving of Planning an simulation

Since - in the worst case - we need to explicitly store the entire program state, including stacks, variable sections, and the contents of all allocated memory blocks, the search may quickly exceed the available memory. To counter this problem, we make use of the speed of IVM to devise a time/memory-tradeoff called *state reconstruction*. Instead of representing each generate state explicitly in memory, we replace a state s by a constant-sized *mini state*, which merely holds a pointer to its immediate predecessor p and the transition, which leads from p to s. Now, the explicit state description of s can be obtained by following the predecessor pointers starting from s up to the initial state i, which is the only state whose explicit representation is kept in memory all the time. Given the sequence $t_1, .., t_n$ of transitions, which occurred in the path from s to i, we apply $t_n, .., t_1$ to i. This is fast, as we only need to execute the corresponding machine instructions. Experimental results show, that state reconstruction allows us to find errors in programs where explicit state representation fails due to the given memory limit. The results are currently under review.

Planning by Model Checking

With StEAM, a powerful and versatile tool was created, whose application is not limited to the verification of software. The tool was also successfully used as a planner for multi agent systems. Here, it is exploited, that StEAM allows the simulation of a program as well as their exploration. The approach is based on a formal description of a multi agent manufacturing problem (MAMP [15]), which constitutes a more complex version of job-shop problems [1]. Through a sequence of consecutive planning and simulation phases, (cf. 2) we get a learning effect that allows us to solve MAMPs significantly better than through a purely autonomous behaviour of the agents. The respective paper was published at the 2004 German conference on artificial intelligence (KI-2004) [15]. Despite the multi agent discourse, upcoming research interest will focus on the verification of Software. On the one hand this includes the further development of StEAM, to be capable of verifying even more complex programs. To cope with the very large state spaces, the iterated search algorithms *iterative deepening* and *IDA** have recently been implemented.

Incremental Hashing

Since the exploration of a program exposes a large set of duplicate states, the use of a hash table is essential. By default, StEAM stores fully expanded states explicitly. In this context, the use of iterated search algorithms does not make sense, since at a backtracking step the corresponding state remains in the hash table and no memory is freed. One solution to this problems lies in the use of compacting hash functions like *bitstate-bashing*. Here, a set of n hash functions $f_1, \ldots, f_n$ map a state s to positions within a bit vector. A state s is regarded as a duplicate only if bits $f_1(s), \ldots, f_n(s)$ are already set in the bitvector. In some cases, it is possible that a state is wrongly seen as a duplicate, leaving parts of the search tree unexplored. Experimental results however showed, that the possibility for this is very low [10]. In the case of StEAM, the size of the state description becomes a problem: If the hash function takes all components of the state description into account, the exploration is slowed down significantly, while hashing only part of the state results in a much higher number of hash collisions. Hence, for the effective use of compacting hash functions we must first find an effective hash function, which also minimizes the number of hash collisions. Here, we can exploit the fact, that state transitions will only change a small part of the state description. For example, we can use an enhanced version of the algorithm of Karp and Rabin [11] to calculate a hash address in $O(k)$, where k is the number of sate components that are changed by the transitions. In recent fundamental research we devised an incremental hashing scheme for state space exploration. Some first results with the Atomix solver *Atomixer* have been presented at the 2004 workshop on planning scheduling and design (PUK'04) [5]. In a subsequent work, the incremental hashing scheme has also been implemented for the sliding-tile puzzle [16] and the for propositional action planning with pattern databases [6].

As the most recent contribution, incremental hashing was implemented for the stacks in StEAM. Some first experimental results show a speedup of the exploration by factor 10 and more if incremental hashing is used. We can expect an even higher gain, if the hashing scheme is extended to the full state description (including the data- and bss-sections and the pools).

4 Future Goals

Pattern Data Bases

Besides the technical improvement of StEAM, a main focus of research will lie on the development of new and more powerful heuristics. Here, the use of *pattern data bases* PDBs will be of particular interest. PDBs use an abstraction function ϕ which maps a state $s = (s_1, \ldots, s_n)$ to a pattern $\phi(s) = (\phi(s_1), \ldots, \phi(s_n))$. In a subsequent complete exploration of the abstract state space induced by ϕ, a table is constructed storing with each abstract state $\phi(s)$ its distance to the abstract error state $\phi(e)$. These distances serve as a heuristics estimate for the distance of an actual state s to the

actual goal state g.

Temporal Logics

In its current form, StEAM can detect deadlocks and assertion violations. In the future it should also be possible to verify properties described in formulae of temporal logics, such as LTL [3]. Temporal logics allow to express more complex properties than assertions. For example, temporal formulae may include quantification over paths such as: *EG p* (there is a path, on which proposition *p* is always true) and $p \Rightarrow AF\ q$ (if *p* holds, then on each path *q* will eventually hold=. As an example we may want to verify for an elevator control program: if a button is pressed at a certain floor, the elevator will eventually arrive there.

Practical Studies

The thesis will equally cover practical and theoretical aspects. The practical part will describe the engineering effort which tailors the model checking engine to the virtual machine. Furthermore the search algorithms, heuristics, hash functions etc. will be evaluated with a carefully chosen set of test programs. The theoretical part will consist of a formal discussion of the same methods with respect to time and space complexity. Furthermore the theory will cover problems that arise directly from the practical part. This includes in particular universal and specific solution to problems that arise out of the extraordinarily large state description. A fist contribution to this is given by the incremental hash functions, whose essence has already been theoretically discussed and experimentally evaluated [5].

The Vision

The ultimate goal of the thesis will be to give a clear perspective of unabstracted software model checking. (in particular for c++ programs). To achieve this, StEAM will be brought to a state of development, which allows it to reliably verify at least a couple real world applications. To cope with the state explosion problem, the tool should posses a set of powerful heuristics, which either target the detection of certain error classes (such as deadlocks) or which exploit structural properties of the underlying programming language (as it is the case e.g. for the read/write) heuristic.

References

[1] Y. Abdeddaim and O. Maler. Preemptive job-shop scheduling using stopwatchautomata. In J.-P. Katoen and P. Stevens, editors, *AIPS Workshop on Planning via Model Checking*, pages 7–13, 2002.

[2] S. Edelkamp, A. Lluch-Lafuente, and S. Leue. Directed model-checking in HSF-SPIN. In *Workshop on Model Checking Software (SPIN)*, Lecture Notes in Computer Science, pages 57–79. Springer, 2001.

[3] S. Edelkamp, A. Lluch-Lafuente, and S. Leue. Directed model-checking in HSF-SPIN. In *Model Checking Software (SPIN)*, Lecture Notes in Computer Science, pages 57–79. Springer, 2001.

[4] S. Edelkamp and T. Mehler. Byte code distance heuristics and trail direction for model checking Java programs. In *Model Checking and Artificial Intelligence (MoChArt)*, pages 69–76, 2003.

[5] Stefan Edelkamp and Tilman Mehler. Dynamic hashing in state space search. In *18th Workshop "New Results in Planning, Scheduling and Design"*, pages 15–19, 2004.

[6] Stefan Edelkamp and Tilman Mehler. Incremental hashing for pattern data bases. In *ICAPS05 poster proceedings (page to appear)*, 2005.

[7] A. Groce and W. Visser. Heuristic Model Checking for Java Programs. *SPIN Workshop on Model Checking of Software*, pages 242–245, 2002.

[8] A. Groce and W. Visser. Model checking Java Programs using structural heuristics. In *International Symposium on Software Testing and Analysis (ISSTA)*, pages 12–21, 2002.

[9] G. J. Holzmann. The model checker SPIN. *Software Engineering*, 23(5):279–295, 1997.

[10] Gerard J. Holzmann. An analysis of bistate hashing. In *Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification XV*, pages 301–314. Chapman & Hall, Ltd., 1996.

[11] R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

[12] P. Leven, T. Mehler, and S. Edelkamp. Directed error detection in C++ with the assembly-level model checker StEAM. In *Model Checking Software (SPIN)*, pages 39–56, 2004.

[13] T. Mehler. Gerichtete Java-Programmvalidation. Master's thesis, Institute for Computer Science, University of Freiburg, 2002.

[14] T. Mehler and P. Leven. *Introduction to StEAM - An Assembly-Level Software Model Checker*. Technical report, University of Freiburg, 2003.

[15] Tilman Mehler and Stefan Edelkamp. Planning in concurrent multiagent systems with the assembly model checker StEAM. In *27th edition of the annual German Conference on Artificial Intelligence (poster procedings)*, pages 16–30, 2004.

[16] Tilman Mehler and Stefan Edelkamp. Incremental hashing in ai puzzle donains. In *under review*, 2005.

[17] W. Visser, K. Havelund, G. Brat, and S. Park. Java PathFinder - second generation of a Java model checker. In *Post-CAV Workshop on Advances in Verification, (WAVe)*, 2000.

[18] W. Visser, K. Havelund, G. Brat, and S. Park. Model Checking Programs. In *International Conference on Automated Software Engineering*, pages 3–12, 2000.

Mixed-Initiative Planning Approach to Rescue Robotic System

Andrea Orlandini
Dipartimento di Informatica e Automazione
Università degli Studi di Roma TRE
Via della Vasca Navale, 79 - Roma - Italy
orlandin@dia.uniroma3.it

Abstract

My doctoral work focuses on the application of a mixed-initiative planning approach to a HRI system for a robotic system (DORO) in a rescue domain. The system's control architecture allows human-planner interaction during rescue mission. NIST test scenarios have been used in order to verify the system performances and we are currently working in order to use our architecture in the next RoboCup Rescue contest.

Introduction and Motivation

Urban search and rescue (USAR) deals with response capabilities for facing urban emergencies, and it involves the location and rescue of people trapped because of a structural collapse. Starting in 2000, the National Institute of Standard Technology (NIST) has initiated the USAR robot competitions (Tadokoro *et al.* 2000; Maxwell *et al.* 2004). NIST, in particular, features future standards of robotics infrastructures, pioneering robotics participation to rescue missions. RoboCup Rescue contests are a test-bed of the technology development of NIST project, and are becoming a real challenge for the robotics community. Rescue robots uphold human operators exploring dangerous and hazardous environments and searching for survivors.

During the mission (20 min.), the operator-robot has to coordinate several activities: exploring and map the environment, avoiding obstacles, localizing itself, searching for victims, correctly locating them on the map, identifying them, and finally describing their status and conditions. In this kind of contest, human-robot interaction has a direct impact on the effectiveness of the rescue team performance.

A crucial aspect of rescue environments, discussed in (Murphy 2004) concerns the operator's situation awareness and human-robot interaction. The difficulties in forming a mental model of the 'robot eye' are endorsed, pointing out the role of the team. In this sense the overall control framework has to capture the operator attention towards "what is important", so as to make the correct choices: following a path, enter a covert way, turn around an unvisited corner, check whether a visible victim is really reachable, according to some specific knowledge acquired during the exploration. In this setting, a fully manual control over a robot rescue is not effective (Bruemmer *et al.* 2003): the operator attention has to be focused over a wide range of activities, losing concentration on the real rescue mission objective: i.e, locating victims. Moreover a significant level of training is needed to teleoperate a rescue rover. On the other hand, fully autonomous control systems are not feasible in a rescue domain where too many capabilities are needed. Therefore, the integration of automonous and teleoperated activities is a central issue in rescue scenarios and has been widely investigated (Kiesler & Hinds 2004; Yanco & Drury 2002; Drury, Scholtz, & Yanco 2003; Michael Baker & Yanco 2004; Yanco & Drury 2002).

In this work we describe a mixed-initiative planning approach (Ai-Chang *et al.* 2004; Myers *et al.* 2003; Allen & Ferguson 2002; Burstein & McDermott 1996) to Human-Robot Interaction (HRI) in a rescue domain and illustrate the main functionalities of a rescue robot system[1]. We deploy a model-based executive monitoring system to interface the operators' activities and the concurrent functional processes. In this setting the user's and the robot's activities are coordinated by a continous reactive planning process which consists of: (i) checking the execution status with respect to a declarative model of the overall system; (ii) providing proactive activity while mediating among conflicting initiatives. In particular, we show that this approach is enhance both the operator's situation awareness and human-robot interaction for the execution and control of the several activities needed during a complex mission such as the rescue one. Moreover, the humans' overall mission can take advantage of the model, that keeps track of the robot/operator execution history, goals, and subgoals, as in fact the proposed control system can provide the operator with a better perception of the mission status.

We implemented our architecture on our robotic platform (DORO) and tested it in a NIST yellow arena. The main modules involved are: *Map*, managing the algorithm of map construction and localization; *Navigation*, guiding the robot through the arena with exploration behaviour and obstacle's avoidance procedures; *Vision*, used in order to automatically locate victims around the arena. (Murphy 2004) propose a high level sequence tasks cycle as a reference for the res-

[1] Doro is the third award winner in RoboCup Rescue contest (2004)

cue system behaviour: Localize, Observe general surroundings, look specially for Victims, Report (LOVR). Our interpretation of the cycle corresponds to the following tasks sequence: map construction, visual observation, vision process execution and victim's presence report.

Control Architecture & Model Based Monitor

Following the approach in (Muscettola *et al.* 2002) we introduce a two-layered system where decision processes (inclucing declarative activities and operator's interventions) are tightly coupled with functional processes through a model-based executive engine.

The physical layer devices are controlled by three functional modules associated to the main robots activities (mapping and localization, visual processing, and navigation). The *state manager* and *task dispatcher* in the figure are designed to manage communication between the executive and functional layers.

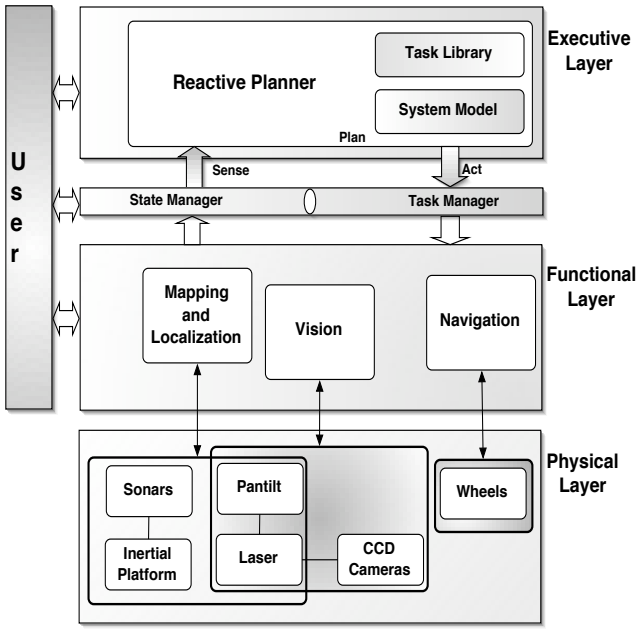


Figure 1: Control architecture

The *state manager* gets from each single module its current status so that any module can query the state manager about the status of any another module. The task dispatcher sends tasks activation signals to the modules upon receiving requests from the planner or the human operator. The overall computational cycle works as follows: the planner gets the modules status querying the state manager. Once the state manager provides the execution context, the planner produces a plan of actions and yields the first set of commands to the task dispatcher. In the execution phase, each module read the signals and start its task modifying its state. At the next cycle start, the planner reads the updated status through the state manager and can check whether the tasks were correctly delivered. If the status is not updated as expected a failure is detected, the current plan is aborted and a suitable recovery procedure is called.

As mentioned above, the functional layer is endowed with three main modules: *Mapping and Localization*, *Navigation*, and *Vision*. These modules provide different tasks that can be activated or stopped according to the *start* or *end* actions communicated by the task dispatcher.

A functional module is a reactive component that changes its internal status with respect to the action received from the task dispatcher. Nevertheless, it can also provide some proactiveness, by suggesting the planner/operator an action to be executed. For instance, the *Slam* module assumes a particular mode in order to communicate to the system that a map's construction cycle is ended, and then the control system can decide an action to stop the mapping phase. Morover, some modules can directly interact among them by communicating some low-level information bypassing the state manager (and the executive layer), e.g. *Slam* devises to *Navigation* the coordinates of the nearest unexplored point during the exploration phases.

The human operator can interact with the control loop both during the plan and the act phase. In the planning phase, the operator can interact with the control system by: (i) posting some goals which are to be integrated in the partial plan already generated; (ii) modifying the generated plan through the user interface; (iii) on-line changing some planning parameters, like the planning horizon, the lenght of the planning cycle, etc.. In the executive phase, the user can directly control some functional modules (e.g., deciding where the rover is to go, or when some activities are to stop). In this case, the human actions are assimilated to exogenous events the monitoring system is to manage and check. Finally, the operator's actions can be accessed by the state manager, and, analogously to the functional modules, can be monitored by the model-based control system.

The role of a model-based monitoring system is to enhance both the system safeness and the operator's situation awareness. Given a declarative representation of the system causal and temporal properties, the flexible executive control is provided by a reactive planning engine which harmonizes the operator activity (commands, tasks, etc.) with the mission goals and the reactive activity of the functional modules. Since the execution state of the robot is continuously compared with a declarative model of the system, all the main parallel activities are integrated into a global view and subtle resources and time constraints violations can be detected. In this case the planner can also start or suggest recovery procedures the operator can modify, neglect, or respect. Such features are implemented by deploying *high-level agent programming* in Temporal Concurrent Golog (Reiter 2001) which provides both a declarative language (i.e. Temporal Concurrent Situation Calculus (Pinto & Reiter 1995; Reiter 1996; Pirri & Reiter 2000)) to represent the system properties and the planning engine to generate control sequences.

The main processes and states of DORO are explicitly represented by a declarative dynamic-temporal model specified in the Temporal Concurrent Situation Calculus (TCSC) . This model represents cause-effect relationships and temporal constraints among the activities: the system is modeled as a set of *components* whose state changes over time.

Each component (including the operator's operations) is a concurrent thread, describing its history over time as a sequence of states and activities. For example, in the rescue domain some components are: *pant-tilt*, *slam*, *navigation*, *visualPerception*, etc. Each of these is associated with a set of processes, e.g. *navigation* can be $nvWand$, $nvGoTo$, or $nvStop$; *pan-tilt* can be: $ptIdle(x)$ (idling in position x), $ptPoint(x)$ (moving toward x), or $ptScan(x)$ (scanning x). The history of states for a component over a period of time is a *timeline*. Hard time constraints among activities can be defined by a temporal model using Allen-like temporal relations, e.g.: $ptPoint(x)$ *precedes* $ptScan(x)$, $ptScan(x)$ *during* $nvStop$, etc..

Our monitoring system is based on a library of Temporal Concurrent Golog scripts representing a set of flexible behaviour fragments. Each of these is associated to a task and can be selected if it is compatible with the execution context. For example, a timeout d can be associated to a task in order to constraint its execution w.r.t. d.

As illustrated before, for each execution cycle, once the status is updated (sensing phase), the Golog interpreter (planning phase) is called to extend the current control sequence up to the planning horizon. When some task ends or fails, new tasks are selected from the task library and compiled into flexible temporal plans filling up the timelines.

Any system malfunctioning or bad behaviour can be detected by the reactive planner (i.e. the Golog interpreter) when world inconsistencies have to be handled. In this case, after an idle cycle a recovery task has to be selected and compiled w.r.t the new execution status. For each component we have classified a set of relevant failures and appropriate flexible (high-level) recovery behaviours.

The planner/Golog interpreter can fail in its plan generation task, when *planner timeout* is raised. Since the reactive planner is the engine of our control architecture, this failure is critical. We identified three classes of recoveries depending on the priority level of the execution. If the priority is high, a safe mode has to be immediately reached by means of fast reactive procedures. In medium priority, some extra time for planning can be obtained by interleaving planning and execution: a greedy action is executed so that the interpreter can use the next time-slot to end its work. In the case of low priority, the failure is handled by replanning: a new task is selected and compiled. In medium and low level priority the operator can be explicitly involved in the decision process in a syncronous way. During a high-priority recovery we have no mixed initiative, unless the operator wants to take care of it, and the monitoring system is bypassed.

Mixed-Initiative Planning

The control architecture introduced before allows us to define some hybrid operative modalities lying between autonomous and teleoperated modes and presenting some capabilities that are crucial in a collaborative planning setting (Allen & Ferguson 2002).

The high-level agent programming paradigm, associated with the short-range planning/interpretation activity, permits an *incremental* generation of plans. In this way, the user attention can be focused on small parts of the problem and the operator can consider possible options on them, without loosing the overall problem constraints. *Plan stability* is guarenteed by flexible behaviours and plan recovery procedures which can harmonize the modification of plans, due to the operator's interventions or exogenous events. Minimal changes to plans lead to short replanning phases minimizing misalignments. Concerning the *open to innovation* issue, the model-based monitoring activity allows one to build novel plans, under human direction, and to validate and reason about them. Depending on the operator-system interaction these features are emphasized or obscured. We distinguish among three different mixed-initiative operational modes.

Planning-based interaction. In this setting, the planning system generates a cyclic LOVR sequences and the operator follows this sequence providing with few modifications, e.g. extending or reducing process durations. Here task's dispatching is handled in an automated way and the operator can supervise the decisions' consistency minimizing the interventions. The human-operator can also act as an executor and manually control some functional activities scheduled by the planner. For example he can decide to suspend automated navigations tools and take the control of mobile activities, in this way he can decide to explore an interesting location or escape from difficult environments. In this kind of interaction the operator initiative minimally interfere with the planning activity and *plan stability* is emphasized.

Cooperation-based interaction. In this modality, the operator modifies the control sequence produced by the planner by skipping some tasks or inserting new actions. The operator's interventions can determine a misalignment between the monitoring system expectations (i.e. the control plan) and the state of the system; this is captured at beginning of the next execution cycle when the state monitor provides the current state of the modules. In order to recover the monitor-system adherence, the planner has to start some recovery operations which are presented to the operator. Obviously, these activities are to be executed in real-time by verifying the satisfiability of the underlaying temporal and causal constraints. This modality permits maximal flexibility for the planner and operator interactive initiatives. Indeed, they can dialogue and work in a concurrent way contributing to the mission completion (*incremental planning*): while the operator tries to modify the plan in order to make it more effective (i.e. the system is *open to innovation*), the monitoring system can validate the operator's choices warn in the case of safety constraints violations and/or suggest suitable corrections.

Operator-based interaction. This modality is similar to teleoperation, the system activities are directly managed by the operator (some autonomy can be always deployed when the operator attention is to be focused on some particular task, e.g. looking for victims). The operator-based interaction is reached when the operators' interventions are very frequent, the planner keeps replanning and cannot support the user with a meaningful proactive activity. In this operative scenario, the planner just follows the operators' choices playing in the role of a consistency checker. The monitoring system can notify the user only safety problems and, in this case, recovery procedures can be suggested (*incremen-*

tal planning can be used only to generate non-critical planning procedures).

Mixed-initiative approach at work

We tested the control architecture and the effectiveness of the mixed-initiative approach in our domestic arenas comparing three possible settings: (i) *fully teleoperated*: navigation, slam, and vision disabled; (ii) *mixed-initiative control*: the monitoring system was enabled and the operator could supervise the rover status and take the control whenever this was needed; (iii) *autonomous control*.

Following the analysis schema in (Scholtz *et al.* 2004) here we briefly discuss some interesting aspects. Concerning *global navigation*, the performance of the mixed-initiative setting are quite stable while the autonomous system performs poorly in small arenas because narrow environments challenge the navigation system which is to find how to escape from them. In greater and more complex arenas the functional navigation processes start to be effective while the fully teleoperated behaviour degrades: the operator gets disoriented and often happens that already visited locations and victims are considered as new one, instead, we never experienced this in the mixed-initiative and autonomous modes. The effectiveness of the control system for *local navigation* and *vehicle state* awareness can be read on the *bumps* row; indeed the bumps are significantly reduced enabling the monitoring system. In particular, we experienced the recovery procedures effectiveness in warning the operator about the vehicle attitude. E.g. a typical source of bumping in teleoperation is the following: the visual scanning process is interrupted (timeout) and the operator decides to go in one direction forgetting the pan-tilt in a non-idle position. Enabling the monitor, a recovery procedure interacts with the operator suggesting to reset the pan-tilt position. The victim identification effectiveness can be assessed considering the victims found in the autonomous mode, considering that visual processing was deployed without any supervision, these results seem quite good (we experienced some rare false-positive).

Our experimental results show that the system performances are enhanced with the presence of an operator supervising the mission. It seems that the autonomous activities are safely performed, but the operator can choose more effective solutions in critical situations. Thus, we can trade off high performances and low risks by exploiting both human supervision and machine control.

References

Ai-Chang, M.; Bresina, J.; Charest, L.; Chase, A.; Hsu, J.-J.; Jonsson, A.; Kanefsky, B.; Morris, P.; Rajan, K.; Yglesias, J.; Chafin, B.; Dias, W.; and Maldague, P. 2004. Mapgen: mixed-initiative planning and scheduling for the mars exploration rover mission. *Intelligent Systems, IEEE* 19(1):8– 12.

Allen, J., and Ferguson, G. 2002. Human-machine collaborative planning. In *Proceedings of the 3rd international NASA Workshop on Planning and Scheduling for Space*.

Bruemmer, D. J.; Boring, R. L.; Few, D. A.; Marble, J. L.; and Walton, M. C. 2003. "i call shotgun!": An evaluation of mixed-initiative control for novice users of a search and rescue robot. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*.

Burstein, M., and McDermott, D. 1996. Issues in the development of human-computer mixed-initiative planning. *Cognitive Technology* 285–303. Elsevier.

Drury, J. L.; Scholtz, J.; and Yanco, H. A. 2003. Awareness in human-robot interaction. In *Proceedings of the IEEE Conference on Systems, Man and Cybernetics*.

Kiesler, S., and Hinds, P. 2004. Introduction to the special issue on human-robot interaction. *Special Issue of Human-Computer Interaction* 19(1,2):1–8.

Maxwell, B. A.; Smart, W. D.; Jacoff, A.; Casper, J.; Weiss, B.; Scholtz, J.; Yanco, H. A.; Micire, M.; Stroupe, A. W.; Stormont, D. P.; and Lauwers, T. 2004. 2003 aaai robot competition and exhibition. *AI Magazine* 25(2):68–80.

Michael Baker, Robert Casey, B. K., and Yanco, H. A. 2004. Improved interfaces for human-robot interaction in urban search and rescue. In *Proceedings of the IEEE Conference on Systems, Man and Cybernetics*. "To appear".

Murphy, R. 2004. Human-robot interaction in rescue robotics. *IEEE Transactions on Systems, Man and Cybernetics, Part C* 34(2):138–153.

Muscettola, N.; Dorais, G. A.; Fry, C.; Levinson, R.; and Plaunt, C. 2002. Idea: Planning at the core of autonomous reactive agents. In *Proc. of NASA Workshop on Planning and Scheduling for Space*.

Myers, K. L.; Jarvis, P. A.; Tyson, W. M.; and Wolverton, M. J. 2003. A mixed-initiative framework for robust plan sketching. In *Proceedings of the 2003 International Conference on Automated Planning and Scheduling*.

Pinto, J., and Reiter, R. 1995. Reasoning about time in the situation calculus. *Annals of Mathematics and Artificial Intelligence* 14(2-4):251–268.

Pirri, F., and Reiter, R. 2000. Planning with natural actions in the situation calculus. *Logic-based artificial intelligence* 213–231.

Reiter, R. 1996. Natural actions, concurrency and continuous time in the situation calculus. In *Proceedings of KR'96*, 2–13.

Reiter, R. 2001. *Knowledge in action : logical foundations for specifying and implementing dynamical systems*. MIT Press.

Scholtz, J.; Young, J.; Drury, J.; and Yanco, H. 2004. Evaluation of human-robot interaction awareness in search and rescue. In *Proceedings of the 2004 International Conference on Robotics and Automation*.

Tadokoro, S.; Kitano, H.; Takahashi, T.; Noda, I.; Matsubara, H.; Shinjoh, A.; Koto, T.; Takeuchi, I.; Takahashi, H.; Matsuno, F.; Hatayama, M.; Nobe, J.; and Shimada, S. 2000. The robocup-rescue project: A robotic approach to the disaster mitigation problem. In *ICRA-2000*, 4089–95.

Yanco, H., and Drury, J. 2002. A taxonomy for human-robot interaction. In *Proc. AAAI Fall Symposium on Human-Robot Interaction*, 111–119.

Planning with non-boolean actions and fluents

Valentina Poggioni
Dipartimento di Informatica e Automazione
Università "Roma Tre"
Via della Vasca Navale 79, 00146 Roma, Italy
poggioni@dia.uniroma3.it

Abstract

This work can be seen as a first approach to a new planning model that takes into account the possibility to express actions and fluents with non-boolean values. According to this model, a planning problem is defined using both graded (multi-valued) and classical (boolean) fluents. Moreover, actions that can have different application degrees can be defined. In this work a PDDL extension allowing to describe such new problems is proposed and a planning algorithm for such problems is presented.

Introduction

In the last years, extensions of the classical planning model have been investigated, such as temporal models, conditional and contingent models, probabilistic models and other mixed models (for example (Hoffmann 2002; Petrick & Bacchus 2002; Bonet & Geffner 2003; Chien & al. 2000)). But, to the best of our knowledge, a feature of the classical model has never been modified: the use of boolean expressions to describe fluents and actions in the domains. This is often too restrictive in order to represent realistic domains because the world is not black and white (i.e. true or false) but it has a lot of colors (i.e. intermediate truth-values, or "degrees of truth"). A different approach to planning that takes into account a numerical "state" associated to an action or fluent is proposed in the probabilistic planning model, but, in this case, "numbers" represent our knowledge or uncertainty about the state or the success of an action, while the real world is always two-valued.

This work can be seen as a first approach to a new planning model that takes into account the possibility to express both actions and fluents with non-boolean values. According to this model, a planning problem is defined using both graded (multi-valued) and classical (boolean) fluents; moreover actions having different application degrees can be defined (i.e. actions having adjustable intensity and that can affect fluents proportionally to how much they are applied). In such a way, also the efficiency of actions can be easily represented. For example, to dry a pavement an action with only the proper amount of power can be chosen; if the pavement is partially dry, less power can be used. If we have a defective machine we can express the result (i.e. the action effect) proportionally to its efficiency. Moreover we provide the possibility to define an objective function in the action application degrees that can be maximized or minimized.

The work defines an extension of the planning language PDDL that allows us to define planning domains and planning problems having graded fluents and graded actions, and a solving algorithm for such problems, where first a candidate plan with partially instantiated actions is constructed, then the plan applicability and correctness is verified by means of a translation into a MIP (Mixed Integer Programming) problem and finally, if a solution of the MIP problem exists, a complete instantiation is made and the solution plan is found.

The Planning Language

The planning language proposed is based on standard PDDL. It provides two kinds of actions and fluents, representing both classical $boolean$ fluents and actions, and $graded$ fluents and actions. Both actions and fluents have an additional argument denoting their "degree of truth": in a fluent it means "how much" the predicate is true (0 means that it is false, 1 that it is true) and in an action it means "how much" the action is applied (0 means that it is not applied at all, 1 means that it is applied with the maximum efficiency).

The type of such terms is declared as a new type *degree* in the PDDL domain definition.

Fluents

If the fluent is $boolean$ then its truth-value is a natural number in $\{0, 1\}$ (as in the classical model), otherwise, if the fluent is $graded$, its truth-value is a real number in $[0, 1]$. There are different declaration sections for boolean and graded predicates. For example:

```
(define(domain example)
  (:types block degree )
  (:boolean_predicates
     (holding ?b- block ?a- arm ?x- degree))
  (:graded_predicates
     (gripper_dry ?a- arm ?x- degree))
  ... )
```

Actions

The additional parameter in an action denotes the degree of application of the action. Again, the value of this parameter

in *boolean actions* belongs to $\{0, 1\}$, while it ranges in $[0, 1]$ in *graded actions*.

Actions may also have other additional parameters referring to fluent degrees. A specific section in action declaration allows one to associate fluents to degree parameters. Such variables maybe used both in preconditions and effects: action preconditions may contain inequality constraints over fluent degrees and effects can be described by means of expressions that define the new "degree of truth" of a fluent as a linear combination of previous fluent values and the application degree of the action.

The choice to include the degrees among the action parameters does not increase the complexity of the operators because, as explained below, such parameters are not instantiated during the solution search phase. Preconditions contain inequality constraints over fluent degrees. They have the form `(cond_type var value)`, where `cond_type` is one of $\{<, \leq, =, \geq, >\}$, `var` is a variable of type degree and `value` is either a real or an integer number (according to the fluent type); the effect standard form contains linear combination in the action and fluent degrees. We can use a syntax PDDL-style `(op var_k Expr(x,var_1,...,var_n))` were `op` is a PDDl operator for numerical expressions (`assign`, `increase` or `decrease`), `var_k` is a variable of type degree referring to the fluent we are modifying, and `Expr(x,var_1,...,var_n)` is a linear combination in the action and fluent degrees.

The objective function

The definition of a graded planning problem may contain an *objective function*, that is a linear function of the action application degrees that must be minimized or maximized when looking for action degree values satisfying a given partially instantiated plan. For example, it can be used in order to minimize the cost of the extracted candidate plan giving different operator costs. It is defined in a new section `:objective_function(...)` in the problem description. It is defined by `(:objective_function(opt_type function))`, where `opt_type` defines the kind of optimization problem (`min` or `max`) and `function` is a linear expression of operator names. The declaration `(:objective_function min` $\mathtt{c}_1\mathtt{*op}_1 + \ldots + \mathtt{c}_k\mathtt{*op}_k$`)` has the following meaning: let $\mathtt{act}_{i1}, \ldots, \mathtt{act}_{in_i}$ be all the instantiations of the operator $\mathtt{op}_i$, and x_{il} the variable associated to the application degree of action $\mathtt{act}_{il}$, then the optimization problem is defined by $\min \sum_{i=1}^{k} c_i \cdot (\sum_{l=1}^{n_i} x_{il})$.

If no objective function is defined in the problem definition we minimize the sum of all operators defined in the domain assuming that they have the same unitary cost, i.e. we minimize the sum of application degrees of all actions in the candidate plan.

The planning model

A *graded ground atom* is an expression of the form $p(t_1, ..., t_n, v)$, where p is a fluent, $t_1, ..., t_n$ are constants and v is

- a real number in $[0, 1]$ if p is a graded fluent
- an integer in $\{0, 1\}$ if p is a boolean fluent

A state is a set of graded ground atoms.

Definition 1 (Graded Planning Problem)
A Graded Planning Problem *is a tuple* $(I, G, f, \mathcal{O})$ *where* I *is the initial state,* G *is the goal,* f *is the* objective function *and* $\mathcal{O}$ *is the operator set. A goal is a set of conditions having the same form as action preconditions.*

Definition 2 (Graded Solution Plan)
A graded plan $\mathcal{P} = (A_1, \ldots, A_m)$ *is a sequence of actions (i.e. fully instantiated operators; in particular their application degrees are constants); a graded plan is a* graded optimal solution plan *of the problem* $(I, G, f, \mathcal{O})$ *if* $A_1, \ldots, A_m$ *are executable starting from* I*, the final state satisfies the conditions in* G *and the action application degrees optimize the objective function* f*.*

Note that, in order for a solution plan to be optimal, the f-value is only required to be minimal/maximal with respect to possible degree values of the actions $A_1, \ldots, A_m$: a plan $\mathcal{P} = (A_1(g_1), \ldots, A_m(g_m))$ – where the displayed g_i are the degrees of the actions A_i – is optimal if there are no feasible solutions plans $\mathcal{P}'$ of the form $(A_1(g'_1), \ldots, A_m(g'_m))$ having a smaller/greater f-value. The objective function allows one to assign different costs to operators.

System Architecture

The system takes as input a graded planning problem $(I, G, f, \mathcal{O})$ and if a solution exists it returns a graded solution plan P, otherwise, if it terminates, it returns a *NoSolution* message. The system architecture is presented in Fig.1: it is composed by three modules, the *plan extractor*, the *system constructor*, the *MIP solver*.

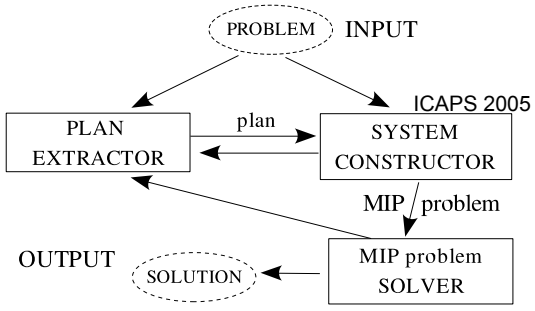


Figure 1: The system architecture

The *plan extractor* synthesizes a "candidate plan" where actions are partially instantiated. Then the *system constructor* computes the world evolution using these actions (the resulting states depend on the action application degrees) and reduces the verification and full instantiation of the plan to a MIP problem; it builds the MIP problem corresponding to a candidate plan and passes it to the *MIP solver*. During the system construction phase some conditions in the action preconditions are directly checked and if they are not satisfied the module fails and another candidate plan must be extracted. In this case the information about which action causes failure is used and another candidate plan is constructed replanning from this point. Finally the

MIP solver computes a solution of the generated problem; if a solution exists it is a set of real and/or integer values $\vec{g} = (g_1, \ldots, g_m)$ and the plan $P = (A_1(g_1), \ldots, A_m(g_m))$ is the graded solution plan of the given problem, otherwise a new candidate plan is extracted and a new MIP problem is generated.

MAIN $(I, G, \mathcal{O}, f)$ **:**

1. CYCLE $(1, I, G, \mathcal{O}, f, [\,], [\,])$

CYCLE $(n, I, Goal, \mathcal{O}, f, plan)$ **:**

1. $cand\_plan =$ PLAN_EXTRACTOR$(n, I, Goal, \mathcal{O}, plan)$
2. **if** $cand\_plan = Null$ **then return** $NoSolution$
   / $(I, G, \mathcal{O})$ has no solution*/*
3. **else**
   . $(sys, k) =$ SYSTEM_CONSTRUCTOR$(cand_plan, I, \mathcal{O})$
   . **if** $sys = Null$ **then**
   . . $(new\_plan, new\_goal) =$ recover$(cand_plan, k)$
   . . */* The good partial solution (from the first to the $(k-1)$-th action) is recovered.*/*
   . . **return** CYCLE $(k, I, new\_goal, \mathcal{O}, f, new\_plan)$
   . **else**
   . . $solution =$ MIP_SOLVER(sys, f)
   . . **if** $solution = Null$ **then**
   . . . **return** CYCLE $(1, I, Goal, \mathcal{O}, f, [\,])$
   . . **else return** $(plan, solution)$

Figure 2: The main algorithm

The *plan extractor* module

This module takes as input an integer index n, a graded planning sub-problem $(I, Goal, \mathcal{O})$ (i.e. without the objective function) and and the partial plan $plan$. It returns a sequence of partially instantiated actions, i.e. a candidate solution plan. In principle, any algorithm can be chosen for plan extraction, provided it is complete and backtrackable.

The technique presently adopted is very naive, and deserves further investigation. It implements a simple backward algorithm with heuristic functions that solves relaxed problems. As a classical backward algorithm at each step an operator is chosen according to the heuristic values defined below, a partial instantiation (i.e. not involving parameters of type degree) is performed and the goal is updated considering both the action preconditions and the action effects in order to construct the goal for the following step.

Here follow some preliminary definitions.

Definition 3

1. *Let $eff_i(o)$ be an effect of the operator $o \in \mathcal{O}$. Then $cap_i(o)$ is the coefficient of the application degree of the operator o in $eff_i(o)$, i.e. if $eff_i(o)$ has the form* `(assign ?v E+k*x)`*, where* `x` *is the application degree of o, then $cap_i(o)$ =* `k`.
2. *Let $g =$*`(cond_type var value)` *be a goal and $eff_i(o)$ be an effect of the operator $o \in \mathcal{O}$. We say that $eff_i(o)$ agrees with g if and only if one of the following conditions holds:*

   `cond_type` is $\geq$ or $>$ and $cap_i(o) \geq 0$
   `cond_type` is $=$ and $value = 1$ and $cap_i(o) > 0$
   `cond_type` is $\leq$ or $<$ and $cap_i(o) \leq 0$
   `cond_type` is $=$ and $value = 0$ and $cap_i(o) \leq 0$

3. (Heuristic function) *Let $\{eff_1(o), \ldots, eff_k(o)\}$ be the effects of the operator $o \in \mathcal{O}$. The value of the heuristic for o with respect to a goal $G = (g_1, \ldots, g_m)$ is defined by*

$$h(o, G) = \sum_{eff_i(o) \in C^+(o)} |cap_i(o)| - \sum_{eff_i(o) \in C^-(o)} |cap_i(o)|$$

   where $C^+(o)$ ($C^-(o)$) is the set of the effects of o that agree (do not agree) with some $g_j \in G$.

The heuristic function takes into account both positive contributions, i.e. those that could help to approach the goal, and negative ones, which would move away from the goal. At each step, the action with the greatest heuristic value is chosen (backtracking point) and the partial goal G is updated taking into account the preconditions and the effects of the chosen action. Let $Goal^{(i)}$ be the set of fluents representing the goal at the i-th step where the action act is chosen, then $Goal^{(i+1)} = (Goal^{(i)} \cup pre(act)) \setminus C^+(op)$, where $pre(act)$ is the set of preconditions of act.

The algorithm stops when either there are no goals to satisfy (a plan is found) or there are no actions to try (the problem has no solution). The sequence of the chosen actions is the candidate plan.

The solution returned can be inconsistent because we do not check possible conflicts generated by the choice of actions modifying a fluent that we have previously resolved with another action. The check is in fact delegated to the *system constructor* and the *MIP solver* modules.

If a plan is extracted it is passed to the *system constructor* that returns either the system of inequalities representing the action executability and the goal satisfiability conditions, or failure with an index k representing the index of the action having a failing precondition; if the algorithm fails the partial solution from the first to the $(k-1)$-th action chosen is recovered, together with the partial goal, and a new candidate plan starting from the k-th action is extracted. In this case the algorithm searches for a solution starting from the k-th choice with a different heuristic that gives a greater weight to actions that modify the preconditions of the action that has caused failure; if there is no solution the algorithm further backtracks, in a standard way, and calls the procedure starting from the $(k-1)$ step.

At the end of the procedure a virtual boolean action representing the goal is added to the candidate plan, so that in the following step its preconditions (the goal) are checked.

The *system constructor* module

This module takes as input the initial state I, the operator set $\mathcal{O}$ and the candidate plan $P = (A_1(x_1), A_2(x_2), \ldots, A_m(x_m))$ – where only the variables x_i standing for the degree of application of A_i are in evidence and the goal is represented by the action A_m.

It returns the pair (sys, k) where sys is the system of inequalities that will be solved if the procedure ends with success, and k is the index of the action that generates failure because some of its preconditions are not satisfied if the procedure fails.

Let $F_1(y_1), ..., F_n(y_n)$ be an enumeration of all the atoms in the language where all arguments are instantiated except for their degree value y_i. The world evolution is represented by a matrix S of size $m \times n$, where m is the number of actions in P and n the length of the above enumeration. Each element s_{ij} in the matrix represents the degree of F_j at the i-th state.

First of all, the first row is filled in with the numerical values defined by the description I of the initial state.

Then, for each step i, starting with $i = 1$, the preconditions of the action A_i are checked in the i-th state (i.e. the i-th row of S): let $F_k(v)$ and $C(v)$ occur in the preconditions of A_i, where $C(v)$ is an inequality constraint on v. If s_{ik} is a constant, $C(s_{ik})$ is immediately checked; if it is false, the algorithm fails and returns i, otherwise the condition is ignored. On the contrary, if s_{ik} is a linear expression depending on some x_j, the inequality $C(s_{ik})$ is added to the system sys.

Then the row $i + 1$ is filled in, using the effects of A_i: for each $F_k(v)$ such that $v = Expr(x_i, ...)$ is an effect of A_i, let $Expr'(x_i)$ be the expression obtained from $Expr(x_i, ...)$ by replacing each degree variable referring to fluent F_j with the expression occurring in s_{ij}. Then $s_{(i+1)k}$ is set equal to $Expr'(x_i)$. For every $F_k(v)$ that is not affected by A_i, $s_{(i+1)k} = s_{ik}$.

The *MIP solver* module

This module takes as input the MIP problem (sys, f) and returns a solution, if it exists; otherwise it calls the procedure CYCLE with CYCLE$(1, I, Goal, \mathcal{O}, f, [\ ])$, so that a new plan is generated. Given the input made up of the *graded planning problem* $\mathcal{P} = (I, G, f, \mathcal{O})$ and the plan $(A_1(x_1), \ldots, A_m(x_m))$ returned by the *plan constructor* module, if the MIP solver returns $\vec{g} = (g_1, \ldots, g_m)$, then the plan $P = (A_1(g_1), \ldots, A_m(g_m))$ is a graded optimal solution plan.

The prototype implementation developed uses the software *Lingo*, distributed by Lindo System Inc. [1] in a free download-able version for students, that can solve problems having up to 150 constraints and 300 variables.

Conclusions and related works

In this work a language and a model of planning with graded fluents and actions are presented. A prototype of the system has been developed.

To the best of our knowledge this is the first system able to manage non boolean actions. Recent works have proposed languages (Fox & Long 2003; Giunchiglia *et al.* 2004; Lee & Lifschitz 2003) and systems (for example (Koehler 1998; Baioletti, Milani, & Poggioni 2003; Hoffmann 2002; Haslum & Geffner 2000)) that can handle numerical values. Graded fluents could be represented in PDDL 2.1 by means of numerical fluents (functions) but respecting some more restriction w.r.t. what is done in this work. Is more difficult to have a good representation of graded action simply using numerical parameters because it is possible only if the parameters have finite domains. As we have chosen the graded/boolean style for the actions we have decided to maintain the same style for fluents.

At the moment we are working in two main directions. From a practical point of view we have to fix some lacks and improve some parts; an algorithm for intelligent backtracking when the *MIP solver* fails, an improvement of the backtracking phase when the *system constructor* fails, more informed heuristics for the *plan extractor* module and an optimization of the objective function over all feasible plan and not only over one skeleton plan are under investigation. Moreover a set of graded domains and graded problems is under construction in order to carry out a wide set of experiments. The second research direction is theoretical: an extension of the algorithm to planning under uncertainty on the initial state is straightforward, introducing variables for fluent degrees in the world construction. Moreover, we are investigating the possibility of representing and reasoning about vague (fuzzy) fluents, using intervals or sets to represent fluent degrees.

[1] Available at http://www.lindo.com

References

Baioletti, M.; Milani, A.; and Poggioni, V. 2003. Planning with fuzzy resources. In *AI*IA 2003: Advances in Artificial Intelligence. In LNAI 2829, 336-348*.

Bonet, B., and Geffner, H. 2003. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proc. of IJCAI 2003*.

Chien, S., and al. 2000. ASPEN: Automated planning and scheduling for space mission operation. In *Proc. of SpaceOps 2000, Colorado (USA)*.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR* 20:61–124.

Giunchiglia, E.; Lee, J.; Lifschitz, V.; N., M.; and Turner, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence* 153:49–104.

Haslum, P., and Geffner, H. 2000. Heuristic planning with resources. In *Proc. of ECAI-00*.

Hoffmann, J. 2002. Extending FF to numerical state variables. In *Proc. of ECAI 2002*.

Koehler, J. 1998. Planning under resource constraints. In *Proc. of ECAI-98*.

Lee, J., and Lifschitz, V. 2003. Describing additive fluents in action language ⌋+. In *Proc. of IJCAI 2003*.

Petrick, R., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. of AIPS 2002*.

A Framework for Multi-Robot Coordination

Sanem Sariel

Istanbul Technical University, Department of Computer Engineering, Istanbul, TURKEY TR34496

sariel@cs.itu.edu.tr

Advisors: Tucker Balch (Georgia Institute of Technology), Nadia Erdogan (Istanbul Technical University)

Abstract

In this thesis work, a complete framework for multi-robot coordination in which robots collectively execute inter-dependent tasks of an overall complex mission requiring diverse capabilities is proposed. Given a heterogeneous team of robots and task dependencies, the proposed framework provides a distributed, robust mechanism for assigning robots to tasks in an order that efficiently completes the mission. The approach is robust to unreliable communication and robot failures. The framework is based on the market-based approach, and therefore scalable. In order to obtain optimum allocations in noisy environments, a coalition maintenance scheme ensuring dynamic reconfiguration is introduced. Additional routines, called precautions are added in the framework for addressing different types of failures common in robot systems and solving conflicts in cases of these failures. The final solutions are close to optimal with the available resources at hand by using appropriate cost functions. The framework has been tested in simulations that include variable message loss rates and robot failures. The experiments illustrate the effectiveness of the proposed system in realistic scenarios.

Introduction

In this thesis work, a coordination framework for multi-robot teams implementing complex missions of tasks having ordering constraints, requiring diverse capabilities and collective work is proposed. The main objective of this framework is dynamically allocating inter-dependent tasks to multi robots in a cost optimal manner without violating the constraints under uncertainties on dynamic environments. The approach used in this framework is a market based scheme supporting scalability. However the optimality of the generated schedules is not addressed in this scheme for changing situations. Robot failures or unreliable communication are such situations common in real world domains. The proposed framework presents different kind of recovery solutions to obtain optimum solutions for dynamic environments.

Recently proposed works somehow address the dynamic task allocation issue against robot failures (malfunction or death) or environmental changes. In these works, the selected test domain missions have usually independent sub-tasks. Closest works to the proposed framework, Zlot's (Zlot and Stentz 2005), Lemarie's (Lemarie, Alami and Lacroix 2004) and Kalra and Stentz's (Kalra, Ferguson, and Stentz 2005) works address task dependency issue for tightly coupled missions. In Kalra and Stentz's work, the implemented system was tested for the collective perimeter sweeping mission. The task dependencies are considered in keeping a formation while obeying some rules. The tasks do not require different capabilities. In Lemarie's work, the main goal is keeping the architecture distributed among the robots so as to gain scalability, robustness and reactivity. However the solution's optimality is not guaranteed. They also inspire from market based approach. Using a token-ring network approach, only one auctioneer is allowed to initiate an auction at a time step. Zlot's recently proposed task allocation scheme (Zlot and Stentz 2005) deals with task tree auctions to obtain globally optimum solutions. Complexity of the task tree auctions increases drastically for more complex task trees. This scheme generates a reconfiguration on the allocations from each robot's point of view. However when there are inter-dependencies among tasks, additional considerations should be taken into. In these approaches, performance for combined tests of failures is not measured.

In the proposed framework, a mechanism for reconfiguration by considering dependencies is provided. The framework relies on a set of task dependencies that are compiled *a priori* by a mission commander, or a planner before the mission. The task dependencies, specifications of the mission, and the robot teams' capabilities are distributed (reliably) to the robots before mission execution begins. At that point, negotiation of task assignments and execution of the mission begins. The proposed framework strives to provide optimal solutions while responding effectively to communication and robot failures. This is the first coordination scheme to address such a broad range of failures for heterogeneous teams executing tightly coupled tasks requiring collective work (Sariel and Balch 2005).

Proposed Framework

The proposed framework is for a multi-robot team ($r_j \in R$, $0 \leq j < ||R||$) that must coordinate to complete a complex mission (M) including tasks T_i s ($0 \leq i < ||M||$) that have

ordering constraints and require diverse capabilities and collective work. The overall objective is completing M in a cost optimal manner. The framework combines *auctions*, *coalition maintenance* and recovery routines called *precautions* to provide an overall system that finds near optimal solutions in the face of noisy communication and robot failures. The *precaution* routines enable the system to dynamically respond to these failures at run time and complete the mission with valid plans at minimum cost.

Task Representation

A simple but effective task representation is used to generate valid plans. The generated plans without a general planner are always valid in the framework by means of the selected task representation. Information on the task definition, required capabilities, hard and soft ordering constraints, and required number of robots is embedded in the representation of each task. After the task dependencies are given initially, the framework generates optimal and valid allocations.

Roles

After being informed about the tasks and dependencies, robots are allowed to be in different roles during runtime to execute the tasks in coordination as required. The framework is designed for missions consisting of sub-tasks having dependencies and requiring either one or more than one robot to execute. Therefore the tasks may be executed by a group of robots. Coalition organizational paradigm (Bryan and Lesser 2004) is selected for teams of robots executing a task. The coalitions (C_i) can contain one or more robots numbers of which are defined in the task representation to execute a task T_i of overall mission M. The capabilities (cap_j) of robots r_j in a coalition should be a superset of the required capability set for T_i ($reqcap_i$). A robot (r_j) may be in different roles for task T_i such as auctioneer, bidder (B_{ij}), coalition leader (CL_i) and coalition member (CM_i) in different time steps.

- An *Auctioneer* robot manages auction negotiation steps and selects $reqno_i$ suitable members of a coalition.
- A *Bidder* robot is a candidate robot to become a member of a coalition executing a task.
- A *Coalition Leader* maintains the coalition and provides synchronization. It executes a portion of the task.
- A *Coalition Member* is one of the members of the coalition, and it executes a portion of the task.

A is the auctioneer set, B_{ij} is the bidder robot r_j for task T_i. A robot r_j may be in more than one B_{ij} roles for different tasks. However it is not allowed that a robot r_j is in more than one of roles A_i, CM_i, or CL_i. The coalition members are selected by auctions.

Precautions

For dealing uncertainties because of the message losses, each robot keeps track of the models of known system tasks and other robots in their world knowledge. When there is reliable communication, the robots update their world knowledge accordingly. Whenever there are message losses in the system, the world knowledge of each robot may be inconsistent. Such inconsistencies occur when robots are not informed about the tasks that are completed, under execution or under auction. *Precaution* routines discover conflicts and resolve them. When inconsistent messages are received, both corrections are made and warnings are released.

Routine Duties
if $r_j \in A$
  AuctionNegotitationProcess

if r_j is a CM_i
  if the "synchronization message" has been arrived
    Set coalition count down
  else if the coalition count down is 0
    Cancel execution of the task
    Send leave message to the CL_i ; break
  else
    if the last communication with CL_i is more than τ (threshold)
      Send "coalition leader query" message
      Decrease coalition count down
  Send updated cost/bid value to the CL_i

if r_j is a CL_i and ||C||>1
  if each CM_i has already sent the execution/updated cost message
    Send "synchronization message" to all CM_is
    Set coalition count down
  else if the coalition count down is 0
    Terminate the coalition; break
  else
    Decrease coalition count down
  Broadcast max cost (CM_i)
  Begin **CoalitionMaintenanceProcess**

if the robot is not executing a task || the task under execution is released
  Begin **SelectAction**
Send query messages for the tasks which are free
If there is a task selected, execute the own portion

Figure 1. Routine Duties of a Robot

Action Selection

Each robot initially updates its world knowledge based on the incoming messages. Then considering the new updated world knowledge and the current states of the processes for different roles, it selects the action. The action may be joining a coalition executing a task or becoming an auctioneer auctioning for a task. Before selection of the action, the robot should perform its routine duties for different roles. These duties are given in Figure 1. If it is leading an auction, it performs the negotiation process. Duties of roles CM_i and CL_i are performed for ensuring synchronization while coordinating in a coalition. If the robot is a CL_i, it checks the current coalition members' situations and can decide to reconfigure the coalition when new robots join or some members are not reachable.

While selecting the action, the robot considers the tasks awarded by an auctioneer, tasks under execution but with a higher value of cost than the robot has, the soft dependencies of the current task, and the free tasks which are not executed or under auction. The robot should be capable of executing all the tasks under consideration.

That means the hard dependencies of them should be completed, and the robot should be able to execute these tasks with its capabilities. Before the robot decides on one of these tasks, it ranks these tasks based on the costs for them and then selects the minimum cost task. This ranking procedure ensures optimum cost selection of the tasks for the given situation. The tasks awarded by an auctioneer have higher priority than free tasks, and the tasks of coalitions with higher maximum cost value than the robot's cost have the highest priority of all.

The selection of the task is performed by considering both the costs and priorities. The task with the minimum cost is selected. If the costs are the same for different tasks, the priorities are used for selection. Another ranking mechanism is used for the selection of a free task, if there are more than one free task having the minimum cost of all, the robot selects the one with the lowest number of soft dependencies and lowest number of robot requirements for executing. Therefore if the costs of the tasks are the same, initially the tasks requiring low coordination is selected. The algorithmic description of action selection process is given in Figure 2.

SelectAction for *rj*
For each known task T_i
  if the $cap_j \subseteq reqcap_i \wedge$ hard dep. of T_i are completed
    if the task is not in one of the following types skip
      The tasks for which the robot is accepted for joining (with higher max cost value than the robots)
      Awarded tasks by the auctioneers
      Free tasks which are not under consideration
    else add the task in a priority queue ordered by cost with the priorities in given order
Select the minimum cost task in the priority queue

if *rj* is in a coalition
  cancel executing the task, send "leave" message to the CL_i

if the selected task is a free task
  begin **AuctionNegotitationProcess**
else
  set the selected task as the current task
  if it is an awarded task
    send "accept become a member" message

Figure 2. Action Selection

Auction Negotiation Process

Each robot offers an auction for a free task if it is selected in the action selection step. When a robot becomes an auctioneer, it manages the auction negotiation process in which the coalition members and the leader are selected for the task to be executed.

Initially the auctioneer offers the auction. The robots can get the necessary task details from the auctions. After receiving an offer, the validity of the auction is checked. If the auction is invalid, a warning message is sent to the auctioneer. This invalidity may occur if the auctioneer has incomplete knowledge about the mission status. Possible situations may be that the task is completed or it has already been executing. If the auction passes the validity check, the candidate robot becomes a bidder of the task (B_{ij}), calculates the cost and sends the cost value as a bid. The other candidate robots behave as so. The auctioneer robot is also a bidder and generates a bid for the task at hand. It waits until the end of the deadline. If the auctioneer cannot get the necessary number of bids from the other robots until the deadline, it cancels the auction. Otherwise it ranks all the bids. It selects the robot with the minimum cost as the CL_i of the coalition. The remaining robots are selected among the other bidders in the ranking. If the necessary number of robots to execute the task is one, the selected leader is the only member of the coalition. In this ranking process, the auctioneer may also select itself either as a CL_i or a CM_i. A bidder robot may be awarded by different auctioneers. However in the action selection step, it selects the optimum cost task for it. Finally it sends a message to become a CM_i to only one of these auctioneers. In the current implementation, each robot involves in only one coalition executing one task.

Coalition Maintenance Process

In the proposed framework, coalition reconfiguration is ensured for obtaining optimal results for changing situations. The coalition leader is responsible to broadcast the maximum cost value of the coalition members in each execution step. If a robot out of coalition has a lower cost value than the maximum cost value for the corresponding task and selects this task in action selection step, it sends a *join request* message to the coalition leader. The leader getting *join request* message, directly adds the robot to the coalition. If the coalition leader detects that the size of the coalition is more than required, it can *release* redundant number of coalition members having the maximum cost value. A releasing and locking mechanism is added to prevent the coalition members leave the coalition until a new more suitable robot joined to the coalition. If a robot gets *released* message, it can select another more suitable task after then. When the coalition leader considers the size of the current coalition, it also checks the failures. Since each robot in the coalition broadcasts *under execution* and updated cost messages, their failure can be detected by the coalition leader. The failed robots are also released if there is enough number of members. If there is not enough number of members to execute the task for a period of time, the coalition leader terminates the execution of the task.

Experimental Design

The proposed framework is tested for collective building construction domain. In the designed experiment, there are different types of objects. The overall mission for the robots contains tasks of finding the necessary objects and making the construction while obeying the specified ordering restrictions. Nine robots with different capabilities are used in the experiments. The pushing requirements of these objects are also different. This mission both contains dependent tasks and requires

cooperative work of the robots. In this domain, experiments are conducted in terms of time to complete the overall mission and active execution time of the robots. The performance is measured against message losses and robot failures. The conducted experiments are run on a simulator simulating the message exchange and execution of the missions. The results are from 100 independent runs with random seeds.

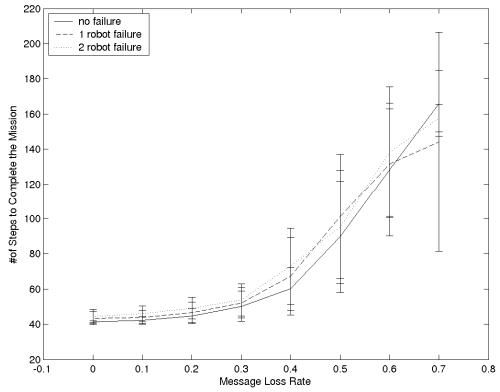


Figure 3. Number of Steps to Complete the Mission Analysis

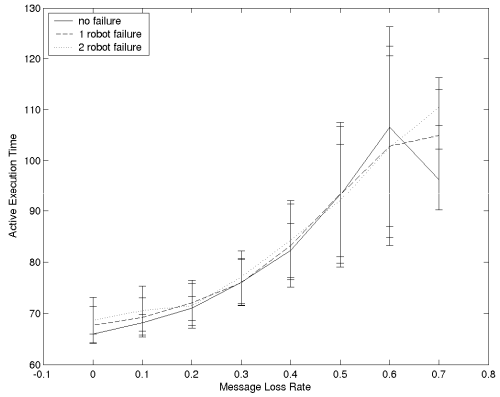


Figure 4. Total Active Execution Time Analysis

Experimental Results

Experiments illustrate that even for 50% message loss rate the robots can complete the mission for the given time period. Since the synchronization is highly required for some of the tasks, the message losses cause the synchronization be lost and also the auction negotiation steps not be completed reliably. If there is enough number of robots having required capabilities after failures, the system can also recover itself and complete the mission with an additional cost of recovery against robot failures. The mission completion time increases for increasing message loss rates logarithmically as in Figure 3. It can also be seen that the framework can easily handle robot failures. In Figure 4, the total execution time analysis can be seen. Combined experiments for robot failures and message losses illustrate that although there is a decrease in performance for the failure cases, the system can recover itself to a great extent while always generating valid plans.

Conclusion and Discussion

The proposed framework is a generic framework for multi robot teams. The generated plans are always valid by means of the selected task representation. The recovery solutions provided by *precaution* routines for different kind of failures ensure the approach is complete. In this framework, close to optimal solutions are generated with available resources at hand. Experiments to validate the approach were conducted in a construction domain.

Analysis of the effects of the cost function selection and proofs of optimality for different domains are part of the current research ongoing. Appropriate cost functions should be applied to obtain optimal results for different domains. This cost function may be defined based on some constraints. Currently the performance is being tested for different domains with different cost functions. One of the possible domains is NP-hard multi robot exploration problem also known as MTSP (Multi TSP). Performance of the proposed framework as a distributed approach for allocating targets to multi robots is being tested for this problem.

The framework is planned to handle online tasks and deadline constraints on the tasks. After observing performance results in simulations for different domains, it is expected to propose a general framework for a multi robot team capable of executing complex missions containing inter-dependent tasks requiring heterogeneity and coordination under uncertainties on dynamic environments.

The final evaluations are going to be implemented on real robots for a complex mission having tasks with ordering constraints.

References

Lemarie T., Alami R. and Lacroix S. 2004. A Distributed Task Allocation Scheme in Multi-UAV Context. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).

Kalra N., Ferguson D. and Stentz A. 2005. Hoplites: A market-based framework for planned tight coordination in multi-robot teams. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).

Sariel S. and Balch T. 2005. Plan B: Robust Multi-Robot Coordination in Noisy and Dangerous Environments. Submitted to the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

Zlot R. and Stentz A. 2005. Complex Task Allocation for Multiple Robots. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).

Bryan H. and Lesser V. 2004. A Survey of Multi-Agent Organizational Paradigms. UMass Computer Science Technical Report. 04-45.

Managing Dynamic Disjunctive Temporal Problems

Peter J. Schwartz

Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI, USA
pschwart@eecs.umich.edu

Introduction

Temporal reasoning has many applications and has been studied in the Artificial Intelligence community for many years (Allen 1983, Dechter, Meiri, and Pearl 1991). A number of different formalisms for temporal reasoning problems exist; one particularly expressive formalism is the Disjunctive Temporal Problem (DTP) (Stergiou and Koubarakis 2000). While DTPs allow one to represent a wide variety of temporal constraints, this expressive power comes at a computational cost. Finding a solution to a DTP is an NP-complete problem; hence significant prior research has been focused on finding efficient heuristic approaches to DTP solving (Stergiou and Koubarakis 2000, Tsamardinos and Pollack 2003, Armando et al. 2004). This prior work has focused on solving a single, static DTP. For many applications, however, it may be necessary to solve a sequence of DTPs in which each problem is very similar to the one before.

For example, the Autominder system (Pollack et al. 2003) is a schedule maintenance and execution monitoring system that issues reminders to keep a user on schedule. Autominder represents a user's schedule as a DTP, and each time the DTP changes, Autominder invokes Epilitis (Tsamardinos and Pollack 2003) to check the DTP for inconsistencies. A user's schedule is modified over time rather than thrown away and completely rewritten, so each DTP is similar to the one before. Epilitis, however, is a static DTP solver, so it does not exploit problem similarity.

The problem encountered by Autominder is that of a dynamic DTP—that is, a sequence of DTPs in which each one is a modification of the one before. Other types of systems may encounter dynamic DTPs as well, including temporal planning systems, mixed-initiative planning and scheduling systems, and DTP optimization systems. In my work, I will explore the problem of improving efficiency and stability (the extent to which one solution is similar to the next) while solving dynamic DTPs.

The Disjunctive Temporal Problem

A DTP is a CSP in which each variable represents a time point, the domain of each variable is the set of all real numbers, and the constraints are disjunctions of difference constraints. Each difference constraint has the form $x - y \leq b$, and is interpreted as "x follows y by no more than b units of time." A disjunctive temporal constraint is satisfied if the time points are assigned times such that at least one of the constraint's disjuncts (i.e., difference constraints) is satisfied. A solution to a DTP is an assignment of times to all time points such that each disjunctive temporal constraint is satisfied.

There are two predominant methods for solving a static DTP. The first is to convert the DTP into a Satisfiability (SAT) problem. This method is used in TSAT++ (Armando et al. 2004), which is currently the fastest DTP solver. The second is to convert the DTP into a finite-domain CSP, called a *meta-CSP*. This method is used in Epilitis, which was the fastest when it was developed several years ago (Tsamardinos and Pollack 2003). Very little research has considered dynamic SAT problems (Hoos and O'Neill 2000), whereas much research has already explored dynamic CSPs (van Hentenryck and Provost 1991, Verfaillie and Schiex 1994a, Verfaillie and Schiex 1994b). For this reason, most of my work will build on the CSP method for solving DTPs, even though the SAT method is currently faster.

We can transform any given DTP into a finite-domain meta-CSP. In the meta-CSP, each variable represents a disjunctive temporal constraint of the DTP, and the domain of each variable is the set of difference constraints in the disjunction. The meta-CSP constraints are implicit—any combination of difference constraints that is chosen as an assignment must be satisfiable. After this transformation, the DTP can be solved with any number of standard CSP techniques.

Epilitis, currently the fastest DTP solver that uses this meta-CSP transformation, uses five different techniques to improve efficiency. Three of these techniques—forward checking, conflict-directed backjumping, and nogood recording—come from the CSP literature. The other two techniques—removal of subsumed variables and semantic

branching—are made possible by the fact that the temporal constraints are linear inequalities. Even though these techniques greatly improve the efficiency of DTP-solving, Epilitis is only designed to solve static problems, not dynamic ones.

The Dynamic DTP

My research will address the problem of the dynamic DTP. A dynamic DTP is a sequence of static DTPs in which each DTP is a modification of the one before. We can represent a dynamic DTP as a tuple $<P_0, C>$, where P_0 is the initial DTP in the sequence, and $C = \{c_1, c_2, \ldots, c_n\}$ is a sequence of changes that describes how the DTP is modified. The system solving a dynamic DTP is initially given only P_0, followed by each change of C once it has solved the previous problem in the sequence. A solution to a dynamic DTP is a sequence $S = \{s_0, s_1, \ldots, s_n\}$, where s_i is the solution to DTP P_i, and P_i is created by applying change c_i to P_{i-1}.

The types changes that are possible in a dynamic DTP can be grouped into restrictions and relaxations. Restrictions can only remove assignments from the solution set of a DTP, and relaxations can only add assignments. The three types of restrictions are (1) tightening the bound b of one of the difference constraints $x - y \leq b$, (2) removing a difference constraint from a disjunctive temporal constraint (i.e., removing a value from a variable's domain in the meta-CSP), and (3) adding a new disjunctive temporal constraint (i.e., adding a variable in the meta-CSP). The relaxations are the opposites of these restrictions. The three types of relaxations are (1) loosening the bound of one of the difference constraints, (2) adding a difference constraint to a disjunctive temporal constraint, and (3) removing a disjunctive temporal constraint. I discuss some generalizations of this definition later in the section on Practical Extensions.

Related Work

As stated earlier, the majority of previous work that is related to dynamic DTPs comes from the literature on dynamic CSPs. A dynamic CSP (Dechter and Dechter 1988) is a sequence of CSPs in which each is a modification of the one before. Dechter and Dechter point out that changes to the variables or domains of a dynamic CSP can be modeled as changes in the constraints, so the majority of dynamic CSP research has assumed that only the constraints can change. The constraints of a meta-CSP of a DTP are implicit, so modeling variable and domain changes as constraint changes in a meta-CSP is not an option. Although the following dynamic CSP techniques can, for the most part, be applied directly to the meta-CSPs of a dynamic DTP, it will be interesting to test their effectiveness when the changes can also affect variables and domains.

Nogood recording (NGR) has been shown to be an effective technique for improving both static and dynamic CSPs (Schiex and Verfaillie 1993). Intuitively, a nogood is a partial assignment of variables that cannot be extended to a complete solution. As a CSP solver searches for a solution, it records a nogood each time it finds a dead end. As search continues, the CSP solver checks each partial assignment against each nogood it has recorded. If it finds a nogood that matches, then that partial assignment can be pruned immediately. In dynamic problems, the nogoods recorded while solving one problem in the sequence can be applied to later problems, allowing for earlier pruning and improved efficiency. Since NGR is already incorporated into Epilitis, it is straightforward to store the nogoods that are recorded while solving one DTP and apply then while solving the next.

Oracles have also been applied to dynamic finite-domain CSPs (van Hentenryck and Provost 1991). During CSP backtracking search, all partial assignments that were tested before the first solution was found must have been pruned. An oracle records the path to the first solution so the next search can try to repeat it, bypassing the pruned portion of the search space. Epilitis is, at its root, a meta-CSP backtracking search algorithm, so it is a simple task to record an oracle after solving one DTP and then use it to select values and variables while solving the next.

Dynamic backtracking (DBT) is an extension of conflict-directed backjumping (CBJ). With CBJ, a back-tracking algorithm maintains enough information that, when a dead end is encountered, it can backtrack through a series of variable assignments until it unassigns a variable that contributed the failure. DBT is similar, except that it unassigns *only* the most recent variable that contributed to the failure, without affecting the assignments that were made later. Verfaillie and Schiex (1994a) showed that DBT can improve both efficiency and stability when solving dynamic CSPs, and one would expect the same to be true when DBT is applied to the meta-CSPs of a dynamic DTP.

The local changes algorithm of Verfaillie & Schiex (1994b) strives to reuse as much of the solution to the previous CSP as possible. If the problem has changed such that the previous solution cannot be reused in its entirety, the local changes algorithm isolates variables involved in the inconsistency and reassigns them until a solution is found. This type of search is a complete departure from the backtracking search of other DTP solvers, so combining this technique with others will be non-trivial.

Enhancements

I have also developed several new techniques of my own that apply to dynamic DTPs, but have not appeared in any previous work.

Besides the techniques to enhance search that were described in the previous section, it is sometimes possible to avoid search entirely. Schiex and Verfaillie (1993) point out that if a consistent problem is relaxed, it is still

consistent, and (conversely) if an inconsistent problem is restricted, it is still inconsistent. Verfaillie and Schiex (1994a) extend this with the notion of an *inconsistency explanation* (also called a *justification*), which is a subset of constraints that are unsatisfiable. Given an inconsistent problem P and its justification J, Verfaillie and Schiex point out that, if the P is relaxed, it can only be consistent if the relaxation affected one of the constraints of J.

I have built on these ideas with the introduction of *justification testing* (JT). If we are given an inconsistent problem P and a justification J for its failure, then J (which is a subset of the constraints of P) and the variables involved in J (which is a subset of the variables of P) define a sub-problem, P', which is also inconsistent. If one of the constraints c in J is then relaxed, the resulting problem can be consistent iff P' (with the relaxed version of c) is consistent. In this situation, JT first attempts to solve the relaxed version of P', which is smaller and easier to solve than the relaxed form of P. It is only necessary to perform a search on the complete relaxed problem if the relaxed sub-problem is found to be consistent. JT can improve efficiency when it finds that the relaxed sub-problem is still inconsistent, but this extra check is a waste of time the sub-problem is consistent.

Even though the sub-problem that JT tests is smaller than the original problem, in the case of CSPs and DTPs, testing its consistency is still NP-complete. One way to speed this up is to apply nogoods from the previous problem. Nogoods are generally recorded as a pair $\langle A, J \rangle$, where A is a partial assignment and J is a justification (A cannot be extended to a solution because of the constraints in J). If nogoods were recorded while searching for a solution to the previous (inconsistent) problem, then any nogood whose justification was not affected by the relaxation can be applied to the next problem. Furthermore, of the set of unaffected nogoods, any nogood whose justification is a subset of the inconsistency explanation can be applied while testing the consistency of the sub-problem with JT. If the sub-problem is found to be consistent, and the complete relaxed problem must then be searched, any additional nogoods that were recorded while searching the sub-problem can then be applied when searching the complete problem.

Schiex and Verfaillie (1993) also point out that if an inconsistent CSP is relaxed by removing a constraint c, then $\neg c$ is an induced constraint of the relaxed problem. In a DTP, the negation of a disjunctive temporal constraint $(x - y \leq b) \vee (w - z \leq a)$ is $(y - x \leq -b - \varepsilon) \wedge (z - w \leq -a - \varepsilon)$, where ε is a small positive number (this is the basis of semantic branching in Epilitis). Hence, when an inconsistent DTP is relaxed by removing a disjunctive temporal constraint c, $\neg c$ is an induced constraint of the relaxed problem and can be added to it before search begins. Likewise, when a disjunctive constraint c is relaxed by adding or loosening one of its difference constraints d, the relaxed problem induces the negation of every difference constraint of c other than d, so these negations can all be added to the relaxed problem prior to search. These additional constraints will improve efficiency by pruning values from the domains of other variables.

The idea of semantic branching can also be used to enhance nogood recording. In Epilitis, when a value v (representing a difference constraint) in the domain of a variable x (representing a disjunctive constraint) is tested and fails, the constraint $\neg v$ is added at that point in the search tree to prune values from the working domains of other variables. This process is called *semantic branching*. Nogood recording can prune values without testing them, so if $x \leftarrow v$ would cause the current partial assignment to match a nogood, it would be pruned, but no semantic branch would be added. Adding the semantic branch $\neg v$ is perfectly valid, though, so enhancing nogood recording with semantic branching should make it a more powerful pruning technique.

When an oracle is used to select variables and values, it can often quickly lead to a solution that is very similar to the previous solution. Occasionally, however, the variables and values the oracle selects are poor choices in the context of the new problem. Specifically, if the new problem is a restriction of the old problem, the tighter constraints can cause more values to be pruned from their domains, leaving only one possible value. If a variable has only one value left in its working domain, most heuristics are designed to select that variable next. If an oracle is selecting variables, it might select one with multiple values in its domain because it is following the variable ordering from the previous search. If the oracle does not lead to a solution, this poor choice of variables will cause unnecessary backtracking and hurt efficiency. This can be overcome by first selecting any variable with only a single value left in its domain, and if no such variable exists, selecting the next variable according to the oracle.

Practical Extensions

The definition of a dynamic DTP given previously works well as a starting point, but many practical applications require a more general formalism. I plan to research two generalizations in particular: sets of changes, and the addition and removal of time points.

The first limitation of the given definition of a dynamic DTP is that the consistency of the DTP must be checked after every atomic change. It is often the case, however, that the schedule will change in such a way as to trigger a *set* of simultaneous changes to the DTP. For example, if an event e is executed exactly t units of time after another event f, a system can represent this by adding two simple temporal constraints to the schedule: $(e - f \leq t)$ and $(f - e \leq -t)$. Both might be added before the new DTP is solved. The definition can be modified easily enough to include a series of sets of changes, rather than a series of atomic changes. The difficulty is in designing algorithms that can efficiently produce stable solutions after a set of changes has occurred.

The second limitation is that the set of time points must remain fixed. In practice, new events will be added to and

removed from the schedule, requiring the addition and removal of time points in the DTP. Although this does not present any particularly interesting theoretical problems, it is a critical part of any practical dynamic DTP solver.

Progress and Research Plans

I have begun my research by comparing the effectiveness of nogood recording and oracles in semi-dynamic DTPs in which the DTPs of the sequence are restricted, but never relaxed. In a semi-dynamic DTP, we can safely assume that any partial assignment that is pruned from one DTP in the sequence can also be pruned from the next DTP. I have performed experiments and found that, in semi-dynamic DTPs, nogood recording improves efficiency and hurts stability, but oracles improve efficiency even more while also improving stability. The performance of the combination of oracles with nogood recording is roughly equivalent to the performance of oracles alone. This work has been accepted to the ICAPS-05 Workshop on Constraint Programming for Planning and Scheduling.

I have also implemented a fully dynamic DTP solver that uses nogood recording, oracles, and justification testing. My experiments with this solver have shown that the performance of different combinations of these techniques depends on the most recent type of change. I used this information to implement a mixed-strategy solver that uses the best combination of techniques for each situation. The mixed-strategy algorithm was 5.8% faster than the second-fastest algorithm. This work has been submitted to the AAAI-05 Technical Program.

The next step in my research will be to compare my dynamic DTP solver, which is based on Epilitis, to one that is based on TSAT++. TSAT++ is a faster static DTP solver than Epilitis, but Epilitis produces a justification when a DTP is found to be inconsistent and TSAT++ does not. I will then compare the effectiveness of the other dynamic CSP techniques, as well as my own enhancements. Finally, I will address sets of simultaneous changes and the addition and removal of time points.

Once I have implemented and tested these various techniques on randomly generated problems, I will compare then using real world data. I have collected data from a small set of users that tracks changes in their daily schedules. I plan to either collect the same data from more users, or obtain additional data from scheduling systems already in use. The combination of randomly generated data and real world data should be adequate to analyze and confidently compare these various techniques when applied to dynamic DTPs.

Acknowledgements

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), through the Dept. of the Interior, NBC, Acquisition Services Division, under Contract No. NBCH-D-03-0010, and on work supported by the Air Force Office of Scientific Research, under Contract No. FA9550-04-1-0043.

References

Allen, J.F. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832-843.

Armando, A., Castellini, C., Giunchiglia, E., and Maratea, M. 2004 A SAT-Based Decision Procedure for the Boolean Combination of Difference Constraints. *Proceedings of SAT-04*.

Dechter, R., and Dechter, A. 1988. Belief Maintenance in Dynamic Constraint Networks. *Proceedings of AAAI-88*, 37-42.

Dechter, R., Meiri, I., and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence*, 49:61-95.

Hoos, H. and O'Neill, K. 2000 Stochastic Local Search Methods for Dynamic SAT—an Initial Investigation. *Technical Report TR-00-01, Computer Science Department, University of British Columbia*.

Pollack, M.E., Brown, L., Colbry, D., McCarthy, C.E., Orosz, C., Peintner, B., Ramakrishnan, S., and Tsamardinos, I. 2003. Autominder: An Intelligent Cognitive Orthotic System for People with Memory Impairment. *Robotics and Autonomous Systems*, 44:273-282.

Schiex, T., and Verfaillie, G. 1993. Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. *International Journal of Artificial Intelligence Tools* 3(2):187-207.

Stergiou, K, and Koubarakis, M. 2000. Backtracking Algorithms for Disjunctions of Temporal Constraints. *Artificial Intelligence*, 120:81-117.

Tsamardinos, I., and Pollack, M. 2003. Efficient Solution Techniques for Disjunctive Temporal Reasoning Problems. *Artificial Intelligence*, 151(1-2):43-90.

van Hentenryck, P.V., and Provost, T.L. 1991. Incremental Search in Constraint Logic Programming. *New Generation Computing*, 9:257-275.

Verfaillie, G., and Schiex, T. 1994a Dynamic Backtracking for Dynamic Constraint Satisfaction Problems. *Proceedings of the ECAI-94 Workshop on Constraints Satisfaction Issues Raised by Practical Applications*.

Verfaillie, G., and Schiex, T. 1994b Solution Reuse in Dynamic Constraint Satisfaction Problems. *Proceedings of AAAI-94*.

Knowledge Acquisition and Engineering for Automated Planning

R. M.Simpson
School of Computing and Engineering
The University of Huddersfield, Huddersfield HD1 3DH, UK
r.m.simpson@hud.ac.uk

Introduction

The proposed research is in the areas of "Knowledge Engineering" and "Visualisation" to support domain specification for "Automated Planning Systems". Despite significant success in developing more efficient automated planning algorithms there is a need to enhance the scope of the technology to bring it to a wider user community. In this research I propose to bring together two strands of research both attempting to address the problem of widening the take up of planning technologies. The first strand of research tries to widen the scope by increasing the expressiveness of the domain modelling languages to provide coverage of a potentially larger number of real world problem areas. The second tries to address the problem by enriching domain modelling languages to support knowledge engineering and to provide rich tool sets to support the creation of domain models. This second strand aims to bring the technology within the potential grasp and use of users from beyond the immediate research community. The knowledge engineering aspects of the project has two broad focuses of interest. The first, focused on the goal of making planning domain specification easier, requires the development of conceptualisations and visual tools to assist in the task. The second, focuses on the validation and re-use of planning domain knowledge and requires that formalisms be either directly or indirectly amenable to reasoning especially with a view to deriving properties of the domain description.

Background

Automatic planning has been a subject studied from the early years of the field of "Artificial Intelligence" with the first significant system being built by Filkes and Nilson (Fikes & Nilsson 1971). Though there has been extensive work in the field since these early days the manner in which planning applications are represented in many modern systems still have their roots in those early systems. Considerable success has been achieved in increasing the expressiveness of domain description languages and in improving the success of planning engines in synthesising plans for large and complex problems. But despite this success, given that planning appears to be an activity undertaken in different forms by all organisations and intelligent organisms, the success falls very far short of the apparent potential for planning systems. There are clearly many reasons why "Planning Technology" has not fulfilled its potential. My work only tries to address a small part of this problem by trying to make the technology as it currently exists more accessible to potential user groups.

Scope of Work

The focus of the this research is on providing support to simplify the task of producing the formal domain specifications necessary for planning. The work builds on the work done for GIPO (Graphical Interface for Planning with Objects) (Simpson *et al.* 2001; 2002; Simpson & McCluskey 2003). In particular it builds on the work on using "Generic Types" (Fox & Long 1997; Simpson *et al.* 2002). Generic types are patterns that re-occur in many problem domain specifications. The primary focus of the new work will be to devise a theoretical foundation for planning patterns and to develop graphical tools to allow the creation of domain specifications to be carried out in a manner achievable by non planning experts. The theoretical foundation for the work will come from building an abstract algebraic domain specification language and link it to a foundation built on the formalisms of "Description Logics". Description logics should provide a basis for a classificatory system for the objects playing the primary roles within the generic types. Using "Description Logics" as the basis for the underlying framework should facilitate integrating the work done as part of this research with wider research in the areas of knowledge management and in particular help link with work done for the "Semantic Web".

At the most basic level the underlying theoretical foundations and the graphical tools should exploit the structure of state machines that can be used to describe the behaviour of the objects within the planning domain that undergo change during the execution of a plan. The new element of this work which builds on previous work trying to exploit an "object centric" view is the conception of "object life histories" and the development of both an algebra and a graphic notation with its accompanying editor to allow domains to be con-

Figure 1: The Barista View of Coffee Making

structed from such conceptions. To briefly illustrate this I present in figure 1 a part of a model of a coffee making system where the states of the *cup* with its contents are shown. This is clearly as presented a linear transition system, which may be what is wanted but if the *cups* are to be re-used "washing up" would have to be incorporated into the model. My belief is that such simple state visualisations helps the domain designer "see" the consequences of modelling decisions.

The initial scope of the work will be limited to devising and creating a system with the built in assumptions of "classical planning". This work will provide the facilities and mechanisms both to create domain specifications from primitives and to build packaged components to be reused in multiple domain models. This ability to create re-usable components should demonstrate a major advance in the manner by which domain specifications can be created. An important constraint on this work is that the resulting algebraic and graphical formalisms should be translatable into current standard planning languages such as PDDL. This constraint guarantees that tools produced to support the new methods of domain specification can be used in conjunction with existing planning engines to create planning system environments.

The range of generic types, or patterns, currently described in the literature needs to be extended to incorporate new patterns. It is to be expected that many complex patterns will be subject domain dependent and not fully general. Within the constructed tools there must accordingly be an API to allow problem centric patterns to be added by system users to libraries of available patterns. The focus of this work will be to provide means for defining new patterns and packaging them up into re-usable components. Work on identifying new patterns will only be done sufficiently to demonstrate the potential for creating complete domain specifications from such re-usable packaged patterns.

The work will also extend the notions of modelling with generic types to richer planning domain modelling languages such as PDDL2.1 and PDDL level 5 (Fox & Long 2001). In addition to the theoretical work necessary it is intended to build prototype tools to enhance the GIPO system and make them available to the larger planning community. A parallel strand of research that will be followed is to investigate the work of the SRI planning group and look at their work on relaxing the commitment to formalism in the construction of "plan authoring" tools. Ideally the work would develop to allow a multi-viewed perspective on planning domains. The "object centric" view is not the only possible view, "agent centric" and "process centric" views should all be available as ways of conceiving of and understanding problem domains.

Evaluation of the work done will be both theoretical and by peer assessment. The work proposes that planning domain specification can be done at a higher conceptual level than has traditionally been the case. This higher level should be closer to the conceptions used by domain experts. The success of the work could be established by showing how a rational reconstruction of the primary domain specifications as used in the ICAPS planning competitions could be recreated in the envisaged higher level form. The theoretical evaluation will concentrate on the adequacy of the developed methods to represent a range of problem domains. Peer assessment will be carried out by submitting the created software to the international competition for knowledge engineering tools in A.I. Planning. This competition is hosted bi-annually by ICAPS (International Conference on Automated Planning and Scheduling). I will present early versions of this new work in that competition. Further peer assessment will be sought by demonstrating the software at conferences and organising workshops to demonstrate the potential of the software. User testing will be done in part by student use of the software as part of the Department's teaching in the area of Artificial Intelligence. One would hope that evaluation could involve deployment in a "real" application but the viability of that depends on the success of developments in other areas of planning as well as that in the knowledge management area.

Current Progress

The "Life History" editor is proposed to be the main editor within GIPO to encapsulate the "object centric" perspective on planning domains. Figure 2 shows a snapshot of the "Life History" editor in use. The editor is being used to edit a version of the "Dock Workers Robots" domain (Ghallab, Nau, & Traverso 2004). The editor enables the creation of object life histories and supports the automatic creation of the textual domain definitions from the developed diagrams. The textual representation includes a translation to PDDL which can be used to run integrated planners from within GIPO.

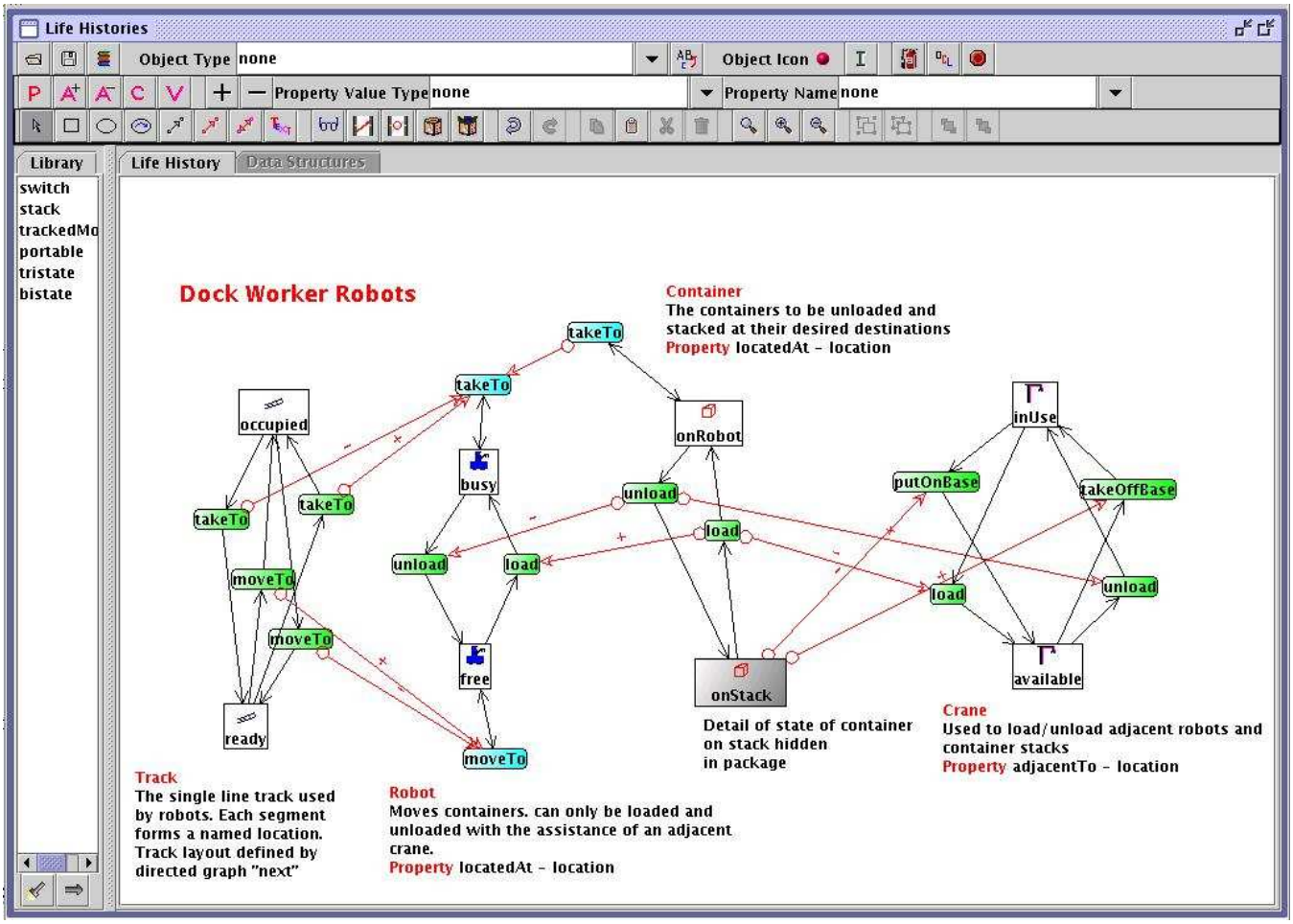


Figure 2: Screen-shot of GIPO editing Dock Workers Domain

The editor supports both the creation of library patterns and the use of patterns to construct new domains. To create a library pattern the user must currently save an entire life history diagram as a library item and supply some *html* formatted text to describe the pattern. This has the consequence that library items must be produced independently of their initial domain of use. In the future we may support defining a fragment of a current domain as a library item. The library viewer is shown in the snapshot in figure 3. The library item being viewed is a stack pattern derived from a version of the classical "Blocks World" domain. Associated with the library facility is the ability to package part of a life history structure to hide the detail. The ability to hide detail is important not only for library items but also in any complex domain where the domain developer needs to control the amount of detail shown at any one point in time. The use of the "stack" pattern is shown in the DWR life history diagram in figure 2. The single state rectangle labelled "onStack" represents the private body of an instance of the "stack" pattern the only visible actions from the stack pattern are the "putOn" and "takeOff" transitions. Within the DWR domain containers are stacked by cranes but otherwise

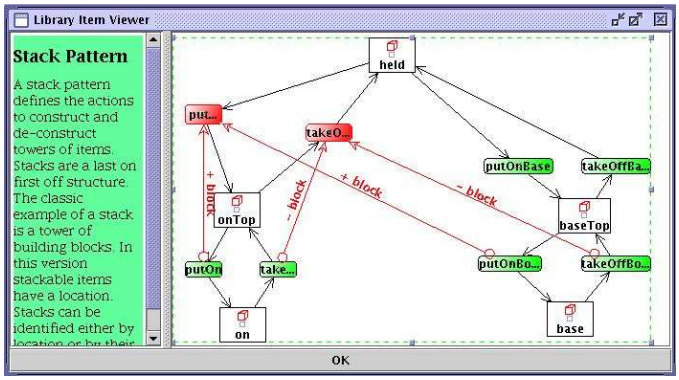


Figure 3: Library Browser View of a Stack

manipulated in a manner identical to that of the blocks in the blocks world. The use of the "stack"structure derived from a blocks world illustrates the potential for re-use within planning domains. It also illustrates the property of encapsulation of re-usable components. In the DWR diagram the com-

plexity of modelling the different states that the container can be in while placed on a stack are hidden from view. The full complexity of the stacking problem is shown in the library viewer but the bulk of that complexity is hidden in the "onStack" node in the DWR diagram. In the DWR domain we can think at the level of placing containers on a stack or removing them but do not need to consider whether or not a particular container is at the bottom of the stack or top of the stack.

The DWR example also gives an indication of the complexity of describing domains in terms of object life histories. It is not just sufficient as illustrated earlier in the coffee cup example to chart the transitions made by object instances from state to state. The domain designer must also define how the transitions of different objects must be coordinated. In the "Life History" editor transitions are marked as coordinated by the arrows connecting the state transitions of the differing object types. The connections between the differing object state machines can indicate that transitions occur together. In effect that the transitions are part of the same action. They can also indicate where instances of one object type must be in a marked state in order to facilitate objects of a different type making a transition between states.

Non-Classical Planning Domains

I have not at this stage tried to fully develop the concepts or software for such language extensions but believe that the "life history model" should be readily adaptable. For example to cope with the formalism of PDDL level 5 as described by Fox and Long et al. (Fox & Long 2001) we need to introduce events, processes and fluent properties of objects. The conceptions of objects changing state all continue to apply but we would need to show how transitions between object states can trigger, update or terminate processes and that processes can trigger events, which are of course just object transitions. A further consequence would be that objects would need more complex numeric properties which are up-dateable when the objects make a state transition. A simplified but possible diagrammatic version of the bath domain is shown in figure 4.

Relation to Previous Work

The proposed work is an extension to work done by the author over the last four years in developing the GIPO software system. The previous work was done in collaboration with others at the Universities of Durham (the Durham team are now located at Strathclyde University) and Salford first as part of the PLANFORM project and following that by the planning team at Huddersfield University. The proposed work will integrate with the earlier work done though that integration will involve re-working substantial elements of the earlier work. Though I am continuing this work I am now doing the work solely as part of my PhD research and not as part of any other funded project.

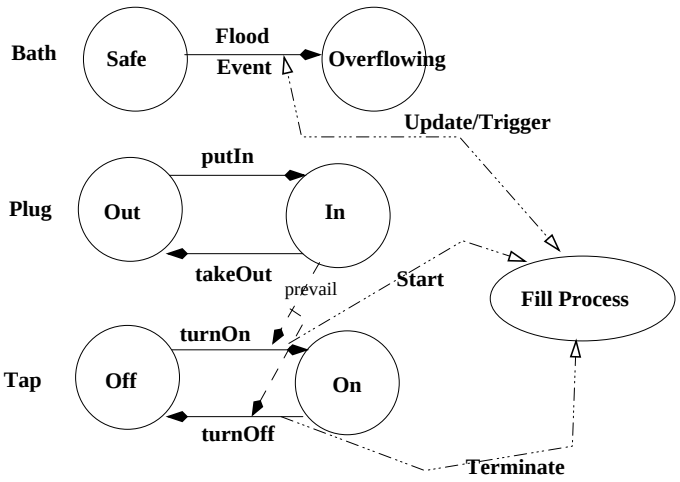


Figure 4: Bath Filling Domain

References

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2.

Fox, M., and Long, D. 1997. The Automatic Inference of State Invariants in TIM. *JAIR* 9:367–421.

Fox, M., and Long, D. 2001. PDDL2.1: An extension to PDDL for expressing temporal planning domains . In *Technical Report, Dept of Computer Science, University of Durham*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning Theory and Practice*. Morgan Kaufmann ISBN 1-55860-856-7.

Simpson, R. M., and McCluskey, T. L. 2003. Plan Authoring with Continuous Effects. In *Proceedings of the 22nd UK Planning and Scheduling Workshop (PLANSIG-2003), Glasgow, Scotland*.

Simpson, R. M.; McCluskey, T. L.; Zhao, W.; Aylett, R. S.; and Doniat, C. 2001. GIPO: An Integrated Graphical Tool to support Knowledge Engineering in AI Planning. In *Proceedings of the 6th European Conference on Planning*.

Simpson, R. M.; McCluskey, T. L.; Fox, M.; and Long, D. 2002. Generic Types as Design Patterns for Planning Domain specifications. In *Proceedings of the AIPS'02 Workshop on Knowledge Engineering Tools and Techniques for AI Planning*.

Modeling Structures for Symbolic Heuristic Decision-theoretic Planning

F. Teichteil-Königsbuch
ONERA-DCSD
2 Avenue Édouard-Belin
31055 Toulouse, FRANCE
florent.teichteil@cert.fr

Introduction

Our research is motivated by an application to exploration or "search and rescue" autonomous aircraft within the *ReSSAC*[1] project at ONERA. We aim at controlling the optimization process, the solution quality and the optimization time, through the enforcement of specific goals to be achieved with maximum priority. An exploration mission

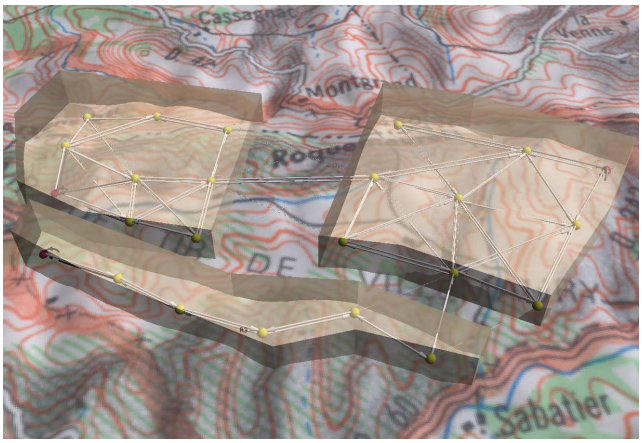


Figure 1: Navigation component of the problem

comprises a problem of navigation (see Figure 1) in a partially known environment on the one hand, and a problem of online information acquisition and replanning on the other hand. Several final, alternative or intermediate goals may be imposed on the agent. Some final goals, such as landing in a safe area, must be achieved in all possible plans. Some goals, such as exploring or searching a region, are the preconditions to be achieved before seeking to realize further goals. Some of the agent's rewards and goals are historic dependent, and there may exist ordering constraints between the goals. Other state variables, such as the agent's energy autonomy level A, are clearly orthogonal to the navigation components, yet not at all decoupled from it: each aircraft motion consumes energy and an insufficient energy level can force the agent to *Abort* its mission and return to its base, or to land on an emergency or security crash base. Last but not least, it may be specified, as in our simple example, that rewards can only be obtained once: so that having achieved goal O_1 in region R_1 nullifies the corresponding reward and thus completely changes the optimal strategy in that region. We want to avoid computing one strategy for each possible combination of goals O_j being achieved or not.

[1] http://www.cert.fr/dcsd/RESSAC

Stochastic planning

Markov Decision Processes (MDPs) (Puterman 1994) are a reference framework for sequential decision making under uncertainty: in order to deal with the uncertain effects of the agent's actions, a *policy* is computed on the state space. It is a function giving, for every enumerated possible state, the action to be performed next. The *optimal policy* maximizes the probability of success, or the mathematical expectation of reward: the *value function* defined on every state. Classical stochastic dynamic programming algorithms are based on an explicitly enumerated and unstructured state space. The size of the state space is an exponential function of the number of features that describe the problem. The state enumeration itself may rapidly become intractable for realistic problems. More generally (Boutilier, Dean, & Hanks 1999) provide an extensive discussion on complexity and modeling issues. (Boutilier, Dearden, & Goldszmidt 2000) show the benefits of factored representations in order to avoid state enumeration, to reason at a higher level of abstraction as in (Dearden & Boutilier 1997) and to take into account non-Markovian aspects of the problem, such as historic dependent rewards or goals as shown in (Bacchus, Boutilier, & Grove 1997). Other approaches, as in (Dean & Lin 1995), introduce a state space hierarchical decomposition of navigation gridworlds into regions. Local policies are then computed in each region and become the macro-actions applicable in the corresponding macro-state of a global abstract and factored MDP. (Teichteil & Fabiani 2005) propose to combine both decomposition and factorization techniques. Other important contributions, such as the SPUDD library (Hoey *et al.* 2000), have improved the efficiency of factored MDP solution algorithms, using decision diagrams, borrowed from the Model Checking community.

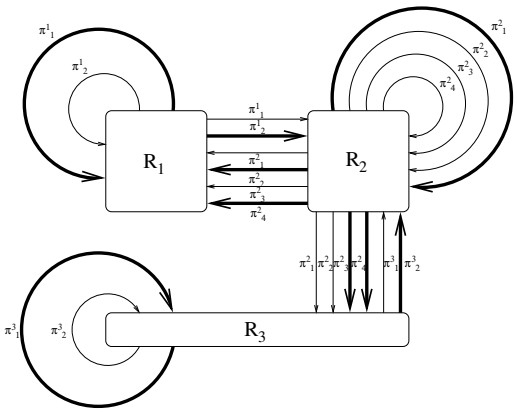


Figure 2: Abstract MDP resulting from the decomposition of the navigation MDP of Figure 1

Compact transitions representation with graphs

In the MDP community, sparse matrices are known to be the natural model of data. MDPs are also often defined as graphs, whose nodes are states and edges are non-zero transitions. For this reason, MDPs transition probabilities can be equivalently represented by graphs or by sparse matrices. Furthermore there exist either direct (Duff, I. S., Erisman, A. M., & Reid, J. K. 1986) or iterative (Saad 2003) algorithms for sparse matrices. For all of these algorithms, the computation time required for a sparse matrix operation is generally proportional to the number of arithmetic operations on non-zero quantities whereas it is proportional to the product of matrices dimensions for a dense matrix operation. Therefore, we should use sparse matrices when the number of transitions is sufficiently smaller than the square of the number of states.

Tests on the computation time and used memory for the solution of grid-like exploration MDPs of varying size with respectively dense and sparse matrices are shown in (Teichteil & Fabiani 2005). The profit in used memory is quite larger than the profit in computer time but the solution of large realistic problems is especially limited by memory constraints rather than by time constraints. On the other hand, both memory and time constraints are important for embedded applications. Therefore, we use a graph-like representation for the navigation component of the state space, as in (Teichteil & Fabiani 2005).

Altough the graph-like definition of MDPs is particularly interesting when the density in non-zero probability transitions is small, the entire enumeration of state space is simply not possible in most realistic problems. We present in the next section a hybrid technique that is well-fitted to stochastic exploration problems over large state spaces.

Decomposition and factorization

The paper (Teichteil & Fabiani 2005) shows the gains that can be achieved in terms of optimization time thanks to factorization: a hierarchical MDP decomposition of the problem results in an abstracted factored MDP as in (Dean & Lin 1995) (see Figure 2). Local policies are computed in each region of the navigation component of the state space. It is done by adapting the Linear Programming approach proposed by (Parr 1998) in order to take into account the fact that rewards can only be obtained once: the local policies are optimized conditionally to the status of the goals in each region. The direct benefits from the decomposition are that, if there are k regions and one goal per region, only $2k$ local policies are optimized, instead of 2^k. Each region is a macro-state of the resulting abstract MDP, the corresponding macro-actions being given by the local policies. The abstract MDP can be represented by Dynamic Bayes Nets (DBNs) as in (Boutilier, Dearden, & Goldszmidt 2000). In DBNs, the transition probabilities and rewards are represented by *Conditional Probability Tables*, i.e. one large matrice for every post-action variable. However, these matrices are sparse in most problems and can generally be encoded using more

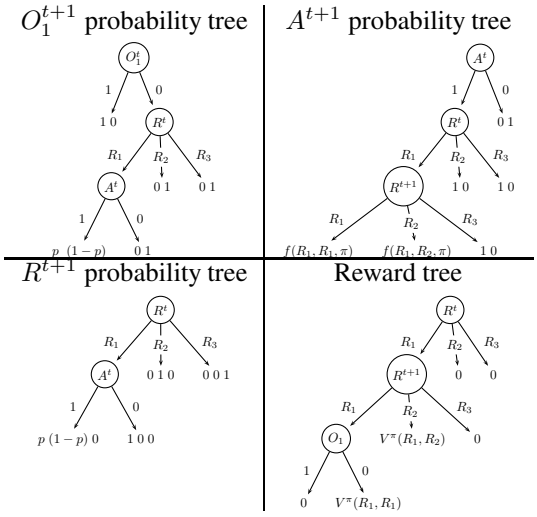


Figure 3: O_1^{t+1}, A^{t+1} and R^{t+1} probability trees, and reward tree (from R_1)

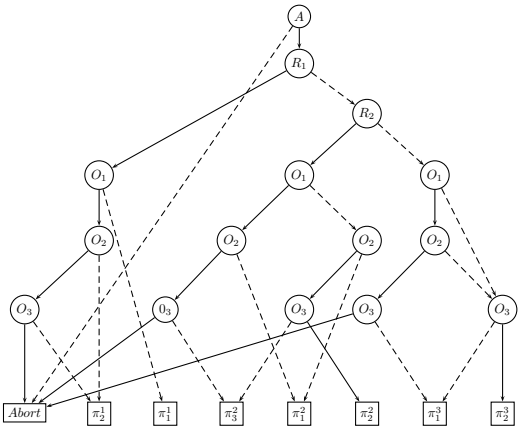


Figure 4: Optimal policy ADD of the abstract factored MDP of Figure 1

efficient structures. Figure 3 shows instances of "decision trees" as in (Boutilier, Dearden, & Goldszmidt 2000), which represent instances of transition probability trees (vectors at the leaves) and transition reward trees for the macro-actions of our instance of abstract MDP. For simplicity, we assume a binary level of energy autonomy in the example. The presented trees say that the reward of goal O_1 can be obtained from region R_1 at time t by staying in region R_1 until time $t+1$ and provided that O_1 has not already been reached before; this can only be achieved with non-zero probability if the autonomy level is 1 at time t. We decided to use "algebraic decision diagrams" (ADDs) as in (Hoey *et al.* 2000) which are more efficient. ADDs offer the additional advantage that nodes of a same value are merged, thus leading to a graph instead of a tree, as shown in Figure 4.

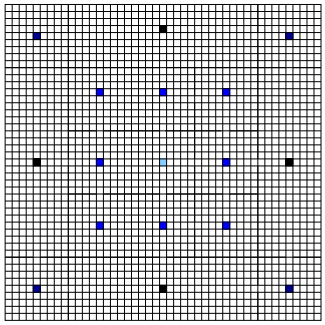


Figure 5: Concentric problems: 17 goals, 1 energy autonomy level, 17 regions

Symbolic heuristic policy iteration

Function HSDP($\mathcal{S}, \mathcal{A}, \mathcal{I}, \mathcal{G}$: **BDD** $\mathcal{T}$,$\mathcal{R}$: **ADD**) : $\mathcal{S}_k, \Pi_k, V_k$
  $\mathcal{H}, \mathcal{S}_k$: **BDD** V_k : **ADD** Π_k : **list**<**BDD**>
  $\mathcal{H} \leftarrow Reachable(\mathcal{I}, \mathcal{A}, \mathcal{G})$
  $(\Pi_0, P(\mathcal{I}) \leftarrow ShortestStochasticPath(\mathcal{H}, \mathcal{I}, \mathcal{G}, \mathcal{T}_{|\mathcal{H}})$
  $\mathcal{S}_0 \leftarrow FilterStates(S, P(\mathcal{I}), \epsilon)$
  $k \leftarrow 0$
  Repeat
    Switch
      SFDP : $\mathcal{S}_{k+1} \leftarrow Reachable(\mathcal{I}, \Pi_k, \mathcal{G})$
      *LAO** : $\mathcal{S}_{k+1} \leftarrow Reachable(\mathcal{S}_k, \Pi_k,$ 1 step lookahead)
    end Switch
    $(V, \Pi)_{k+1} \leftarrow DynamicProgramming\left(\mathcal{S}_{k+1}, (\mathcal{T}, \mathcal{R})_{|\mathcal{S}_{k+1}}\right)$
    $k \leftarrow k+1$
  until $(\|V_{k+1} - V_k\| < \epsilon$ over $\mathcal{S}_k)$
  return $(\mathcal{S}_k, V_k, \Pi_k)$
End

Algorithm 1: Heuristic Search Dynamic Programming

A key idea is to apply dynamic programming only on a subset of the state space which is computed through iterative reachability analysis. Recently, heuristic search schemes have been proposed such as *LAO** (Hansen & Shlomo 2001), or *LRTDP* (Bonet & Geffner 2003). We propose a *Symbolic Focused Dynamic Programming (SFDP)* heuristic search algorithm (see Algorithm 1), that is an original contribution to our knowledge. *SFDP* conforms, like *LAO**, with a two-phased scheme of *planning space expansion*-and-*optimization*. (Teichteil & Fabiani 2005) presents Algorithm 1 and experimentations on it, that show the benefits of our hybrid approach combining decomposition and factorization techniques in large state spaces. Moreover, *SFDP* finds quite quickly (sub-optimal) solutions, whereas *LAO** cannot give any answer after more than one hour. Nevertheless, *SFDP* appears as much more sensitive to goal constraints than *LAO**. Figure 5 illustrates the family of problems that we have tested, with different starting and ending points.

Faster solution, weaker value

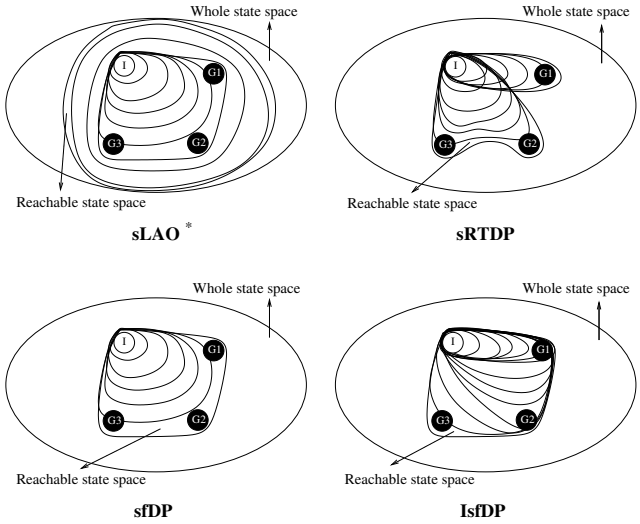


Figure 6: Reachable state spaces of sLAO*, sRTDP, sfDP and IsfDP

Function IsfDP(L_b : **list**<**BDD**> $\mathcal{S}, \mathcal{A}, \mathcal{I}$: **BDD** $\mathcal{T}$, $\mathcal{R}$: **ADD**) : $\mathcal{S}_k, \Pi_k, V_k$
  $\mathcal{S}_0 \leftarrow \mathcal{I}$
  $k \leftarrow 1$
  While (L_{sg} non empty) **do**
    $\mathcal{I}_k \longleftarrow \mathcal{S}_{k-1}$
    $\mathcal{G}_k \longleftarrow$ head of L_{sg}
    Switch
      IsfDP : $(\mathcal{S}, \Pi, V)_k \leftarrow sfDP(\mathcal{S}, \mathcal{A}, \mathcal{I}_k, \mathcal{G}_k, \mathcal{T}, \mathcal{R})$
      IsfLAO : $(\mathcal{S}, \Pi, V)_k \leftarrow sfLAO(\mathcal{S}, \mathcal{A}, \mathcal{I}_k, \mathcal{G}_k, \mathcal{T}, \mathcal{R})$
    end Switch
    remove head of L_{sg}
    $k \leftarrow k+1$
  done
  return $(\mathcal{S}_k, \Pi_k, V_k)$
End

Algorithm 2: $IsfDP$ algorithm for on-line planning

Following the previous ideas, another version of the *focused dynamic programming* scheme was developed by weakening the stopping condition of the $Reachable$ function, that holds now as soon as **at least one state is reached where the required goal conditions are achieved**. This new stopping condition is obviously weaker and the new algorithms, called *sfDP* and *sfLAO* still follow the scheme presented in Algorithm 1. Nevertheless, the *sfDP* and *sfLAO* algorithms are obviously not optimal but they solve

problems with more than 8×10^5 enumerated states, that are intractable for *SPUDD* (see Figure 7). Moreover, the solution quality grows with the number of goal conditions imposed up to the optimal solution (see Figure 8). Thus, we developed Incremental versions of these algorithms, respectively $IsfDP$ and $IsfLAO$, that iteratively call $sfDP$ or $sfLAO$ in order to incrementally – and possibly on-line – improve the solution (see Algorithm 2 and Figure 6). Figure 8 shows on a small example that $IsfDP$ outperforms $IsfLAO$ but remains sub-optimal. On that example, *SPUDD* is still able to tackle the problem, and *sfDP* finds an optimal solution.

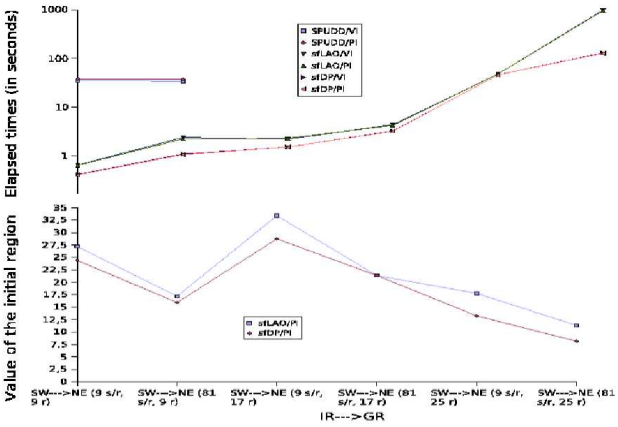


Figure 7: sfDP and sfLAO compared with SPUDD on different problems of increasing sizes (values are not comparable between problems)

Conclusion

We have proposed a hybrid MDP modelling framework, where some state components correspond to an explicitely enumerated subspace that can be defined as a decomposable graph and then integrated in an abstract factored MDP. We have proposed an original algorithm *SFDP* for which the optimization time and the solution quality can be controlled through the definition of planning goals constraints. Such goals could be adapted off-line or on-line, thus opening interesting directions for future work on decision under time and ressources constraints. This is particularly interesting in our application perspectives on autonomous aircraft. We have developed *sfDP*, an even faster, but weaker, version of *SFDP*. An incremental version of *sfDP*, named *IsfDP*, was developed for on-line planning: it couples *sfDP* with a higher level optimization controller in order to reuse the sub-optimal solution obtained with *sfDP* in a higher level optimization process. We will now develop an optimal version of *IsfDP*, that could switch to *LAO* for the last iteration.

References

Bacchus, F.; Boutilier, C.; and Grove, A. 1997. Structured solution methods for non-markovian decision processes. In *Proceedings 14th AAAI*, 112–117.

Bonet, B., and Geffner, H. 2003. Labeled rtdp: Improving the convergence of real-time dynamic programming. In *Proceedings 13th ICAPS 2003*, 12–21.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *J.A.I.R.* 11:1–94.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 2000. Stochastic dynamic programming with factored representations. *Artificial Intelligence* 121(1-2):49–107.

Dean, T., and Lin, S.-H. 1995. Decomposition techniques for planning in stochastic domains. In *Proceedings 14th IJCAI*, 1121–1129.

Dearden, R., and Boutilier, C. 1997. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence* 89:219–283.

Duff, I. S.; Erisman, A. M.; and Reid, J. K. 1986. *Direct Methods for Sparse Matrices*. Oxford: Clarendon Press.

Hansen, E. A., and Shlomo, Z. 2001. Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129:35–62.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 2000. Optimal and approximate stochastic planning using decision diagrams. Technical Report TR-2000-05, University of British Columbia.

Parr, R. 1998. Flexible decomposition algorithms for weakly coupled markov decision problems. In *Proceedings 14th UAI*, 422–430.

Puterman, M. L. 1994. *Markov Decision Processes*. John Wiley & Sons, INC.

Saad, Y. 2003. *Iterative Methods for Sparse Linear Systems*. Society of Industrial and Applied Mathematics, second edition.

Teichteil, F., and Fabiani, P. 2005. Influence of modeling structure in probabilistic sequential decision problems. *RAIRO Operations Research* to appear.

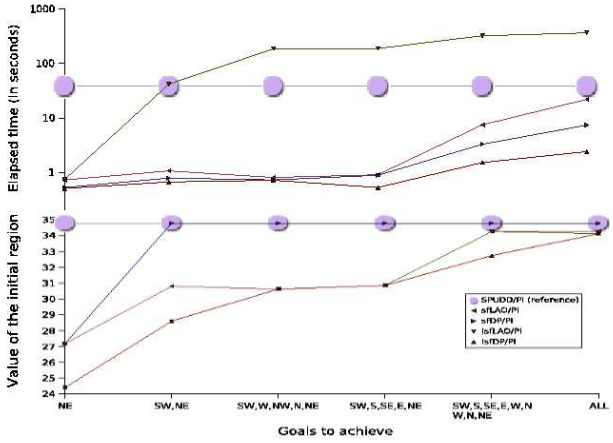


Figure 8: (I)sfDP and (I)sfLAO compared with SPUDD while increasing the number of planning goals to achieve

Integer Programming Approaches for AI Planning

Menkes van den Briel
Department of Industrial Engineering
Arizona State University
Tempe AZ, 85287-8809
menkes@asu.edu

Abstract

The focus of my research is on the formulation and analysis of mathematical programming techniques in AI planning. This extended abstract provides a brief overview of the paper (Van den Briel, Vossen, & Kambhampati 2005) that will be presented at ICAPS 2005. In addition, it provides an overview of some of my future research plans.

Introduction

The conventional wisdom in the AI planning community is that integer programming-based (IP) systems cannot compete with satisfiability-based (SAT) and constraint satisfaction-based planners. We challenge this current perception of IP-based systems by presenting novel formulations that (1) use multi-valued state variables that are represented by networks, and that (2) progressively generalize the notion of parallelism. The resulting IP encodings are solved within a branch-and-cut framework and outperform the most efficient SAT-based planner.

The use of integer programming (IP) to solve AI planning problems has an intuitive appeal, given its remarkable successes in similar problem domains such as scheduling, production planning and routing (Johnson, Nemhauser, & Savelsbergh 2000). In addition, one potential advantage is that IP techniques can provide a natural way to incorporate several important aspects of real-world planning problems, including numeric constraints and objective functions.

Nevertheless, the application of IP techniques to AI planning has only received limited attention. The first appears to have been Bylander (1997), who proposed a linear programming (LP) formulation that could be used as a heuristic in partial order planning. Vossen *et al.* (1999) discuss the importance of developing "strong" IP formulations, by comparing two formulations for classical planning. While a straightforward translation of sat-based encodings yields mediocre results, a less intuitive formulation based on the representation of state transitions results in considerable performance improvements. Dimopoulos (2001) discusses a number of ideas that further improve this IP formulation. A somewhat different approach that relies on domain-specific knowledge is proposed by Bockmayr and Dimopoulos (1998; 1999). The use of LP and IP has also been explored for non-classical planning. Dimopoulos and Gerevini (2002) describe an IP formulation for temporal planning and Wolfman and Weld (1999) use LP formulations in combination with a satisfiability-based planner to solve resource planning problems. Kautz and Walser (1999) use IP formulations for resource planning problems that incorporate action costs and complex objectives.

So far, none of these IP approaches have been able to produce a planner whose performance compares with today's most advanced satisfiability and constraint satisfaction-based planners. In this research we challenge this current perception by presenting novel IP formulations that outperform the best SAT-based planners. The formulations we propose rely on two key innovations:

1. We model changes in individual state variables during planning as flows in an appropriately defined network. As a consequence, the resulting IP formulations can be interpreted as a network flow problem with additional side constraints. While this idea can be used with any state variable representation, it is particularly synergistic with multi-valued state variables. We thus adapt existing methods to automatically convert PDDL domain encodings into multi-valued domain encodings.

2. One difficulty in scaling IP encodings has been the dependency between the size of the encoding and the length of the solution plan. This dependency often leads to encodings that are very large. To alleviate this dependency, we separate causal considerations from the action sequencing considerations to generalize the common notion of parallelism based on planning graphs (Srivastava, Kambhampati, & Do 2001). This concept suggest that it should be possible to arrange parallel actions in any order with exactly the same outcome (Blum & Furst 1995). By relaxing this condition, we develop new concepts of parallelism that are similar yet strictly more general and powerful than the relaxation proposed by Cayrol *et al.*

(2001) for the LCGP planning system.

A naive encoding of this decoupling will not be effective as the sequencing phase will add exponentially many ordering constraints. Instead, we propose and implement a so-called *Branch-and-Cut* framework, in which certain constraints are dynamically generated and added to the formulation only when needed. This approach has been extremely successful for a number of large-scale optimization problems (Caprara & Fischetti 1997). We show that the performance of the resulting planning system is superior to Satplan04(Siege) (Kautz 2004), which is currently the most efficient SAT-based approach to planning. This is a significant result in that it forms the basis for other more advanced IP-based planning systems capable of handling numeric constraints and non-uniform action costs.

The remainder of this extended abstract is organized as follows. In the next section, the ideas of (1) using multi-valued state variables and (2) progressively generalizing the notion of parallelism are discussed. Due to size restriction, the actual IP formulations are not presented, instead we refer the reader to Van den Briel, Vossen, and Kambhampati (2005). Subsequently, conclusions and a discussion of avenues for future research are described.

IP formulations

The IP formulations use (multi-valued) state variables instead of the (binary-valued) propositional variables that were used in the formulations by Vossen *et al.* (1999). The use of multi-valued state variables is based on the SAS+ planning formalism (Bäckström & Nebel 1995). SAS+ is planning formalism that uses multi-valued state variables instead of propositional atoms, and it uses a *prevail condition* on top of the regular pre- and post-conditions (pre-, add-, and delete-lists). A prevail is a condition imposed by an action that specifies for one or more state variables a specific value that must hold before and during the execution of that action. Another way to look at a prevail is that it implies the persistence of a specific value. To obtain a state variable description from a PDDL description of a planning problem we use the translator that is implemented in the planner Fast (Diagonally) Downward (Helmert & Richter 2004). This translator is a general purpose algorithm that transforms a classical planning problem into a multi-valued state variable description. It provides an efficient grounding that minimizes the state description length and is based on the ideas presented by Edelkamp and Helmert (1999).

In addition, the IP formulations are based on a more general notion of parallelism. We propose a *set of alternative conditions* for parallel actions. The basic idea is to relax the condition that parallel actions can be arranged in any order. Instead we will require a weaker condition, which states that *there exists* a valid ordering of the actions within a time step. More specifically, within a time step a set of actions is feasible if (1) there exists an ordering of the actions such that all preconditions are satisfied, and (2) there is at most one state change in each of the state variables.

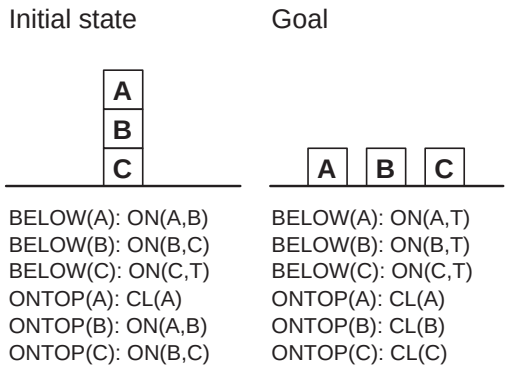


Figure 1: Another Blocksworld instance

To illustrate the basic concept, let us examine the Blocksworld instance given in Figure 1. The obvious solution is to first execute action $MOVE(A, B, T)$ and then $MOVE(B, C, T)$. Clearly, this is not a solution that would be allowed *within a single step* under Graphplan's parallelism, since we cannot execute the actions in an *arbitrary* order (that is, $MOVE(B, C, T)$ cannot be executed unless $MOVE(A, B, T)$ is executed first). Yet, the number of state changes within any state variable is at most one, and while the two actions cannot be arranged in any order with exactly the same outcome, there does exists *some* ordering that is feasible. The key idea behind this example should be clear: while it may not be possible to find a set of actions that can be linearized in any order, there may nevertheless be an ordering of the actions that is viable. The question is, of course, how to incorporate this idea into an IP formulation.

This example illustrates that we are looking for a set of conditions that allow, within each plan step, those sets of actions for which:

- All the actions' preconditions are met,
- There exists an ordering of actions at each plan step that is feasible, and
- Within each state variable, the value is changed at most once.

The incorporation of these two ideas are implemented in IP formulations that yield impressive results on a set of benchmark problems taken from the international planning competitions.

Conclusions and Future Work

Despite the potential flexibility offered by IP encodings for planning, in practice planners based on IP encodings have not been competitive with those based on CSP and SAT encodings. This state of affairs is more a reflection on the type of encodings that have been tried until

now, rather than any inherent shortcomings of IP as a combinatorial substrate for planning. In this research a sequence of novel IP formulations are introduced whose performance scale up and surpass that of state-of-the art SAT-based approaches to planning. The success of our encodings is based on three interleaved ideas: (1) modeling state changes in individual (multi-valued) state variables as flows in an appropriately defined network, (2) generalizing the notion of action parallelism to loosen the dependency between encoding length and solution length, and (3) using a branch and cut framework (rather than a branch and bound one), to allow for incremental addition of constraints during the solving phase.

In the future, I intend to exploit the competitive foundation that this framework provides to explore more complex classes of planning problems that have natural affinity to IP encodings. These include, handling of numeric variables and constraints, and generation of cost sensitive plans (in the context of non-uniform action costs). I would also like to explore the interface between planning and scheduling by coupling IP-based schedulers to our planner (using the same general branch-and-cut framework). In the near term, I plan to (1) improve the engineering of the branch-and-cut framework, (2) strengthen the IP formulations by taking into account mutexes (these will be different from Graphplan mutexes due to the different notion of parallelism), (3) further analyze the impact of more general notions of parallelism, and (4) increase the scale of problems that can be solved using column generation techniques.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS^+ planning. *Computational Intelligence* 11(4):625–655.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1636–1642.

Bockmayr, A., and Dimopoulos, Y. 1998. Mixed integer programming models for planning problems. In *Working notes of the CP-98 Constraint Problem Reformulation Workshop*.

Bockmayr, A., and Dimopoulos, Y. 1999. Integer programs and valid inequalities for planning problems. In *Proceedings of the European Conference on Planning (ECP-99)*, 239–251. Springer-Verlag.

Bylander, T. 1997. A linear programming heuristic for optimal planning. In *AAAI-97/IAAI-97 Proceedings*, 694–699.

Caprara, A., and Fischetti, M. 1997. *Annotated Bibliographies in Combinatorial Optimization*. John Wiley and Sons. chapter Branch and Cut Algorithms, 45–63.

Cayrol, M.; Régnier, P.; and Vidal, V. 2001. Least commitment in graphplan. *Artificial Intelligence* 130(1):85–118.

Dimopoulos, Y., and Gerevini, A. 2002. Temporal planning through mixed integer programming. In *Proceeding of the AIPS Workshop on Planning for Temporal Domains*, 2–8.

Dimopoulos, Y. 2001. Improved integer programming models and heuristic search for ai planning. In *Proceedings of the European Conference on Planning (ECP-01)*, 301–313. Springer-Verlag.

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In *Proceedings of the European Conference on Planning (ECP-99)*, 135–147. Springer-Verlag.

Helmert, M., and Richter, S. 2004. Fast downward: making use of causal dependencies in the problem representation. In *Notes on the International Planning Competition (IPC-2004)*, 41–43.

Johnson, E.; Nemhauser, G.; and Savelsbergh, M. 2000. Progress in linear programming based branch-and-bound algorithms: An exposition. *INFORMS Journal on Computing* 12(?):?–?

Kautz, H., and Walser, J. 1999. State-space planning by integer optimization. In *AAAI-99/IAAI-99 Proceedings*, 526–533.

Kautz, H. 2004. Satplan04: Planning as satisfiability. In *Notes on the International Planning Competition (IPC-2004)*, 44–45.

Srivastava, B.; Kambhampati, S.; and Do, M. 2001. Planning the project management way: Efficient planning by effective integration of causal and resource reasoning in realplan. *Artificial Intelligence* 131(1-2):73–134.

Van den Briel, M.; Vossen, T.; and Kambhampati, S. 2005. Reviving integer programming approaches for ai planning: A branch-and-cut framework. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS-2005)*, (Accepted for publication).

Vossen, T.; Ball, M.; Lotem, A.; and Nau, D. 1999. On the use of integer programming models in ai planning. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 304–309.

Wolfman, S., and Weld, D. 1999. The lpsat engine and its applicationto resource planning. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 310–317.

Search-Based Online Job Scheduling of Parallel Computer Workloads

Sangsuree Vasupongayya

Computer Science Department, Portland State University
P.O. Box 751, Portland, OR 97207-0751, USA
vsang@cs.pdx.edu

Abstract

Scheduling of parallel computer workloads is an interesting online job scheduling problem. In this domain, typically the scheduler has more than one goals, including minimizing average job response time, maximizing system utilization, and achieving certain quality-of-service, fairness, and special priority. It is challenging to design the job scheduler as these objectives often conflict with each other. In this research, we attack the on-line parallel job scheduling problem using combinatorial search techniques. To deal with the often inaccurate estimates of job runtimes and unknown future job arrivals, we also plan to model and predict such information to improve the performance of the scheduler. This paper focuses on the search-based scheduling engine.

In our preliminary experiments, we formulate a complex objective that deals with two common goals. Our results demonstrate that systematic search can outperform priority backfilling methods in achieving the overall objective. Works need to be done on more complex objectives, improving the scheduling engine, and modeling component of the system.

Introduction

Online job scheduling is a challenging problem: in general it is NP-complete, yet it usually requires making scheduling decisions quickly. Further, decisions are often made on the basis of incomplete information. One interesting online job scheduling problem is the scheduling of parallel computer workloads. In this domain, the scheduler must consider objectives like minimizing average job response time, maximizing system utilization, and achieving certain quality-of-service, fairness, and special priority. These objectives may conflict. In this research, we focus on the on-line non-preemptive parallel job scheduling problem which is the most commonly used scheme in parallel computing centers. We apply combinatorial search techniques to the problem.

The solution to an online parallel job scheduling problem is a selection of jobs to be scheduled at the current decision point. In on-line non-preemptive parallel job scheduling, users submit a job and also supply its estimated number of processors, use of memory, and runtime. When resources become available and there are jobs waiting to be selected, a scheduler is called to decide which jobs get the resources. Resources are partitioned, and the selected jobs are executed concurrently on their assigned partitions until they terminate. Selection is difficult even when there are no precedence dependencies between jobs. The current decision has consequences for later decisions: work deferred now must be performed later.

In practice, a complex combination of objectives is often desired. Components of this complex objective may conflict. For example, minimizing response time favors small and short jobs while some longer and larger jobs may suffer. A complex objective makes designing an optimal job scheduler more difficult. Priority scheduling is the most popular method for job scheduling problems with a complex objective. In priority scheduling available jobs are sorted in priority order: the scheduler selects the highest-priority jobs currently executable. The priority function is usually a weighted function of the individual objective components. In practice, selecting a set of weights that will optimize for any given complex objective is hard.

Combinatorial search techniques allow a complex objective to be expressed in declarative format which is easier for human to understand. We also apply modeling techniques to reduce uncertainty of estimates. Modeling techniques enable the scheduler to cope with uncertainty of job arrivals and inaccurate estimated runtime. However, this paper only focuses on the scheduling engine. Modeling techniques enables the scheduler to cope with uncertainty of job arrivals and inaccurate estimated runtime. Furthermore, separating the scheduling engine from the modeling enables each piece to be modified or changed more easily.

The rest of this paper proceeds as follows. First, background is given on our testbed domain: parallel supercomputer non-preemptive job scheduling, and on combinatorial search techniques. Next our proposed solution is described. Some preliminary results are reported. Finally, conclusions are drawn from this research

Parallel Job Scheduling

Job scheduling on parallel computer workloads has been an important research area in the past decade. An early

approach is simple strict pure space partitioning on a priority ordering such as First Come First Served (FCFS). Under this scheme, the jobs are ordered by their priority. While the highest priority job in the queue cannot be started, all remaining jobs are also delayed. This scheme guarantees freedom from starvation and is very simple to implement. However, this strict scheme fails to utilize processors efficiently.

Backfilling algorithm, which allows small jobs in the back of the queue to be scheduled as long as the highest priority job is not delayed, has been shown to improve utilization (e.g. (Skovira et al. 1996), (Mu'alem & Feitelson 2001)). While *Backfilling* improves utilization, it is not sufficient to optimize for a complex objective. Thus, a weighted priority function computed at each decision point is commonly used instead of a static priority ordering (Jackson, Snell & Clement 2001; Talby & Feitelson 1999).

In practice, choosing weight settings to achieve a given complex objective using any of these schedulers is non-trivial. The appropriate weight settings vary widely depending on the workload. Several researches have shown that workload characteristic can affect the performance of scheduling policies (e.g. (Chiang & Vernon 2001a), (Subhlok, Gross & Suzuoka 1996)). Therefore, a scheduler that is aware of changes in workload behavior can benefit from such knowledge. As demonstrated by Talby and Feitelson (2005), a simple change in one of the workload characteristics can identify a policy change for better performance. This work, however, only considered single-objective scheduling.

Workload modeling has been done and compared on various real-world workloads (Chiang & Vernon 2001b; Feitelson 2005). However, there is no generally-accepted best-practice method for modeling parallel job scheduling workloads.

Combinatorial Search

Combinatorial search techniques have been successfully applied to problems in various domains, e.g. (Crawford 1993), (Korf 2004), and (Climer & Zhang 2004). In this work, we apply combinatorial search as a scheduling engine. The scheduling engine is used to evaluate alternative schedules, returning the expected best schedule for the current decision point.

Our initial search techniques include complete methods while local search methods are left for future works. Complete methods search in a space of partial schedules. Starting from an empty schedule, a complete method adds an assignment of a job to the schedule one-by-one until a total schedule is reached. This way an optimal solution is guaranteed to be found if one does exists. However, the search space of a complete method grow exponentially on the number of jobs thus searching all possible total schedules becomes infeasible. To cope with a large search space, a heuristic is introduced to guide the search to the most promising candidate solutions first.

With a limited search time, a simple complete search such as Depth First Search (DFS) can be stuck at a small portion of the search tree when heuristic makes early mistakes. Thus, alternative method such as Limited Discrepancy Search or LDS (Harvey & Ginsberg 1995), which searches in an increasing order of heuristic mistake count, can help avoid the early mistake problem.

To cut down the search time further, Branch-and-Bound (BnB) are used to prune search space and gets to the promising schedules fast. However, BnB requires a mechanism to evaluate a partial schedule and an admissible heuristic (Ginsberg 1993). Other techniques used in this paper to speed up the search include variable and value ordering heuristic (Purdom 1983) and forward checking (Wolfram 1989).

Proposed Solution

Our proposed scheduler consists of two main components: the scheduling engine and the modeling engine as shown in Figure 1. The scheduling engine selects a set of jobs to be scheduled at each decision point. The modeling engine collects information about the system and workload characteristic and performance of the scheduler. The modeling engine then provides useful information to the scheduling engine.

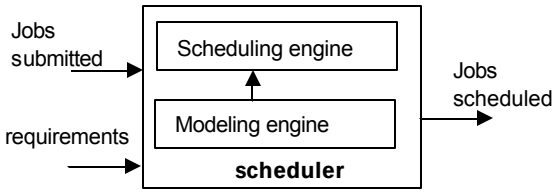


Figure 1: Structure of the proposed scheduler

By employing combinatorial search techniques in the scheduling engine, we can replace the setting of priority weights with the direct description of a complex objective. Thus, the administrator of the system can directly describe exactly what the goal is and he/she does not have to tune a set of weights in an attempt to achieve this goal. Furthermore, the scheduling engine can be modified and replaced without changing any underlying mechanism.

The modeling part uses domain knowledge to collect and provide useful information to the scheduling engine. This component can be seen as (1) a information provider unit to the scheduling engine, helping to give an accuracy of user runtime estimate and a prediction of future arrivals; (2) a feedback unit to the scheduling engine, helping to improve decision making. Approaches from both queuing theory and machine learning will be explored in this research to build the modeling engine. Both approaches have shown some promising results in the past research (Smith, Taylor, and Foster 1998).

Two simplifying assumptions are made in this research: i) the environment is not distributed: the data and resources

to be used by the job are available locally; and ii) the resource requirements are fixed throughout the lifetime of the job. Even though our current research focuses on the pure space slicing approach for non-preemptive scheduling, a slight modification to the objective that take into account the effects of resource location should enable our scheduling framework to fit for the distributed environment as well.

Preliminary Results

Our current focus is on exploring search engine design. An event-driven simulator is implemented in JAVA. The simulator is capable of handling both synthetic and real world traces. Several static ordering, *Backfilling* and systematic search methods have been implemented in the simulator, and preliminary experiments have been performed. Preliminary results of experiments with an objective with two conflicting components are shown.

Two components of the objective in this experiment are minimizing maximum wait-time and average bounded slowdown. Slowdown of a job is defined as the ratio of its response time and its duration. However, we use bounded slowdown (BSD) instead of slowdown because slowdown of a very short job can be very large which is not fair to the large jobs. To reduce the effects of very short jobs, we set the lower bound of job durations to be 10 minutes. The BSD of a job is defined as the ratio of its response time and the maximum value between its duration and 10 minutes.

The objective is defined as follows: schedule S_i is better than schedule S_j if it has a smaller total excessive-wait-time. Ties are broken by average BSD. The wait-time bound used was 50 hours. The 50-hour wait-time bound was selected based on the nature of this workload. Number of processors was relaxed for estimating the goodness of each partial schedule in order to perform Branch-n-Bound.

These experimentations were performed on 9-monthly-traces of NCSA-IA64 (see (Chiang and Fu 2005) for more details). The job actual runtime was used but the estimated future arrival was not known. The arrival time of jobs were manipulated to achieve the load level of 0.9 on all months. Limited job duration from July 2003 to November 2003 was 12 hours and 24 hours during the remaining of the year 2004.

We compare our LDSBnB results with FCFS-Backfill and LXF-Backfill. FCFS-Backfill has a potential to preserve the fairness thus it tends to provide a small maximum wait-time since jobs are backfilled in the order of their arrival time. LXF-Backfill has a potential to minimize the average BSD since jobs are backfilled in the order of their BSD.

Figure 2 shows the performance of LDSBnB with FCFS-Backfill and LXF backfill. According to the total-excessive-wait metric, LDSBnB showed improvement over *Backfilling* policy except in February and January. However under the average waittime metric, LDSBnB showed improvement over FCFS-Backfill in both January and February workloads. Even though LDSBnB showed a slightly higher average BSD in July, the improvement in total-excessive-wait metric was dramatic. Since the LDSBnB objective was defined in a hierarchical fashion, the average BSD metric was not considered as long as the total-excessive-wait of the current schedule was no better than that of the best schedule. Thus, the comparable performance of LDSBnB in average BSD is actually an impressive accomplishment.

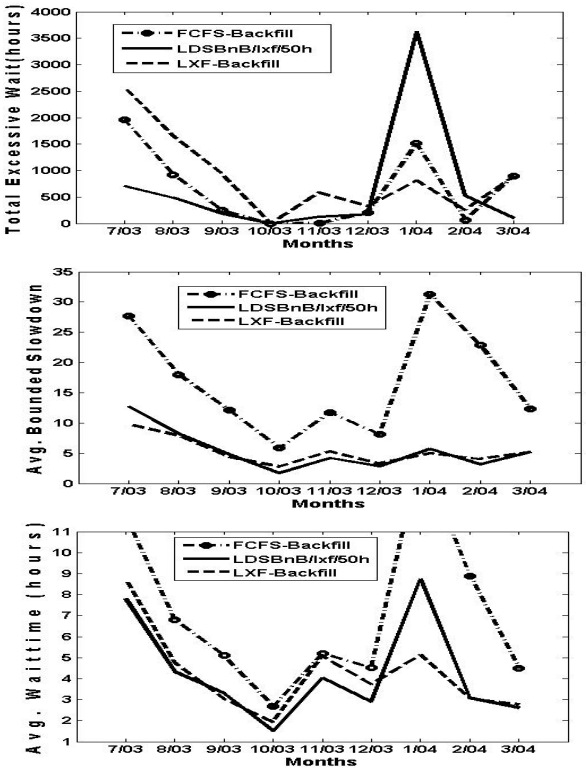


Figure 2: Performance of *Backfilling* vs LDSBnB

In conclusion, LDSBnB produces a compromised schedule such that (a) the total-excessive-wait-time is comparable or better than that of FCFS-Backfill and (b) the average BSD is comparable or better than that of LXF-Backfill.

At this point, we search for the cause of LDSBnB bad performance in January and February workloads. First, we consider the average queue length because reducing the average queue length results in the decreasing of the job wait-time. Figure 3 shows the average queue length of each policy. Even though our search method results in smaller average queue length, the average queue length of January is still large.

Second, we consider the decision points where there are a lot of jobs in the waiting queue. Figure 4 shows the ratio of decision points where the search can cover less than one percent of the search space. Approximately 80 percents of the time in February and January our search method cover less than one percent of the search space. These results confirm our suspicion that the search space may be very large and our search method may not cover enough space.

The immediate future works are (1) exploring the pruning technique to cover more search space in limited time since the above results shows that cover more space lead to better schedules; (2) studying the sensitivity of hour bound used in the objective; (3) studying the effect of estimated job runtime because such information in the real world is always inaccurate.

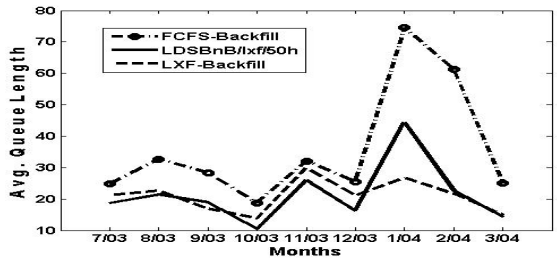


Figure 3: Average queue length of each policy

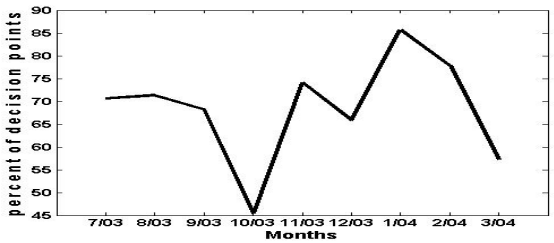


Figure 4: Ratio of decision points when LDSBnB covers less than one percent of the search space

Conclusions

While space limitations preclude full discussion of these results, they have been highly promising. Experiments with an objective of two conflicting components have demonstrated that systematic search can outperform priority backfilling methods in achieving the overall objective. Works need to be done on more complex objectives involving Quality-of-Service, fairness and special priority. Then the experimentations on local search methods and the alternatives must be conducted. Finally, the modeling component of the system needs to be constructed.

Acknowledgement

The author would like to thank Dr. Chiang and Dr. Massey for their valuable discussions and advice in crafting this research.

References

Chiang, S. & Fu, C. 2005. Benefit of Limited Time-Sharing in the Presence of Very Large Parallel Jobs, *in Proc. Int. Parallel & Distributed Processing Symp.*, Denver

Chiang, S. & Vernon, M. 2001a. Production Job Scheduling for Parallel Shared Memory Systems, *in Proc. Int. Parallel & Distributed Processing Symp*., San Francisco, California

Chiang. S. & Vernon, M. 2001b. Characteristics of a large shared memory production workload, *LNCS* Springer-Verlag. 2221:159-187.

Climer S., Zhang W., 2004, A Linear Search Strategy using Bounds, in *Proc ICAPS*, British Columbia, Canada

Crawford, J., 1993. Solving Satisfiability Problems Using a Combination of Systematic and Local Search. *Second DIMACS Challenge: Cliques, Coloring, and Satisfiability*, Rutgers University, NJ.

Feitelson, D. 2005. Experimental Analysis of the Root Causes of Performance Evaluation Results: A Backfilling Case Study. *IEEE Trans. Parallel & Distributed System.* 16(2):175-182.

Ginsberg, M. 1993. *Essentials of Artificial Intelligence*. Morgan Kaufmann

Harvey, W. & Ginsberg, M. 1995. Limited Discrepancy Search. *In Proc. of Int. Joint. Conf. in AI*, 607-613.

Jackson, D., Snell Q. & Clement, M. 2001. Core Algorithms of the Maui Scheduler. *LNCS*,Springer-Verlag.2221:87-102.

Korf. R. 2004, Optimal Rectangle Packing: New Results, in *Proc ICAPS*, British Columbia, Canada

Lifka, D. 1995. The ANL/IBM SP scheduling System, LNCS Springer-Verlag. 949:295-303.

Mu'alem, A. & Feitelson, D. 2001, Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Trans. Parallel & Distributed Sys*., 12(6):529-543

Purdom, P. 1983. Search rearrangement backtracking and polynomial average time. *AI* 21(1-2):117–133.

Skovira, J. Chan W, Zhou H, & Lifka D., The EASY-LoadLeveler API project, Springer-Verlag. 1162:41-47.

Smith, W., Taylor, V. & Foster I 1998. Using Run-Time Predictions to Estimate Queue Wait Times and Improve Scheduler Performance. Springer-Verlag. 1459:122-142.

Subhlok, J., Gross T., & Suzuoka T. 1996. Impact of Job Mix on Optimizations for Space Sharing Schedulers. *In Proc. of ACM/IEEE conf. on Supercomputing*, Pittsburgh

Talby, D. & Feitelson, D. 1999. Supporting Priorities and Improving Utilization of the IBM SP Scheduling Using Slack-Based Backfilling. *In the Intl. Parallel Processing Symp*., 513-517, San Juan, Puerto Rico.: IEEE Press.

Talby, D. & Feitelson, D. 2005. Improving and Stabilizing Parallel Computer Performance Using Adaptive Backfilling., *in Proc. Int. Parallel and Distributed Processing Symp*., Denver, Colorado.

Wolfram, D. 1989. Forward checking and intelligent backtracking. *Information Processing Letters*, 32: 85-87.

Planning as inference over DBNs

Deepak Verma and **Rajesh P. N. Rao**
Deptt of CSE, Univ. of Washington,
Seattle WA- 98195-2350
{deepak,rao}@cs.washington.edu
An extended version submitted to AAAI 2005

Abstract

We show that the problems of planning and policy learning can be addressed within a unified framework based on probabilistic inference in graphical models. Planning is viewed as probabilistic inference of an action sequence in a Dynamic Bayesian Network (DBN), given an initial state and a desired goal state. We describe how planning can be used to bootstrap the learning of goal-dependent policies. In contrast to conventional planning and policy learning methods, our approach does not require a reward or utility function, although constraints such as shortest path to goal can be incorporated within the graphical model. More importantly, we show that the approach extends easily to the challenging case of partially observable states (e.g., POMDPs). The method's performance on a set of benchmark POMDP problems was found to be comparable to or better than current state-of-the-art POMDP algorithms.

1 Introduction

Two fundamental problems in AI are planning and policy learning. Planning involves computing a sequence of actions that will take an agent from its current state to a desired goal state. Classical planning techniques focused on deterministic environments, allowing the problem to be formalized as deduction (Green 1969). Satisfiability solvers can be employed in such cases to efficiently compute the optimal plan (Kautz and Selman 1992). However, in real-world domains where the effects of actions are stochastic and noisy (e.g., (Bonet and Geffner 2000)), agents need to be "reactive" and choose actions based on current state. Considerable research has focused on learning "policies," which prescribe the optimal action to take in any given state so as to maximize total future expected reward (Boutilier *et al.* 1999; Sutton and Barto 1998). However, such algorithms do not generalize easily to the case of partially observable MDPs (POMDPs), where the state is not directly observable and actions are computed based on probability distributions (or "beliefs") over states

In this paper, we show that the problems of planning and policy learning can be solved in a unified manner using inference in probabilistic graphical models. We demonstrate the strength of the approach by applying it to the difficult problem of planning in POMDPs. The main contributions of this paper are: (1) An efficient method for planning under uncertainty based on the most probable explanation (MPE) of hidden variables in a graphical model (cf. (Attias 2003)), (2) A planning-based method for learning optimal policies that does not require an artificial reward structure to be imposed on the problem, (3) A new planning method for the challenging case of partially observable MDPs (POMDPs) Our approach opens the door to solving the problem of planning in uncertain environments using powerful new algorithms for exact and approximate inference in graphical models, in much the same way as viewing the deterministic planning problem as deduction paved the way for using satisfiability solvers for efficient planning (Kautz and Selman 1992).

2 Graphical Models for Planning and Policy Learning

Let Ω_S be the set of states in the environment, Ω_A the set of all possible actions available to the agent, and Ω_G the set of possible goals (all finite). For the present paper, we assume that goals represent states that the agent wants to reach, so each goal g represents a target state $S_g \in \Omega_S$. At time t the agent is in state S_t and executes action A_t. G_t represents the current goal at time t. Executing the action A_t changes the agent's state in a stochastic manner given by the transition probability $P(S_{t+1} \mid S_t, A_t)$, which is typically assumed to be independent of t i.e., $P(S_{t+1} = s' \mid S_t = s, A_t = a) = \tau_{s'sa}$.

Starting from an initial state $S_1 = s$ and a desired goal state $G_1 = g$, the agent's aim is to reach the goal state by a series of actions $A_{1:T}$, where T represents the maximum number of time steps allowed (the "episode length"). Note that we do not require T to be *exactly* equal to the shortest path to the goal, just as an upper bound on the shortest path length. We use capital letters (e.g., S, A) to denote variables and lowercase letters (e.g., s, a) to denote specific instances. Also, when obvious from context, we use s for $S_t = s$ and a for $A_t = a$, etc. The problem formulated above (for a given goal) can be considered a special case of a *Markov Decision Process* (MDP) (Boutilier *et al.* 1999) if we choose a suitable reward function. As in the case of MDPs, the strategy to choose the optimal set of actions is critically dependent

on the *observability* of the state. There exist three cases of interest:

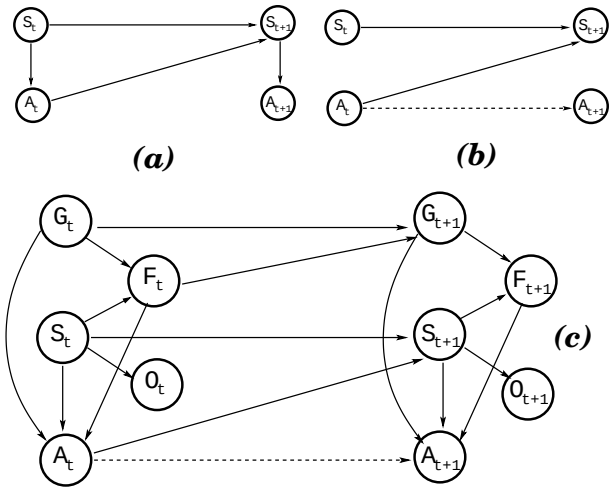


Figure 1: **Graphical Models:** (a) The standard MDP (or FOMDP) model. (b) The Non Observable MDP (NOMDP) model: Dotted line represents a possible dependence between consecutive actions. (c) The POMDP model (with goal and "finished" nodes) used in this paper.

Fully Observable MDP (FOMDP or just MDP): The state S_t is fully observed. The agent needs to compute a stochastic *policy* $\hat{\pi}_t(a \mid s, g)$ that maximizes the probability $P(S_{T+1} = S_g \mid S_t = s, G_t = g)$. The graphical model for this case (ignoring goals) is shown in Fig. 1a.

Non Observable MDP (NOMDP): In this case, the state S_t cannot be observed and the only information available to the agent is the initial state. The agent thus needs to compute an optimal *plan*, i.e.,a sequence of actions $\hat{a}_{1:T}$. We use the algorithm in (Attias 2003) to compute the optimal plan for NOMDPs.

Partially Observable MDP (POMDP): In this case, the state S_t is hidden but it is possible to observe some aspects of it where an (discrete) observation o is produced with the probability $P(O_t = o \mid S_t = s) = \zeta_{so}$. We extend the POMDP model to include a goal variable G_t representing the current goal and a boolean "reached" variable F_t, that assumes the value 1 whenever the current state equals the current goal state and 0 otherwise. These are used to infer the shortest path to the goal state and demonstrate the potential extensibility using graphical models.

The Maze Domain: To illustrate the proposed approach, we use the standard stochastic maze domain (Sutton and Barto 1998; Attias 2003) (Figure 2). There are five possible actions: `up,down,left,right` and `stayput`. Each action takes the agent into the intended cell with a high probability and into the neighboring cells with probability η.

3 Planning and Plan Execution Strategies

In this section, we investigate solutions to the problem of planning an action sequence given the general graphical model in Fig. 1c. This is used to bootstrap policy learning. For simplicity of exposition, we assume full observability ($\zeta_{so} = \delta(s, o)$) in this section and generalize the solutions for partial observability in section 5. We also assume that the environment model τ is known (the problem of learning τ is addressed in the next section). The problem of planning can then be stated as follows: Given a goal state g, an initial state s, and number of time steps T, what is the sequence of actions $\hat{a}_{1:T}$ that maximizes the probability of reaching the goal state? The *maximum a posteriori* (MAP) action sequence is:

$$\hat{a}_{1:T} = \operatorname*{argmax}_{a_{1:T}} P(a_{1:T} \mid S_1 = s, S_{T+1} = S_g) \quad (1)$$

Planning can thus be viewed as a probabilistic inference problem over a Dynamic Bayesian Network (DBN) (Attias 2003). The exact solution to inference of a subset of hidden variables in a Bayesian Network is known to be NP-complete (Cooper 1990). This problem is also a special case of solving a NOMDP problem with a specific reward function, an NP-complete problem in its most general form (Papadimitriou and Tsisiklis 1987). An efficient solution for a specific case was proposed recently by (Attias 2003), but the approach does not generalize easily to arbitrary graphical models.

We propose an alternate solution, namely, computing the most probable explanation (MPE) for a graphical model, given a set of known variables. This is a straightforward inference problem computable using standard techniques (such as the junction tree algorithm used for the results in this paper) and generalizes easily to arbitrary dynamic graphical models. When applied to the graphical model in Fig. 1c, our proposal for planning amounts to computing:

$$\bar{a}_{1:T}, \bar{s}_{2:T+1}, \bar{g}_{1:T}, \bar{f}_{1:T} = \operatorname{argmax} P(a_{1:T}, s_{2:T}, g_{1:T}, f_{1:T} \mid s_1, g_{T+1}, F_{T+1} = 1) \quad (2)$$

Note that there may exist cases where $\bar{a}_{1:T} \neq \hat{a}_{1:T}$. However, the MPE-based plan is typically a very good approximation to the MAP plan, is efficiently computable, and provides flexibility by allowing arbitrary graphical models. More importantly, the MPE-based plan can be used to learn optimal policies whereas the MAP plan typically cannot.

The "reached" variable F_t is used to compute the shortest path to the goal. For $P(a \mid g, s, r)$, we set the prior for the `stayput` action to be very high when $F_t = 1$ and uniform otherwise.Thus, the `stayput` action is discouraged unless the agent has reached the goal and hence MPE takes the shortest path[1]. The success of a plan is very sensitive to noise in the environment, A plan may fail when a single action in the plan "fails" (results in an unexpected state). This becomes more likely as plan length increases[2] (see Fig. 2).

4 Policy Learning

As discussed above, in noisy environments, it is preferable to execute actions based on the current state rather than only

[1]This can be used in any other domain also as one can always add in a *no-op* action

[2]For a detailed comparison of MAP vs. MPE approach and other action selection strategies (including greedy and why that does not work) see (Verma and Rao 2005)

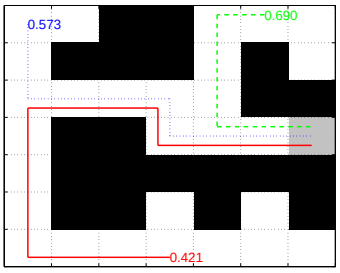


Figure 2: **Three Example Plans (Action Sequences) Computed using the MPE Method**. The plans are shown as colored lines capturing the direction of actions. The numbers denote probability of success of each plan.

the initial state. This requires a policy $\hat{\pi}(a \mid s, g)$, which represents the probability for action a in state s when the goal state to be reached is g. In this section, we describe how the MPE-based plan-and-execute strategy can be used to learn an optimal policy . We define optimality in terms of reaching the goal using the shortest path. Note that the optimal policy may differ from the prior $P(a|s, g)$ which would count all actions executed in state s for goal g, regardless of whether the plan was successful.Also, the MAP estimate (Eq. 1) of (Attias 2003) while providing the best *plan*, would be unsuitable for learning policy. (See (Verma and Rao 2005) for details.)

Algorithm 1 shows a planning-based method for learning policies for an MDP (both τ and π are assumed unknown and initialized to a prior distribution, e.g., uniform). To learn an accurate τ, the algorithm is biased towards exploration of the state space initially based on the parameter α (the "exploration probability") decayed by γ.

Algorithm 1 Policy learning in an unknown environment

1: Initialize transition model $\tau_{s's a}$, policy $\hat{\pi}(a \mid s, g)$, α.
2: **for** $iter = 1$ to $numTrials$ **do**
3: Choose random start location s_1 based on prior $P(s_1)$.
4: Pick a goal g according to prior $P(G_1)$.
5: With probability α:
6: $a_{1:T}$= Random action sequence.
7: Otherwise:
8: Compute MPE plan as in Eq.2 using $\tau_{s's a}$; $a_{1:T} = \bar{a}_{1:T}$
9: Execute $a_{1:T}$ and record observed states $s^o_{2:T+1}$.
10: Update $\tau_{s's a}$ based on $a_{1:T}$ and $s^o_{1:T+1}$.
11: If the plan was successful, update policy $\hat{\pi}(a \mid s, g)$ using $a_{1:T}$ and $s^o_{1:T+1}$.
12: $\alpha = \alpha \times \gamma$
13: **end for**

Experiments and Results: We tested the above algorithm in the maze domain with three goals and $\eta = 0.02$, $\alpha = 1$ and $\gamma = 0.98$. Figure 3a shows the error in the learned transition model and policy as a function of the number of iterations of the algorithm. Error in $\tau_{s's a}$ was defined as the squared sum of differences between the learned and true transition parameters. Error in the learned policy was defined as the number of disagreements w.r.t. the optimal deterministic policy Both errors decrease to zero with increasing number of iterations. The policy error decreases only after the transition model error becomes significantly small because without an accurate estimate of τ, the MPE plan is typically incorrect . Fig. 3b shows the optimal policy learnt. In separate experiments, we found the policies learned using Algorithm 1 for different mazes and goal states to be identical to the optimal policies learned using Q-learning (Sutton and Barto 1998) with a reward of +1 for the goal state and -1 for other states.

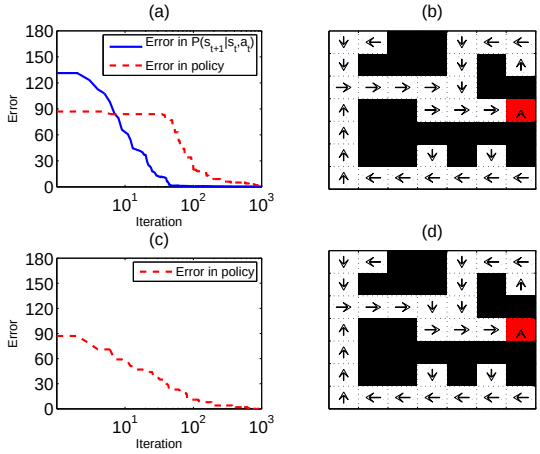


Figure 3: **Learning Policies for an MDP and a POMDP:** (a) shows the error in the transition model and policy (b) The optimal policy (c) and (d) show corresponding results for the policy in the POMDP case.

5 Partial Observability

Algorithm 1 assumes that agent has access to the true state of the environment. In case of partial observability problem thus becomes a POMDP problem. We consider two cases:

Solving the Underlying MDP: In the case where the observations are strongly correlated , one can modify Algorithm 1 as follows: (1) The MPE plan, $\bar{a}_{1:T}$, is computed based on observation $O_1 = o$ as evidence instead of $S_1 = s$ in Eq.2. (2) The plan is executed to record observations $o_{2:T+1}$, which are then used to compute the MPE estimate for the hidden states: $\bar{s}^o_{1:T+1}, \bar{g}_{1:T}, \bar{f}_{1:T+1} = \text{argmax}\ P(s_{1:T+1}, g_{1:T}, f_{1:T+1} \mid o_{1:T+1}, \bar{a}_{1:T}, G_{T+1} = g)$. The MPE estimate $\bar{s}^o_{1:T+1}$ is then used instead of $s^o_{1:T+1}$ to update $\hat{\pi}$ and τ. Note that it is the use of graphical models that enables this relatively easy extension to POMDPs.

Experiments: We focused on learning $\hat{\pi}$ for a POMDP version of the maze task and assumed τ was given. We selected $P(o_t \mid s_t)$ such that $o_t = s_t$ with probability 0.85 and one of the nearby states otherwise. The results are shown in Fig. 3c and d. The policy error decreases and reaches zero in spite of perceptual ambiguity.

Planning in POMDPs using Beliefs: In case of heavy perceptual aliasing, a plan based on an initial observation is bound to fail in most cases. Also, the strategy of learning the optimal policy for the underlying MDP does not work well in many cases. Below, we propose a new algorithm for

computing an optimal action at each time step based on the current belief. To simplify the exposition, we do not include the terms for G_t and F_t. As before, define:

$$\bar{a}_{1:T}, \bar{o}_{1:T}, \bar{s}_{1:T} = \text{argmax } P(a_{1:T}, o_{1:T}, s_{1:T} \mid o_{1:t}, a_{1:t-1}, S_{T+1} = S_g) \quad (3)$$

Algorithm 2 A MPE-based POMDP Solution

1: **Given:** Initial obs o_1 , desired goal state g and T.
2: Compute $\bar{a}_{1:T}, \bar{o}_{1:T+1}$ as in Eq.3 for $t = 1$
3: **for** $t = 1$ to T **do**
4: Execute $\bar{a}_t$ to generate $O_{t+1} = o_{t+1}$.
5: **if** $o_{t+1} \neq \bar{o}_{t+1}$ **then**
6: Update $\bar{a}_{1:T}, \bar{o}_{1:T+1}$ as in Eq.3 for $t + 1$.
7: **end if**
8: **end for**

The algorithm above computes actions based on all current and past observations and actions, and can handle non-trivial perceptual aliasing. To evaluate our algorithm, we ran it on a set of benchmarks maze problems introduced in (Littman *et al.* 1995) which have been used to test POMDP algorithms. The state spaces in these problems are reasonably large (57 and 89 respectively). The results[3] are shown in Table 1. As can be seen, our algorithm outperforms the QMDP algorithm (Littman *et al.* 1995) and PBVI[4] (Pineau *et al.* 2003) which is an anytime algorithm for solving POMDP using point-based value iteration. It matches the performance of HSVI (Smith and Simmons 2004) which solves the POMDP using heuristic search value iteration.

Domain	Q-MDP	PBVI	HSVI	MPE
Hallway	47.4	96	100	100
Hallway2	25.9	98	100	100

Table 1: **Comparison with other POMDP algorithms on Benchmarks:** The numbers denote the percentage of times goal was reached starting from a location chosen randomly.

6 Conclusions

This paper proposes the use of graphical models for addressing the problems of planning, policy learning, goal inference, and imitation within a single unified framework. A major advantage of the proposed approach is its ability to handle partial information, especially the challenging case of POMDPs. We showed that the method achieves results comparable to or better than some well-known algorithms for POMDPs in a set of benchmark problems with reasonably large state spaces and perceptual aliasing.

Our approach builds on the proposals of several previous researchers. It extends the approach of (Attias 2003) from planning in a traditional state-action Markov model to a full-fledged graphical model involving states, actions, and goals with edges for capturing conditional distributions denoting policies. The indicator variable F_t used in our approach is similar to the ones used in some hierarchical DBNs but these these papers do not address the issue of action selection or planning.

Although our initial results are encouraging, several important questions remain. First, how well does the proposed approach scale to MDP and POMDP problems with larger numbers of states and actions? For larger state spaces, we intend to explore hierarchical extensions of the current model, potentially allowing planning at multiple time scales and policy learning at multiple levels of abstraction. Another line of research actively being pursued is the investigation of graphical models for continuous state and/or action spaces. Clearly, applications such as controlling a robotic arm or maintaining balance in a biped robot are better expressed in the continuous domain. We expect the insights gained from the current discrete state space model to be extremely helpful in formulating graphical models for planning and policy learning in continuous state spaces.

[3]We ran the experiments as 251 runs with T set to 251 as in (Littman *et al.* 1995). The non-MPE results are taken from (Smith and Simmons 2004).

[4]PBVI trades-off performance for speed.

References

H. Attias. Planning by probabilistic inference. In *Proceedings of the 9th Int. Workshop on AI and Statistics*, 2003.

B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In *Proc. 6th International Conf. on Artificial Intelligence Planning and Scheduling*, pages 52–61, 2000.

C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of AI Research*, 11:1–94, 1999.

G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artif. Intell.*, 42(2-3):393–405, 1990.

C. Green. Application of theorem proving to problem solving. pages 219–239, 1969.

H. Kautz and B. Selman. Planning as satisfiability. pages 359–363, 1992.

M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *ICML*, pages 362–370, 1995.

C. Papadimitriou and J. N. Tsisiklis. The complexity of Markov decision processes. *Math. Oper. Res.*, 12(3):441–450, 1987.

J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for pomdps. In *IJCAI*, pages 1025 – 1032, August 2003.

T. Smith and R. Simmons. Heuristic Search Value Iteration for POMDPs. In *Proc. of UAI 2004*, Banff, Alberta, 2004.

R. S. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

D. Verma and R. Rao. Graphical models for decision theoretic planning. Technical Report UW-CSE-05-02-01, University of Washington, Seattle, WA, February 2005.

Quality-based Resource Management

Xiaofang Wang and Stephen F. Smith
The Robotics Institute, Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh,PA 15213
xiaofanw, sfs@cs.cmu.edu

Quality is an important dimension of operational excellence in many sectors, including news reporting, healthcare, intelligence gathering, new product research and development. Achieving high quality while controlling for costs is a major challenge for managers. Traditional research in scheduling and resource management tends to focus on the time-based performance measure such as makespan or weighted tardiness, while the objective of maximizing the *quality* of the outputs (or products) of scheduled processes is typically ignored. This is because quality is hard to define scientifically and quantitatively. Its definition highly depends on the problem context.

Instead of dealing with quality in general, our research focuses on duration-dependant quality, that is, quality is an increasing function of task duration. The optimization problem is solved by integrating artificial intelligence techniques(specifically, heuristic-guided constraint-based methods) and operations research. The broad goal of our research is to understand the connections and differences between this quality-based resource management and traditional time-based resource management.

In the sections below we summarize our work to date in this area and the set of research issues we are currently pursuing.

Previous Work

Our previous work (Wang & Smith 2004; 2005) explores a type of scheduling problem where each task is associated with a duration-dependent quality profile. In that problem, task durations must be determined to maximize overall quality while respecting process deadlines and resource capacity constraints. We develop and empirically evaluate a new precedence constraint posting (PCP) algorithm(as shown in Fig. 1), along with a number of search control heuristics for solving this class of problems. Our PCP algorithm incorporates linear optimization to set activity durations at each step, and search control heuristics direct the search toward resource feasibility. Overall, the experimental analysis indicates that a good heuristic must strike the right balance between minimizing quality loss at each step and retaining flexibility for future duration reduction.

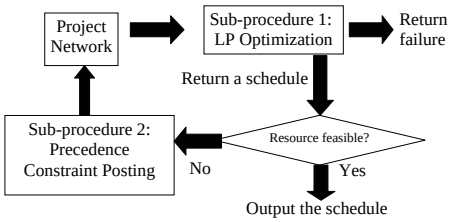


Figure 1: The Precedence Constraint Posting Framework

Ongoing Work

From the above work, we have seen it is hard to balance the quality loss in each constraint posting step and retaining flexibility for achieving resource feasibility. An alternative way to solve this scheduling problem is to separate the two aspects of the problem: find a resource feasible solution first and then achieve good quality.

To explore the potential advantages of this approach, we extend the two-step procedure of generating robust schedule (Cesta, Oddi, & Smith 1998; 2000; Policella *et al.* 2004). The solution framework is shown in Fig. 2. In the first step, we use the Earliest Start Time Algorithm(ESTA) to find a resource feasible solution assuming all activities have minimum durations. Then we take all the posted constraints away and post chaining constraints to link activities competing for the same resource into precedence chains. Finally, a linear programming solver is called to determine the durations in order to maximize the quality. After this last step, the resulting schedule is still resource feasible because chaining has established consistent sequences of activities, no matter how their durations are changed. ESTA is not a complete algorithm. If it can't find a resource feasible solution, the algorithm will return failure. We refer to this extended two step approach as the *Separated Algorithm*. And we call the previous algorithm the *Integrated Algorithm*. The ongoing work includes:

Algorithm Performance Analysis and Comparison

Initial experimental results have shown that the Integrated Algorithm achieves better quality when the resource capacity is large while the Separated Algorithm achieves better quality when resource capacity is small. The Separated Al-

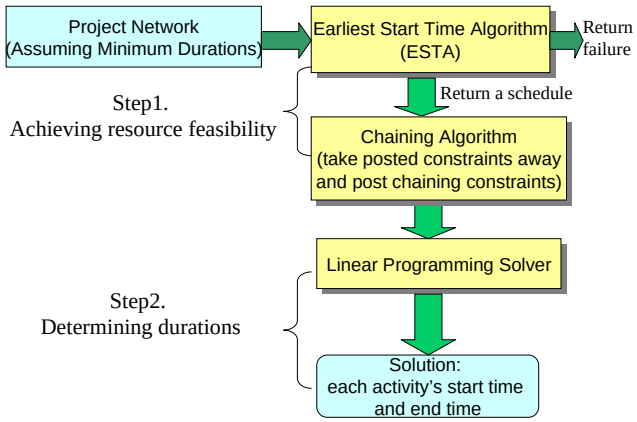


Figure 2: The Two Step Procedure

gorithm does a better job of finding feasible solutions. From this, we can see the Integrated Algorithm suffers more from the resource levelling process than the Separated Algorithm.

The explanation could be the Integrated Algorithm has the pathological behavior of posting unnecessary levelling constraints. The negative effect on quality becomes more obvious when it is applied to problems with smaller capacity. One goal of our current research is to establish this hypothesis, through both experimental and analytical results.

The Earliest Start Time Algorithm(ESTA) (Cesta, Oddi, & Smith 1998)has been shown to perform very well in make-span based scheduling problems. But it is somewhat surprising that it works well in the quality-based scheduling problems. From intuition, it shouldn't because it doesn't consider quality-related information. What is the performance of ESTA under quality-centered scheduling environment and how can we improve it? Those are the research questions in our current work.

Search Control with the Realtime Network Structural Information

Another focus of our work is the design of scheduling algorithms based on the information of the current network structure. We have already partially implemented this idea (Wang & Smith 2005). The central concept in the heuristics we define is use of a measure that combines an activity's *reducible duration* with its quality profile as a basis for determining how to resolve resource conflicts. This concept is found to yield heuristics that exhibit superior performance. *Reducible duration* is defined as the difference between current duration and minimum duration, which is actually the realtime network information. What if other more complex network structural information is considered during search, such as the path length, the size of an activity's successor(predecessor) set?

In the meanwhile, we need to define new measures for quality-centered networks. For example, how can we compare one network with the other in terms of the potential to achieve good quality? Some existing measures in robust scheduling literature provide us with a good start point.

Incremental Algorithm for Dynamic Scheduling Problems

The real world is dynamic. Many new tasks will arrive with different resource requirements and different quality returns. Acceptance decisions and duration decisions(if this activity is accepted) must be made correctly and promptly. Under such a fast-changing situation, incremental scheduling algorithms become natural alternatives, both to speed up the scheduling and rescheduling process and to hedge against possibility of future higher quality tasks. The design of incremental quality-based scheduling algorithms is another research objective.

More Complex Quality-Centric Scheduling Problems

A final goal of our current research is to extend our framework to solve and analyze more complex quality-centric scheduling problems. In our previous work, the only dependencies among activities are precedence relationships. Quality dependencies, which imply that one activity's output may influence it's successors' quality profiles, are more realistic. In that case, instead of a simple summation form, the objective will be a more complex function. We have assumed in our previous work that the expected quality of activities can be expressed by linear profiles. Actually, with little change, our solution procedure can be applied to piece-wise linear profiles, although the development of good heuristics for the extended problem remains an important open question. Profile information can also be associated with resources. This latter extension allows for the possibility of differentiating the skill level of different resources.

References

Cesta, A.; Oddi, A.; and Smith, S. 1998. Profile-based algorithms to solve multi-capacitated metric scheduling problems. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems(AIPS-98)*.

Cesta, A.; Oddi, A.; and Smith, S. 2000. Iterative flattening: A scalable method for solving multi-capacity scheduling problems. In *Proceedings of the Seventeenth National Conference in Artificial Intelligence (AAAI'00)*.

Policella, N.; Smith, S. F.; Cesta, A.; and Oddi, A. 2004. Generating robust schedules through temporal flexibility. In *Proceedings 14th International Conference on Automated Planning and Scheduling*.

Wang, X., and Smith, S. 2004. Generating schedules to maximize quality. Technical Report TR-04-25, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Wang, X., and Smith, S. F. 2005. Retaining flexibility to maximize quality: When the scheduler has the right to decide activity durations. In *Proceedings 15th International Conference on Automated Planning and Scheduling*.

Generic PSO Heuristic for Constrained Planning

Chi Zhou
Industrial & Manufacturing System Engineering
Huazhong Univ.of Sci&Tech, Wuhan 430074, P.R.China
cutecheng@163.com

Introduction

Among population-based heuristics, Particle Swarm Optimization (PSO) algorithm (Kennedy and Eberhart 1995; Shi and Eberhart 1998) receives more and more attentions for its efficient information flow efficiency and convergence performance. However, most applications of PSO algorithms aim at unconstrained optimization problems. For constrained engineering optimization, the generic approach simply combines it with penalty function strategy. However, it was investigated that simple penalty function strategy cannot be well integrated with PSO algorithms because it does not consider the historical memory information-an essential mechanism of PSO heuristic. A new generic constraints handling strategy that complies with the optimization mechanism of PSO is proposed. In addition, local search procedure is hybridized with PSO to intensify its search ability in specific promising regions and thus hybrid PSO model for constrained planning is put forward. To demonstrate its efficiency, a few applications have been reported on engineering problems such as cutting parameters selection, tolerance allocation, economic load dispatch and constrained layout planning. Experimental results validate the effectiveness of HPSO model for constrained planning problems.

HPSO Model for Constrained Planning

Real-world problems especially the engineering optimization problems contain many complex practical constraints and can be formulated as difficult nonlinear programming mathematical models. The methods for solving this kind of problems include traditional operational research methods (such as linear programming, quadratic programming, dynamic programming, gradient methods and Lagrangian relaxation approaches) and modern heuristic methods (such as artificial neural networks, simulated annealing and evolutionary algorithms). Some of these methods are successful in locating the optimal solution, but they are usually slow in convergence and require much computational cost. Some other methods may risk being trapped to a local optimum that is the problem of premature convergence. In view of the above analysis and the successful applications of PSO in nonlinear continuous optimization, maybe PSO is a potential remedy to these drawbacks. PSO is a novel optimization tool based on swarm intelligence, which utilizes the swarm intelligence generated by the cooperation and competition between the particles in a swarm. Compared with evolutionary algorithms (genetic algorithm, evolutionary programming, evolutionary strategy, and genetic programming), PSO maintains the population based global search strategy but adopts the velocity-displacement model with more efficient information flow and easier implementation procedures.

However, the applications of PSO are mainly focused on unconstrained optimization problems. Some researchers attempt to solve the constrained problems using traditional penalty function strategy. Penalty function is an effective constraint-handling tool for constrained optimization problems. It is also the most popular strategy for its simplicity and ease of implementation. Nevertheless, since the penalty function approach is generic and applicable to any type of constraint, the performance of the corresponding algorithms depend much on the dedicated design of penalty parameters (Carlos 2002). Especially when the problems are difficult and the imposed constrained conditions become more complex, this method usually fails to generate the constrained optimum solution, sometimes even cannot achieve a feasible one. This paper analyzes the underlying limitations of simple combination of PSO and penalty function strategy. Unfair competition exists in the swarm population under dynamic and adaptive penalty coefficients. This problem is obvious in that PSO has an inherent mechanism based on memory information. This mechanism has contributed much to high efficiency and effectiveness of PSO algorithm, but also lead to low flexibility for constrained optimization. That is, the penalty parameters cannot be changed during the iteration. The effectiveness of this strategy has been validated (Kalyanmoy 2000). Thus, the most difficult aspect of finding appropriate penalty parameters to guide the search towards the constrained optimum can be avoided. It is desirable to design a new generic constraint-handling scheme suitable for PSO and maintain high convergence efficiency. Integrating the memory mechanism of PSO with the penalty strategy, the proposed constraint-handling scheme is presented in Figure 1.

The core characteristics of the proposed strategy can be described as follows:

1) Corresponding to the memory mechanism of PSO, a special notation-Particle has been Feasible (PF) is introduced, which is used to record whether the current particle has ever satisfied all the constraint conditions. This notation preserves historical constraint status for each particle.

2) Each particle updates its individual best and neighborhood best according to the historical constraint information PF, the current constrain status (Current particle is Feasible, CF) and the objective function with the penalty term.
3) The algorithm selects the velocity updating strategy according to the historical information by PF.
4) When updating the personal and neighborhood best, the algorithm adopts the static penalty strategy instead of dynamic and adaptive ones to guarantee the fairness. The detailed procedure for updating the personal and neighborhood best values based on the above constrain handling strategy is presented in Figure1.

```
For Each Particle {
 If PF=true Then
  If f(x_i) ≤ f(p_i) and CF= true Then
    p_i = x_i
   If f(p_i) ≤ f(l_i) Then
     p_i = l_i
   End if
  End if
 Else if PF=false Then
  If CF= true Then
    p_i = x_i
   PF=true
   If f(p_i) ≤ f(l_i) Then
     p_i = l_i
   End if
   Else if f(x_i) ≤ f(p_i) Then
    p_i = x_i
  End if
 End if
```

Figure 1: The constraint handling strategy for PSO.

Special attention should be paid that the PSO algorithm based on the proposed constraint handling strategy does not have to guarantee the existence of feasible solutions in the initial population. With the randomized initial velocity, the PSO itself has the ability to explore the feasible space. In addition, the penalty function imposed on the violated particles also direct the search of PSO towards the feasible region. Therefore once feasible solutions emerge in the neighborhood population, the neighborhood best will be preserved in the subsequent iteration procedure. According to the velocity updating formula, each particle will obtain updating information from its neighborhood best particle, so the corresponding particle would return to the feasible solution space immediately.

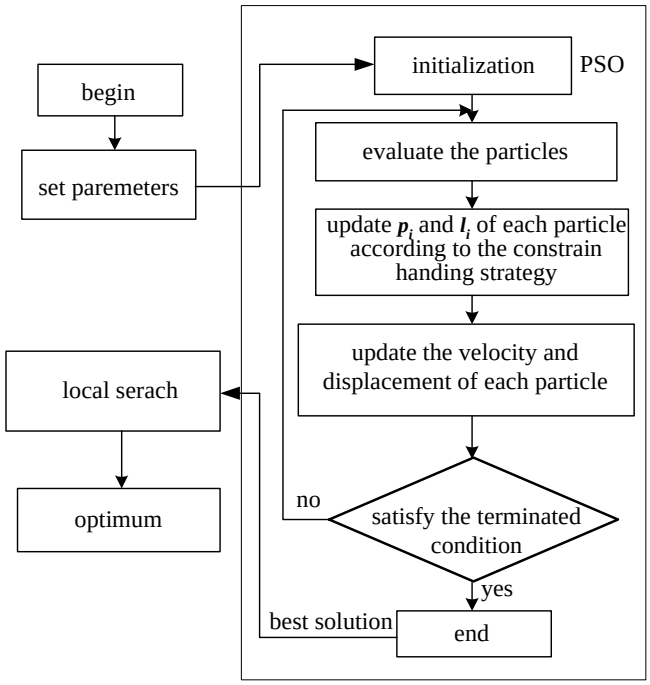


Figure 2: Framework of the proposed Hybrid PSO model.

Based on the constraint handling strategy presented in Figure 1, PSO can converge quickly to the optimal or near optimal region for constrained planning problems. However, despite its superior global search ability, PSO fails to meet the high expectation that theory predicts for the quality of the solution and search efficiency. As widely accepted that PSO is capable of identifying the region that contains possible optimum or near optimum solutions, but due to the lack of refined search in the local region, the convergence process is too slow to find the satisfactory solution with high precision within reasonable computational time. Considering the above limitation, a local search procedure such as direct search or simplex is introduced in the latter procedure to improve the convergence speed and the solution quality. That is, once the poetical region is identified by PSO, the hybrid PSO model will start a local search procedure. The corresponding procedure of hybrid PSO model can be summarized in Figure 2.

Experimental results

We use HPSO model based algorithms under proposed constraint handling strategy and local search procedure that includes direct search or simplex (Chun-Lung and Nanming 2001; Nelder and Mead 1965) to solve some constrained planning problems. Firstly, we choose some complex benchmarks to examine the efficiency of the proposed constraint handling method. It is supposed that the successful applications on benchmarking problems will reveal the potential in more complex problems with strong engineering background. Fortunately, the results on the benchmarks reached the positive conclusion. To further

investigate its performance, several engineering problems such as cutting parameters selection, tolerance allocation, economic load dispatch, and constrained layout planning are tested in the order of complexity level, with the number constraints from 2 to 980.

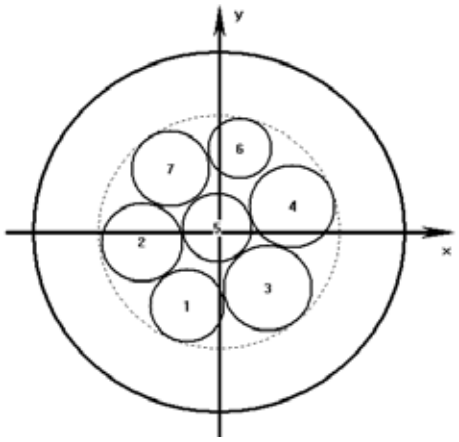
Here we just took the constrained layout planning problem as an example to demonstrate the performance of HPSO. In this problem, the layout to be optimized is required to satisfy the following three constraints: (1) capacity constraints, (2) non-overlap constraints, (3) static non equilibrium constraints. The distance between the center of the baseboard and centroid of all the pieces should be minimized or the error of the whole system should not exceed a permissible value. A great deal of work devoted to the layout planning problems has been done for many years because of its great significance in both theory and economy. The mathematical model and the corresponding instances can be referred to (Tang and Teng 1999). The objective is to minimize the radius of the circle whose origin coincides with the center of the vessel, and to minimize the circle that can encloses all the objects. The constraints have been listed in the above reference, the complexity of which significantly depends on the number of object. Here we referred two instances to demonstrate the efficiency of the proposed algorithm. In the instances, the radius of the round container equals to 50mm in the 7 objects instance and 880mm in the 40 objects one respectively.

Table 1: Computational results of case 1 (7 objects).

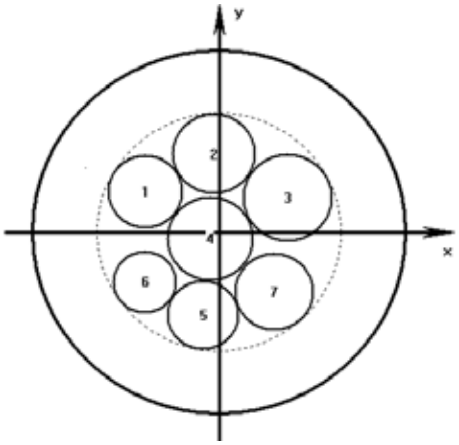
Algorithms	Maximum Radius (mm)	Non Equilibrium (g.mm)	CPU Time(s)
PSO+LS	32.230	7.04E-05	9.545
PSO	32.308	8.95E-05	13.439
Multiplier	32.559	7.87E-07	31.515
GA	32.837	0.102	288.010
HCIGA	32.662	0.029	166.332

Table 2: Computational results of case 2 (40 objects).

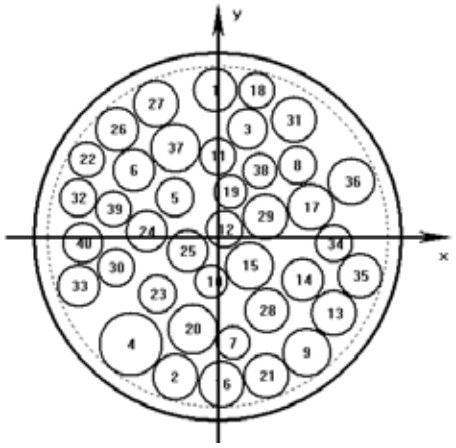
Algorithms	Maximum Radius (mm)	Non equilibrium (g.mm)	CPU Time (s)
PSO+LS	811.806	0.002	187.283
PSO	812.311	0.005	267.694
Multiplier	869.750	2.32E-05	304.422
GA	874.830	11.395	274.896
HCIGA	870.331	0.006	225.428



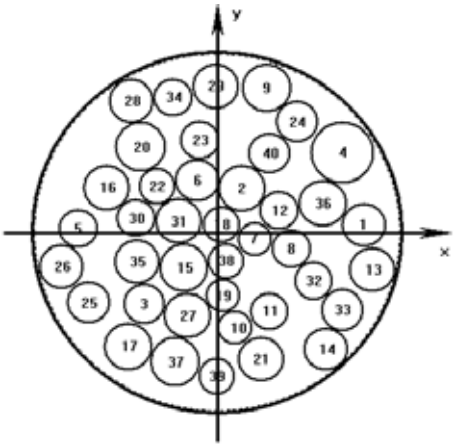
(a) Planning results of HPSO for case 1.



(b) Planning results of Multiplier for case 1.



(c) Planning results of HPSO for case 2.



(d) Planning results of Multiplier for cases 2.

Figure 3: Planning generated by HPSO and Multiplier.

Table 1 lists the statistical results achieved by the representative approaches, that is HPSO, traditional PSO, Multiplier, Hybrid GA and a Human-Computer interactive GA. The above four figures depict the optimized layout planning status. From the data in the tables and the corresponding figures, the maximum radius obtained by HPSO is much smaller than by the other approaches, which leads to the significantly space saving for this problem and thus the computational expense saving for the algorithm. Due to its delicate process, the Multiplier is slightly better than HPSO in terms of non equilibrium, but the corresponding results achieved by HPSO are good enough. In addition, the computational expense of the HPSO is much lower, which will reveal significant advantages in the lager scale problems.

Conclusion and future directions

In view of the memory mechanism of PSO, a new generic constraints handling strategy suitable for PSO is presented in this paper for constrained planning problems. This new strategy can adequately utilize the historical information in PSO algorithm.

To intensify the refined search ability, local search procedure is employed and hybridized with PSO model. Based on the constraints handling strategy proposed and local search procedure, the HPSO model is proposed for nonlinear constrained planning problems including standard benchmarks and several engineering problems. The computational results verify its effectiveness in terms of solution quality, computational cost as well as the convergence stability.

However, when we attempt to extend the proposed approach to the constrained optimization with large number of complex equality constraints, subtle drawbacks emerged, as the constrained make the feasible regions so narrow that the equality constraints are hard to met with high standard. This problem reveals the research direction that is the effective equality constraint handling strategy desirable for PSO based heuristic. Furthermore, more powerful local search methods should be introduced to improve its refined search ability. In view of the successful applications in the constrained planning problems especially those engineering ones, PSO can be considered as a generic constraint planning method, and thus could be applied to any engineering planning problems that can be modeled as nonlinear programming problems.

References

Kennedy, J., and Eberhart, R.C. 1995. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neutral Networks*, 4: 1942-1948.

Shi, Y.H., and Eberhart, R.C. 1998. A modified particle swarm optimizer. In *Proceedings of IEEE Conference on Evolutionary Computation*, 69-73.

Carlos, A. 2002. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer methods in applied mechanics and engineering* 191: 1245-1287.

Kalyanmoy, D. 2000. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering* 186: 311-338.

Chun-Lung, C., and Nanming, C. 2001. Direct Search Method for Solving Economic Dispatch Problem Considering Transmission Capacity Constraints. *IEEE Transaction on power system* 16(4): 764-769.

Nelder, J.A., and Mead, R. 1965. A simplex method for function minimization. *The computer Journal* 7: 308-313.

Tang F., Teng H. F. 1999. A modified genetic algorithm and its application to layout optimization [J]. *Journal of Software* 10(10): 1096–1102.

Local replanning within a team of cooperative agents

Olivier Bonnet-Torrès
ENSAE-Supaero and ONERA-CERT/DCSD
2, avenue Édouard Belin 31055 Toulouse cedex 4 FRANCE
olivier.bonnet@onera.fr

The general framework of the thesis is a mission specified in terms of objectives: agents are operated in order to carry out the mission and they are hierarchically organised in a team. This thesis aims at formalising the relationship between the team plan and individual agents' plans and the replanning process through the use of Petri nets (PN) (Bonnet-Torrès & Tessier 2005a; 2005b).

Related Work

In the wake of HTN, Grosz *et al.* (Grosz & Kraus 1996) base the *SharedPlan* approach on the hierarchical decomposition of shared plans into a sequence of recipes to be applied by a team of agents. Their work also inherits from the logics of beliefs and intentions (Cohen & Levesque 1990). Tambe *et al.* (Tambe 1997) have focused in STEAM on reactive team behaviour based on rules. Van der Krogt (van der Krogt & de Weerdt 2004) identifies two effects in the repair process: removing actions from the plan and adding actions. Moreover the representation of the plan itself tends to make use of the automata theory and the Petri net formalism (Chanthery, Barbier, & Farges 2004).

Mission, Goals and Agenticity

The *objective* of the mission is decomposed into a hierarchy of goals to be carried out. The leaves in the hierarchy are *elementary goals* and *recipes* give courses of actions for achieving them. Several recipes may be available for the same elementary goal. Agents' resources are modelled by coloured Petri nets. The team plan is extracted by organising a subset of recipes using constraint programming. The plan is attached a possible organisation of the team.

Since a group of closely interacting agents can be considered as an agent in itself, a team of agents is equivalent to a composite agent. This agent bears an *agenticity hierarchy*, whose leaves are elementary agents and whose nodes are subteams, *i.e.* composite agents. Each node has for children nodes the agents that compose the subteam it represents. There is no requirement that an individual agent be represented only once.

More formally the team X is composed of hierarchically organised elementary agents $\{x_1, x_2, \ldots, x_n\}$. Let $A = \{a_1, a_2, \ldots, a_m\}$ be the set of agents in X. The agenticity of agent a_i with regards to any subteam $a_j, a_i \subset a_j$ (including X) is its depth in the hierarchy $\mathcal{H}_{a_j}$ whose root is the considered subteam (fig. 1).

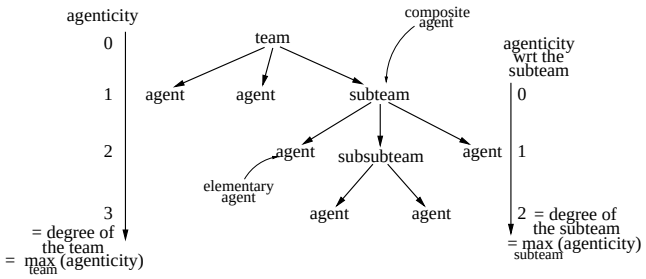


Figure 1: Hierarchy of agenticity

Team Plan

The team plan is designed in terms of a detailed sequence of tasks, represented as a coloured Petri net (CPN) (Jensen 1997) (see fig. 2). The set of token colours is the set of elementary agents. Each reachable marking $\mathcal{M}$ represents the allocation of the agents to the corresponding recipes. Petri net analysis can be performed through the use of the incidence matrix $\mathcal{A}$ (Murata 1989). $\mathcal{A}$ represents the relations between places and transitions, namely the arcs. The team plan bears some typical structures that can be identified as modifications of the team organisation, which allow to hierarchise the plan. The plan is then projected on the elementary agents.

Construction

During mission preparation a set of recipes that allow to achieve – maybe in a number of fashions – each of the elementary goals is defined by the (human) mission manager. A recipe is a piece of PN consisting in a single place. It is associated to a subnet that organises primitive actions so as to achieve one elementary goal. It specifies what resources are needed, how to use them and what the predicted duration for completion is. The other assumptions of the model are: (1) the elementary goals may come along with a completion

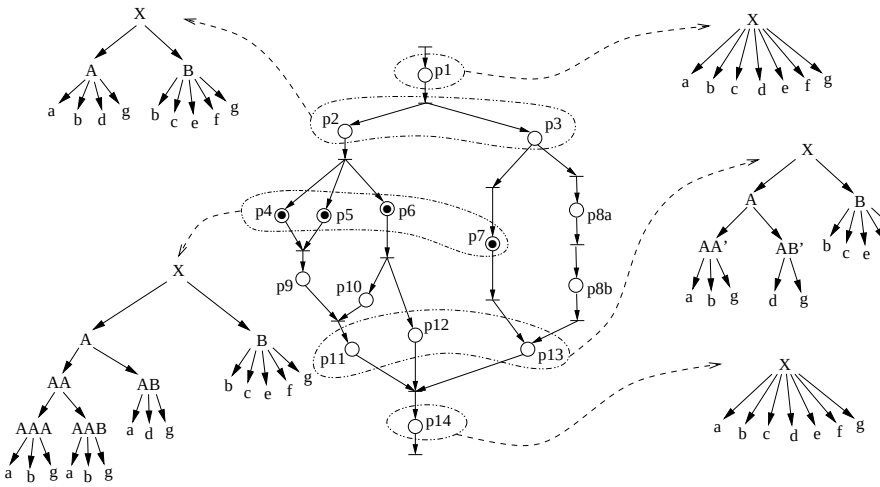


Figure 2. Team plan with some agenticity hierarchies derived from the initial allocation

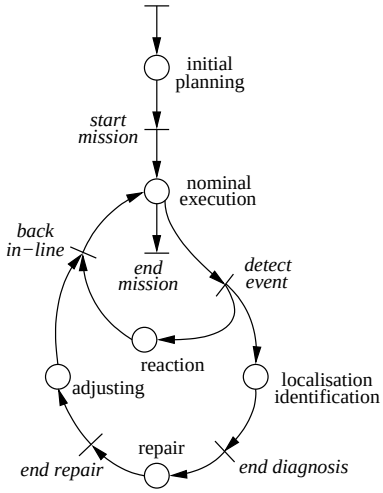


Figure 3. Execution control PN

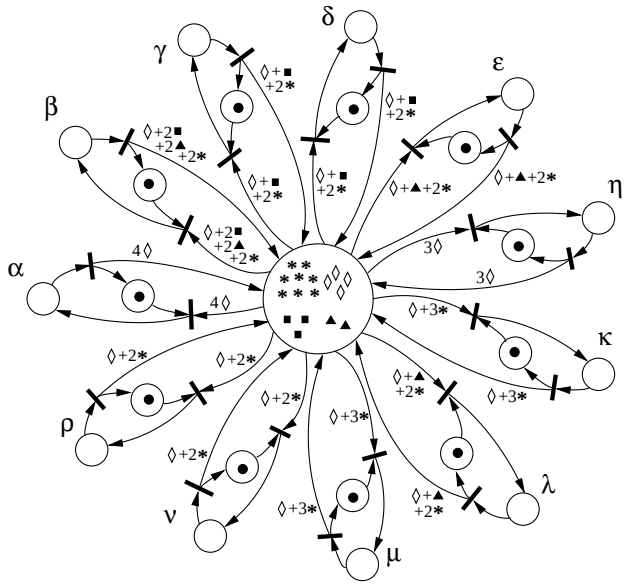


Figure 5: An agent viewed as a collection of resources. (Each type of symbol is a colour; greek letters designate the resources.)

time requirement or a mutual temporal precedence specification; (2) the (heterogeneous) agents can be regarded as resource aggregates managed by CPNs (fig. 5). Each colour represents a mutual exclusion constraint relation between a number of resources within the considered agent.

The team plan is a solution of a constrained problem: knowing that resources may have mutual exclusion constraints within each agent and that several recipes are available to perform each elementary goal, the construction process output is a team plan, *i.e.* a partial preorder on the goals that respects their constraints. Each goal is performed according to a recipe that is attached some agents so that both agent-internal constraints (resource exclusion) and external constraints (resource requirements for each recipe and time specifications) are satisfied. The problem is currently under investigation with two main leads:

- a resolution through constraint satisfaction/optimisation;
- a resolution with heuristics derived from the *open shop* scheduling framework.

Reduction

Representing a team plan as a hierarchical Petri net (HPN) allows for more flexibility than CPN and helps incorporating hierarchical information. The net is therefore abstracted so as to represent the activities at each level of agenticity. To build this information we extend the ordinary Petri net reduction rules according to the semantics of basic team management structures, namely *arrival* or *withdrawal of an agent, create* or *merge two subteams, transfer agents between subteams* and *choose a possible course of actions*. The CPN is decoloured and the agent information is transferred to the places. The reduction rules are iteratively applied until the net is reduced to a single place. The tokens within the HPN only represent activities. The hierarchy is born by the structure of the HPN (fig. 6). The algorithm is traced back so that each reduction place is hierarchically unfolded. The resulting plan then consists in a HPN whose levels correspond to the levels of agenticity in the agent team. Each place develops into a subnet of higher agenticity whose places hold the agent(s) performing the activity corresponding to the marking. Hence each reachable marking $\mathcal{M}$ corresponds to an agenticity hierarchy $\mathcal{H}_X(\mathcal{M})$ of the whole team X. The Petri net in figure 2 in fact corresponds to the *detailed global plan* built from the leaf-places of the hierarchical team plan of figure 6.

Projection

The hierarchical structure of the team plan allows the agents' individual plans to be derived. In the detailed global plan, the plan of agent a_i consists of the sequence of places where a_i appears and all levels above. The corresponding activities all involve a_i or its ancestors in the agenticity hierarchies.

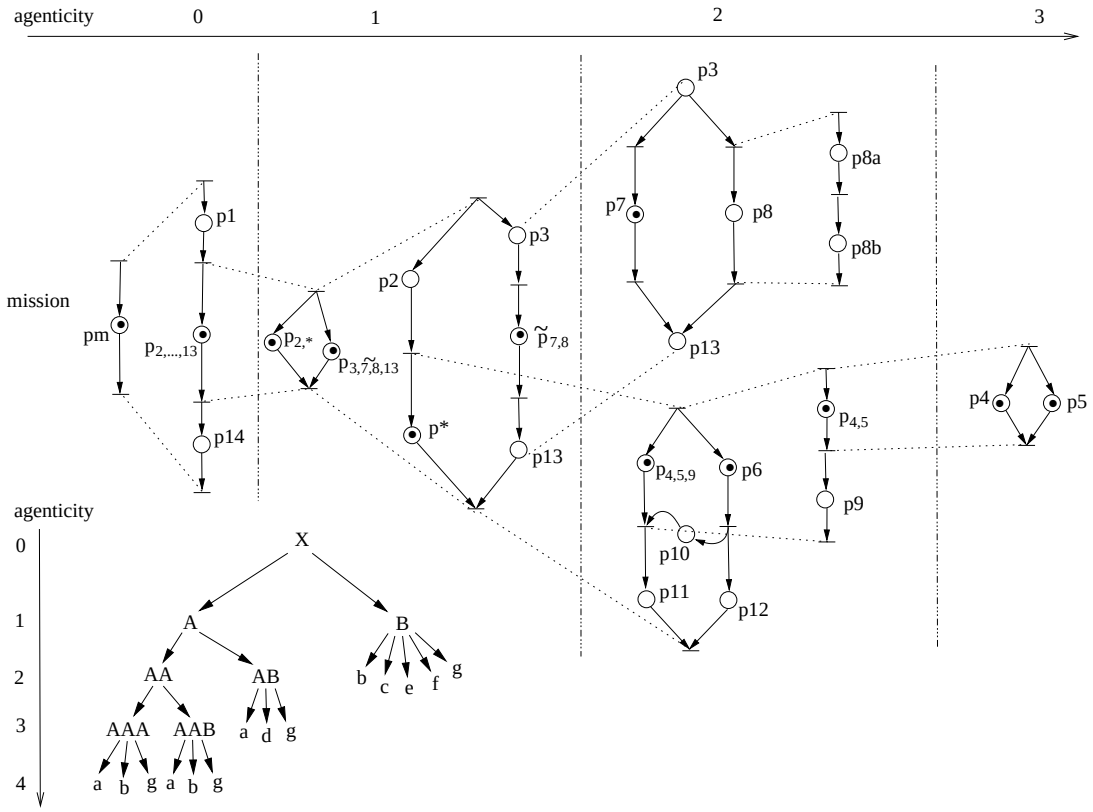


Figure 6: A hierarchical team plan with agenticity hierarchy tokens

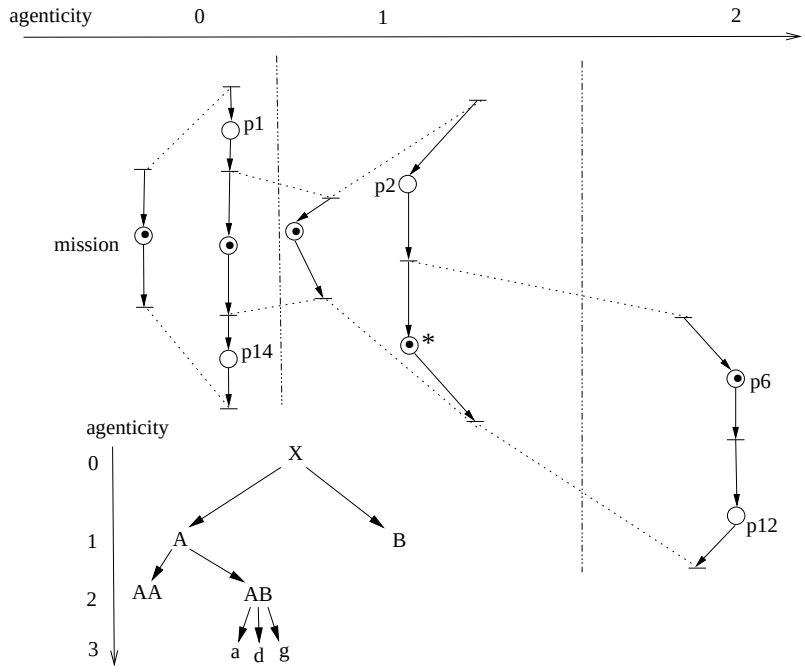


Figure 7: Projection of the team plan on agent d

The projection of the team plan on an agent a_i consists in isolating the places of the corresponding level of agenticity in which a_i is involved and extracting the hierarchies of places and agenticity associated to the places (fig. 7). This definition extends the projection operator in coloured Petri nets to hierarchical nets. In the detailed global plan (fig. 2) the projection on a_i prunes the branches of the Petri net that are not labelled by a_i. It results in an individual plan for a_i.

Plan Execution

A General Overview

Another Petri net models the execution control (fig. 3). The controller is the same for all agents at any level of agenticity including for the team itself. An abstract instance of the controller is considered during mission design and monitoring. It is distributed on the team when the mission is performed. Before the mission begins an initial planning phase (first place in the PN on figure 3) is performed: the plan is prepared out of the set of recipes, then reduced and projected onto individual agents, as mentioned in the previous section. Then the plan is executed. At the occurrence of an unexpected event a replanning step is triggered. At that time a reaction is deployed in the form of a contingency plan while the system goes under diagnosis. When the failure is located the plan is repaired as locally as possible. Once the new plan is elaborated an adjusting step is required to ensure a smooth switching between the contingency plan and normal execution of the new plan.

More Details on Replanning

During plan execution an unexpected event may be detected. Some of these events, usually the most probable or the most critical, are foreseeable and hence taken care of thanks to alternate plans. For the others they are not directly distinguishable: only their effects are detectable. For instance no answer (*i.e.* a timeout) to a communication request may denote either a malfunction of the sender's or recipient's communication systems, on either the emit or the receive circuit. It may as well correspond to the withdrawal of the unanswering agent from the team. A timeout for an expected action may denote a failing communication link (that prevented a synchronisation), a withdrawal from the team or an obstacle to the realisation of the task. These events may therefore be categorised according to their indirect characteristics. For each category or foreseen event a generic adapted reaction is stored: a contingency plan that has to be instantiated according to the features of the situation, *e.g.* which agent(s) is(are) in trouble and which is(are) impacted.

At the same time some agents or resources may not be available any more. This results in some new constraints on the remaining recipes. The plan is repaired by applying the planning process locally. Locality is ensured in trying to solve the problem at the lowest possible level in the agenticity hierarchy. The plan repair consists in replacing the failing recipe by another recipe or subset of recipes that realise the same goal. The subsequent activities may be modified so as to respect the constraints. If this fails it is necessary to involve other parts of the initial plan in the repair in ascending the agenticity hierarchy. One can notice that, as far as some subteams are concerned, there exists a discontinuity between the already-executed part of the plan and the new plan at the repaired level, but not at the above levels in the hierarchy.

When the repaired plan is constituted the current state of the team may not correspond to the initial state of the repaired plan. An adjustment is then necessary. Two possibilities have been identified: either the beginning of the new plan is appended a dispatch phase that will be suited to the expected end point of the reaction plan or the contingency plan is interrupted and the dispatch is organised from that point. In both cases the transition to standard mission execution is smooth.

Conclusion

In the context of teams of robots, this approach may aims at dynamically responding to an unexpected event, such as a failure or an external action, at a relevant level. Current works concern the reallocation problem in the repair. An identified difficulty is to avoid global repairs that involve the whole team: the repair must be attempted at the lowest agenticity level (recipe level) so as to ensure its locality; if unsuccessful the next level is considered. The midterm objective is the development of ΕΛΑΙΑ, a Petri net-based decision architecture for local replanning within the team. Experiments are prepared in order to validate the principles with a team of *PeKee* robots at Supaero. The envisioned applications concern the implementation of cooperative robots for missions ranging from search and rescue operations to military UAV/robot team operation.

References

Bonnet-Torrès, O., and Tessier, C. 2005a. From multiagent plan to individual agent plans. In *ICAPS'05 Workshop on Multiagent Planning and Scheduling*.

Bonnet-Torrès, O., and Tessier, C. 2005b. From team plan to individual plans: a Petri net-based approach. In *AAMAS'05*.

Chanthery, E.; Barbier, M.; and Farges, J.-L. 2004. Integration of mission planning and flight scheduling for unmanned aerial vehicles. In *ECAI'04 - Workshop on Planning and Scheduling: Bridging Theory to Practice*.

Cohen, P., and Levesque, H. 1990. Intention is choice with commitment. *Artificial intelligence* 42:213–261.

Grosz, B., and Kraus, S. 1996. Collaborative plans for complex group action. *Artificial Intelligence* 86(2):269–357.

Jensen, K. 1997. *Coloured Petri nets. Basic concepts, analysis methods and practical use*. MTCS. Springer, 2nd edition.

Murata, T. 1989. Petri nets: properties, analysis and applications. In *Proc. of the IEEE*, volume 77-4, 541–580.

Tambe, M. 1997. Towards flexible teamwork. *Journal of Artificial Intelligence Research* 7:83–124.

van der Krogt, R., and de Weerdt, M. 2004. The two faces of plan repair. In *Proc. of the 16th Belg.-Neth. Conf. on Art. Int.*, 147–154.