ICAPS 2006

The English Lake District, Cumbria, UK

Doctoral Consortium

DC

Jeremy Frank
NASA Ames Research Center, USA

Marie desJardins
UMBC, USA.














ICAPS 2006

The English Lake District, Cumbria, UK

Doctoral Consortium

Jeremy Frank
NASA Ames Research Center, USA

Marie desJardins
UMBC, USA.

ICAPS 2006
Doctoral Consortium

Table of contents

ICAPS 2006
Doctoral Consortium

Preface

Welcome to the 16th International Conference on Automated Planning and Scheduling Doctoral Consortium.

This is the fourth Doctoral Consortium hosted by ICAPS, and it continues the tradition established by the first Doctoral Consortium in 2003. A total of 36 doctoral students from 10 countries presented their work, interacted with senior practitioners in the field and with each other, explored new ideas, experienced criticism and praise, grew as researchers, and enriched our community. These students bring a range of experience and interests to planning and scheduling, and during the Doctoral Consortium discussed a wide variety of topics, ideas and techniques for addressing problems of planning and scheduling. The Doctoral Consortium provides these students a valuable opportunity not only to present their work, but also to discuss their ideas in depth with a senior researcher or user of these technologies, who will look at their work as an interested outsider. We are confident that this year's Consortium will set the stage for more to come.

The 2006 Doctoral Consortium was funded by DARPA, the National Science Foundation, NASA Ames Research Center, Honeywell, NICTA/Australian National University, and QSS Group Inc. We would like to thank Subbarao Kambhampati, Adele Howe, Ken Brown, Francesca Rossi, Wheeler Ruml for assistance in reviewing student applications. The Doctoral Consortium also owes special thanks to Stephen Smith for additional fundraising support, and to Carnegie Mellon University for handling the finances. Finally, special thanks to Blazej Bulka for assistance in tracking and assembling the final list of submitted papers.

Marie des Jardins
Jeremy Frank
ICAPS 06 Doctoral Consortium Co-Chairs

Organizers

- *Marie desJardins (mariedj@cs.umbc.edu), University of Maryland Baltimore County*
- *Jeremy Frank (frank@email.arc.nasa.gov), NASA Ames Research Center*

Reviewers

- *Ken Brown (k.brown@cs.ucc.ie), University College Cork*
- *Adele Howe (howe@cs.colostate.edu), Colorado State University*
- *Subbarao Kambhampati (rao@asu.edu), Arizona State University*
- *Francesca Rossi (frossi@math.unipd.it), University of Padova*
- *Wheeler Ruml (ruml@parc.com), Palo Alto Research Center*

Planning with Temporally Extended Goals using Heuristic Search

Jorge A. Baier and **Sheila McIlraith**
Deptartment of Computer Science, University of Toronto,
Toronto, ON, M5S 3H5, Canada

Abstract

Temporally extended goals (TEGs) refer to properties that must hold over intermediate and/or final states of a plan. Current planners for TEGs prune the search space during planning via goal progression. However, the fastest classical domain-independent planners rely on heuristic search. In this paper we propose a method for planning with propositional TEGs using heuristic search. To this end, we translate an instance of a planning problem with TEGs into an equivalent classical planning problem. With this translation in hand, we exploit heuristic search to determine a plan. We represent TEGs using propositional linear temporal logic which is interpreted over finite sequences of states. Our translation is based on the construction of a nondeterministic finite automaton for the TEG. We prove the correctness of our algorithm and analyze the complexity of the resulting representation. The translator is fully implemented and available. Our approach consistently outperforms existing approaches to planning with TEGs, often by orders of magnitute.

1 Introduction

In this paper we address the problem of generating finite plans for temporally extended goals (TEGs) using heuristic search. TEGs refer to properties that must hold over intermediate and/or final states of a plan. From a practical perspective, TEGs are compelling because they encode many realistic but complex goals that involve properties other than those concerning the final state. Examples include achieving several goals in sequence (e.g., book flight after confirming hotel availability), safety goals (e.g., the door must always be open), and achieving a goal within some number of steps (e.g., at most 3 states after lifting a heavy object, the robot must recharge its batteries).

Planning with TEGs is fundamentally different from using temporally extended domain control knowledge to guide search (e.g., TLPLAN [1], and TALPLAN [11]). TEGs express properties of the plan we want to generate, whereas domain control knowledge expresses general properties of the search for a class of plans [10]. As a consequence, domain control knowledge is generally associated with an additional final-state goal.

A strategy for planning with TEGs, as exemplified by TLPLAN, is to use some sort of blind search on a search space that is constantly pruned by the progression of the temporal goal formula. This works well for safety-oriented goals (e.g., $\Box open(door)$) because it prunes those actions that falsify the goal. Nevertheless, it is less effective with respect to liveness properties such as $\Diamond at(Robot, Home)$. Our objective is to exploit heuristic search to efficiently generate plans with TEGs.

To achieve this, we convert a TEG planning problem into a classical planning problem where the goal is expressed in terms of the final state, and then we use existing heuristic search techniques. An advantage of this approach is that we can use any heuristic planner with the resulting problem.

In contrast to previous approaches, we propose to represent TEGs in f-LTL, a version of propositional linear temporal logic (LTL) [14] which can only be interpreted by finite computations, and is more natural for expressing properties of finite plans. To convert a TEG to a classical planning problem we provide a translation of f-LTL formulae to nondeterministic finite automata (NFA). We prove the correctness of our algorithm. We analyze the space complexity of our translations and suggest techniques to reduce space.

Our translator is fully implemented and available on the Web. It outputs PDDL problem descriptions, which makes our approach amenable to use with a variety of classical planners. We have experimented with the heuristic planner FF [9]. Our experimental results illustrate the significant power heuristic search brings to planning with TEGs. In almost all of our experiments, we outperform existing (non-heuristic) techniques for planning with TEGs.

There are several papers that addressed related issues. First is work that compiles TEGs into classical planning problems such as that of Rintanen [15], and Cresswell and Coddington [3]. Second is work that exploits automata representations of TEGs in order to plan with TEGs, such as Kabanza and Thiébaux's work on TLPLAN [10] and work by Pistore and colleagues [12]. We discuss this work in the final section of this paper.

2 Preliminaries

We represent TEGs using f-LTL logic, a variant of a propositional LTL [14] which we define over *finite* rather than infinite sequences of states. f-LTL formulae augment LTL formulae with the propositional constant final, which is only true in final states of computation. An f-LTL formula over a set $\mathcal{P}$ of propositions is (1) final, true, false, or p, for any $p \in \mathcal{P}$; or (2) $\neg\psi$, $\psi \wedge \chi$, $\bigcirc\psi$, or $\psi \mathsf{U} \chi$, if ψ and χ are f-LTL formulae.

The semantics of an f-LTL formula over $\mathcal{P}$ is defined over finite sequences of states $\sigma = s_0 s_1 \cdots s_n$, such that $s_i \subseteq \mathcal{P}$, for each $i \in \{0, \ldots, n\}$. We denote the suffix $s_i \cdots s_n$ of σ by σ_i. Let φ be an f-LTL formula. We say that $\sigma \models \varphi$ iff $\sigma_0 \models \varphi$. Furthermore,

- $\sigma_i \models$ final iff $i = n$, $\sigma_i \models$ true, $\sigma_i \not\models$ false, and $\sigma_i \models p$ iff $p \in s_i$

- $\sigma_i \models \neg\varphi$ iff $\sigma_i \not\models \varphi$, and $\sigma_i \models \psi \wedge \chi$ iff $\sigma_i \models \psi$ and $\sigma_i \models \chi$.
- $\sigma_i \models \bigcirc\varphi$ iff $i < n$ and $\sigma_{i+1} \models \varphi$.
- $\sigma_i \models \psi \mathsf{U} \chi$ iff there exists a $j \in \{i, \ldots, n\}$ such that $\sigma_j \models \chi$ and for every $k \in \{i, \ldots, j-1\}$, $\sigma_k \models \psi$.

Standard temporal operators such as *always* ($\Box$), *eventually* ($\Diamond$), and *release* (R), and additional binary connectives such as $\vee$, $\supset$ and $\equiv$ can be defined in terms of the basic elements of the language (e.g., $\psi \mathsf{R} \chi \stackrel{\text{def}}{=} \neg(\neg\psi \mathsf{U} \neg\chi)$).

As in LTL, we can rewrite formulae containing U and R in terms of what has to hold true in the "current" state and what has to hold true in the "next" state. The following f-LTL identities are the basis for our translation algorithm.

1. $\psi \mathsf{U} \chi \equiv \chi \vee \psi \wedge \bigcirc(\psi \mathsf{U} \chi)$.
2. $\psi \mathsf{R} \chi \equiv \chi \wedge (\mathsf{final} \vee \psi \vee \bigcirc(\psi \mathsf{R} \chi))$.
3. $\neg\bigcirc\varphi \equiv \mathsf{final} \vee \bigcirc\neg\varphi$.

Identities 2 and 3 explicitly mention the constant final. Those familiar with LTL, will note that identity 3 replaces LTL's equivalence $\neg\bigcirc\varphi \equiv \bigcirc\neg\varphi$. In f-LTL $\bigcirc\varphi$ is true in a state iff there exists a next state that satisfies φ. Since our logic is finite, the last state of each model has no successor, and therefore in such states $\neg\bigcirc\varphi$ holds for every φ.

The expressive power of f-LTL is similar to that of LTL when describing TEGs. Indeed, f-LTL has the advantage that it is tailored to refer to finite plans, and therefore we can express goals that cannot be expressed with LTL. Some examples of TEGs follow.

- $\Box(\mathsf{final} \supset at(Robot, R1))$: In the final state, $at(Robot, R1)$ must hold. This is one way of encoding final-state goals.
- $\Diamond(p \wedge \bigcirc\bigcirc\mathsf{final})$: p must hold true two states before the plan ends. This is an example of a goal that cannot be expressed in LTL, since it does not have the final constant.

Planning Problems A planning problem is a tuple $\langle \mathcal{I}, \mathcal{D}, \mathcal{G} \rangle$, where set $\mathcal{I}$ is the *initial state*, composed by first-order (ground) positive facts; $\mathcal{D}$ is the *domain description*; $\mathcal{G}$ is a TEG in f-LTL.

A domain description is a tuple $\mathcal{D} = \langle \mathcal{C}, \mathcal{R} \rangle$, where $\mathcal{C}$ is a set of *causal rules*, and $\mathcal{R}$ a set of *action precondition rules*. Intuitively, a causal rule defines when a fluent literal becomes true after performing an action. We represent causal rules by the triple $\langle a(\vec{x}), c(\vec{x}), \ell(\vec{x}) \rangle$, where $a(\vec{x})$ is an *action term*, $\ell(\vec{x})$ is a *fluent literal*, and $c(\vec{x})$ is a first-order formula, each of them with free variables among those in $\vec{x}$. Intuitively, $\langle a(\vec{x}), c(\vec{x}), \ell(\vec{x}) \rangle$ expresses that $\ell(\vec{x})$ becomes true after performing action $a(\vec{x})$ in the current state if condition $c(\vec{x})$ holds true. As with ADL operators, the condition $c(\vec{x})$, can contain quantified first-order subformulae. Moreover, ADL operators can be constructed from causal rules and vice versa [13]. Finally, we assume that for each action term and fluent term, there exists at most one positive and one negative causal rule in $\mathcal{C}$. All free variables in rules of $\mathcal{C}$ or $\mathcal{R}$ are regarded as universally quantified.

Regression The causal rules of a domain describe the dynamics of individual fluents. However, to model an NFA in a planning domain, we need also know the dynamics of arbitrary complex formulae, such as for example, the causal rule for $at(o, R_1) \wedge holding(o)$. This is normally accomplished by goal regression [18, 13]. For example, if the following are causal rules for fluents α and β:

$$\langle a, \Phi^+_{a,\alpha}, \alpha \rangle, \quad \langle a, \Phi^-_{a,\alpha}, \neg\alpha \rangle, \quad \langle a, \Phi^+_{a,\beta}, \beta \rangle \quad \langle a, \Phi^-_{a,\beta}, \neg\beta \rangle,$$

we would add the causal rule $\langle a, \Phi^+, \alpha \wedge \beta \rangle$ for $\alpha \wedge \beta$, where $\Phi^+ = \{\Phi^+_{a,\alpha} \wedge \Phi^+_{a,\beta}\} \vee \{\alpha \wedge \neg\Phi^-_{a,\alpha} \wedge \Phi^+_{a,\beta}\} \vee \{\beta \wedge \neg\Phi^-_{a,\beta} \wedge \Phi^+_{a,\alpha}\}$

The size of the resulting causal rule (before simplification) for a boolean combination of fluents can grow exponentially:

Proposition 1 *Let $\varphi = f_0 \wedge f_1 \wedge \ldots \wedge f_n$. Then, assuming no simplifications are made, $|\Phi^+_{a,\varphi}| = \Omega(3^n(m^+ + m^-))$, where $m^+ = \min_i |\Phi^+_{a,f_i}|$, and $m^- = \min_i |\Phi^-_{a,f_i}|$.*

Moreover, the simplification of a boolean formula is also exponential in the size of the formula. Despite this bad news, below we present a technique to reduce the size of the resulting translation for formulae like these.

3 Translating f-LTL to NFA

It is well-known that for every LTL formula φ, there exists a Büchi automaton A_φ, such that it accepts an infinite state sequence σ iff $\sigma \models \varphi$ [17, 16]. To our knowledge, there exists no pragmatic algorithm for translating a finite version of LTL such as the one we use here[1]. To this end, we have designed an algorithm based on the one proposed by Gerth et al. [7]. The automaton generated is a *state-labeled NFA* (SLNFA), i.e. an NFA where states are labeled with formulae. Given a finite state sequence $\sigma = s_0 \ldots s_n$, the automaton goes through a path of states $q_0 \ldots q_n$ iff the formula label of q_i is true in s_i. The automaton accepts σ iff q_n is final.

Space precludes displaying the complete algorithm. Nevertheless, the code is downloadable from the Web[2], and the algorithm is described in detail in [2]. Briefly, there are three main modifications to the algorithm of Gerth et al [7]. First, the generation of successors now takes into account the final constant. Second, the *splitting* of the nodes is done considering f-LTL identities in Section 2 instead of standard LTL identities. Third, the acceptance condition of the automaton is defined using the constant final and the fact that the logic is interpreted over finite sequences of states. We prove that our algorithm is correct:

Theorem 1 *Let A_φ be the automaton built by the algorithm from φ. Then A_φ accepts exactly the set of computations that satisfy φ.*

Simplifying SLNFAs into NFAs Our algorithm often produces automata that are much bigger than the optimal. To simplify it, we have used a modification of the algorithm presented in [5]. This algorithm uses a simulation technique to simplify the automaton. In experiments in [6], it was shown to be slightly better than LTL2AUT [4] at simplifying Büchi automata. The resulting automaton is an NFA, as the ones shown in Figure 1. In contrast to SLNFA, in NFA transitions are labeled with formulae.

Size complexity of the NFA Although simplifications normally reduce the number of states of the NFA significantly, the resulting automaton can be exponential in the size of the formula in the worst case. E.g., for the formula $\Diamond p_1 \wedge \Diamond p_2 \wedge \ldots \wedge \Diamond p_n$, the resulting NFA has 2^n states. Below, we see that this is not a limitation in practice.

[1] In [8], finite automata are built for a $\bigcirc$-free subset of LTL, that does not include the final constant.

[2] http://www.cs.toronto.edu/~jabaier/planning_teg/

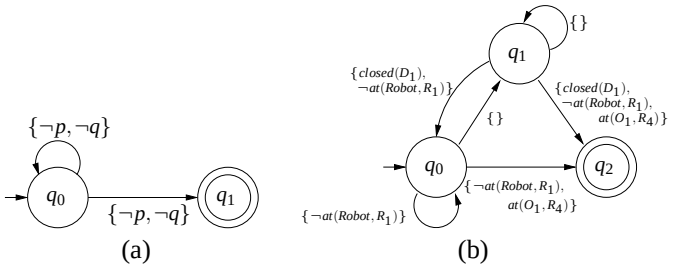


Figure 1: Simplified NFA for (a) $\Box(p \supset \bigcirc q) \wedge \Box(q \supset \bigcirc p)$, and (b) $\Box(at(Robot,R_1) \supset \bigcirc\Diamond closed(D_1)) \wedge \Diamond\Box at(O_1,R_4)$.

4 Compiling NFAs into a Planning Domain

Now that we are able to represent TEGs as NFAs, we show how the NFA can be encoded into a planning problem. Once the NFA is modeled inside the domain, the temporal goal in the newly generated domain in reduced to a property of the final state alone. Intuitively, this property corresponds to the accepting condition of the automaton.

In the rest of the section, we assume the following. We start with a planning problem $L = \langle \mathcal{I}, \mathcal{D}, \mathcal{G} \rangle$, where $\mathcal{G}$ is a TEG in f-LTL. The NFA $A_\mathcal{G} = (Q, \Sigma, \delta, Q_0, F)$ is built for $\mathcal{G}$. We denote by $\lambda_{p,q}$ the formula $\bigvee_{(q,L,p)\in\delta} \bigwedge L$. E.g., in Fig. 1(b), $\lambda_{q_1,q_0} = closed(D_1) \wedge \neg at(Robot,R_1)$. Finally, we denote by $Pred(q)$ the states that are predecessors of q.

In the planning domain, each state of the NFA is represented by a fluent. For each state q we add to the domain a new fluent E_q. The translation is such that if sequence of actions $a_1 a_2 \cdots a_n$ is performed in state s_0, generating the succession of states $\sigma = s_0 s_1 \ldots s_n$, then E_q is true in s_n if and only if there is a run of $A_\mathcal{G}$ on σ that ends in state q.

For each fluent E_q we generate a new set of causal rules. New rules are added to the set $\mathcal{C}'$, which is initialized to $\emptyset$.

For each action a, we add to $\mathcal{C}'$ the causal rules $\langle a, \Phi^+_{a,E_q}, E_q \rangle$ and $\langle a, \Phi^-_{a,E_q}, \neg E_q \rangle$ where:

$$\Phi^+_{a,E_q} = \bigvee_{p\in Pred(q)\setminus\{q\}} E_p \wedge (\Phi^+_{a,\lambda_{p,q}} \vee (\lambda_{p,q} \wedge \neg\Phi^-_{a,\lambda_{p,q}})),$$
$$\Phi^-_{a,E_q} = \neg\Phi^+_{a,E_q} \wedge \neg(\Phi^+_{q,\lambda_{q,q}} \vee \lambda_{q,q} \wedge \neg\Phi^-_{a,\lambda_{q,q}}).$$

where $\Phi^+_{a,\lambda_{p,q}}$ (resp. $\Phi^-_{a,\lambda_{p,q}}$) is the condition under which a makes $\lambda_{p,q}$ true (resp. false). Both formulae must be obtained via regression. Formula $\lambda_{q,q}$ is false if there is no self transition in q.

The initial state must give an account of which fluents E_q are initially true. The new set of facts $\mathcal{I}'$ is the following $\mathcal{I}' = \{E_q \,|\, (p,L,q) \in \delta, p \in Q_0, L \subseteq \mathcal{I}\}$.

Intuitively, the automaton $A_\mathcal{G}$ accepts iff the temporally extended goal $\mathcal{G}$ is satisfied. Therefore, the new goal, $\mathcal{G}' = \bigvee_{p\in F} E_p$, is defined according to the acceptance condition of the NFA. The final planning problem L' is $\langle \mathcal{I} \cup \mathcal{I}', \mathcal{C} \cup \mathcal{C}', \mathcal{R}, \mathcal{G}' \rangle$.

Size complexity The size of the translated domain has a direct influence on how hard it is to plan with that domain. We can prove that the size of the translated domain is $O(n|Q|2^\ell)$, where ℓ is the maximum size of a transition in $A_\mathcal{G}$, n is the number of action terms in the domain, and $|Q|$ is the number of states of the automaton.

Prb. #	Comp. time	No. Sts.	$\|\mathcal{G}\|$	FF t	FF ℓ	TLPLAN t	TLPLAN t-ctrl	TLPLAN ℓ
1	.02	2	6	.02	6	.07	.01	6
2	.02	2	6	.02	8	.04	.03	8
3	.09	15	21	.04	10	.20	.02	10
4	.06	5	12	.03	6	.38	.10	6
5	.07	6	21	.04	15	.5	.19	13
6	.49	37	71	.19	16	.51	.17	18
7	.05	6	21	.05	9	.96	.31	10
8	.07	15	9	.05	10	1.40	.04	10
9	.01	4	11	.03	18	13.90	.15	14
10	.04	6	12	.07	32	17.52	.40	14
11	.08	5	23	.06	22	m	m	–
12	.09	5	25	.50	25	m	m	–
13	.09	6	15	m	–	m	m	–
14	.32	5	31	m	–	m	m	–
15	.07	5	18	.11	31	m	m	–
16	.09	10	22	m	–	m	m	–

(a)

Prb.	FF t	FF ℓ	TPBA+C t	TPBA+C ℓ
1	0.02	2	0.24	2
2	0.02	5	0.96	5
3	0.01	5	1.3	5
4	0.02	7	3.29	7
5	0.02	10	11.66	10
6	0.02	12	28.87	12
7	0.02	15	82.57	15
8	0.02	19	35.69	17
9	0.02	21	13.37	20
10	0.23	52	126.25	35
11	0.07	54	m	–
12	0.23	47	m	–
13	0.54	72	m	–
14	4.03	82	m	–
15	11.19	95	m	–

(b)

Table 1: Our approach compared to TLPLAN (a) and search control with Büchi automata (b)

Reducing $|Q|$ We previously saw that $|Q|$ can be exponential in the size of the formula. Fortunately, there is a workaround. Consider for example the formula $\varphi = \Diamond p_1 \wedge \ldots \wedge \Diamond p_n$, which we know has an exponential NFA. φ is satisfied if each of the conjuncts $\Diamond p_i$ is satisfied. Instead of generating a unique NFA for φ we generate different NFA *for each* $\Diamond p_i$. Then we plan for a goal equivalent to the conjunction of the acceptance conditions of each of those automata. This generalizes to any combination of boolean formulae.

5 Implementation and Experiments

We implemented a compiler that given a domain and a f-LTL TEG, generates a classical planning problem following Section 4. The compiler can further convert the new problem into a PDDL domain/problem, thereby enabling its use with a variety of available planners.

We conducted several experiments in the Robots Domain [1] to test the effectiveness of our approach. In each experiment, we compiled the planning problem to PDDL. To evaluate the translation we used the FF planner.

Table 1(a) presents results obtained for various temporal goals by our translation and TLPLAN. The second column shows the time taken by the translation, the third shows the number of states of the automata representing the goal, and the fourth shows the size of the goal formula, $|\mathcal{G}|$. The rest of the columns show the time (t) and length (ℓ) of the plans for each approach. In the case of the TLPLAN, two times are presented. In the first (t) no additional search control was added to the planner, i.e. the planner was using only the goal to prune the search space. In the second (t-ctrl) we added (by hand) additional control information to "help" TLPLAN do a better search. The character 'm' stands for *ran out of memory*.

Our approach significantly outperformed TLPLAN. TLPLAN is only competitive in very simple cases. In most cases, our approach is one or two orders of magnitude faster than TLPLAN. Moreover, the number of automata states is comparable to the size of the goal formula, which illustrates that our approach does not blow up easily for natural TEGs. We also observe that FF cannot solve all problems. This is because FF transforms the domain to a STRIPS problem, and tends to blow up when conditional effects contain large for-

mulae. This problem, can be overcome if one uses *derived predicates* in the translation as proposed in [2].

Table 1(b) compares our approach's performance to that of the planner presented in [10] (henceforth, TPBA), which uses Büchi automata to control search. In this case we used goals of the form $\Diamond(p_1 \wedge \bigcirc(\Diamond p_2 \wedge \ldots \wedge \bigcirc\Diamond p_n)\ldots)$, which is one of the four goal templates supported by this planner. Again, our approach significantly outperforms TPBA, even in the presence of extra control information added by hand (this is indicated by the '+c' in the table).

The results presented above are not surprising. None of the planners we have compared to uses heuristic search, which means they may not have enough information to determine which action to choose during search. The TLPLAN family of planners is particularly efficient when control information is added to the planner. Usually this information is added by an expert in the planning domain. However, control information, while natural for classical goals, may be hard to write for temporally extended goals. The advantage of our approach is that we do not need to write this information to be efficient. Moreover, control information can also be added in the context of our approach by integrating it into the goal formula.

6 Discussion and Related Work

In this paper we proposed a method to generate plans for TEGs using heuristic search. We proposed a translation method that takes as input a planning problem with an f-LTL TEG and produces a classical planning problem. Experimental results demonstrate that our approach outperforms—often by several orders of magnitude—existing (non-heuristic) planners for TEGs in the Robots Domain. [2]. Our approach is limited to propositional TEGs. In [2] we show how we can extend it to capture a compelling subset of first-order f-LTL. We also provide analogous performance results on multiple domains.

There are several notable pieces of related work. TPBA, the temporal extension of TLPLAN that uses search control, and that we use in our experiments [10], constructs a Büchi automaton to represent the goal. It then uses the automaton to guide planning by following a path in its graph from an initial to final state, setting transition labels as subgoals, and backtracking as necessary.

Approaches for planning as symbolic model checking have also used automata to encode the goals (e.g. [12]). These approaches use different languages for extended goals, and are not heuristic.

In [3] a translation of LTL formulae to PDDL has been proposed. They translate LTL formulae to a *deterministic* finite state machine (FSM). The FSM is generated by successive applications of the *progress* operator of [1] to the TEG. The use of deterministic automata makes it prone to exponential blowup even with simple goals, e.g., $\Diamond(p \wedge \bigcirc^n q)$. The authors' code was unavailable for comparison with our work. Nevertheless, they report that their technique is no more efficient than TLPLAN, so we infer that our approach has better performance.

Finally, [15] proposes a translation of a subset of LTL into a set of ADL operators. Their translation does not use automata, and therefore is limited to a small subset of LTL.

Acknowledgments We are extremely grateful to Froduald Kabanza, who provided the TPBA code for comparison. We also wish to thank Sylvie Thiébaux, Fahiem Bacchus, and the anonymous reviewers for insightful comments on this work. Finally, we gratefully acknowledge funding from NSERC Research Grant 229717-04.

References

[1] F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Ann. of Math Art. Int.*, 22(1-2):5–27, 1998.

[2] J. A. Baier and S. McIlraith. Planning with first-order temporally extended goals using heuristic search. Forthcoming.

[3] S. Cresswell and A. Coddington. Compilation of LTL goal formulas into PDDL. In *ECAI-04*, pages 985–986, 2004.

[4] M. Daniele, F. Giunchiglia, and M. Y. Vardi. Improved automata generation for linear temporal logic. In *Proc. CAV-99*, volume 1633 of *LNCS*, pages 249–260, Trento, Italy, 1999.

[5] K. Etessami and G. J. Holzmann. Optimizing büchi automata. In *Proc. CONCUR-2000*, pages 153–167, 2000.

[6] C. Fritz. Constructing Büchi automata from ltl using simulation relations for alternating büchi automata. In *Proc. CIAA 2003*, volume 2759 of *LNCS*, pages 35–48, 2003.

[7] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV-95*, pages 3–18, 1995.

[8] Dimitra Giannakopoulou and Klaus Havelund. Automata-based verification of temporal properties on running programs. In *Proc. ASE-01*, pages 412–416, 2001.

[9] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Art. Int. Research*, 14:253–302, 2001.

[10] F. Kabanza and S. Thiébaux. Search control in planning for temporally extended goals. In *Proc. ICAPS-05*, 2005.

[11] J. Kvarnström and P. Doherty. Talplanner: A temporal logic based forward chaining planner. *Ann. of Math Art. Int.*, 30(1-4):119–169, 2000.

[12] U. Dal Lago, M. Pistore, and P. Traverso. Planning with a language for extended goals. In *Proc. AAAI/IAAI*, pages 447–454, 2002.

[13] E. P. D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. KR-89*, pages 324–332, 1989.

[14] A. Pnueli. The temporal logic of programs. In *Proc. FOCS-77*, pages 46–57, 1977.

[15] J. Rintanen. Incorporation of temporal logic control into plan operators. In *Proc. ECAI-00*, pages 526–530, 2000.

[16] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, volume 1043 of *LNCS*, pages 238–266. Springer, 1995.

[17] M. Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[18] R. Waldinger. Achieving several goals simultaneously. In *Mach. Intel. 8*, pages 94–136. Ellis Horwood, 1977.

Continuous Planning for the Control of an Autonomous Agile Satellite

Grégory Beaumet
Supaéro, ONERA, CNES
2, avenue Édouard Belin 31055 Toulouse cedex 4 FRANCE
gregory.beaumet@onera.fr

Abstract

Because of the complexity of the encountered problems, controlling autonomous satellites is an interesting field for the AI research community. This document introduces the current thesis about planning the activities of an agile autonomous Earth-observing satellite.

Application Domain

Mission

The application domain of the thesis is an ONERA-CNES project of development of a ground demonstrator of an autonomous satellite (AGATA project, (Charmeau & Bensana 2005)). In this project we consider an Earth-observing mission. A satellite on a heliosynchronous low circular orbit around the Earth aims to acquire images of specified areas on the Earth surface, and to download them to one or more ground mission centers.

Agile satellite

The satellite that we consider is an agile satellite, like the Pleiades satellites (Boussarie & Boissin 2006), able to operate freely and quickly along the three axes of rolling, lacing and pitching (Figure 1) thanks to a cluster of gyroscopic actuators.

Figure 1: A Pleiades satellite

This satellite is equipped with (1) an optical high-resolution instrument to acquire images, (2) a cloud cover detection instrument, (3) a radio antenna allowing the satellite to download the observation data, and (4) solar generators and batteries producing and storing electric energy. These components are not mobile: they are fixed to the satellite. Finally, the satellite has a fixed size mass memory to save the detection and observation data.

Planning the activities

Currently, the activities of the Spot satellites are planned offline, as will be those of the Pleiades satellites: the ground mission center builds plans over a horizon of 24 hours and downloads them daily to the satellites. These plans are very precise: fixed schedule of activities with fixed starting times. They are executed without any possibility for replanning.

Spacecraft Autonomy

Potential advantages of autonomy

During a revolution period round the Earth, an Earth-observing satellite has limited visibility windows with the ground stations. Autonomy would allow the satellite to make decisions between two visibility windows in order to react to unforeseen events such as:

- **subsystem failure**. The autonomy allows the system to react immediatly if a failure arises during the execution of a task.
- **unexpected level of resources**. Some actions of the satellite have nondeterministic effects on the consumption or the production of the onboard resources: for example, it is impossible to foresee the quality of an image and its compression rate before its realization, and thus to know the memory space it will use. Autonomy would make the satellite able to make decisions by knowing the actual current state of the onboard resources.
- **unexpected cloud cover**. The detection instrument may detect a cloud cover different from that provided by the weather forecast, authorizing or preventing some observations. Because the detection can be performed by pointing the satellite 30 degrees ahead, it must decide autonomously within a few seconds wether to add or remove these observations from its actions plan.

State of the art

The EO-1 Autonomous Science Agent This software enables the Earth-Observing One (EO-1) spacecraft to autonomously detect and respond to science events occuring

on the Earth. It is organized into a traditional three-layer architecture. At the highest level of abstraction, the Continuous Activity Scheduling Planning Execution and Replanning (CASPER) software is responsible for mission planning functions. CASPER uses a local search approach (Rabideau *et al.* 1999) to develop operations plans. It plans within limited CPU resources by using a hierarchical, continuous (Chien *et al.* 2000) planning paradigm. Rather than attempt to plan out an entire week of operations in a single batch timeslice, it utilizes a long-term, more abstract plan for the longest planning horizon (one week), and plans at a detailed level for the next day of operations. As time proceeds forward, it incrementally replans for the new observations that fall within this one-day horizon.

Non-agile satellite The work presented by S. Damiani (Damiani 2005; Damiani, Verfaillie, & Charmeau 2005) allowed us to design, implement and test successfully an autonomous decision mechanism onboard a non-agile satellite. It is supported by a permanently active planning module, reasoning on more and more complex problems to improve quality of the proposed decisions, using all the time it has at its disposal, but able to provide a realizable decision at any time, even if it is not necessarily optimal according to the principles of the anytime algorithms (Zilberstein 1996).

Application to an agile satellite The observation instrument of a non-agile Earth-watching satellite like Spot is permanently pointed under the satellite, and a mobile mirror in front of it allows it to observe ground areas laterally. The starting times of observations are thus fixed.

On the contrary an agile satellite is able to bring forward or delay the starting time of an observation by a simple change of its attitude; then the observations have starting time windows which relax planning but make the selection and scheduling of observations significantly more difficult, due to the larger search space for potential solutions (Figure 2).

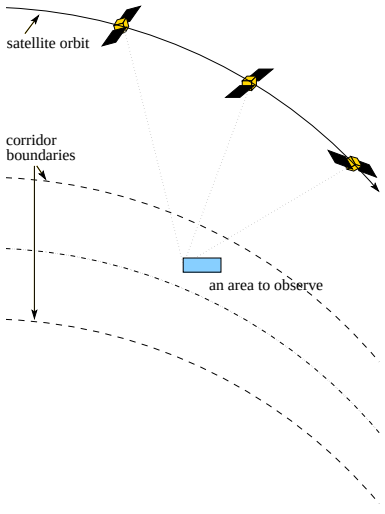


Figure 2: Three possible attitudes of an agile satellite for starting an observation

Some works (Lemaître *et al.* 2002) deal with the problem of offline selecting and scheduling observations of agile satellites. They present different methods which have been investigated in order to solve a simplified version of the complete problem: a greedy algorithm, a dynamic programming algorithm, a constraint programming approach and a local search method.

However, these works deal with the planning of the observations independently of the other activities of the satellite (cloud cover detection, data downloading...). But many activities of an agile satellite need to control its attitude which can be seen as a shared resource. For example, an observation cannot be executed in parallel of a cloud cover detection which requires an orientation of the satellite 30 degrees ahead. Thus it becomes necessary to plan together all the activities controlling the attitude of the satellite.

Contribution

Objective

This study aims to extend the work developed by S. Damiani (Damiani 2005) to an agile satellite: permanent planning task of all the activities of the satellite by using as well as possible the time available to reason.

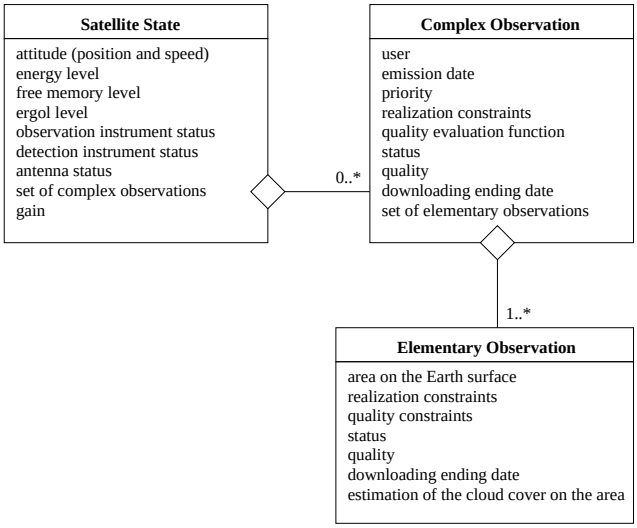


Figure 3: Model of the current state of the system

Achieved work

List of possible activities We distinguish two categories of activities realizable by the satellite: activities with controlled attitude trajectory during which the attitude of the satellite, in position and speed, is entirely determined, and activities with uncontrolled attitude trajectory which can be executed parallel with the other activities.

We listed seven activities with controlled attitude trajectory: the *observation* of an area on the Earth surface, the *detection* of the cloud cover in front of the satellite, the *recharge* of the batteries (to point the solar panels to the sun), the *downloading* of observation data (to point the satellite

to the ground station), a *change in attitude*, an *orbital manoeuvre* (to correct the orbit of the satellite if necessary), a *geocentric pointing* (when the satellite "does nothing" or is in safety mode) and three activities with uncontrolled attitude trajectory: the *parallel downloading* of observation data, the *analysis* of the results of an observation (to evaluate the quality of an image, compress or delete the saved image), the *analysis* of the results of a detection (to evaluate the cloud cover).

Model The first step of our work consisted in modelling the decision problem by using the PDDL language and its extension to the durative actions (Fox & Long 2003): the current state of the system (Figure 3), the various actions realizable by the satellite, their preconditions, their effects (deterministic or not) on the state of the system and on the satisfaction of the objectives.

The figure 4 presents the model of the "observe" action using the PDDL language. The satellite starts watching the area $?o_i$ at the date $?t_s$.

```
(: durative-action observe
 : parameters (?o_i - observation ?t_s - date)
 : duration (= ?duration (observationDuration ?o_i ?t_s))
 : condition (and (at start (= (status ?o_i) notAcquired))
                  (at start (visible ?o_i))
                  (at start (= attitude (obsStartAttitude ?o_i ?t_s)))
                  (at start (= obsInstrStatus available))
                  (at start (not assignedAttitude))
                  (at start (≥ energy (energyConsum ?o_i ?t_s)))
                  (at start (≥ memory (memoryConsum ?o_i)))
                  (over all (visible ?o_i))
                  (over all (= obsInstrStatus used))
                  (over all (assignedAttitude))
                  (over all (≥ energy 0))
                  (over all (≥ memory 0))
                  (at end (= obsInstrStatus used))
                  (at end (assignedAttitude))
                  (at end (visible ?o_i))
                  (at end (≥ energy 0))
                  (at end (≥ memory 0))

 : effect (and (at start (decrease energy (energyConsum ?o_i ?t_s)))
               (at start (decrease memory (memoryConsum ?o_i)))
               (at start (assign obsInstrStatus used))
               (at start (assignedAttitude))
               (at end (assign attitude (obsEndAttitude ?o_i ?t_s)))
               (at end (assign (status ?o_i) acquired))
               (at end (assign obsInstrStatus available))
               (at end (not assignedAttitude))
               (at end (increase energy (energyProd ?o_i ?t_s)))
          )
)
```

Figure 4: Model of the "observe" action

Current work

To plan online, we need to estimate (1) the duration of each activity of the satellite, depending on its starting time and on the attitude profile of the satellite and (2) the production and the consumption of energy and memory for each activity of the satellite. For that, we try to compare two different methods : an analytical one using a simplified model of the satellite kinematics, and a learning one based on the use of neural networks to approximate the quantities of interest.

Future work

We plan to solve, initially offline, this problem of planning with a dynamic programming approach like this one used by S. Damiani (Damiani 2005), then with a local search method. A second step will consist in adapting the algorithms to a mode of anytime reasoning in order to be able to use them online.

References

Boussarie, E., and Boissin, B. 2006. Pléiades - Dual Optical System for Metric Resolution Observations. http://smsc.cnes.fr/PLEIADES/.

Charmeau, M.-C., and Bensana, E. 2005. AGATA: A Lab Bench Project for Spacecraft Autonomy. In *8th International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS-05)*.

Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and G., R. 2000. Using iterative repair to improve responsiveness of planning and scheduling. In *Fifth International Conference on Artificial Intelligence Planning and Scheduling*.

Damiani, S.; Verfaillie, G.; and Charmeau, M.-C. 2005. An earth watching satellite constellation : How to manage a team of watching agents with limited communications. In *Fourth International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2005)*, volume 1, 455–462.

Damiani, S. 2005. *Gestion d'une constellation de satellites de surveillance de la Terre : autonomie et coordination*. Ph.D. Dissertation, Supaéro, Toulouse, France.

Fox, M., and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research (JAIR)* 20:61–124.

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.; Knight, R.; Chien, S.; Fukunaga, A.; and Govindjee, A. 1999. Iterative repair planning for spacecraft operations in the aspen system. In *International Symposium on Artificial Intelligence Robotics and Automation in Space*.

Zilberstein, S. 1996. Using anytime algorithms in intelligent systems. *AI Magazine* 17(3):73–83.

Solving Goal Utility Dependencies* and Simple Preferences in Partial Satisfaction Planning

J. Benton
Computer Sci. & Eng. Dept.
Arizona State University
Tempe, AZ 85287
j.benton@asu.edu

Abstract

This doctoral research focuses on the analysis and development of techniques for solving domain-independent partial satisfaction planning (PSP) problems and planning problems with preferences. Recently, these areas have gained the attention of the planning community. This has been underscored by the recent introduction of preferences to the Fifth International Planning Competition (IPC5). This extended abstract outlines contributions made to the area of PSP and shows that planning with "simple preferences", as defined in the IPC5, is can be compiled to PSP. Afterwards, it outlines future steps to be taken for advancing this line of research.

Introduction

In many real world problems, users prefer some goals over others. In this sense, they have preferences among specified "soft" goals. For instance, a user may prefer brown flour over white flour but if white flour is all that is available, the user will accept it. Goal preferences like this may also be balanced with the cost of achieving the goal. For example, if brown flour costs more than some measurable utility it brings, and white flour costs less than its utility, then white flour is the obvious choice despite the preference for brown flour. This type of problem is called a partial satisfaction planning (PSP) problem (van den Briel *et al.* 2004) and provides a starting point for this work. Work on this type of planning has been given recent attention (van den Briel *et al.* 2004; Do & Kambhampati 2004; Smith 2004; Brafman & Chernyavsky 2005; Nigenda & Kambhampati 2005; Benton, Do, & Kambhampati 2005). The primary focus of this doctoral research is to extend the expressiveness of this type of planning by using the current state of the art planning graph heuristics as a means for solving these problems.

The most recent work on this doctoral thesis has focused on extending PSP into handling goals with utility dependencies. That is, some goals are worth much more or less in conjunction with other goals. For instance, having both a left and right shoe is worth much more than having just one or the other and having two books on the same subject is worth less than the sum of having either book independently.[1] Another recent extension is the ability to handle "simple preferences" as defined by PDDL3 in the 5^{th} International Planning Competition. It turns out that planning problems with preferences defined in this way are very similar to PSP problems.

The rest of this extended abstract is organized as follows. First, we motivate the need for representing and handling goal utility dependencies in PSP and provide a framework of representing them using the General Additive Independence (GAI) model (Bacchus & Grove 1995) and give an outline of heuristic methods for handling them. To show the effectiveness of our framework, we provide empirical results on some benchmark planning domains. We then briefly outline the method of generating a PSP problem from "simple preferences" defined in PDDL3. Afterwards, we discuss future work.

Goal Utility Dependency

Classical planning problems define each goal as a member of a conjunctive set that must be satisfied at a plan's end. In partial satisfaction planning (PSP) we relax the constraint of ending a plan with every goal satisfied. Instead we define soft goals and provide each with a numeric utility value. This allows the planner to solve for a subset of the goals. We also attribute to each action a numeric cost. The planner then aims to find a plan with the best *net benefit*, where net benefit is defined as the difference between the satisfied goal utility and the action costs.

The process of finding plans in PSP is complicated by two types of dependencies between goals: (i) A set of goals may have *cost dependencies* in that there are dependencies among the plans to achieve them (making the cost of achieving them together significantly more or less than the total cost of achieving them in isolation) (ii) A set of goals may have *utility dependencies* in that achieving the goals together may lead to significantly different utility than the sum of

*Joint work with Minh Do, Palo Alto Research Center, Palo Alto, CA and Subbarao Kambhampati, Arizona State University, Tempe, AZ

[1] These are examples of mutual dependency. There is also the idea of conditional dependency, in which the utility of having one item is conditional on whether we have the other item. The difference is subtle, but the general idea is that conditional dependency is based upon an "if" relationship rather than an "and" relationship. Also note that the "and" relationship is more general and can be used to represent an "if" relationship by listing possible goal combinations.

achieving individual goals. Part of this dissertation work is on investigating heuristic approaches to handle both utility and cost dependencies together in PSP.

We have developed an approach for representing these utility dependencies between planning goals using the *Generalized Additive Independence (GAI)* model (Bacchus & Grove 1995) and a planning algorithm based on forward search that solves this extended PSP problem. The algorithm is based on the forward heuristic search described in the *Sapa*$^{\mathcal{PS}}$ planner (van den Briel *et al.* 2004). The main innovation is our heuristic, which is able to take into account both goal utility and goal achievement cost dependencies.

Problem Formulation & Heuristics

A classical planning problem is a 4-tuple $\langle F, I, G, A\rangle$ where: F is a set of predicate symbols representing state facts; I is the initial state, completely defined by predicates in F; G is a goal state, which is partially defined by a set of predicates in F; A is a set of actions with $a \in A$ is defined by pre and post-conditions $Precond(a), Effect(a) \subseteq F$. The plan is a sequence of actions in A such that, when executed from I, will achieve all goals $g \in G$. In PSP (Smith 2004; van den Briel *et al.* 2004), goals $g \in G$ have utility values $u_g \geq 0$, representing how much each goal is worth to a user, and each action $a \in A$ has an associated positive execution cost c_a. Moreover, not all goals in G need to be achieved. Let P be the lowest cost plan that achieves a subset $G' \subseteq G$ of those goals. The objective is to maximize the tradeoff between total utility $U(G')$ of G' and total cost of actions $a \in P$.

Work on PSP until now assumed that goals have no utility dependencies and thus their utilities are additive: $U(G') = \Sigma_{g \in G'} u_g$. To represent the goal utility dependencies we adopt the *Generalized Additive Independence(GAI)* model (Bacchus & Grove 1995). We named the PSP problem with utility dependencies represented by GAI model *PSP*$^{\mathcal{UD}}$. We chose this model because it is simple, intuitive and expressive. It also is more general than other commonly used models such as CP-Net (Brafman & Chernyavsky 2005) or UCP-Net (Boutilier *et al.* 2001). Because of this, representing goals specified using GAI may result in a problem size increase in comparison with these other modeling methods. However, its generality allows problem specification to be more straightforward for the user (i.e. there are no "inferred" utility values). A cost propagation process is used on the planning graph to estimate the achievement cost for each individual goal. After the propagation process is done we have an estimated cost $c(g)$ for each goal $g \in G$. As shown in (Do & Kambhampati 2001), if we use *max* propagation, then $c(g)$ will underestimate the cost to achieve g while there is no such guarantee for *sum* propagation.

The max family of heuristics tend to perform badly in practice. Therefore, we use an alternative approach of utilizing the relaxed plan employed by *Sapa*$^{\mathcal{PS}}$ for PSP[2]. For each state S explored in a progression planner, after building the relaxed planning graph and doing forward cost propagation on the graph, we extract a relaxed plan RP to support a subset of goals $G' \subseteq G$. Let $RP(G')$ be the relaxed plan with highest net-benefit value among those achieving $G' \subseteq G$, the relaxed plan heuristic for *PSP*$^{\mathcal{UD}}$ is:

$$h_{relax} = max_{G' \subseteq G} \left(u(G') - \sum_{a \in RP(G')} c_a\right) \qquad (1)$$

To capture the mutual cost dependencies between the goal achievement costs (i.e. cost dependencies), we find the set of actions shared between different partial plans achieving different goals. This allows the generation of $GS(a)$ which specifies the set of goals for which the action a contributes.

Given the utility dependencies represented by GAI local functions f^u and the goal achievement cost dependencies represented by goal supporting action set $GS(a)$, we set up an ILP encoding for h_{relax}. The purpose of this encoding is to capture the set of goals $G' \subseteq G$ that gives the maximum tradeoff between utility of G' and the cost of actions in the relaxed plan supporting G'.

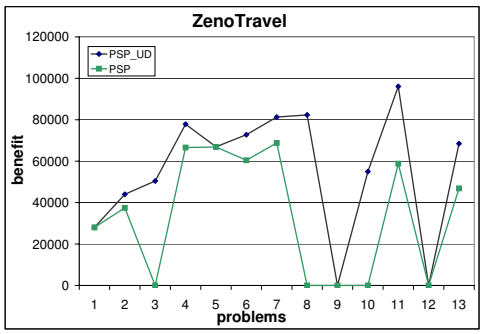


Figure 1: *Sapa*$^{\mathcal{PS}}_{\mathcal{UD}}$ and *Sapa*$^{\mathcal{PS}}$ in ZenoTravel domain.

Results

We have implemented the heuristic search algorithm for PSP problems discussed in this paper on top of the *Sapa*$^{\mathcal{PS}}$ planner. We call the new planner *Sapa*$^{\mathcal{PS}}_{\mathcal{UD}}$ and tested it on two sets of random *ZenoTravel* and *Satellite* problems. These problems were generated on top of the problem sets used in the Third International Planning Competition (Long & Fox 2003).

All tests were run using a Pentium IV 2.66GHz with 1GB RAM and a 1200 second time limit. Because A^*_{PSP} continuously finds better solutions given more time (or the termination node is found), the results reported in this section represent the plan with the highest benefit value found within the time limit. For solving the ILP encoding, we use the C version of `lp_solve ver5.5` software, a free solver, with a Java wrapper.

While *Sapa*$^{\mathcal{PS}}_{\mathcal{UD}}$ is sensitive to both cost and utility dependencies, *Sapa*$^{\mathcal{PS}}$ only accounts for cost dependencies. The empirical evaluation is designed to test whether *Sapa*$^{\mathcal{PS}}_{\mathcal{UD}}$ is able to solve the *PSP*$^{\mathcal{UD}}$ problems more effectively (i.e. with higher net benefit). Figure 1 and 2 show the comparison between those two planners.

PDDL3 "Simple Preferences" to PSP

The Fifth International Planning Competition defined preferences as a new language feature for PDDL3(Gerevini &

[2]Variants of this approach are also used in several other PSP planners such as *AltAlt*^{ps} (van den Briel *et al.* 2004; Nigenda & Kambhampati 2005) or the orienteering planner (Smith 2004).

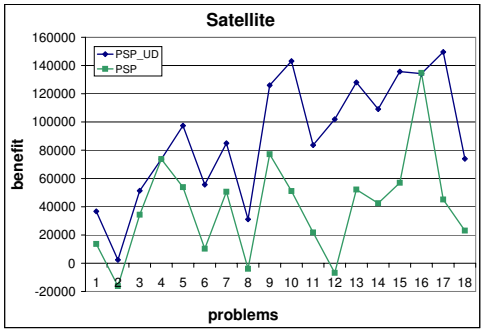


Figure 2: $Sapa^{\mathcal{PS}}_{\mathcal{UD}}$ and $Sapa^{\mathcal{PS}}$ in Satellite domain.

Long 2005). This feature allows domain modelers to express soft constraints on action conditions and goals. Each preference is given a name and an associated violation count. This count can then be used as part of a metric specifying how to measure the quality of the resulting plan.

In the planning competition, the "simple preferences" category of domains specifies preferences and plan metrics in a manner that allows problems to be converted from PDDL3 to PSP. The domains in this category define preferences on actions as well as goals. An example is the *drive* action of the trucks domain:

```
(:action drive
 :parameters
   (?t - truck ?from ?to - place)
 :precondition (and
   (at ?t ?from) (connected ?from ?to)
   (preference p-drive (and
    (ready-to-load goods1 ?from level0)
    (ready-to-load goods2 ?from level0)
    (ready-to-load goods3 ?from level0))
    ))
 :effect (and (not (at ?t ?from))
              (at ?t ?to)))
```

A plan metric assigns a weight to this preference in the following manner:

```
(:metric (+ (* 10 (is-violated p-drive))
))
```

A domain specified in this way can be compiled into a PSP problem (Benton, Kambhampati, & Do 2006). This is done by generating an action for each preference combination on the original action. The cost of executing the action is equal to the cost of not satisfying the preferences excluded from the action definition. Preferences on goals are handled similarly except actions provide a "has preference" goal with a utility that matches the cost of not having the preference.

Conclusion & Future Work

In this extended abstract, we discussed a framework of solving partial satisfaction planning (PSP) problems with utility dependencies and a way to handle IPC5 problems with "simple preferences" by compiling them to PSP problems. The former methods show that there exists expressive power in combining problem heuristics with declarative formulations. In this case, re-formulating the relaxed plan as an ILP allows us to impose more constraints that cannot easily be handled procedurally. Though there exists additional computational cost for generating the new heuristics, this is offset by the extra guidance achieved toward better quality plans. This line of reasoning is especially important to consider as the planning community's concerns have begun to focus on finding plans of quality as evidenced with the Fifth International Planning Competition (Gerevini & Long 2005).

For the future, this dissertation work will extend the heuristic and search architectures used to solve utility dependencies for dealing with trajectory preferences and constraints in PDDL3 (Gerevini & Long 2005). The idea is to include in the ILP formulation information about the time points on which actions are executed in the relaxed plan so that we may find good estimates of the best action orderings. We also plan on extending this work to take more negative information into account, following the example of AltWlt (Nigenda & Kambhampati 2005).

References

Bacchus, F., and Grove, A. 1995. Graphical models for preference and utility. In *Proc. of UAI-95*.

Benton, J.; Do, M. B.; and Kambhampati, S. 2005. Over-subscription planning with numeric goals. In *IJCAI*, 1207–1213.

Benton, J.; Kambhampati, S.; and Do, M. B. 2006. Yochan$\sqrt{^{\int}}$: Pddl3 simple preferences as partial satisfaction planning. In *Fifth International Planning Competition Extended Abstract Booklet*.

Boutilier, C.; Brafman, R.; Hoos, H.; ; and Poole, D. 2001. Reasoning with conditional ceteris paribus preference statements. In *Proc. of UAI-2001*.

Brafman, R. I., and Chernyavsky, Y. 2005. Planning with goal preferences and constraints. In *ICAPS*.

Do, M., and Kambhampati, S. 2001. Sapa: A domain-independent heuristic metric temporal planner. In *Proc. of ECP-01*.

Do, M., and Kambhampati, S. 2004. Partial satisfaction (over-subscription) planning as heuristic search. In *Knowledge Based Computer Systems*.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3: The language of the fifth international planning competition. Technical report, University of Brescia, Italy.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*.

Nigenda, R. S., and Kambhampati, S. 2005. Planning graph heuristics for selecting objectives in over-subscription planning problems. In *ICAPS*.

Smith, D. 2004. Choosing objectives in over-subscription planning. In *Proc. of ICAPS-04*.

van den Briel, M.; Sanchez, R.; Do, M. B.; and Kambhampati, S. 2004. Effective approaches for partial satisfaction (over-subscription) planning. In *Proc. of AAAI-04*.

Sequential Monte Carlo in Probabilistic Planning Reachability Heuristics

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

Abstract

In this work, we apply heuristic search to conformant probabilistic planning by adapting planning graph heuristics developed for non-deterministic planning. We consider a straight-forward application of these planning graph techniques, which amounts to exactly computing the distribution over reachable relaxed planning graph layers. In reality, computing these distributions is costly, so we apply Sequential Monte Carlo to approximate them. We demonstrate on several domains how our approach enables our planner to far outscale existing (optimal) probabilistic planners and still find reasonable quality solutions.

This work adds to a much broader dissertation on improving the scalability of planning under uncertainty. We have designed heuristics for conformant and conditional non-deterministic planning, in addition to this latest addition to probabilistic planning. Future work will build on these heuristic techniques to address conditional probabilistic planning and devise new search algorithms for probabilistic planning.

Introduction

We address the problem of conformant probabilistic planning, where the planning agent has no observability and must formulate plans with uncertain actions. A conformant plan is a sequence of actions that will guarantee goal satisfaction with some probability. While agents can potentially improve their success by devising conditional plans (that use observations), conformant planning is a special case that is useful when sensing is too expensive or impossible (e.g., because of broken sensors). As has been seen in non-deterministic conformant and conditional planning (Bryce, Kambhampati, & Smith 2006; Brafman & Hoffmann 2005), conformant planning heuristics are useful in conditional planning because ignoring observations is sometimes a reasonable relaxation. Future application of the work discussed herein will validate how well conformant probabilistic planning heuristics adapt to conditional probabilistic planning. Initial experiments indicate the adaptation is appropriate.

Despite long standing interest (Kushmerick, Hanks, & Weld 1994; Hyafil & Bacchus 2003; 2004), probabilistic plan synthesis algorithms have a terrible track record in terms of scalability. The current best conformant probabilistic planners are only able to handle very small problems. In contrast, there has been steady progress in scaling deterministic planning. Much of this progress has come from the use of sophisticated reachability heuristics. In this work, we show how to effectively use reachability heuristics to solve conformant probabilistic planning (CPP) problems. We use work on planning graph heuristics for non-deterministic planning (Bryce, Kambhampati, & Smith 2006; Hoffmann & Brafman 2004) as our starting point.

We investigate an extension of the work by Bryce, Kambhampati, & Smith (2006) that uses a planning graph generalization called the labelled uncertainty graph (*LUG*). The *LUG* is used to symbolically represent a set of relaxed planning graphs (much like the planning graphs used by Conformant GraphPlan, Smith & Weld, 1998), where each is associated with a possible world. While the *LUG* (as described by, Bryce, Kambhampati, & Smith, 2006) works only with state uncertainty, it is necessary in CPP to handle action uncertainty. Extending the *LUG* to consider action uncertainty involves symbolically representing how at each level CGP creates a new literal layer for each joint outcomes of the uncertain actions.

With uncertain actions, an explicit or symbolic representation of planning graphs for all possible worlds at each time step is *exactly* representing an exponentially increasing set of literal layers. Since we are only interested in planning graphs to compute heuristics, it is both impractical and unnecessary to exactly represent all of the reachable possible worlds. We turn to approximate methods for representing the possible worlds. Since we are planning in a probabilistic setting, we can use Monte Carlo techniques to construct planning graphs.

There are a wealth of methods, that fall under the name sequential Monte Carlo (SMC) (Doucet, de Freitas, & Gordon 2001) for reasoning about a hidden random variable over time. SMC applied to "on-line" Bayesian filtering is often called particle filtering, however we use SMC for "off-line" prediction. The idea behind SMC is to represent a probability distribution as a set of samples (particles), which evolve recursively over time by sampling a transition function. In our application, each particle is a (simulated) deterministic planning graph and the transition function describes the Conformant GraphPlan (Smith & Weld 1998) construction semantics. By using more particles, we capture more possible worlds, exploiting the natural affinity between SMC

approximation and heuristic accuracy.

The SMC technique requires multiple planning graphs (each a particle), but their number is fixed. We could represent each planning graph explicitly, but they may have considerable redundant structure. Instead, we generalize the *LUG* to symbolically represent the set of planning graph particles in a planning graph we call the Monte Carlo *LUG* ($\mathcal{M}cLUG$). We show that by using the $\mathcal{M}cLUG$ to extract a relaxed plan heuristic we are able to greatly out-scale the current best conformant probabilistic planner CPplan (Hyafil & Bacchus 2004; 2003) in a number of domains, without giving up too much in terms of plan quality.

This work appears as a full paper with Subbarao Kambhampati and David E. Smith in the ICAPS'06 Technical Session. Our presentation starts by describing a worked example of how to construct planning graphs that exactly compute the probability distribution over possible worlds versus using SMC, as well as how one would symbolically represent planning graph particles. We then present an empirical analysis of our technique compared to CPplan, and conclusions. Please consult the full paper for the formal details.

Monte Carlo Planning Graph Construction

We illustrate an example to give the intuition for Monte Carlo simulation in planning graph construction. Consider a simple logistics domain where we wish to load a specific freight package into a truck and loading works probabilistically (because rain is making things slippery). There are two possible locations where we could pick up the package, but we are unsure of which location. There are three fluents, $F = \{$ atP1, atP2, inP $\}$, our initial belief state b_I is 0.5: $s0 = \{$atP1, ¬atP2, ¬inP $\}$, 0.5: $s1 = \{$¬atP1, atP2, ¬inP $\}$, and the goal is $G = \{$inP$\}$. The package is at location 1 (atP1) or location 2 (atP2) with equal probability, and is definitely not in the truck (inP). Our actions are LoadP1 and LoadP2 to load the package at locations 1 and 2, respectively. Both actions have an empty enabling precondition $\{\}$, so they are always applicable, and have two outcomes. The first outcome with probability 0.8 loads the package if it is at the location, and the second outcome with probability 0.2 does nothing. We assume for the purpose of exposition that driving between locations in not necessary.

Figure 1 illustrates several approaches to planning graph based reachability analysis for our simplified logistics domain. (We assume we are evaluating the heuristic value $h(b_I)$ of reaching G from our initial belief state.) The first is in the spirit of Conformant GraphPlan, where uncertainty is handled by splitting the planning graph layers for all outcomes of uncertain events. CGP creates a planning graph that resembles a tree, where each branch corresponds to a deterministic planning graph.

CGP: In Figure 1a, we see that there are two initial literal layers (denoted by literals in boxes), one for each possible world at time zero. We denote the uncertainty in the source belief state by X_0, which takes on values $s0, s1$ (for each state in our belief state). Both load actions are applicable in both possible worlds because their enabling preconditions are always satisfied. The edges leaving the actions denote the probabilistic outcomes (each a set of conditional effects). While it is possible for any outcome of an action to occur, the effects of the outcome may or may not have their secondary precondition supported. In world $s0$, if outcome the first outcome of LoadP1 occurs, then effect denoted by atP1→inP is enabled and will occur, however even if the first outcome of LoadP2 occurs its effect is not enabled and will not occur.

The set of possible worlds at time one is determined by the cross product of action outcomes in each world at time zero. For instance, possible world $x00$ is formed from world $s0$ when outcomes the first outcome of LoadP1 and the first outcome of LoadP2 co-occur. Likewise, world $x12$ is formed from world $s1$ when outcomes the second outcome of LoadP1 and the first outcome of LoadP2 occur.

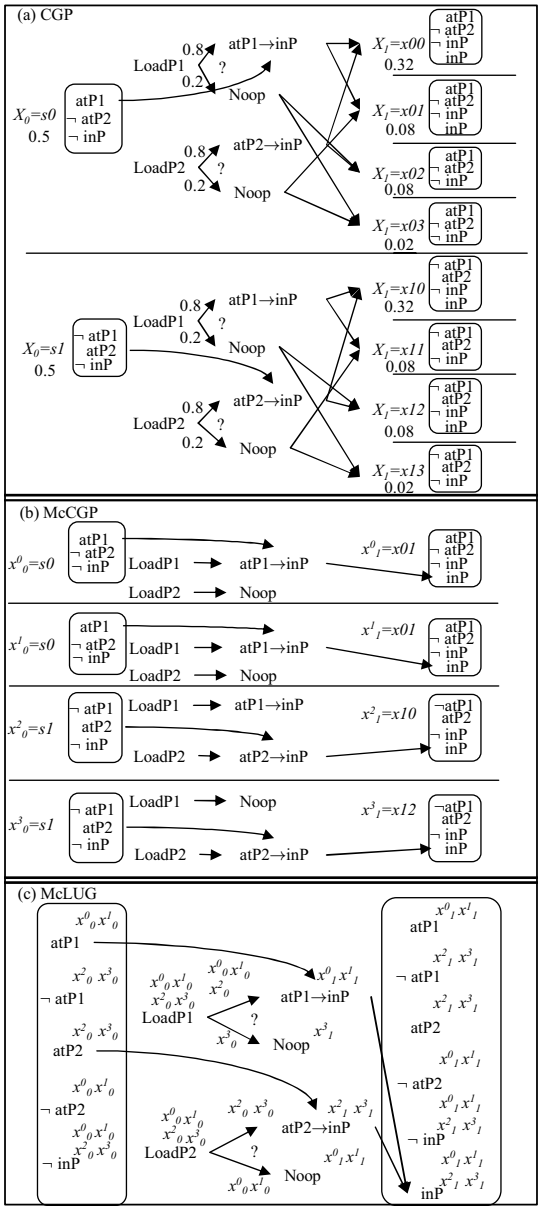


Figure 1: *Variations on planning graph representations.*

CGP is exactly representing the reachable literal layers for all possible worlds. In our example, CGP could determine the exact distribution over X_1 for every value of X_0. We see that our goal is satisfied in half of the possible worlds at time 1, with a total probability of 0.8. It is possible to back-chain on this graph to extract a relaxed plan (by ignoring mutexes) that satisfies the goal with 0.8 probability. However, we note that this is not efficient because it is exactly representing all possible worlds (which can increase exponentially).

McCGP: Next, we illustrate a Monte Carlo simulation approach we call Monte Carlo CGP (McCGP), in Figure 1b. The idea is to represent a set of N planning graph particles. In our example we sample $N = 4$ states $\{x_0^n\}_{n=0}^{N-1} \sim P(X_0) = b_I$ and create an initial literal layer for each. To simulate a particle we first insert the applicable actions. We then insert effects by sampling from the distribution of joint action outcomes. Finally, the subsequent literal layer is constructed, given the sampled outcomes. Note that each particle is a deterministic planning graph.

In our example, the simulation was lucky and the literal layer for each particle at time 1 satisfies the goal, so we may think the best one step plan achieves the goal with certainty. From each of these graphs where the goal is satisfied it's possible to extract a relaxed plan, which can then be aggregated to give a heuristic as described by Bryce, Kambhampati, & Smith (2006). While McCGP improves memory consumption by bounding the number of possible worlds, it still wastes quite a bit of memory. Of the planning graphs many literal layers are identical. Symbolic methods allow us to compactly represent these planning graph particles.

***McLUG*:** Using ideas from Bryce, Kambhampati, & Smith (2006) , we can represent a single literal layer at every time step for all samples in a planning graph called the Monte Carlo *LUG* ($\mathcal{M}cLUG$), in Figure 1c. The idea is to union the connectivity of multiple planning graphs into a single planning graph skeleton, and use labels on the actions and literals to signify the original, explicit planning graphs in which an action or literal belongs. The contribution in the $\mathcal{M}cLUG$ is to represent a set of particles symbolically and provide a relaxed plan extraction procedure that takes advantage of the symbolic representation. From the $\mathcal{M}cLUG$ we are able to extract a relaxed plan that supports the goal for every particle that reaches the goal.

Empirical Analysis

We externally evaluate our planner and its heuristic based on the $\mathcal{M}cLUG$ by comparing with the leading approach to CPP, CPplan (Hyafil & Bacchus 2003; 2004). We also internally evaluate our approach by adjusting the number of particles N that we use in each $\mathcal{M}cLUG$. We show results for a logistics domain. CPplan finds the optimal probability of goal satisfaction for a given plan length, but our planner, like Buridan (Kushmerick, Hanks, & Weld 1994), finds plans that satisfy the goal with probability no less than τ. We find plans with a forward-chaining A* search in the space of belief states. To compare with CPplan, we run CPplan on a problem for each plan length until it exceeds our time or

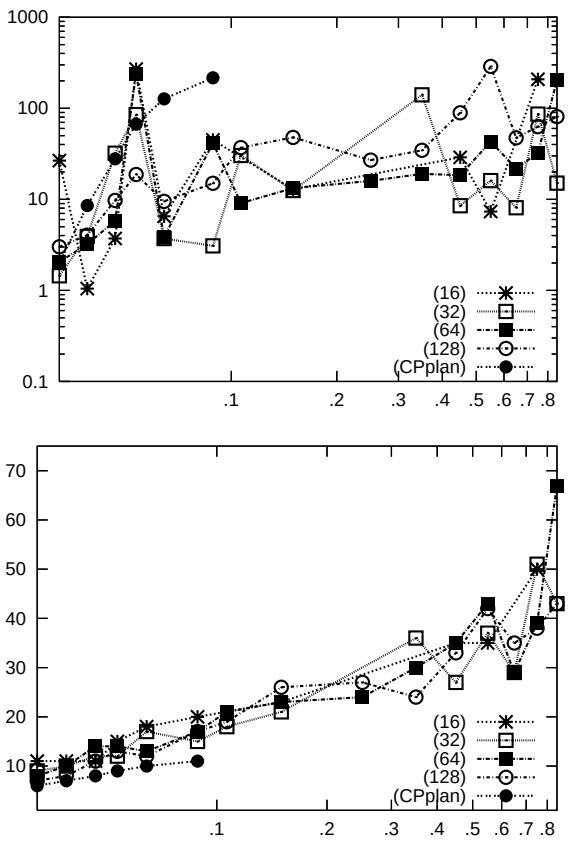


Figure 2: *Run time in seconds (top), and Plan lengths (bottom) vs. τ (log scale) for Logistics p4-2-2*

memory limit. We record the probability that CPplan satisfies the goal for each plan length. We then give our planner a series of problems with increasing values of τ that match the values found by CPplan (and fixed increments thereafter). We ran our planner five times on each problem and present the average run time, and plan length.

The logistics domain has the standard logistics actions of un/loading, driving, and flying, but adds uncertainty. Hyafil & Bacchus (2004) enriched the domain to not only include initial state uncertainty, but also action uncertainty. In each problem there are some number of packages whose probability of initial location is uniformly distributed over some locations and un/loading is only probabilistically successful. Plans require several loads and unloads for a single package at several locations, making a relatively simple deterministic problem a very difficult stochastic problem. We compare on problem p4-2-2, where there are 4 possible initial locations for a package, 2 cities, and 2 packages.

The plots in Figure 2 compare the total run time in seconds (top) and the plan lengths (bottom) of our planner with 16/32/64/128 particles in the $\mathcal{M}cLUG$ versus CPplan. In this domain we also use helpful actions from the relaxed plan. We notice that CPplan is able to at best find solutions where $\tau \leq 0.09$. In most cases our planner is able to find plans much faster than CPplan for the problems they

both solve. It is more interesting that our planner is able to solve problems for *much larger* values of τ. Our planner finds solutions where $\tau \leq 0.85$ which is 9.6 times the maximum values of τ solved by CPplan. In terms of plan quality, the average increase in plan length for the problems we both solved was 4.2 actions. Where CPplan exactly evaluates plan suffixes to find pruning conditions for plan prefixes, we use a heuristic to estimate plan suffixes. As the results demonstrate, our heuristic effectively guides search toward good plans.

Conclusion & Future Work

We have presented an approach called $\mathcal{M}cLUG$ to integrate Monte Carlo into heuristic computation on planning graphs. The $\mathcal{M}cLUG$ enables us to quickly compute effective heuristics for conformant probabilistic planning. With the heuristics, our planner is able to far out-scale the current best conformant probabilistic planner. At a broader level, our work shows one fruitful way of exploiting the recent success in deterministic planning to scale stochastic planners.

Our future work will concentrate on adapting the heuristics described here to handle conditional probabilistic planning. We also intend to develop a Monte Carlo based heuristic search algorithm for search in belief space that combines the work of Thrun (2000) and Barto, Bradtke, & Singh (1995).

References

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

Brafman, R., and Hoffmann, J. 2005. Contingent planning via heuristic forward search with implicit belief states. In *Proceedings of ICAPS'05*.

Bryce, D.; Kambhampati, S.; and Smith, D. 2006. Planning graph heuristics for belief space search. *JAIR*. (To appear).

Doucet, A.; de Freitas, N.; and Gordon, N. 2001. *Sequential Monte Carlo Methods in Practice*. New York, New York: Springer.

Hoffmann, J., and Brafman, R. 2004. Conformant planning via heuristic forward search: A new approach. In *Proceedings of ICAPS'04*.

Hyafil, N., and Bacchus, F. 2003. Conformant probabilistic planning via CSPs. In *Proceedings of ICAPS' 03*.

Hyafil, N., and Bacchus, F. 2004. Utilizing structured representations and CSPs in conformant probabilistic planning. In *Proceedings of ECAI'04*.

Kushmerick, N.; Hanks, S.; and Weld, D. 1994. An algorithm for probabilistic least-commitment planning. In *Proceedings of AAAI'94*.

Smith, D., and Weld, D. 1998. Conformant graphplan. In *Proceedings of AAAI'98*.

Thrun, S. 2000. Monte Carlo POMDPs. In *Advances in Neural Information Processing 12*, 1064–1070.

Analyzing, Learning, and Shaping Planning Search Spaces

Blazej Bulka
Department of Computer Science
University of Maryland, Baltimore County
bulka1@umbc.edu

Introduction

The complexity of a task faced by a planner depends strongly on the search space. The importance of the planner's search space is reflected in the two main classes of classical AI planners: *state-space* planners, which explore the search space by considering totally ordered sequences of directly neighboring states of the world, and *plan-space* planners, which explore the search space by considering incomplete plans consisting of partially ordered actions. The search space also depends on the structure of subgoals (their interactions) and on the representation of knowledge used by the planner (e.g., the level of abstraction at which the planning is done or the expressiveness of the description language). Taking these factors into account can lead to finer-grained classification of planning domains and search spaces.

Knowing the properties of a search space is useful when a human designer creates a new planner or decides which of the existing planners to use. However, knowledge of the properties of a search space may also be useful to the planner itself. In my research, I plan to explore which features of the search spaces can be automatically determined or learned by planners (during repeated executions), and how such information may be used by a planner to improve the search process (e.g., by choosing an appropriate planning technique, abstraction level, or representation).

This research is still in the idea stage: I am in the process of developing my dissertation proposal, and plan to take the preliminary exam (proposal defense) in July, after the conference. Attending the doctoral consortium will give me the ideal opportunity to receive early feedback that can help guide my dissertation research. I also look forward to the opportunity to attend the conference in order to get a better sense of the current state of the field, and to meet other graduate students and senior researchers.

Overview of the approach In my approach, the planner will discover (either through a preliminary analysis or learning) features of the structure of the planning domain. Information about these features can improve the planning process by using them as heuristics, selecting a more appropriate search method, tuning the parameters of the planner, or by changing to a different representation of the domain knowledge (shaping the search space).

In this paper, I propose two approaches to discover features of a planning domain. The first one is a preliminary analysis of the domain description. Although the cost of such analysis would be high, it can be amortized over multiple planning attempts in the future that would benefit from it. The second approach includes observing multiple valid plans (either produced by the planner itself, other planners or humans), and learning the features from the observations. This approach may be particularly effective in discovering statistical features of the search space for the domain (e.g., intermediate states that are often included in a plan).

Detection of search space features

Analysis In my approach, the planner will run a preliminary analysis of the planning domain (i.e., actions, their preconditions, and their effects) in order to discover dependencies among the actions. Such dependencies can later be used to make planning decisions for any specific problem (i.e., any initial state and goals). This approach is inspired by work of Kambhampati, Parker, and Lambrecht (1997) and Hoffmann, Porteous, and Sebastia (2004).

Kambhampati, Parker, and Lambrecht analyzed Graphplan. They point out that the planning graph created by Graphplan is a way of representing the search tree in a compact but approximate manner. Each level represents a set of states in the search space, and the exact path in the state-space can be retrieved, which happens during the extraction of the plan produced by the planner. The representation of the search tree is approximate because it includes only information for certain goals interactions (*mutexes*—i.e., sets of size of two containing mutually exclusive propositions). They also show that there are domains where this information is not sufficient, yet looking for the additional information during each generation of a planning graph is not feasible. This would justify preprocessing, which needs to be done only once since it does not depend on the particular initial state and goals. An open question remains: how to automatically detect whether a particular domain has the property that information in a planning graph created by Graphplan is not sufficient.

Hoffman, Porteous, and Sebastia also did preprocessing in order to find ordering of subgoals, as described later. Their research, however, was dependent on the initial and goal states, which means that the preprocessing has to be exe-

cuted during each run of the planner. If a more general analysis of the search space's structure (not dependent on initial and goal states) could be performed, the preprocessing could be performed only once. Moreover, the information obtained during such an analysis could be used in multiple ways. The first one would be to perform reachability analysis of a search space (which would allow the planner to reject some paths in the search space and to declare some goals not achievable from certain starting points). Another way would be through the identification of *hub states*: states or preconditions that *always* or *often* appear in valid plans in the domain. This would improve planning by adding hubs as subgoals, or by precomputing actions that lead from one hub state to another one. An extension of this idea would be treating a group of states as a hub (a minimal group of states such that at least one of the states appears in almost all valid plans).

Hoffmann, Porteous, and Sebastia also used heuristics based on parsing techniques (e.g., lookahead). I would also like to explore the similarities between planning, given a domain representation, and parsing given a grammar. Many efficient parsers do not analyze the grammar while parsing: they usually use precomputed information such as a parser table. Most parsers, however, cannot process arbitrary grammars. Instead, they are limited to a few basic classes like context-free or LALR(n) grammars. It would be interesting to identify and analyze analogous classes of planning spaces. Such classes of spaces could either be based on specific structures of subgoals or relations among planning states (especially in state-space search). (The relation of planning to context-grammars has already been noticed by Erol, Hendler and Nau (1994), but they mostly focused on HTN planning, which has an explicit structure among actions.)

Learning The search space analysis described above may be most efficient for features that are present in every valid plan, but an approach that considers only such features may be too restrictive (especially if we consider preprocessing of the space for all initial states). However, detection of features that are often (but not always) included in plans may require enumerating a large part of the search space. Therefore, it could be feasible to learn the existence of such hub nodes by observing multiple generated plans for the same domain, instead of analyzing.

Many domains (including benchmark domains for planning) are reported to have regularities in their local search topology. For example, Hoffmann (2003) analyzed heuristics that ignore delete lists of operations in the context of phenomena that occur in the local search topology. Similarly, Haslum and Geffner (2000) showed that the successful use of heuristics to guide the planning process can be linked to the regularities in the domains. Learning the local features and regularities of the search space can possibly lead to the development of good, domain-specific heuristics. Moreover, for a search guided by a heuristic, it could also be possible to incrementally learn a better heuristic.

Shaping the search space

Having discovered the features of a planning domain (e.g., by analyzing the search space or by learning the features from observation of valid plans), a planner may take advantage of this knowledge by reasoning about planning in this domain (meta-planning) and shaping the search space.

One way to change the shape of the search space is to change the representation of facts. My idea is partially inspired by work by Haslum and Jonsson (2000), who focused on the idea of removing redundant operators *given an initial state*. I believe that planning could be improved by removing particular effects of the operators, while still preserving correctness comparing to the plan with non-modified operators (*redundant effects*). It may also be possible to remove operators that are rarely used in plans (as learned by the planner). As a result, removing some operators would allow easier preprocessing of the domain regardless of the initial state (e.g., if it would reduce the class of a search space to a simpler one) at the expense of producing less optimal plans in terms of the plan's length.

Another possible way of changing the search space is the selection of an alternative search method for the whole plan or part thereof. (A similar approach was used in the FLECS algorithm (Veloso & Stone 1995), but the selection condition in that case was given by the designer, not learned by the planner.) For example, it may be possible to learn which search method performs the best given the set of detected features of the planning domain. Alternatively, the planner could learn a hierarchical representation of the problem (inspired by work by Knoblock (1994)), decide to first solve the planning problem at a higher level, and then solve the subproblems independently (similar to HTN planning).

Related work

The problem of structure and interactions among the goals has previously been analyzed by Barrett and Weld (1994). They described different classes of planning domains, and tested the behavior of both total-order and partial-order planners on these domains. Hoffmann, Porteous, and Sebastia (2004) described different kinds of possible ordering relations between subgoals. They also introduced the concept of *landmarks*, which can be perceived as a particular type of hub states mentioned earlier in this paper. (A landmark is a subgoal that must be satisfied at some point in *every* plan in the domain; my definition of a hub state is slightly broader and also includes the goals or states that are included in *most* plans in the domain.) Additionally, Hoffmann in his earlier work described how local features of the topology of the search space (as opposed to "global" landmarks and orderings) may influence planning, and how such features of the space can be detected and used in FF planner (Hoffmann 2001; 2003). The work by Smith and Peot (1996) described analyzing the search space by using *operator graphs* to avoid recursion and prune it. Preanalysis of the search space is also used in the work by Fox and Long (1998), which finds state invariants using type inference. Their later work (Fox & Long 1999; Porteous, Long, & Fox 2004) focuses on finding regularities

in planning problems (symmetry and almost symmetry).

Other work has explored the appropriate level of abstraction during planning. This issue is the focal point of HTN planning (Sacerdoti 1975), where different levels of abstraction are explicitly represented. Nevertheless, information about an abstract hierarchy may also be used in planners that do not support hierarchy explicitly: for example, Veloso and Stone (1995) mention treating some intermediate goals as *milestones*, which divide states in a plan into independent groups. Each such group can be treated as a goal at a higher level of abstraction. Information about these groups is obtained from an external source. (Veloso and Stone mention work by Knoblock (1994) as a method to generate abstractions automatically.) In fact, Kambhampati (1995) presents a comparison of "pure" partial-order planners (no use of information regarding hierarchy) and HTN planning, and discusses the advantages of having an explicit representation for abstract or higher-level goals.

Different ways of representing the same domain are also a popular research topic, especially the tradeoff between the complexity of planner's data structures and the size of the search space (Kambhampati & Yang 1996; Kambhampati, Parker, & Lambrecht 1997).

There is also a body of work on learning in planning. Learning appropriate heuristics by planners based on their previous experiences in planning can be found in work by Likhachev and Koenig (2005). Botea *et al.* (2005) presented an approach that exploits the underlying domain structure, and learns ordering of operators (actions) and combining them into groups (macros) by observing plans in the domain.

Status of the work

This paper presents preliminary work done under supervision of my advisor, Prof. Marie desJardins. I plan to have the ideas further extended by July 2006 by providing details of the proposed methods and examples of planning domains where these methods are applicable. At this time, this work should be developed far enough to form a Ph.D. thesis proposal.

References

Barrett, A., and Weld, D. S. 1994. Partial-Order Planning: Evaluating Possible Efficiency Gains. *Artificial Intelligence* 67(1):71–112.

Botea, A.; Müller, M.; and Schaeffer, J. 2005. Learning Partial-order Macros from Solutions. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS-05)*, 231–240.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. HTN Planning: Complexity and Expressivity. In *Proceedings of Eleventh National Conference on Artificial Intelligence (AAAI-94)*, 1123–1128.

Fox, M., and Long, D. 1998. The Automatic Inference of State Invariants in TIM. *Journal of Artificial Intelligence Research (JAIR)* 9:367–421.

Fox, M., and Long, D. 1999. The Detection and Exploitation of Symmetry in Planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*.

Haslum, P., and Geffner, H. 2000. Admissible Heuristics for Optimal Planning. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2000)*, 140–149.

Haslum, P., and Jonsson, P. 2000. Planning with Reduced Operator Sets. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2000)*, 150–158.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered Landmarks in Planning. *Journal of Artificial Intelligence Research (JAIR)* 22:215–278.

Hoffmann, J. 2001. Local Search Topology in Planning Benchmarks: An Empirical Analysis. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, 453–458.

Hoffmann, J. 2003. Where Ignoring Delete Lists Works: Local Search Topology in Planning Benchmarks. Technical Report 185, Institute for Computer Science, Albert-Ludwigs-University, Freiburg, Germany.

Kambhampati, S., and Yang, X. 1996. On the Role of Disjunctive Representations and Constraint Propagation in Refinement Planning. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR-96)*, 135–146.

Kambhampati, S.; Parker, E.; and Lambrecht, E. 1997. Understanding and Extending Graphplan. In *Recent Advances in Planning: Fourth European Conference on Planning (ECP-97)*, 260–272.

Kambhampati, S. 1995. A Comparative Analysis of Partial Order Planning and Task Reduction Planning. *SIGART Bulletin* 6(1):16–25.

Knoblock, C. A. 1994. Automatically Generating Abstractions for Planning. *Artificial Intelligence* 68(2):243–302.

Likhachev, M., and Koenig, S. 2005. A Generalized Framework for Lifelong Planning A*. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS-05)*.

Porteous, J.; Long, D.; and Fox, M. 2004. The Identification and Exploitation of Almost Symmetry in Planning Problems. In *Proceedings of the 23rd UK Planning and Scheduling SIG*.

Sacerdoti, E. 1975. The Nonlinear Nature of Plans. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 206–214.

Smith, D. E., and Peot, M. A. 1996. Suspending Recursion in Causal-link Planning. In *Proceedings of the Third Artificial Intelligence Planning Systems Conference (AIPS-96)*, 182–190.

Veloso, M., and Stone, P. 1995. FLECS: Planning with a Flexible Commitment Strategy. *Journal of Artificial Intelligence Research (JAIR)* 3:25–52.

Linear Logic in Planning

Lukas Chrpa
Department of Theoretical Computer Science and Mathematical Logic
Faculty of Mathematics and Physics
Charles University in Prague
chrpa@kti.mff.cuni.cz

Abstract

Linear Logic is a powerful formalism used to manage a lot of problems with resources. Linear Logic can also be used to formalize Petri Nets and to solve simple planning problems (for example 'Block World'). Research goes ahead also in Linear Logic Programming, which means that we have tools, that can solve Linear Logic problems. In this paper I will show the possible connection between solving planning problems and Linear Logic Programming.

Introduction

Planning problems can be solved by translation into another formalism like SAT, CSP, BDD, etc. Linear Logic is another formalism to which a planning problem can be translated. There already exists a planning system based on Linear Logic called RAPS (Küngas 2003). RAPS was introduced at Doctoral Consortium at ICAPS 2003. The author of RAPS compared RAPS with the best planners that participated at IPC 2002. This comparison showed very interesting results, a Skeleton version of RAPS showed almost the best computation time in computing the plans (the typed Depots domain), but on the other hand the solution length (in the typed Depots domain) was almost the highest, which means that the plans weren't optimal. The Skeleton version of RAPS first converts a planning problem into propositional Linear Logic (which means that predicates in a planning operator description are abstracted to propositional constants by removing predicates' arguments) and from that it calculates skeleton plans, which means that we obtain a sequence of actions needed to reach the goal. The final plan is obtained from the skeleton plan by unification with corresponding arguments. RAPS (including the Skeleton version) exploits the fact that a planning problem coded into Linear Logic is easily converted to the problem of Petri Net reachability so the system mainly exploits the algorithms for Petri Net reachability.

Instead of coding the planning problems in Linear Logic and solving the Petri Net Reachability problem like in RAPS, I propose to study possibilities of solving the planning problems using Linear Logic Programming tools (described bellow). I believe that Linear Logic Programming tools can achieve better efficiency than 'classical' Logic Programming tools and hence might be more appropriate for solving planning problems (Banbara 2006). Nevertheless I will also study the possibilities of efficient solving Linear Logic problems in Prolog. Prolog itself has many extensions which may support some techniques for optimization of planning problems.

In the next part of this paper I will give a short introduction to Linear Logic and Linear Logic Programming. Then I will describe how to solve Petri Net reachability problem using Linear Logic and how to convert planning problems to Linear Logic. Finally I will present my future research plans in this area.

Linear Logic

Linear Logic was introduced by J.Y. Girard at 1987 (Girard 1987; 1995). Unlike the 'classical' logic we can handle resources in Linear Logic. The basic operator in Linear Logic is a (linear) implication ($A \multimap B$), which is defined as B is obtained by using one resource A. Linear Logic defines more operators (not only implication), but I will describe here only the multiplicative conjunction $\otimes$ and the additive disjunction $\oplus$ (the description of other operators can be found in (Girard 1987; 1995)). The expression $(A \otimes B) \multimap (C \otimes D)$ means that C and D are obtained using A and B. The expression $A \multimap (B \oplus C)$ means that B or C (we don't know which one) is obtained using A. Proving in Linear Logic is quite similar to proving in the 'classical' logic (hypotheses $\Rightarrow$ conclusion) , but the calculus of Linear Logic is more complicated. To find out more about proving in Linear Logic and the whole calculus of Linear Logic, see (Girard 1987; 1995).

Linear Logic Programming

Linear Logic Programming is derived from classical logic programming (Prolog based) by including linear facts and linear operators. Syntax of common Linear Logic Languages is quite similar to Prolog syntax (Banbara 2006). As I mentioned above, the efficiency of these languages in solving problems describable in Linear Logic is better than in Prolog. The good efficiency of Linear Logic Programming languages is reached by using optimization techniques based on the theory of proving in Linear Logic,

more in (Banbara 2002).

In my diploma thesis I proposed a Linear Logic Programming language SLLL (Chrpa 2005), which was constructed as a compiler to Prolog. However, the problem of this language is a low computational efficiency caused by emulating the linear facts as lists. Fortunately, there are other Linear Logic Programming languages: Lolli (Hodas 1994; 1992), LLP (Banbara 2002), Lygon (Winikoff 1996), LTL[1] and more. Lolli is possibly the strongest Linear Logic Programming language, which contains almost all of Linear Logic features. LTL and LLP are possibly the most effective Linear Logic Programming languages today.

Solving Petri Net reachability problem by Linear Logic

In the next paragraphs, I will describe how Linear Logic can be used in solving the Petri Net reachability problem, that is, the problem of finding whether a given marking is reachable from the initial marking. To find more about Petri Nets (and the problem of Petri Net reachability), see (Reisig 1985).

Now I explain how the problem of Petri Net reachability can be easily encoded using Linear Logic. Tokens in places are encoded as linear facts (resources), in particular the initial marking (in this case: one token in each place $p_1, \ldots, p_k$) is encoded in the following way:

$$\vdash p_1 \otimes p_2 \otimes \ldots \otimes p_k$$

Transitions are also encoded as axioms. For transition t and places $p_{i_1}, \ldots, p_{i_m} \in$ IN(t)...input places and $p_{o_1}, \ldots, p_{o_n} \in$ OUT(t)...output places we get:

$$\vdash (p_{i_1} \otimes \ldots \otimes p_{i_m}) \multimap (p_{o_1} \otimes \ldots \otimes p_{o_n})$$

If the goal (in this case: one token in each place $p_{g_1}, \ldots, p_{g_l}$) $(p_{g_1} \otimes \ldots \otimes p_{g_l})$ is provable (by using the above axioms), the marking (one token in each place $p_{g_1}, \ldots, p_{g_l}$) is reachable.[2] More about the topic can be found in (Oliet & Meseguer 1989).

Planning with Linear Logic

Problem of using Linear Logic in planning have been studied by several authors (Masseron, Tollu, & Vauzeilles 1993; Kanovich & Vauzeilles 2001). Encoding of planning problems in Linear Logic is quite similar to encoding of Petri Nets (planning problems can also be encoded directly using Petri Nets). In planning we have states, that are represented by the set of predicates, that are true in the given state. We can encode these states as a multiplicative conjunction of (true) predicates, that belong to the corresponding state. The encoding of state s:

$$(p_1 \otimes p_2 \otimes \ldots \otimes p_n), \qquad s = \{p_1, p_2, \ldots, p_n\}$$

Actions in planning contain preconditions p (must be satisfied before preforming the action), negative effects e^-(removed after the action), and positive effects e^+ (added after the action). The action $a = \{p, e^-, e^+\}$ is encoded as:

$$\forall i \in \{1, 2, \ldots, k\}, l_i \in p \cup e^-$$
$$\forall j \in \{1, 2, \ldots, m\}, r_j \in e^+ \cup (p - e^-)$$
$$(l_1 \otimes l_2 \otimes \ldots \otimes l_k) \multimap (r_1 \otimes r_2 \otimes \ldots \otimes r_m)$$

This expression means that the predicates on the left side of the implication will no longer be true after performing action a and the predicates on the right side of the implication will become true after performing action a. The plan exists if and only if the encoding of the goal state is provable from the encoding of the initial state using the actions encoded as axioms.[3]

Encoding negative predicates

The above formalism worked only with positive predicates. However sometime we also need to encode negative predicates.[4] We extend the encoding of predicates with symbols for negative predicates (p will obtain a twin $\overline{p}$ which represents a negative form of predicate p). The encoding of state s, where predicates $p_1, \ldots, p_m$ are true in s and $p_{m+1}, \ldots, p_n$ are false in s:

$$p_1 \otimes \ldots \otimes p_m \otimes \overline{p_{m+1}} \otimes \ldots \otimes \overline{p_n}$$

For every action $a = \{p, e^-, e^+\}$, we create an action $a' = \{p, e'^-, e'^+\}$, where $e'^- = e^- \cup \{\overline{p} \,|\, p \in e^+\}$ and $e'^+ = e^+ \cup \{\overline{p} \,|\, p \in e^-\}$. Now we can encode all actions a' in the same way as described above.

Example

Let us present now an example of the conversion (without negative predicates). Imagine the version of "Block World", where we have slots and boxes, and every slot may contain at most one box. We have also a crane, which may carry at most one box.

Initial state: 3 slots (1,2,3), 2 boxes (a, b), empty crane, box a in slot 1, box b in slot 2, slot 3 is free.

Actions:

$$PICKUP(Box, Slot) = \{$$
$$\begin{aligned} p &= \{empty, in(Box, Slot)\}, \\ e^- &= \{empty, in(Box, Slot)\}, \\ e^+ &= \{holding(Box), free(Slot)\} \end{aligned}$$
$$\}$$
$$PUTDOWN(Box, Slot) = \{$$
$$\begin{aligned} p &= \{holding(Box), free(Slot)\}, \\ e^- &= \{holding(Box), free(Slot)\}, \\ e^+ &= \{empty, in(Box, Slot)\} \end{aligned}$$
$$\}$$

[1] developed by Dr. Arnost Vecerka, my diploma supervisor

[2] multiple tokens in places or multiple (input, output) places can be easily encoded as n-times $p \otimes \ldots \otimes p$

[3] To obtain a full plan, we must keep information about used axioms (encoded actions) during proving.

[4] Negative predicates often appear in preconditions.

Goal: Box a in slot 2, Box b in slot 1, empty crane, free slot 3.

The encoding of the problem:

$INIT$:
$in(a,1) \otimes in(b,2) \otimes free(3) \otimes empty$

$PICKUP(Box, Slot)$:
$empty \otimes in(Box, Slot) \multimap holding(Box) \otimes free(Slot)$

$PUTDOWN(Box, Slot)$:
$holding(Box) \otimes free(Slot) \multimap empty \otimes in(Box, Slot)$

$GOAL$:
$in(b,1) \otimes in(a,2) \otimes free(3) \otimes empty$

A solution is a sequence of actions. In this case, this sequence looks like: $PICKUP(a,1)$, $PUTDOWN(a,3)$, $PICKUP(b,2)$, $PUTDOWN(b,1)$, $PICKUP(a,3)$, $PUTDOWN(a,1)$.

Possible optimizations

In the previous subsections I described the pure encoding of planning problems to Linear Logic. In this subsection I will show that we are able to encode some optimizations to Linear Logic as well.

In the above example we have two actions: $PICKUP(Box, Slot)$ and $PUTDOWN(Box, Slot)$. These actions are inverse, which means that if we perform these actions with same parameters $Box, Slot$ consecutively, we obtain the state that we had before performing these actions. The main idea how to block the consecutive performing of inverse actions is an extension of the encoding of the actions. The encoding of the action $PICKUP(Box, Slot)$ from the above example id shown bellow (encoding of the action $PUTDOWN(Box, Slot)$ is analogical):

$PICKUP(Box, Slot)$:
$canpick(Box, Slot) \otimes canput(Box, Slot) \otimes nopick(X, Y) \otimes empty \otimes in(Box, Slot) \multimap holding(Box) \otimes free(Slot) \otimes canpick(Box, Slot) \otimes noput(Box, Slot) \otimes canpick(X, Y)$

The predicates $canpick(Box, Slot)$ ($canput(Box, Slot)$) mean that actions $PICKUP(Box, Slot)$ ($PUTDOWN(Box, Slot)$) can be performed (allowed). The predicates $nopick(Box, Slot)$ ($noput(Box, Slot)$) mean that actions $PICKUP(Box, Slot)$ ($PUTDOWN(Box, Slot)$) can't be performed (blocked). The encoding of the action $PICKUP(Box, Slot)$ means that this action can be performed if and only if the predicate $canpick(Box, Slot)$ is true. After performing this action the predicate $canput(Box, Slot)$ becomes false, the predicate $noput(Box, Slot)$ becomes true, the predicate $nopick(X, Y)$ (represents exactly one blocked action $PICKUP(X, Y)$) becomes false and the predicate $canpick(X, Y)$ becomes true. In the other words this means that performing some (allowed) action blocks the inverse action and unblocks the action blocked by the previously performed inverse action.

Another optimization of the previous example is blocking the action $PICKUP(Box, Slot)$ forever if the predicate $in(Box, Slot)$ is true in goal state. The action $PICKUP(Box, Slot)$ is blocked when both predicates $canpick(Box, Slot)$ and $nopick(Box, Slot)$ are false. This is obtained by removing the predicate $nopick(Box, Slot)$ from the right side of the linear implication in the encoded $PUTDOWN(Box, Slot)$ action.

I showed that Linear Logic can easily encode some optimizations for the planning problems. Using these optimizations may lead the to better efficiency.

Comparing to SAT

Linear Logic itself has some advantages that can be exploited in the encoding of planning problems. The main advantage is the linear size of the encoding of the planning problems. For example the size of a SAT encoding of planning problems can be exponential. On the other way, SAT problems are in general NP-complete unlike the undecidability of whole Linear Logic. In planning we are using only a part of Linear Logic, but we still have no evidence about decidability and complexity of this restricted problem.

Future Research

In my future research, I will study the problem of efficient usage of Linear Logic in planning problems (for example encoding optimizations). I will study the possibilities of using Linear Logic Programming tools and possibilities of emulating Linear Logic in Prolog. I will also make a comparison to some models (Gelfond & Lifschitz 1993). The following paragraphs will present my future research plans in more detail:

Using Linear Logic Programming Tools

As I have mentioned above, we have several tools that could solve Linear Logic problems efficiently. The preliminary experiments showed that the existing Linear Logic Programming tools are not powerful enough to solve the planning problems, because these tools can't still handle the linear implication well. This means that I am still emulating Linear Logic in Prolog, which isn't much efficient. Nevertheless, I believe that these tools may be useful as a support to other planning techniques. I also believe that possible improvements of these tools may help with solving the planning problems. I will study the possibilities of using these tools to solve the planning problems.

Emulating Linear Logic in Prolog

Linear Logic can be easily emulated in Prolog.[5] Linear facts are in a special list. We must define two predicates, one for deleting the facts from the list (lin_del) and one for adding the facts to the list (lin_add):

```
lin_del(V,[V|L],L).
lin_del(V,[H|L],[H|NL]):-lin_del(V,L,NL).
lin_add(V,L,[V|L]).
```

[5] We don't need whole Linear Logic, we need to emulate only the support for the operators $\otimes, \oplus, \multimap$.

Emulation of the multiplicative conjunction $\otimes$ and the additive disjunction $\oplus$ is very easy, because we can replace them by 'classical' conjunction and disjunction which are presented in Prolog. Emulation of the linear implication $\multimap$ is also easy. All linear facts on the left side of the linear implication are deleted from the list and all linear facts on the right side of the linear implication are added to the list. A formula $a \otimes b \multimap c \oplus d$ can be written in Prolog like this:

```
(lin_del(a,L1,L2),lin_del(b,L2,L3)),
(lin_add(c,L3,L4);lin_add(d,L3,L4))
```

Variables $L1, L2, L3, L4$ represent the list of linear facts, because we must keep this list consistent and vulnerable to backtracking.

This emulation isn't very efficient, but we can do some optimizations. If we have the list of linear facts sorted, we don't need to use the predicates lin_del n-times consequently. We can improve the lin_del predicate such that it will accept a sorted list of n facts and make the same effect like using the old predicates lin_del n-times consequently. This approach will result in a fact that the list of linear facts can be explored only once. To keep the list sorted we must also improve the lin_add fact. In future I will try to find out more and better optimizations in emulating Linear Logic in Prolog.

Temporal Logic extensions

There are also Linear Logic Programming tools that support Temporal Logic extensions. For example the extension of LLP is called TLLP (Banbara 2002). This could provide a formalism to time extensions, especially for qualitative modeling of time. With Temporal Logic we can also model features like an action that must be performed before another action. This may lead to PSP (Plan-Space Planning).

Using Linear Logic in probabilistic planning

When performing an action in probabilistic planning we could reach more states (instead of one like in deterministic planning). Reachability of a particular state depends on probability of obtaining that state after performing the planned action. The main advantage of Linear Logic is additive disjunction, so we are able to encode the actions in probabilistic planning in the following way ($s, s_1, s_2, \ldots, s_n$ are states, A is the action):

$$s \times A \rightarrow \{s_1, s_2, \ldots, s_n\}$$

$$A : s \multimap (s_1 \oplus s_2 \oplus \ldots \oplus s_n)$$

This expression means that only one state from $s_1, s_2, \ldots, s_n$, could be reached after performing action A from state s in a certain step, but we don't know which one (depends on probability). Unfortunately the main disadvantage of Linear Logic is that it can't handle probabilities directly. Nevertheless, there is still an option, which consists of possible cooperation with other techniques. This problem needs to be more studied, so in future I will also try to find out more about this extension.

Conclusion

The paper showed that Linear Logic can be used to encode planning problems. Like for other encodings, the advantage of this approach is that an improvement of the Linear Logic solver leads to improved efficiency of the planner based on Linear Logic. Still, the efficiency of current Linear Logic solvers applied to planning problems should be explored in more detail.

Acknowledgements

I thank to my supervisor Roman Bartak for help with writing this paper. The research is supported by the Czech Science Foundation under the contract no. 201/04/1102 and by the Grant Agency of Charles University (GAUK) under the contract no. 326/2006/A-INF/MFF.

References

Banbara, M. 2002. *Design and Implementation of Linear Logic Programming Languages*. Ph.D. Dissertation, The Graduate School of Science and Technology, Kobe University.

Banbara, M. 2006. http://bach.istc.kobe-u.ac.jp/llp/.

Chrpa, L. 2005. Linearni logika. Master's thesis, Department of Computer Science, Palacky University, Olomouc. (in Czech).

Gelfond, M., and Lifschitz, V. 1993. Representing actions and change by logic programs. *Journal of Logic Programming* 17(2,3,4):301–323.

Girard, J.-Y. 1987. Linear logic. *Theoretical computer science* 50:1–102.

Girard, J.-Y. 1995. *Linear Logic: Its Syntax and Semantics*. Cambridge University Press.

Hodas, J. 1992. Lolli: An extension of lambdaprolog with linear logic context management. *Proceedings of the 1992 Workshop on the lambdaProlog Programming Language*.

Hodas, J. 1994. *Logic Programming in Intuitionistic Linear Logic: Theory, Design, and Implementation*. Ph.D. Dissertation, University of Pennsylvania, Department of Computer and Information Science.

Kanovich, M., and Vauzeilles, J. 2001. The classical ai planning problems in the mirror of horn linear logic: Semantics, expressibility, complexity. *Mathematical Structures in Computer Science 11(6)*.

Küngas, P. 2003. Linear logic for domain-independent ai planning. *Proceedings of Doctoral Consorcium ICAPS*.

Masseron, M.; Tollu, C.; and Vauzeilles, J. 1993. Generating plans in linear logic i-ii. *Theoretical Computer Science*.

Oliet, N. M., and Meseguer, J. 1989. From petri nets to linear logic. *Springer LNCS 389*.

Reisig, W. 1985. *Petri Nets, An Introduction*. Springer Verlag, Berlin.

Winikoff, M. 1996. Hitch hiker's guide to lygon 0.7. Technical Report 96/36, The University of Melbourne, Australia.

Generic Types and their Use in Improving the Quality of Search Heuristics

Andrew Coles
Department of Computer and Information Sciences
University of Strathclyde
26 Richmond Street,
Glasgow, G1 1XH
email: andrew.coles@cis.strath.ac.uk

Abstract

This abstract discusses work looking into techniques for improving the quality of the search heuristics used to guide forward-chaining planning. The improvements in heuristic quality are made by performing a static analysis of the planning problem to identify commonly occurring 'generic types', and providing additional heuristic guidance based on their known properties. In doing so, the heuristic is tailored to the identified properties of the domain and can provide a more realistic heuristic value and refined relaxed plan. This can potentially lead to reduction in the time taken to find a plan, and the generation of shorter plans.

Introduction

Forward-chaining planning guided by a heuristic has proved to be an effective planning strategy in a range of planning domains. At recent international planning competitions, many of the participating planners followed this search approach; of particular note is FF (Hoffmann & Nebel 2001), which participated with great success in the 2002 and 2000 competitions. Work on HSP (Bonet & Geffner 2000) and Downward (Helmert 2004) has explored alternative heuristics. What all these planners share, however, is that the heuristic goal-distance estimate they provide is obtained from a 'relaxed' version of the original problem, i.e. one from which some constraints have been removed. The relaxation of the original problem in this manner is necessary to allow a heuristic value to be obtained in a reasonable time; however, it does reduce the accuracy with which the relaxed problem is able to model certain aspects of the original problem.

Using static analysis techniques, such as those performed by TIM (Long & Fox 2000), it is possible to identify 'generic types' of objects within planning problems: for instance, self-propelled mobile objects capable of moving from one location to another. These generic types form subproblems with known properties with which type-specific heuristics can be used: for instance, using the Floyd Walshall algorithm to calculate the cost of moving a mobile from one location to another. HybridSTAN (Fox & Long 2001), a forward-chaining heuristic planner, took the approach of isolating these known sub-problems when planning, removing all predicates pertaining to the location of mobiles from the domain. Once a solution plan was found, actions were inserted into the plan to move the mobile objects to the locations needed for the actions used.

The decomposition approach of HybridSTAN relies on being able to cleanly isolate the sub-problem, which is only possible if it is wholly described by the generic type. For example, if the move action for the mobile requires another condition to be satisfied (such as one defining whether a door is open between the two locations) then the subsolver cannot handle the additional constraints imposed. In these cases, is not possible to add the missing actions to the plan as required once the remainder of the problem has been solved, as it is no longer clear which actions are needed.

To this end, this work is concerned with investigating whether the static domain analysis used to discover subproblems can be used to improve the quality of the relaxation heuristic used, in this case the Relaxed Planning Graph heuristic, without relying on being able to solve the identified subproblems in isolation. By improving the heuristic, and the guidance it provides through state space, the aim is to reduce the time taken to find solution plans and to improve the quality of plans found.

Background

Generic Types

TIM is capable of identifying objects, or groups of objects, within planning problems as having a recognisable generic behaviour and thus being of a certain generic type. TIM first analyses planning problems to discover the 'property spaces' relating to each of the objects. From these, generic types are identified by looking for hand-coded patterns of transitions within the property spaces. Included in these generic types are mobiles and resources. Mobiles have a location property, the value of which is changed by the application of 'move' actions to move the mobile from one location value to another. The locations at which the mobile can be located are arranged into a map; directed edges exist in the map between pairs of locations where a feasible move action exists to move the mobile from the source location to the destination. At no point, either in the initial state or any sound, reachable, state is is it possible for a mobile to be located at more than one location.

Resources are a special case of mobiles, whose map consists of a series of linearly interconnected nodes. An edge

can be drawn from a node A to a node B if there exists an action capable of moving the mobile denoting the resource level from A to B; an edge can be drawn from B to A if there exists an action capable of moving the resource level from B to A. Edges in one direction correspond to increasing the resource level; edges in the other direction correspond to decreasing the resource level.

Known generic types can sometimes arise in unexpected situations, where human intuition might not have expected them. Any object which has a predicate relating it to one of a series of other objects and a corresponding action schema which changes this assignment is identified as a mobile object.

Generic Types and the Relaxed Planning Graph Heuristic Landscape

The relaxed planning graph heuristic, as first used in FF, has proved to be a useful heuristic for guiding forward-chaining planning. The relaxation used as a basis for the heuristics is to ignore the delete lists (negative effects) of the domain actions; Graphplan (Blum & Furst 1995) is then used to solve this relaxed problem, although only a subset of the algorithm needs to be implemented as the planning graph does not contain mutexes due to the removal of delete effects.

When delete lists are ignored, once a fact has been established by an action, it is available for use as a precondition to all the subsequent actions in the plan. This has some interesting effects on how well the relaxed problem is able to model some aspects of known generic types within planning problems. When the move actions of mobile objects are invoked, the effects of the action normally establish two facts: the mobile is now located at the destination; and the mobile is no longer located at the source. Similarly, when action increasing or decreasing resource levels are invoked: the resource level is now that resulting from the action; and no longer holds the previous value. Ignoring the delete effects of move actions (or resource-level-altering actions), as done when forming the relaxed planning problem, removes the effects that establish that once a mobile has moved it is no longer at its previous location. Effectively, when executing a relaxed plan, mobiles are simultaneously available at all the locations they have ever been, and resources are available at all levels they have held.

When dealing with resources, this can have a substantial impact on how well the relaxed planning problem models the original: if a resource level is non-zero in the initial state from which a relaxed-plan is built, it is available at that non-zero level throughout. In FreeCell, for instance, if there is one free cell available in a given state, the relaxed plan to the goal from that state can make use of an effectively unlimited number of free cells. No action is able to reduce the number of free cells available by subsequent actions, as the delete effect that would establish that the free cell count is lowered when a card is placed in a free cell has been removed. This can lead, for instance, to relaxed plans which state that as many cards as necessary should be moved to a free cell and then the cards should be moved to the home cells in the correct order.

This over-optimism in the presence of a resource level of one by the relaxed planning graph heuristic has a profound effect on search: up to, and including the point, where there is still a non-zero resource level, as much of the resource as desired is available so a relaxed solution plan can be found. However, as soon as an action reduces a resource level to zero, the nature of the relaxed plan changes dramatically: if that resource is required then actions must be added to the relaxed solution plan to increase the resource level (assuming such resource-increasing actions are available). This sudden change in relaxed plan can lead to unforeseen dead-ends, or a sudden increase in relaxed plan length - both of which have a negative impact on search performance.

The multi-locatedness of mobiles under the ignore-delete-lists relaxation—that is, a mobile is available at all the locations it has ever been at thus far in the relaxed plan—can lead to some interesting relaxed plans being formed. Consider, for instance, a logistics problem in which a truck, beginning in location A, must collect a package from location E and deliver it to location A. The relaxed plan forwards from the initial state moves the truck from A to E (via B, C and D), loads the package into the truck and immediately unloads it at A: this ‘teleportation’ of the package from E to A, without the truck having to move back again, occurs because the fact that the truck is in location A was never deleted and, thus, the unload action placing the package at A is immediately applicable.

Relaxed Plan Refinement using Generic Types

Refining Relaxed Plans

In many cases, one can identify actions that are logically missing from relaxed plans that would need to be inserted in order to make the plan executable if delete lists were considered. Through analysis of the behaviour of known-typed objects in the plan, it is possible to suggest what some of the missing actions are, and produce a relaxed plan which is somewhat ‘less relaxed’ than it was previously.

When dealing with mobiles, if a precondition of one action demands that a mobile be in one location, and the precondition an action immediately following it demands that it be in another, then it is clear that actions to move the mobile from the former location to the latter would be necessary. As the map describing how the mobile can traverse between its locations is known, a path between all possible pairs of locations that may arise can be determined, in polynomial time, using the Floyd Walshall algorithm. The additional actions corresponding to the mobile moving along this path can be added to the relaxed plan, making it a closer analogue of a real solution plan, and increasing the heuristic cost by number of actions added.

The level of a resource is denoted by an assignment to a series of ranked objects. Actions which increase the level of the resource change the assignment denoting the resource level to a higher-ranked object; actions which decrease the level of the resource change the assignment to a lower-ranked object. By starting with the resource level in the state from which the relaxed plan was built, the cumulative resource-level effects of the actions in the relaxed plan

can be monitored: resource-increasing actions move the current resource level one place higher up the rank; resource-decreasing actions move it one place lower. If at any point an action attempts to move the resource level to off the top of the rank or off the bottom of the rank, a decreasing or increasing action needs to be inserted as appropriate. Such actions are not available in all cases; if they are not, a penalty can be added the heuristic value returned (the plan length) to dissuade search from considering plans whose relaxed solutions appear to violate resource limits. This is similar to the adjusted cost heuristic used in Sapa (Do & Kambhampati 2003), but as TIM provides finite bounds on the resource levels it is possible to penalise resource flows through the relaxed plan that would take the resource level both below and above its bounded values.

The effect of relaxed plan extraction on plan refinement

The process to extract a relaxed plan from a GraphPlan planning graph is designed to be as efficient as possible, to reduce the overhead of heuristic evaluation. When choosing an achiever for each fact, the first achiever found when building the planning graph is used. The first achiever found, however, varies between states, and can lead to dramatically different relaxed plans being built, even if the plan lengths are similar.

When refining the relaxed plans built in the conventional manner, the penalty is heavily dependent on the first-achieving actions found; in this case, adding actions to the relaxed plan adds noise to the relaxed plan length, making it difficult to decide which states are the most likely to lead to a goal. In an attempt to address this problem, two alternative plan extraction approaches are being investigated:

- A stochastic approach, called several times in an attempt to minimise noise, in which one of the achievers for each fact is chosen at random, rather than the first one found;
- A guided approach, called once, which uses a heuristic to choose which successor to use.

These alternatives will lead to differing heuristic values being found; which may lead to improved performance and/or shorter plans.

Using Lookahead with Refined Plans

The heuristics discussed are invariably more expensive than the baseline, unrefined, relaxed planning graph heuristic. The 'less-relaxed' plans found are, however, closer to being solutions to the original planning problem than unrefined relaxed plans; suggesting that it would be beneficial to use more than just the plan length as a heuristic value to guide search.

YAHSP (Vidal 2004), a planner which competed at the 2004 international planning competition, uses a lookahead approach to generate an additional successor to each state. The additional successor state is formed by applying as many of the sequenced actions from the relaxed plan as possible. In YAHSP, in an attempt to satisfy some of the unsatisfied preconditions of the actions in the relaxed plan, an attempt is made to find one action that would add the unsatisfied precondition. Adding action sequences to satisfy preconditions is not, however, considered: if satisfying a precondition requires more than one action, lookahead terminates.

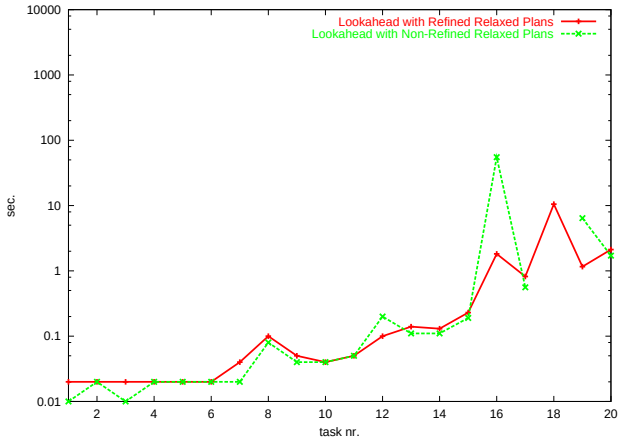


Figure 1: Time Taken To Solve Problems in the DriverLog Domain with Lookahead on Refined and Non-Refined Plans

Performing lookahead on the less-relaxed plan provided by the generic-type refinement, rather than the conventional relaxed plan, should allow more actions to be applicable in domains with recognised generic types. Within the refined plan, move action sequences to satisfy locatedness preconditions have been added; something which the lookahead procedure itself cannot do, as it only considers adding single actions to satisfy preconditions. The combination of these two techniques allows the low-cost of the lookahead procedure to be maintained, by it only considering adding single actions, whilst allowing action sequences to be inserted where these can be determined using the generic types analysis.

Using lookahead provides a further possibility: using the non-refined relaxed plan to provide a heuristic value; but performing lookahead over the refined plan. Such a configuration would have two benefits:

- lookahead can apply more actions than it would have done otherwise, as action sequences to achieve mobile locations have been added;
- the low-cost greedy relaxed plan extraction procedure can still be used, as the length of the non-refined plan (without the aforementioned noise) is taken to be the heuristic value.

Initial results in the DriverLog domain using this planner configuration, presented in figure 1, suggest that the use of refined plans in this manner increases the effectiveness of lookahead, providing a reduction in planning time. It can be seen that a small overhead is incurred through the analysis of the generic types in the domain, but in larger problems the reduction in planning time far outweighs this overhead. In particular, problems 16 and 19 are solved in less time, and problem 18 is solved where previously it was not (within the 30 minute time-limit to which the tests were subjected).

Selectively Introducing Delete Lists based on Generic Type Information

Another approach to making the relaxed problem more realistic would be to introduce some of the delete effects which are known to have controllable interactions within the problem, forming a 'partially relaxed' planning problem. In particular, if the delete effect when mobiles moved was maintained, the actions in the relaxed plan could not make use of a mobile being in two locations at once.

Two approaches are being investigated to use to solve the partially relaxed problem and return a heuristic measure:

- Using GraphPlan, as with the conventional relaxed planning problem, but handling the mutexes introduced by the added delete effects
- Using a simple partial-order approach, dealing with the mutexes by adding the necessary actions during plan time - for example, a mutex between two actions requiring a mobile to be at two locations can be dealt with by adding actions between the two to move the mobile from one location to the other.

Conclusions

This paper presented an overview of work investigating improved search guidance; with a particular focus on the idenfication and use of generic type information to provide better heuristic knowledge. To date, the relaxed plan extraction and lookahead techniques have been implemented and an evaluation is being performed. The implementation of the selective introduction of delete effects into the relaxed problem still in progress.

References

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Inteligence (IJCAI-95)*.

Bonet, B., and Geffner, H. 2000. HSP: Heuristic search planner. *Artificial Intelligence Magazine* 21.

Do, M. B., and Kambhampati, S. 2003. SAPA: A multi-objective metric temporal planner. *Journal of Artificial Intelligence Research* 20:155–194.

Fox, M., and Long, D. 2001. Hybrid STAN: Identifying and managing combinatorial optimisation sub-problems in planning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, 445–452. Morgan Kaufmann.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 161–170.

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

Long, D., and Fox, M. 2000. Automatic synthesis and use of generic types in planning. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2000)*, 196–205.

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS-2004)*, 150–159.

Integrating Macro-Operators and Control-Rules Learning

Rocío G. Durán
Departamento de Informática, Universidad Carlos III de Madrid
Avda. de la Universidad, 30 - 28911 Leganés (Madrid). Spain
rgduran@inf.uc3m.es

Abstract

Nowadays, planning is still a computationally unsolved task and many different learning techniques have been applied in order to improve its capabilities. In this paper we propose the integration of two learning methods sequentially: macro-operators and search control rules. Macro-operators provide us with a sequence of actions that are often executed in a given order. Thus, they avoid to plan that sequence each time it is required. However, the use of macro-operators increases the branching factor of the planning search tree, so the complexity of the planning process grows, and may produce a decrement of the planning performance. Our goal is to learn control rules that let us know when to use the macro-operators. Therefore, the search through the planning tree can be efficiently guided by the control rules. We show that this combination can be successfully applied in classical planning domains.

Introduction

Planning is a process that chooses and organizes a set of given actions by anticipating their expected outcomes. It is a task of Artificial Intelligence considered very complex and computationally hard, in which the search tree reaches a very big size and makes it difficult to find a solution. To reduce the difficulty of finding a solution plan, many solvers employ learning techniques, that acquire macro-operators, heuristics, search control rules, etc, whose results improve noticeably their original behaviour.

In this paper, we propose to use two of these learning techniques sequentially that acquire: macro-operators and search control rules. Firstly, we select the most common macro-operators, composed by two or three simple operators, obtained from the solution plans of a set of random problems. Secondly, we use the search tree of some problems solved with the macro-operators to learn control rules. These rules may include the macro-operators used in the plans previously generated. Finally, we compare the results of both learning techniques together with both techniques individually.

None of these two learning techniques are new, but the sequential use of both provides a novel way of applying the macros in the planning process. This method is much more selective that without the control rules, reducing the number of nodes of the search tree expanded and reducing the planning time.

In the next section we describe the planner and learning modules used in the experiments. The third section describes the method to decide the macro-operators and generate the control rules. The forth section shows some experiments with a version of the Logistics and the Miconic domains of the International Planning Competition. Finally, the last section introduces some conclusions and outlines future work.

The IPSS planner

Nowadays there are very different kinds of planners with different results for each domain. The planner used in this work is the IPSS planner, which provides the two learning modules we need in this work: macro-operators and control rules learning modules. IPSS is an integrated tool for planning and scheduling (Rodríguez-Moreno *et al.* 2004), which is based on PRODIGY (Veloso *et al.* 1995) as the planner component. PRODIGY is a nonlinear planning algorithm and it has been used for studying several machine learning techniques in the context of planning.

IPSS planner inputs are the domain and problem descriptions, generating as output a total-ordered plan, and the planning search tree. They can be used to learn macro-operators and seach control rules respectively, as explained next.

Macro-operators learning

A macro-operator is an operator composed by several simpler operators. It produces the same result than executing the simple operators sequentially. Their principal drawback is the utility problem (Minton 1988; McCluskey & Porteous 1997). The addition of macro-operators increases the branching factor and the processing cost per node, which can mean that they have worse search performance than not using them. Some other effects of using the macro-operators can be disadvantageous too: change of the order in which the search space is traversed (they change the order in which the primitive operators are used for obtaining a solution), change of the path costs, and increase of redundancy.

However, they can show significant improvement in different domains (Botea, Mueller, & Schaeffer 2005), by including into the macro-operators a partial ordering of its simple operators or combining the use of macro-operators

with techniques such as the relaxed graphplan computation implemented in FF. Therefore, a key issue consists on finding the good macro-operators, which can find faster a better plan.

In this work, we have selected the macro-operators using the frequency of appearance of several simple operators sequentially together in a set of obtained solution plans. IPSS provides a module to obtain a macro-operator from solving a problem in a given domain. It is also possible to select one operator subsequence from the solution plan to obtain a smaller macro-operator.

The HAMLET learning module

HAMLET is an incremental learning method based on EBL (Explanation Based Learning) and inductive refinement of control rules (Borrajo & Veloso 1997). The inputs of HAMLET are a domain, a set of training problems, and other learning-related parameters. HAMLET calls IPSS and receives as input the search tree expanded by the planner, in order to decide where and what to learn. HAMLET output is a set of control-rules that potentially guide the planner towards good quality solutions. In the context of this work, we use HAMLET to find a set of control rules that are able to learn when to use the acquired macro-operators.

Integration of macro-operators and control-rules

In this work, we have used both learning techniques together, with the aim of generating control rules that define when a specific macro-operator shall be used. To show the effectiveness of this approach, we show the results of using the two techniques separately and together: control rules in the original domain, macro-operators in the original domain and control rules after the macro-operators are acquired.

The first step is to select some macro-operators composed by two and three simple operators. We provide IPSS a set of random training problems to be solved. From the resulting total-ordered plans, all the different combinations of two and three operators have been obtained that appear one after the other and have, at least, one constant in common. The most common of them are selected for the second step.

The next step is, for each macro-operator, to insert them separately into the given domain and let the system learn control rules, using always the same training set of random problems. Learning control rules using the original domain (without macro-operators) is also executed, in order to compare the results.

Finally, the same test set is used for each resulting domain: (i) the original domain, (ii) the domains with each selected macro-operator, (iii) the original domain with its own learned control rules and (iv) the macro-operators and the control rules together. The main objective of this approach is to obtain good control-rules for each macro-operator and so, better results with this combination than using both techniques separately.

Experiments

This section describes the experiments performed in both the Logistics and Miconic domains. For each experiment, the learning parameters are the default ones. The only parameter modified is the time limit given to learn from the training problems and to solve the test problems. For both domains and in both cases, this value is always 30 seconds.

Logistics domain

We use the version of the Logistics domain, as it was first defined (Veloso 1994). The difference with the version created for the first IPC is that the predicates for describing where packages, trucks and airplanes are, have changed to at-object, at-truck and at-airplane.

We have used a random problem generator to create different problem sets for learning and test. These sets are the following:

- Macro-operator learning set (to obtain the most common macro-operator composed by 2 and 3 simple operators): 30 random problems with 3 cities, 3 objects and a maximum of 3 goals.
- Control-rules learning set: 30 random problems with 3 cities, 3 objects and a maximum of 3 goals.
- Test set 1: 30 random problems with 7 cities, 10 objects and from 1 to 10 goals.
- Test set 2: 40 random problems. 10 of them are of type (3, 5, 5), other 10 are (5, 10, 10), the next 10 problemas are (8, 15, 15) and the last 10 are (10, 20, 20), where (c, o, g) refers to number or cities (c), number of objects (o) and number of goals (g) respectively.

The characteristics of the sets are different because their different use. For instance, the problems generated for learning control-rules and macro-operators are “simple” problems (with small number of cities, goals and objects) to ensure that the planner is able to: (i) find solutions from which to generate macro-operators; and (ii) expand the whole search tree to obtain control rules. Test sets are also different. We have first created a test set with easy problems (from 1 to 10 goals), and a more complex set that contains problems with up to 10 cities, 20 objects and 20 goals.

We have learned several macro-operators of different types, following the approach introduced in the second section. The complete list is enumerated next, where we describe the operators that compose the macro-operator.

1. Macro m2-1: drive-truck unload-truck
2. Macro m2-2: fly-airplane unload-airplane
3. Macro m2-3: load-truck drive-truck
4. Macro m3-1: load-truck drive-truck unload-truck
5. Macro m3-2: drive-truck load-truck drive-truck
6. Macro m3-3: load-airplane fly-airplane unload-airplane

Tables 1 and 2 show the results of solving the problems of both test files respectively. They present percentages of solved problem (*Solved*) and number of used rules (*Rules*).

The IPSS column shows the results obtained with IPSS without control rules. The HAMLET column shows the results obtained by IPSS when using the learned control rules. The different rows describe the results obtained when different macro-operators are used. In the first row, no macro-operator is used. In the second one, the m2-1 macro is used, and so on.

Domain	IPSS	HAMLET	
	Solved	**Solved**	**Rules**
Logistics	20%	27%	9
Logistics + m2-1	13%	20%	4
Logistics + m2-2	43%	63%	8
Logistics + m2-3	8%	8%	6
Logistics + m3-1	13%	20%	4
Logistics + m3-2	13%	13%	9
Logistics + m3-3	23%	76%	9

Table 1: Percentage of solved problems of the test set 1.

The results obtained for the first test set are very satisfactory in two cases: m2-2 (fly-airplane+unload-airplane) and m3-3 (load-airplane+fly-airplane+unload-airplane). For both macros, the results are good with and without control rules. In the first case, the percentage of solved problems is 43%, more than double when compared with IPSS alone, that obtains a 20%. If we learn the control rules for that macro, the percentage increases up to 63%. When using control rules and the m3-3 macro-operator, this percentage increases up to 76% of solved problems. However, the table shows that the results depend on the macro-operator used and, for instance, when using the macros m2-1, m2-3, m3-1 and m3-2 without control rules, the performance is lower (13%) than when the macro-operator is not used (20%). In two of these cases (m2-1 and m3-1), their results using HAMLET improve the results of IPSS alone and, oddly, equal the results of IPSS in the original domain. In the other two cases (m2-3 and m3-2) their results using HAMLET are the same than the results of IPSS.

The results with the second test set provide a similar reading as described in Table 2. For macros m2-2 and m3-3 the performance raises from a 2% of problems solved up to 15% and 58% respectively. Thus, the macros that were useful in the previous test set are useful in this one too. IPSS default uses trucks before airplanes to load and unload objects in a location. With these airplane macro-operators it changes the preference and it seems to learn control rules that decide when to use the airplane macro-operators (m2-2 and m3-3). With the macros m2-1, m2-3 and m3-1, IPSS is not able to solve any problem, nor with HAMLET. Finally, the macro m3-2 keeps its results with IPSS equal than with HAMLET, only 3% solved problems.

Miconic domain

The version of this domain is the one used in the IPC-2000, as well as the 150 used problems. In this domain there are two types of objects: passengers and floors. The goal is to bring people using an elevator to different floors. We used the 10 most simple problems of the 150 problems of the competition to learn the control rules in HAMLET and the rest 140 to test. The first group are problems with two and four floors, while the second set has problems with from six up to sixty floors. The learned macro-operators in this domain are:

Domain	IPSS	HAMLET	
	Solved	**Solved**	**Rules**
Logistics	2%	0%	9
Logistics + m2-1	0%	0%	4
Logistics + m2-2	13%	15%	8
Logistics + m2-3	0%	0%	6
Logistics + m3-1	0%	0%	4
Logistics + m3-2	3%	3%	9
Logistics + m3-3	10%	58%	9

Table 2: Percentage of solved problems of the test set 2.

1. Macro m2-1: up board
2. Macro m2-2: board down
3. Macro m2-3: down depart
4. Macro m3-1: board down depart
5. Macro m3-2: up board down
6. Macro m3-3: board up depart

Table 3 shows the results of solving the test problems in the same format that in the previous tables.

Domain	IPSS	HAMLET	
	Solved	**Solved**	**Rules**
Miconic	3%	11%	3
Mic+m2-1	22%	17%	4
Mic+m2-2	12%	26%	3
Mic+m2-3	51%	52%	3
Mic+m3-1	17%	28%	3
Mic+m3-2	27%	34%	3
Mic+m3-3	51%	53%	3

Table 3: Percentage of solved problems of the test set.

Every macro-operator configuration has better results than the original domain, even with control-rules in HAMLET. So, except for the first macro-operator (up+board), the results with both techniques together improve over using only one of them or not using them. After analysing the solution plans from the Miconic domain using the macro-operators, these solutions are not semantically correct. In Figure 1, we can see the obtained solution plan using the macro-operator up+board for one simple problem. The first thing we can observe is the unnecessary use of operators. The second and third actions, for example, could be better replaced by down and board, instead of down and up+board. But the real problem is the fact of repeating the action **up-board f0 f2 p1** after boarding already the passenger p1 into the lift and, even **up-board f0 f1 p0** after serving the passenger p0 in the floor **f2**.

The reason of this behaviour is the definition of the board simple operator, which does not delete the predicate **origin**

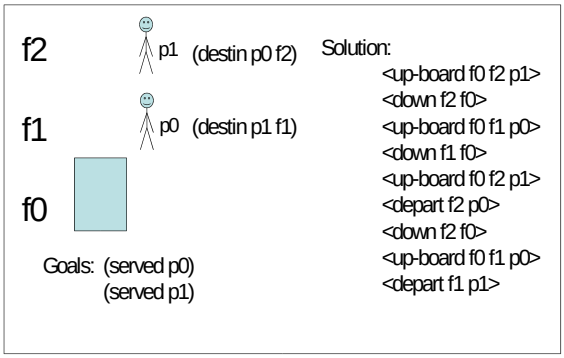


Figure 1: Example with macro-operator up+board.

p f. That means that the planner can board a passenger as many times as it needs, because no operator deletes the origin predicate. So, for example, if we have to go up to the second floor to leave passenger p0, the planner is going to select first the macro-operator up+board, which will try to move the lift up to the second floor, and board someone into the lift: passenger p1, who has there the origin. Nothing in this domain avoids this problem and the solution plan is incorrect.

In order to solve this problem, we added a new predicate: **(at-passenger p f)**, to know exactly where each passenger is and to avoid boarding them into the lift many times. This includes changing the definition of the Miconic domain and generating the correct definition of the macro-operators again. Finally, the new results for the Miconic domain are given in the Table 4.

Domain	IPSS	HAMLET	
	Solved	**Solved**	**Rules**
Miconic	3%	10%	2
Miconic + m2-1	16%	15%	4
Miconic + m2-2	1%	1%	3
Miconic + m2-3	11%	10%	3
Miconic + m3-1	4%	7%	3
Miconic + m3-2	1%	4%	3
Miconic + m3-3	15%	16%	3

Table 4: Percentage of solved problems of the test set.

Now, the results are not as good as before, but the plans are valid this time. Only with macro m3-3 we obtain better results with both learning techniques together than both techniques alone.

Conclusions and future work

In this paper, we have shown that the combination of macro-operators and control rules in the Logistics and Miconic domains can improve the results of the IPSS planner alone. We demonstrate that different macro-operators can be learned, and that their use does not always outperform the results of IPSS alone. However, when learning control rules to guide the search, the results over using the macro-operator alone improve.

We show, however, that there are some risks on the application of macro-operators: the learned macro-operators may solve no problem. Thus, to define which kind of macro-operators is good for this integration and which training problems are good to obtain the right rules, are two of the future research lines. That can include a new method to find good macro-operators.

Also, there are many domains in which this integration must be tested and we have to increase even more the number of simple operators that compose the used macro-operators.

A side effect of learning control rules on planning domains with macro-operators has also been finding extra knowledge about macro-operators: after acquiring macro-operators, we have seen a bug in the Miconic domain description that would be difficult to detect without using them, given when not using them IPSS would always generate valid plans.

Acknowledgements

This work has been partially supported by the Spanish MCyT project TIC2002-04146-C05-05, MEC project TIN2005-08945-C06-05 and regional CAM-UC3M project UC3M-INF-05-016.

References

Borrajo, D., and Veloso, M. 1997. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review Journal. Special Issue on Lazy Learning* 11(1-5):371–405. Also in the book "Lazy Learning", David Aha (ed.), Kluwer Academic Publishers, May 1997, ISBN 0-7923-4584-3.

Botea, A.; Mueller, M.; and Schaeffer, J. 2005. Learning partial-order macros from solutions. In *Proceedings of ICAPS'05*.

McCluskey, T. L., and Porteous, J. M. 1997. Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence* 95(1):1–65.

Minton, S. 1988. *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. Boston, MA: Kluwer Academic Publishers.

Rodríguez-Moreno, M. D.; Oddi, A.; Borrajo, D.; Cesta, A.; and Meziat, D. 2004. IPSS: A hybrid reasoner for planning and scheduling. In de Mántaras, R. L., and Saitta, L., eds., *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004)*, 1065–1066. Valencia (Spain): IOS Press.

Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI* 7:81–120.

Veloso, M. 1994. *Planning and Learning by Analogical Reasoning*. Springer Verlag.

Techniques for Generating Optimal, Robust Plans in the Presence of Temporal Uncertainty

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

Abstract

Planning under uncertainty has been well studied, but usually the uncertainty is in action outcomes. This work instead investigates uncertainty in the amount of time that actions require to execute. In addition to this temporal uncertainty, the problems being studied must have robust solution plans that are optimized based on an objective function. This extended abstract details two iterative approaches that have been used to solve these type of problems and discusses future work including over-subscription of goals and MDP approaches.

Introduction

Uncertainty applies to several aspects of planning problems and many planners have been built that prepare contingency plans when actions may affect the world in uncertain ways (Bresina *et al.* 2002). However, less work has been done with planners that assume action durations are 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 time consumed by the actions in the plan. This approach is often undesirable as it leads to missed opportunities and slack time in the plan when actions complete quickly (Bresina *et al.* 2002). 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 of a rock 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. My research focuses on finding ways to create robust plans where suboptimal actions are taken when time dictates, but optimal actions are executed when time allows.

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

Problem Specification

I am considering a class of problems with solutions that combine temporal uncertainty, optimality, and robustness, each of which is difficult to deal with individually and more so in combination. In this class of problems, action durations cannot be specified exactly and are represented by a closed interval [*min-d*, *max-d*], specifying the lower and upper bounds for the duration. Under this model, the actual duration required for an action is only known through observation after the action has executed. These duration intervals complicate the problem because solution plans are ranked by an objective function and the optimal solution is only attainable when actions complete quickly. This means that solutions found with the pessimistic assumption that all actions require *max-d* will be suboptimal, but optimal solutions found under the optimistic assumption that all actions require *min-d* (or any value less than *max-d*) are not guaranteed to execute to completion. The best solutions for these problems must be robust plans that are guaranteed to run to completion regardless of the amount of time actions require to complete. Plans that are robust in this sense are classified as *safe*. Considering all of these attributes, a *temporally uncertain planning problem* is defined as a quadruple $<$**D**, **I**, **G**, **M**$>$, where **D** is a domain description that lists the available actions (including interval durations and temporal constraints), **I** is a description of the initial state, **G** is a description of the goals, and **M** is a plan metric that represents the objective function for ranking plans.

Temporal Contingency Planning

One way to create optimal plans that are also robust in the face of temporal uncertainty is to build temporal contingency plans (i.e., plans with contingency branches that are taken based on the observed time at execution). At present, I have developed two related iterative approaches for generating temporal contingency plans. They differ in that one is a greedy algorithm and the other is a hill climbing algorithm. These algorithms are implemented in the planners PHOCUS-G (Foss & Onder 2005) and PHOCUS-HC (Foss & Onder 2006).

Both approaches follow a Just-In-Case style algorithm (Drummond, Bresina, & Swanson 1994) where a seed plan is generated, the points where it is likely to fail are located, and then contingency branches are inserted (when available)

at those points (Fig. 2). The two algorithms differ in the way that repairs are found when failure is possible. To generate the seed plan (line 1 in Fig. 2), temporal uncertainty is removed from the problem. This allows generation of plans using any planner that can handle durative actions, timed initial literals, and optimize based on an objective function[1]. Because it is assumed that the optimal plan is only attainable when actions complete quickly, *min-d* is assigned as the duration of each action. The resulting seed plan *P* returned by such a planner is temporally deterministic. My algorithm factors temporal uncertainty back in by converting *P* to a directed, edge-weighted graph called a *distance graph DG*, thus expressing *P* as a simple temporal network (STN) (Dechter, Meiri, & Pearl 1991). Figure 1 (b) shows a distance graph for a plan from a simplified rover domain. This conversion is described in detail in earlier work (Foss & Onder 2005).

Since *DG* contains all temporal constraints given in the domain, it can be used to determine when *P* becomes unsafe (line 11) (Dechter, Meiri, & Pearl 1991). In the loop that contains line 11, the plan is analyzed one step at a time to find the latest action *i* which makes the rest of the plan unsafe. If an action is found to be safe in line 11, the domain and the corresponding distance graph are updated to provide topmost flexibility to the earlier actions (lines 12,13), assuming the action requires its maximum duration. Otherwise, modifications are made so that *i* minimally uses the duration that causes the plan to fail and a new plan meeting the new constraints is sought for in one of the REPAIR-PLAN-* algorithms.

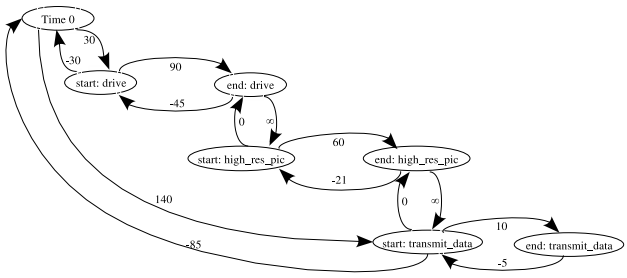
Figure 1 (c) shows how the distance graph in (b) has changed after several actions have been analyzed. First, it was found that the transmit_data action could execute to completion if it required its maximum duration of 10 time units. The distance graph was then updated to constrain transmit_data to always take 10 time units by changing the weight of the arc from end:transmit_data to start:transmit_data to -10. Next, it was found that even with the updated transmit_data constraint, high_res_pic could execute safely with any duration in its interval. The figure shows that this action was also then constrained to require its maximum duration. However, when the drive action was analyzed, it was found that the shortest path from start:drive to end:drive had a weight of 50 (this path is bolded in the figure). This indicates that high_res_pic and/or transmit_data may not have enough time to complete if drive executes for longer than 50 time units. At this point, a repair function must be called.

To apply the greedy approach, REPAIR-PLAN-G is called (Fig. 3). In this version, the initial conditions of the world are changed to represent the state of the world after all actions up to and including *i* are executed, assuming that *i* requires the amount of time that would cause failure in the original plan. Then, an attempt is made to generate a new plan which could be added as a temporal contingency branch on the original plan. If no such plan is found, this algorithm returns null and thus finds no solution. When applying the greedy approach, the seed plan is optimal in respect to the objective function. This optimal plan is never abandoned and it is augmented with branches that are each optimal, given the constraints used when generating them.

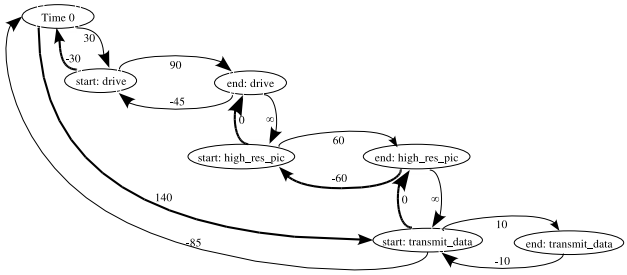
For the hill-climbing approach, REPAIR-PLAN-HC is

Execution Time	Action
30	drive_to_target
76	take_high_res_pic
107	transmit_data

(a)



(b)



(c)

Temporal Contingency Plan

at time 30: drive
if time < 81
  high_res_pic
  transmit_data
else
  low_res_pic
  transmit_data

(d)

Figure 1: (a) A seed plan for a problem from the rover domain. Note that the times given by the seed plan assume actions require their minimum durations. (b) The distance graph for the seed plan in (a), incorporating temporal uncertainty. For clarity, only the most important edges are shown. (c) The updated version of the distance graph in (b) after the transmit_data, high_res_pic, and drive actions have been analyzed. The bold arcs show the shortest path from start:drive to end:drive. (d) The temporal contingency plan generated by both the greedy and hill-climbing approaches.

[1]Currently LPG-td(Gerevini *et al.* 2004) is being used for this step because it handles durational actions and timed initial literals (used for specifying deadlines), creates parallel plans, and considers the objective function at planning time. It has also performed well in the International Planning Competition.

called (Fig. 4). Instead of modifying initial conditions, in this case the domain is modified so that *i* minimally requires the amount of time that would cause failure in the original plan. Then, an entire new plan is generated. If the new plan shares a head with the current plan, a contingency plan is formed. Otherwise, the new plan is returned and replaces the seed plan. As with the greedy approach, the initial seed plan is optimal with respect to the object function. However, the hill-climbing approach will abandon and replace the original seed plan either if no branches can be added to the seed plan to make it safe, or if a new seed plan has higher utility than the plan created by adding a branch to the old seed plan. In this way, the safest branch of the plan is optimized.

Both the greedy and hill-climbing versions of the algorithm benefit from the fact that they allow parallelism. This is especially important when deadlines are taken into consideration. Each approach has individual advantages, also. Intuitively, the greedy approach is faster when contingency branches can be added to repair the optimal plan. There are two related factors that contribute to this. First, the domain is modified so that the head of the plan will not be regenerated, restricting the search space. Second, because a contingency branch is shorter than a full plan, it is faster to generate it than to regenerate the entire plan as is done in the hill-climbing algorithm. However, the greedy approach fails to find any solution when no contingency branches can be added to the optimal plan. Since the hill-climbing approach always regenerates the whole plan, it is able to escape local minima/maxima. Also, the greedy algorithm may start with an optimal plan that is unlikely to be executed and augment this plan with very undesirable branches that are likely to be executed. In this situation the hill-climbing algorithm would abandon the optimal plan and find a sub-optimal, but likely to succeed plan that would have higher utility than the branches in the greedy algorithm's plan. Each algorithm has been independently implemented and tested (Foss & Onder 2005; 2006) and more experiments are planned to verify that these intuitive conclusions hold.

Related Work

The main framework of this algorithm is very close to Just-In-Case (JIC) scheduling (Drummond, Bresina, & Swanson 1994). The JIC scheduler analyzes a seed schedule, finds possible failure points, and inserts contingency branches so that valuable equipment time is not lost when an experiment fails. My work extends this framework to multiple planner goals, parallel plans, and nontemporal metrics, but does not currently consider probability of failure.

Several planners dealing with problems similar to those I am working with have been developed recently. Tempastic (Younes & Simmons 2004) is a planner that models continuous time, probabilistic effects, probabilistic exogenous events and both achievement and maintenance goals. It uses a generate-test-debug algorithm that generates an initial policy and fixes the policy after analyzing the failure paths. In producing a better plan, the objective is to decrease the probability of failure. Nontemporal resources are not modeled.

```
PHOCUS-* (D, I, G, M)
1: P_0 ← GENERATE-SEED-PLAN (D, I, G, M)
2: P_current ← P_0
3: loop do
4:    DG ← CONSTRUCT-DISTANCE-GRAPH(P_current,D,I)
5:    if SAFE-PLAN (P_current, DG, D, I, G, M) return P_current
6:    P_next ← MAKE-PLAN-SAFE (P_current, DG, D, I, G, M)
7:    if P_next is null return failure
8:    P_current ← P_next
MAKE-PLAN-SAFE (Plan P, DistanceGraph DG, D, I, G, M)
9:  for i = downto 1 in P
10:   maxAllowedDuration ← SHORTEST-PATH-DISTANCE(s_i, e_i, DG)
11:   if maxAllowedDuration ≥ max-d of i
12:     DG, D ← DG, D updated to constrain i to always require
                max-d of i
13:     DG, D ← DG, D updated to constrain i to always start at
                latest possible time that allows max-d of i
14:   else
15:     return REPAIR-PLAN-*(i,Plan P, D, I, G, M)
```

Figure 2: The shared PHOCUS-* algorithms.

```
REPAIR-PLAN-G (i, Plan P, D, I, G, M)
1: newMinDuration ← maxAllowedDuration + 1
2: I_mod ← I modified to represent the world after all steps up to i
          have completed and i has consumed newMinDuration
3: P_new ← generate plan with D, I_mod, G, M
4: if P_new is not null
5:    return a contingency plan created out of P and P_new
6: else
7:    return null
```

Figure 3: The REPAIR-PLAN-G algorithm. A greedy algorithm for finding temporal contingency branches.

Mausam and Weld (2005) describe a planner that can handle actions that are concurrent, durative and probabilistic. They use novel heuristics with sampled realtime dynamic programming in this framework to generate policies that are highly optimal. The quality metric includes makespan but nontemporal resources are not modeled in the planning problem. Prottle (Little, Aberdeen, & Thiebaux 2005) is a planner that allows concurrent actions that have probabilistic effects and probabilistic effect times. Prottle uses effective planning graph based heuristics to search a probabilistic AND/OR graph consisting of advancement and placement nodes. Prottle's plan metric includes probability of failure but not makespan or metric resources. Schaffer, Bradley and Chien (2005) developed a probabilistic approach for reasoning about uncertainty in continuous activity duration and resource usage. Their approach does not include contingency planning. They have shown robustness improvements over traditional non-probabilistic methods.

Future Work

Temporal contingency planning improves on conservative planning techniques by including the most conservative plan as the least desirable contingency branch, executed only

REPAIR-PLAN-HC (i, Plan P, D, I, G, M)

1: $newMinDuration \leftarrow maxAllowedDuration + 1$
2: $D_{mod} \leftarrow D$ modified so that action i requires $newMinDuration$
3: $P_{new} \leftarrow$ generate plan with D_{mod}, I, G, M
4: **if** P and P_{new} have the same steps through step i
5: **return** a contingency plan created out of P and P_{new}
6: **else**
7: **return** P_{new}

Figure 4: The REPAIR-PLAN-HC algorithm. A hill-climbing algorithm for finding temporal contingency branches.

when more desirable options may cause failure. The techniques currently implemented begin with an optimistic assumption that actions complete quickly and assume a uniform distribution over the uncertain duration interval. As I continue to work on these iterative approaches, I plan to consider what happens when the distribution is not uniform. The most likely case is that action durations will have a Gaussian distribution where most of the probability mass lies in the center of the interval. Considering this, it does not make sense to start with the assumption that each action requires only its minimum duration because that will result in a plan that is unlikely to execute to completion. Instead, it will be better to start with a value from the duration that is likely to occur, based on the given distribution. In this situation, opportunity branches can be added for when actions complete faster than expected, and contingency branches can be added for when actions run long. This may be a good anytime approach to be applied when there is a limited amount of time available for planning. In this circumstance it is important to spend the time available for planning to generate branches that will improve the plan in a significant way. By incorporating non-uniform distributions, I will be able to better determine when to stop branching because the expected utility gained is too small.

In addition to the rover domain, I have been working with problems from a travel domain and an evacuation domain. In the travel domain, the goal is to travel from home to some destination within a given time constraint. There are several different ways to reach the destination, but some modes of transportation are more expensive and the objective function in this domain is to minimize the amount of money spent. The challenge is that more expensive options, such as taking a taxi, are faster than less expensive options, like taking a bus. Optimally, the bus would be taken, but if this action comes after a flight that is running late, there may only be enough time to take the taxi. In the evacuation domain, the goal is to evacuate as many people as possible within a given period of time. This is further complicated by the fact that there are intermediate deadlines for rescuing different groups of people. As such, it is easy to create problems where it is not possible to evacuate all people, resulting in over-subscribed goals (Smith 2004).

Over-subscription is also an issue in the rover domain and most real world problems. I would like to develop techniques that directly address this issue. One approach is to simply achieve more goals when actions complete quickly and only the highest priority goals, otherwise. Another possibility is that entirely disjoint sets of goals may be attained on different branches of the plan.

Finally, I would like to investigate MDP approaches to solving planning problems with temporal uncertainty. Unlike the iterative planning approaches, MDPs do not naturally allow parallel actions. Even so, MDPs can be useful in this context because they naturally deal with uncertainty and take cost and rewards into account. One challenge in using MDPs to solve these type of problems is how to represent states when time is a factor. A naive approach is to include time in the state and thus have one state for each possible time increment. However, this would very quickly cause a blow-up in the size of the state space. It is likely that many states in this naive approach would be identical, only differing in time stamp. I plan to investigate ways to group states by time to reduce the number of states without sacrificing quality in the solution policy.

References

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*.

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.

Foss, J., and Onder, N. 2005. Generating temporally contingent plans. In *IJCAI 2005 Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains*.

Foss, J., and Onder, N. 2006. A hill-climbing approach for planning with temporal uncertainty. In *FLAIRS 2006 Conference*. To appear.

Gerevini, A.; Saetti, A.; Serina, I.; and Toninelli, P. 2004. Planning in PDDL2.2 domains with LPG-*TD*. In *International Planning Competition booklet (ICAPS-04)*.

Little, I.; Aberdeen, D.; and Thiebaux, S. 2005. Prottle: A probabilistic temporal planner. In *Proc. 20th National Conf. on Artificial Intelligence (AAAI-05)*.

Mausam, and Weld, D. S. 2005. Concurrent probabilistic temporal planning. In *Proc. 15th International Conf. on Automated Planning and Scheduling (ICAPS-05)*.

Schaffer, S. R.; Clement, B. J.; and Chien, S. A. 2005. Probabilistic reasoning for plan robustness. In *Proc. IJCAI 2005*.

Smith, D. 2004. Choosing objectives in over-subscription planning. In *Proc. 14th International Conference on Automated Planning and Scheduling (ICAPS-04)*, 393–401.

Younes, H. L., and Simmons, R. G. 2004. Policy generation for continuous-time stochastic domains with concurrency. In *Proc. 14th International Conf. on Automated Planning and Scheduling (ICAPS-04)*.

Controlability and Makespan Issues with Robot Action Planning and Execution

Matthieu Gallien
LAAS-CNRS*
7, Avenue du Colonel Roche, 31077 Toulouse Cedex 4, France
{matthieu.gallien}@laas.fr

Introduction

Nowadays, many robotic applications need autonomous decision-making capabilities. Among them, some make intensive use of planning. Yet, planning is an activity whose algorithmic complexity is often incompatible with the reactivity requirement of an exploration rover or a space probe.

In past years, some planners have proven their ability to handle complex situations required by autonomous systems. Some of these systems (e.g. RAXPS [Jonsson *et al.* 2000], CASPER [Chien *et al.* 2005]) have been deployed.

The IxTeT planner[1] [Ghallab & Laruelle 1994] was developed to handle such robotic planning problems. It was extended to handle complex resources [Laborie & Ghallab 1995], continuous domains and constraints between both atemporal and temporal variables [Trinquart & Ghallab 2001]. Further work [Lemai 2004] added a temporal executive to IxTeT.

Reasoning about time is necessary to address these planning problems. The planner must be able to take into account strict deadlines, temporal windows for some tasks, durative actions, and durative goals. The STN[2] [Dechter, Meiri, & Pearl 1991] formalism is often used in temporal planning because the requests on these networks are solved very efficiently by polynomial algorithms. Nowadays, an extension to uncertain constraints has been studied and a polynomial algorithm [Morris, Muscettola, & Vidal 2001] has been proposed.

Actual robotic space exploration missions are very expensive, with a high requirement for quality scientific returns. During the MER mission, the use of MapGen has allowed a 25% increase of such returns [Rajan 2004]. In a fully autonomous planner, optimization can be made in two ways: finding directly one good plan or searching through the whole search space several plans to find the optimal one. Due to limited computational capacity, the second approach is often unreasonable. So we have to modify the planner to search for high quality solutions.

New issues were raised while experimenting with IxTeT new executive. Some are related to temporal uncertainty, thus we decided to experiment another time framework. Some are related to efficiency, thus we decided to make a different heuristic to solve this issue. We use a robot simulator to make intensive tests of the proposed solutions. During the tests, it becames apparent that existing plan repair capabilities in IxTeT were in some cases unacceptably inefficient. In this paper, we describe a preliminary solution and some commented results. Future works will extend this work to try to make the plan repair mechanism complete and efficient.

Planning

IxTeT [Ghallab & Laruelle 1994] is a temporal constraint-based causal link planner using partially instantiated actions. Its planning algorithm is adapted from SNLP [McAllester & Rosenblitt 1991]. A time reified logic describes the evolution of state variables across the whole plan. IxTeT uses CSP techniques[3] to maintain the consistency of the plan constraints. In particular, the planner uses a Simple Temporal Network [Dechter, Meiri, & Pearl 1991] to represent the temporal constraint.

Definition 1 *A* temporal assertion *on a state variable v is either an* event *or a* persistence condition *on v.*

Definition 2 *A plan $\mathcal{P}(S, \Phi, G, CA, F, T)$ is described by the state variables contained in S. Φ is a chronicle describing all the temporal assertions of the plan. F is the set of defaults in the plan. $CA \subset \Phi$ contains temporal assertions on variables of S describing the predicted evolution of contingent attributes. The goals are in $G \subset \Phi$, they are persistence conditions on state variables of S. T is the set of tasks in the plan.*

The planner begins with a plan describing the initial situation, the initial goals and the known predicted evolutions of contingent attributes such as visibility windows. The search is performed until the plan contains no default. These defaults are temporal assertions unexplained in the current plan[4], conflicts between two temporal assertions or possible resource conflicts. At each search step, a default is chosen

*Part of this work has been funded by a grant from the ESF (European Social Fund)

[1] IxTeT is a system used for chronicle recognition, planning and temporal execution.

[2] STN: Simple Temporal Network

[3] Constraint Satisfaction Problem [Mackworth 1977] (CSP)

[4] A temporal assertion is not explained by a plan if it is not an initial condition or if no causal link establishes the assertion.

according to a given heuristic. One of the resolvants of this default is then chosen and applied. The planner only needs to backtrack on resolvant choices and not on default choices to be complete.

IxTeT uses a least commitment heuristic to evaluate a cost for each resolvant of each default. Then, a notation Opp(ρ) is computed for each one (see Lemai thesis [Lemai 2004] for more details). The basic idea is to minimize the size of the search space and to ease the choice between the resolvants of one default.

In order to make plans with a shorter makespan, we design a new heuristic. We have implemented it by modifying two costs of the old heuristic. The first considers one single ordering resolvant. The new cost depends on the earlier date of one of the first timepoint instead of the commitment of the resolvant. The second cost evaluates one causal link. Instead of using the maximum duration for computing the commitment, we now use the minimum duration. The idea is that the planner will make shorter links and thus makes shorter plans. This heuristic is called makespan minimizing heuristic.

Underlying CSPs

IxTeT uses classical CSPs algorithms for managing constraints on atemporal variables. It uses an STN for managing all the temporal constraints, and a general arc-consistency filtering algorithm for managing symbolic and numeric constraints.

In some cases, we want to link the effects of a task to its duration. For example, you need a mixed constraint between temporal and atemporal variables if a navigation duration depends on the navigation length and speed of the robot. IxTeT features a mechanism to propagate these constraints [Trinquart & Ghallab 2001].

On the STN, IxTeT always needs the minimal graph to be computed. If the network is not always propagated, the complexity of a request is not constant (i.e. $O(n)$). IxTeT makes a number of requests that is much higher than constraint updates [Vidal 1995]. It uses a path consistency algorithm like PC-2. An incremental version (only for constraint addition) is used during planning with a complexity of $O(n^2)$. For a constraint relaxation, the complete one is used in $O(n^3)$.

During execution, we will update the plan for example at each start or end of task. The CSP framework allows us to do this. A special care is taken to always keep the STN complete and minimal during execution. In fact the resource conflict detection, the plan repair mechanism and the propagation of mixed constraints need a complete graph. So the executive does not use a local temporal propagation like the one in [Muscettola, Morris, & Tsamardinos 1998]. The atemporal CSP is only kept arc-consistent for computational reason and because the system can repair or replan.

Simple Temporal Network with Uncertainties

Definition 3 *An STNU [Vidal & Fargier 1999]* $\Theta = (V, D, C_{clb}, C_{ctg})$ *with* V *the set of variables,* D *the set of domains. All constraints are in the form* $lb \leq v_i - v_j \leq ub$*. The set* C_{clb} *is all the controllable constraints equivalent to STN constraints.* C_{ctg} *is a set of contingent constraints. The duration of these constraints can only be observed.*

The introduction of a new type of constraint changes the consistency notion inherited from the STN. Three main levels of controllability have been defined [Vidal & Fargier 1999]. In IxTeT, we use the dynamic controllability. An STNU is dynamically controllable if the execution controller must take decisions knowing only the past observations and timepoint instantiations. The 3DC+ algorithm [Morris, Muscettola, & Vidal 2001] is known to establish it in polynomial time. The result is similar to STN's result (i.e. the minimal network). It introduces a new ternary constraint type called “wait" necessary to safely execute the STNU.

We have made two little improvements to this algorithm. The first is that the STNU in IxTeT are dynamic ones (i.e. constraints and variables are added during planning). Before any constraint addition, we remove all existing “waits”. The second one replaces the complete algorithm used to keep the STNU minimal during 3DC+ loop by the same incremental one used on STN.

Execution

IxTeT's executive runs a classical execution cycle corresponding to a “sense/plan/act" scheme. The executive begins with an initial plan produced by the planner.

All executable timepoints[5] are started as soon as possible except the end of actions labelled as “late preemptible" or “not preemptible".

The executive receives task reports, new goals or resource capacity changes. It has to check the validity of the task reports considering the current plan. If the report is not nominal, the system integrates the report, thus partially invalidate the current plan and triggers a plan repair if possible. All causal links possibly in conflict with new inserted tasks are removed during the relaxation. The execution can continue interleaved with the plan repair. If the failed plan does not anymore support the running tasks, all tasks are interrupted and a complete replanning is made. The new goals and resource capacity changes are integrated in the same manner.

Simulation and Results

IxTeT runs on the robot Dala and on a simulator of this robot. The simulator allows us to perform accurate tests of the different IxTeT strategies presented in the paper. The environment and the initial conditions can be exactly the same between runs.

We illustrate our contributions with an exploration rover like mission. The robot must acquire scientific data from several places. During its mission, it must communicates with an orbiter during visibility windows.

IxTeT now features two different planning heuristics and two different time management systems. This defines four IxTeT instantiations and we compare their performances using the simulator and the robot.

[5] IxTeT currently executes only a subset of the plan's timepoints: start and end of actions, goal and contingent timepoints.

During some missions with an STN, the system has a bad comportment. Due to multiple faillures, the system repaired many times the plan. The makespan was the maximum allowed duration for the mission. In that case, the system must cancel unachievable goals according to their priority. The system does not make that but due to STN propagations decided to keep a low priority goal instead of cancelling it. This is due to uncertain durations of tasks that have been squeezed.

The execution of the tasks to satisfy the low priority goal makes impossible to satisfy other goals with higher priority. The system makes exactly the worst case. Thanks to STNU, it may be impossible because uncertain durations are never reduced, thus it keeps enough time to execute the remaining tasks in the plan.

Using an STN and the makespan minimizing heuristic can produce up to 30% shorter plan. With an STNU, the value is approximatively 15%. During execution of the mission and depending on the world, the mission duration can increase by 15% removing the advantage of the new heuristic.

Results show that the combination of an STN and the makespan minimizing heuristic makes plans very unstable and breakable most of the times and sometimes make a very good and shorter execution. In general, the correct execution of the mission highly depends on the uncertainties. The new heuristic gives good results for the initial plan with STN or STNU, but if some plan repairs are made during execution the quality decreases significantly. The STNU produces stable and robust plans. Thanks to this, the whole mission is executed in a more reliable way.

Improve the Plan Repair Mechanism

We identify a drawback of the current plan repair process during our tests. Sometimes, a repaired plan contains unnecessary tasks leading to a suboptimal plan. For example, during our tests, we add new “take picture" goals. The planner produces a plan resulting in navigation from an existing waypoint to a new goal location and back from the new goal to the old one. This may lead to a very low quality plan.

This situation arises when the planning decision taken to satisfy a new goal make the old tasks not supported by the plan. So new tasks are inserted to support these tasks. In fact the set of tasks added to restore the state variable to their values before the new goals may be unnecessary, for example the navigation tasks. A better way is to relax the existing tasks so that they may be adapted to the new plan.

A preliminary solution

The example is a mission with initially five “take picture" goals and two communication goals. One “take picture" goal is added during the first communication. The initial plan is found in 1.7s. The simulator runs on a Pentium4 at 3GHz.

The problem comes from a limited relaxation of the plan before the plan repair process. The plan repair solution, described in the precedent sections of this paper, removes only causal links. A POCL planner using partially instantiated tasks, adds constraints on variables to make causal links valid. If these constraints remain after the removal of the link, the plan repair may produce a suboptimal plan. The solution is to remove the constraints at the same time than the link.

We integrated the algorithms described in [Surynek & Barták 2004]. We adapt it to continuous domains and use it to manage the filtering in the atemporal CSP. This permits to remove the atemporal constraints supporting a causal link. The current implementation does not remove temporal constraints because of the very small benefit. In fact, the number of temporal constraints added during resolution of conflicts between temporal assertions is much higher than constraints added with causal links. The relaxation of causal links temporal constraints does not significantly relax the plan, contrary to atemporal constraints.

Our proposition does not remove all removable causal links anymore. The ones added with task insertions are no more removed to keep as much as possible satisfiability decisions.

The planner finds a solution containing only necessary tasks or navigations. This solution is yet limited to simple cases where actions partial order allows the planner to find a new solution. The duration is rather similar than for the initial planning. From initial tests, the answer is that replanning may be faster to find a new plan but must interrupt all running tasks. In our test, the rover is navigating to its next goal and interleaving it with the plan repair resulting in zero delay for mission execution. The replanning, will introduce a delay before the navigation can be made. Yet a comparison of the duration of the plans produced by repair or replannning has not been done.

Ongoing and Future Works

The recording of constraints associated to causal links permitts to remove only some constraints before a plan repair. We will try to generalize this idea to record more explanations inside the plan. In fact, we want to be able to change task ordering when doing plan repair. By recording not only feasibility decisions but also the satisfiability decisions, we may be able to do that in the same way than the precedent work.

A promising way of research is to be able to explain why a task is in the plan and why it is in a specific time window. Using such explanation, one will be able to make local change on a plan in order to repair or improve it.

Any of these research ways may invalidate some hypothesis of the executive and may need to review all them in order to be able to use new repair capabilities. Clearly if one want to use plan repair, it must be globally interesting for the overall mission even if plan repair is longer to find a plan or if the duration of a repaired plan is greater.

We need also to improve the propagation of temporal constraint removals. In fact, this is the longest operation made during an execution cycle. The maximum duration of an execution cycle influences the task models and the reactivity to exogenous events. To safely execute the plan, the value must always be greater than the real cycle duration.

Conclusion

We have describe a temporal planner and executive whose plan execution raises new issues. The first one is to deal with uncontrollable durations. We use a temporal framework with explicit uncertainties. The second one is the bad quality of the plans when compared with a duration optimal plan. We modify the search control of the planner to find better plans by modifying the planning heuristic.

The integration of an STNU shows that it is usable on a rover. It shows a better robustness of the mission execution. If one goal is achievable, with 3DC+, it is executed.

A simulation architecture is used to evaluate the two solutions. During the test, the heuristic has shown a good robustness. Yet, an identified drawback limits the performance of this work. A solution using the plan repair ability is described in the last part of the paper.

We see that ongoing work improves the plan repair mechanism but this work is limited to only some case and may take more time than a complete replanning. Ongoing work is made to evaluate the opportunity of using new relaxation methods before a plan repair and to extend the relaxation.

References

Chien, S.; Tran, D.; Rabideau, G.; Cichy, B.; Davies, A.; Sherwood, R.; Castano, R.; Mandl, D.; Frye, S.; Trout, B.; D'Agostino, J.; Shulman, S.; and Boyer, D. 2005. The autonomous sciencecraft on earth observing one. In *i-SAIRAS-2005*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal Constraint Network. *Artificial Intelligence* 49(1-3):61–95.

Ghallab, M., and Laruelle, H. 1994. Representation and Control in Ixtet, a Temporal Planner. In *AIPS*, 61–67.

Jonsson, A. K.; Morris, P. H.; Muscettola, N.; Rajan, K.; and Smith, B. D. 2000. Planning in Interplanetary Space: Theory and Practice. In *Artificial Intelligence Planning Systems*, 177–186.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *IJCAI*.

Lemai, S. 2004. *IxTeT-eXeC : planning, plan repair and execution control with time and resource management*. Ph.D. Dissertation, LAAS-CNRS and Institut National Polytechnique de Toulouse, France.

Mackworth, A. 1977. Consistency in networks of relations. *Artificial Intelligence* 8:99–118.

McAllester, D. A., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *AAAI*, 634–639.

Morris, P. H.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In *IJCAI*.

Muscettola, N.; Morris, P.; and Tsamardinos, I. 1998. Reformulating temporal plans for efficient execution. In *Principles of Knowledge Representation and Reasoning*.

Rajan, K. 2004. Invited talk: Mapgen. In *IWPSS 2004, 4th International Workshop on Planning and Scheduling for Space, June 23 - 25*.

Surynek, P., and Barták, S. 2004. A New Algorithm for Maintaining Arc Consistency After Constraint Retraction. In *Principles and Practice of Constraint Programming*.

Trinquart, R., and Ghallab, M. 2001. An extended functional representation in temporal planning : towards continuous change. In *ECP*.

Vidal, T., and Fargier, H. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. *JETAI* 11(1):23–45.

Vidal, T. 1995. *Le temps en planification et ordonnancement*
Vers une gestion complète et efficace de Contraintes hétérogènes et entachées d'incertitudes. Ph.D. Dissertation, LAAS-CNRS.

Backdoors in Planning and Scheduling Problems

Peter Gregory and **Derek Long** and **Maria Fox**
University of Strathclyde
Glasgow, UK
firstname.lastname@cis.strath.ac.uk

Abstract

A backdoor is a set of assignments to a problem that yield the rest of the problem polynomially determinable. Backdoors have been shown to have interesting relationships with problem hardness, backbones and other properties.

We show deeper insight into the relationship between backdoors and backbones than has previously been shown. The effect of no-good clause learning on backdoors is also discussed.

The aim of this work is to find ways to exploit backdoors to solve problems more efficiently.

Introduction

Boolean Satisfiability (SAT) underlies the optimal STRIPS planner, Blackbox (Kautz & Selman 1999). It gained the first place in the 'optimal track' of the 2004 International Planning Competition (Hoffmann & Edelkamp 2004). The translation is effective because modern SAT-solvers incorporate sophisticated search features such as rapid-restarts and clause learning; these features in combination with efficient data-structures make SAT a high performance general problem solving framework. This work furthers previous studies on the relationship between two structures in SAT problems, the *backbone* and *backdoors*. The backbone of a SAT problem is the variables that are set in the same way in every solution. A backdoor of a SAT problem, is a set of variables that lead to the rest of the problem being polynomially determinable.

The *backdoor* has recently been seen as an important structure in SAT problems (Williams, Gomes, & Selman 2003), as it can explain how a rapid restart policy can improve chronological backtracking search. However, there are many unanswered questions about backdoors.

- How are backdoors distributed across the variable space?
- How *exactly* are backdoors and backbones related, if at all?
- How does the amount of symmetry in a problem relate to backdoor distribution?
- Does no-good clause learning affect the backdoor distribution of a problem?
- Are backdoor properties domain-independent, or are there specific *planning* backdoors, etc.

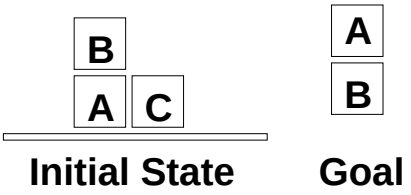


Figure 1: A blocksworld example to illustrate the major points in this paper. Can be solved in two time-steps, there are two different plans that solve the problem at this length. Note, in the goal, it is not specified that any block is on the table.

- Can backdoor variables be predicted with enough accuracy to directly solve problems with them?

The following example will be used throughout the paper as illustration of the concepts discussed. It is intentionally trivial, so that ideas can be clarified; but its structure is not so trivial that it isn't representative of harder problems. Figure 1 is a blocksworld planning problem. The goal is to reach a state where A is on B. This can be achieved in two ways: the first being 'put B on *the table* and then put A on B', the second being 'put B on *C* and then put A on B'.

Blackbox encodes STRIPS planning problems by first representing the problem as a plan-graph. The graph is then translated to a SAT instance where each variable represents either a fact (at some layer), a possible actions (including no-ops). The clauses represent the preconditions, effects, and mutexes between both facts and actions (Kautz, McAllester, & Selman 1996). The example SAT instance is the translation of the plan-graph at the first satisfiable layer.

Motivation

The aim of any academic study into the structure of problems should be to better understand that problem, and to develop novel problem solving techniques that exploit this new understanding. If backdoors can be characterised effectively, then it should be possible to search over small "candidate backdoors" at a much lower cost than searching the entire set of variables.

The challenges involved in this are manifold. A good prediction of the size of backdoor is required. Searching for an

unreasonably small backdoor would never find one, whilst searching for a backdoor much larger than the smallest ones would lead to redundant search. If we are to identify those variables in a problem that are most likely to be backdoor variables, then we need to have a good characterisation of how a backdoor is structurally composed. It is this characterisation that is detailed in this paper, along with some interesting effects that clause learning can have on backdoors. In previous research on backdoors, the backbone and backdoor sets been shown to be typically disjoint. For example, (Kilby *et al.* 2005) shows an empirical distinction between the two structures. In this work, we show analytically that there is a certain part of the backdoor that *necessarily* is not part of the backbone and that even if all of the backbone variables were set correctly, that would not be enough information to solve the problem (unless all variables are in the backbone).

SAT and Backdoors

In this work, we are concerned with the problem of Boolean Satisfiability. Boolean Satisfiability (SAT herein) is a special case of CSP. SAT restricts the domains of every variable to two values, $true$ and $false$. It also restricts the constraints to a set of clauses. A clause is a disjunction of literals. The variables correspond to logical variables, and the clauses disjunctions of logical literals, rather than writing $x_i = true$ and $x_i = false$, we will use the shorthand x_i and $\neg x_i$ instead. We will also occasionally refer to x_i and $\neg x_i$ as being in positive and negative *phase* respectively.

A sub-solver is an algorithm that solves a tractable subproblem of the general problem class. Paraphrasing Garey and Johnson (Garey & Johnson 1979), a subproblem of a general problem is obtained whenever we place additional restrictions on the allowed instances of that problem class. A sub-solver is an algorithm that determines only problem instances of a given subproblem.

The backdoor structure is reliant on the definition of a *sub-solver*, A, that has the following properties (given as input a CSP, C):

Trichotomy A either rejects the input C, or "determines" C correctly.

Efficiency A runs in polynomial time.

Trivial Solvability A can determine if C is trivially true (has no constraints) or trivially false (has a contradictory constraint).

Self-Reducibility If A determines C, then for any variable x, and value v, then A determines $C[v/x]$. (Williams, Gomes, & Selman 2003)

The informal definition of a backdoor is those variables which lead to a solution in polynomial time, when assigned correctly, or prove no solution exists for unsatisfiable problems. A *weak* backdoor can determine if a problem is satisfiable. That is, a set of variables wBD, for which there is at least one assignment such that A returns a satifying assignment. A *strong* backdoor can determine both satisfiability and unsatisfiability. For an unsatisfiable instance, a strong backdoor is defined as a set of variables, such that each assignment to those variables leads to A determining the instance unsatisfiable.

The definition of a backdoor requires a polynomial-time *sub-solver* (A in the definition). In this work, the sub-solver is assumed to be unit-propagation, and the CSP is always a SAT instance. Even when restricted to SAT, the sub-solver need not be unit propagation, it could be an algorithm that solves only 2SAT problems, or an algorithm that solves only horn-SAT problems, for example. The work will be as general as possible, as the concepts discussed (backdoors, backbones, no-good learning) are ubiquitous in CSP, planning, scheduling, and all search problems that can be modelled using constraints. Whenever I refer to *backdoors* from here, I refer to *minimal backdoors* as these are more interesting to study (The entire set of variables is trivially a non-minimal backdoor but it isn't a very interesting one, for example.)

Distribution of Backdoors

If we can find all of the minimal backdoors in a problem, we can easily calculate the total number of backdoors in the problem. As this is computationally prohibitive, we can use a sampling method to find representatives from the global set of minimal backdoors. The algorithm we use to find these is as follows:

```
ALGORITHM: MINIMAL BACKDOORS
-------------------------------------
1. s <- number of variables

2. while (!cutoff_limit){
3.  BD <- pick random variables (size s)
4.  backtrack over BD, if backdoor then{
5.   minBD <- MinimiseBackdoor(BD)
6.   s' <- |minBD|
7.   if (s' < s)
8.     s <- s'
9.   BDlist <- BDlist U {minBD}
    }
   }
-------------------------------------
```

The algorithm seeds the size of the backdoor to be the size of the instance. Then, while a cutoff limit is not reached, new backdoor candidates are selected. The algorithm backtracks over the variables in the candidate, if a solution is found (or unsatisfiability proven), we minimise the backdoor.

Minimisation is achieved by simply removing each variable, in order, and testing if the remaining structure is a backdoor. If it is, then the variable is not part of the minimal backdoor, and is discarded. If not, then the variable is reintroduced into the candidate. This is similar to the MINWEAKBACKDOOR algorithm in (Kilby *et al.* 2005). The difference being, that algorithm used *literals* and not variables as the constituents of the backdoor. This means that different instantiation of the variables in their backdoors could give a smaller *weak* backdoor. It also means that the (Kilby *et al.* 2005) procedure cannot detect (or minimise) *strong* backdoors. It would however have better runtime performance than MINIMAL BACKDOORS.

Problem	#Backdoors	Variables	Median BD Size	Backbone Size	Median BB Overlap
qg1-07	8	343	5.5	189	0
qg2-07	11	343	5	169	0
qg7-09	13	729	2	505	0
bw-medium	30	116	2	97	1
bw-large.a	31	459	3	459	3
bw-huge	38	459	3	459	3
flat30-5	9771	90	5	0	0
flat75-5	141	225	12	0	0

Table 1: Table of statistics for the studied instances. The studied instances are 3 quasi-group completion problems, 3 blocksworld problems, and 2 graph colouring problems, each from the satlib benchmark suite. The results show the number of minimal backdoors found in 10 minutes using the above algorithm.

Let us consider the blocksworld instance. Using unit propagation as the subsolver, there are two minimal backdoors. These relate to the actions (stack B C) and (move-to-table B) (both at timepoint 1). This is because, in two steps, the goal can be achieved by *either* putting B on the table or on block C first, then stacking A on B. Once we have decided which option to take, the mutex between the two actions causes propagation to imply the other one false. Everything else then propagates from this decision.

The Backbone and Backdoors

The backbone of a SAT instance is the set of variables that are implied by the model. More intuitively, it is the set of variables that take *the same* assignments in every solution. There are two variables in the SAT encoding of our example that are not in the backbone. These variables correspond to the actions (stack B C) and (move-to-table B). These have already been identified as the backdoor variables, no backbone variables are backdoor variables in this instance.

It has been previously observed that backdoor variables are not often backbone variables (Kilby *et al.* 2005). There is occasionally an intersection between the two structures, but it appears accidental. So a better question is: what is the reason that backbones and backdoors appear to be (typically) disjoint? Let us start by making some observations.

If all of the backbone variables are set correctly, could this be a backdoor? No. The backbone variables are those whose assignments are implied by the problem. Thus, if setting the backdoor correctly implied another variable/value assignment, this other variable *must be* in the backbone also. Once we have this piece of information, we can see that partial/ full assignments to backbone variables only have the capacity to imply *other backbone variables*. Since a backdoor implies *every* variable's value for a given solution, the backbone variables cannot be a backdoor.

As variables are assigned in search, the sub-spaces that we move into have monotonically growing backbones. Indeed, when the problem is solved using assignment and propagation, all of the variables are trivially in the 'backbone' (as in the final state all variables are set). Since we have shown backbone variables can only imply themselves, it is true that in any sub-space of the search tree, an algorithm would not want to make the choice of next variable one which is in the augmented backbone, as this can't imply any variables other than those in the augmented backbone.

Identifying Unique Solutions

When all variables are in the augmented backbone, then there is a single solution (in that sub-space). This doesn't mean that search is necessarily complete: some problems with single solutions can be hard to solve. But it does mean the problem is simplified to a state where it *might* be possible to solve the problem using propagation, because there is now a single solution in the sub-space.

Therefore, we have a necessary, but not sufficient, property of any backdoor – *assignment of part of the backdoor must identify a unique solution*. The next enquiry naturally concerns the question: how is the remainder of the backdoor composed? In this situation, several variables have been assigned such that, in the current sub-space, there is a unique solution to the studied instance (but the problem is not solved). However, there is not enough information in the current clauses to cause propagation of the remaining variables. One reason this can happen is that there are cyclic relationships in the clauses. Once a unique solution is found, if there remain cycles in the constraint graph, it may be necessary to 'cut' them in order to finish solving the problem. This is equivalent to finding the *cycle cutset* (Dechter 2003) of the remaining problem after a unique solution has been identified.

Another conjecture could be posited: if we had enough inferred knowledge of the problem at hand, then *no* backdoors would contain any backbone variables. This claim may seem unlikely, but some preliminary work has been carried out to suggest that it may not be. The conjecture centres around *inferred knowledge*. What *exactly* has to be inferred from the problem to reduce the size of backdoors? A "perfect" set of clauses can be imagined. With these clauses, any partial assignment that is made would lead to the entire augmented backbone being propagated. This would mean that the problem would have a unit clause for each variable in the backbone. If we actually had the "perfect" set of clauses, we could make any assignment in full confidence that the assignment led to a solution. Clearly we could never find such a model in reasonable time. But there are ways of bringing our model closer to this "perfect" model during search. We

can infer extra knowledge of a SAT instance using conflict-clause learning.

Clause Learning and Backdoors

Many SAT solvers make use of conflict clause learning. What happens to the backdoors in a problem as new clauses are learnt? It appears that as we learn new information, the size of a typical backdoor to a problem reduces. To illustrate how this happens, consider the following tiny SAT instance:

$(1 \vee 2) \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 1 \vee 2 \vee 3) \wedge (\neg 1 \vee 2 \vee \neg 3)$

In this example 2 is the only backbone member ($\neg 2$ implies 1 which in turn implies $3 \wedge \neg 3$, a contradiction, hence 2 is true in all solutions). There are two minimal backdoors to the instance, (1) and $(2, 3)$ (this is because 3 alone cannot cause any propagation).

The effect of choosing $\neg 2$ as the first decision, causes the contradiction on variable 3. Using FirstUIP conflict analysis, the generated conflict clause would be simply the unit-clause (2). With this clause in place, one of the previous minimal backdoors is now *non-minimal*. The backdoor $(2, 3)$ need not have 2 in it, as this is already implied by the conflict clause. Clause learning has reduced the size of the largest minimal backdoor and removed the backdoor values from it.

This one example only shows that clause learning can be used to reduce the size of backdoors.

Results and Discussion

The results in Table 1 were found by running MINIMAL BACKDOORS on each problem instance for 10 minutes. There are three classes of problems studied. The first are quasigroup completion problems. These are partially complete latin squares with additional constraints. These are useful in experimental design, for example scheduling a drugs trial. The second class of problem is the blocksworld planning problem. The third class of problem are graph colouring instances. The problems are benchmarks picked from the satlib web resource (Hoos & Sttzle 2000).

It is interesting to note the fact that the planning instances have very large backbones. This is because of the the fact that at the first satisfiable plan graph layer, the number of valid solutions to these blocksworld problems is extremely limited. The median backdoor sizes for every instance studied is tiny in proportion to the total number of variables in the respective problems. This is what was expected, and further indication that direct exploitation of backdoors is a promising research avenue.

Conclusions and Future Work

Backdoors and backbones are related structures, even though they do not often contain the same variables as each other. A backdoor causes the backbone to "grow" so that it covers all of the variables. The rest of the backdoor is composed of assignments that "fill-in" the missing information that renders the problem easily soluble. It also appears that clause learning can reduce the size of backdoors. Immediate future work includes rigourous empirical analysis of the relationship between backdoors and backbones; and between backdoors and clause learning.

Algorithms that predict which variables are backbone variables (Dubois & Dequen 2001) have been used previously to guide DPLL based search. If the algorithm they use can successfully predict backbone variables, then these variables are unlikely to be backdoor variables. Once the likely backdoor variables are found, then candidate backdoor sets can be generated from these, and tested using backtracking.

In DPLL based search, certain types of learnt clause are not useful. The decision variables are never used to generate conflict clauses as the same assignments will never again be visited in chronological search. However, this changes with restarts occurring, and will be just as important with search using backdoors. Which type of clause learning will be most useful in the development of a solver that exploits backdoors directly isn't clear, and is worthy of further work.

The final goal of this work is to create a SAT Solver that uses analysis of the structure of problems to find backdoors efficiently, so that problems can be solved faster and and problems that are currently out of reach of current solvers can be solved.

References

Dechter, R. 2003. *Constraint Processing*. Morgan Kauffman.

Dubois, O., and Dequen, G. 2001. A backbone-search heuristic for efficient solving of hard 3-sat formulae. In Nebel, B., ed., *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, 248–253. Morgan Kaufmann.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Hoffmann, J., and Edelkamp, S. 2004. International planning competition. http://ipc.icaps-conference.org/.

Hoos, H. H., and Sttzle, T. 2000. SATLIB: An Online Resource for Research on SAT. In I.P.Gent; H.v.Maaren; and T.Walsh., eds., *Proceedings of the Third International Conference on the Theory and Applications of Satisfiability Testing*, 283–292. IOS Press.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and graph-based planning. In Minker, J., ed., *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*. College Park, Maryland: Computer Science Department, University of Maryland.

Kautz, H. A.; McAllester, D.; and Selman, B. 1996. Encoding plans in propositional logic. In *Proceedings of the Fifth International Conference on the Principle of Knowledge Representation and Reasoning (KR'96)*, 374–384.

Kilby, P.; Slaney, J.; Thiebaux, S.; and Walsh, T. 2005. Backbones and backdoors in satisfiability. In *Proceedings of AAAI-2005*.

Williams, R.; Gomes, C.; and Selman, B. 2003. Backdoors to typical case complexity.

Computing action equivalences for planning

Natalia H. Gardiol, Leslie Pack Kaelbling
MIT Computer Science and Artificial Intelligence Lab
Cambridge, MA 02139
nhg@mit.edu,lpk@csail.mit.edu

Abstract

In order for autonomous artificial decision-makers to solve realistic tasks, they need to deal with searching through large state and action spaces under time pressure. We study the problem of planning in such domains. We show how structured representations of action effects can help us partition the action space in to a smaller set of approximate equivalence classes at run time. The pared-down action space can be used to identify a useful subset of the state space in which to search for a solution. This analysis allows us to collapse the action space and yields large gains in planning efficiency.

Introduction

In many logical planning domains, the crux of finding a solution often lies in overcoming an overwhelmingly large action space. In the blocks world domain, for example: the number of ways to make a stack of a certain height grows exponentially with the number of blocks on the table, so this apparently simple task becomes daunting very quickly. We want planning techniques that can deal with large state spaces and large, stochastic action sets, since most compelling, realistic domains have these characteristics.

One way to describe large stochastic domains compactly is to use relational representations. Such a representation allows dynamics of the domain to be expressed in terms of object *properties* rather than object identities, and, thus, yields a much more compact representation of a domain than the equivalent propositional version can.

Even planning techniques that use relational representations, however often end up operating in a fully-ground state and action space when it comes time to find a solution, since such spaces are conceptually much simpler to handle. In this case, a key insight gives us leverage: often, several action instances produce similar effects. For example, in a blocks world it often does not matter which block is picked up first as long as a stack of blocks is produced in the end. If it were possible to identify under what conditions actions produce equivalent kinds of effects, the planning problem could be simplified by considering a representative action (from each equivalence class) rather than the whole action space.

This work is about taking advantage of structured, relational action representations. We want to identify logically similar effects in order to reduce the effective size of the action space.

Related Work

The idea of exploiting symmetries in a planning problem in order to reduce the search space has a rich history. Fox and Long present a notion of symmetric states that is used to simplify planning (Fox & Long 1999; 2002; Fox, Long, & Porteous 2005). Two objects are defined to be equivalent if they have the same initial and final properties and attributes. In their most recent work, object symmetry (computed with respect to a pre-specified abstraction of the object relationships) is used to supplement the FF algorithm (Hoffmann & Nebel 2001) during search.

Guere and Alami (Guere & Alami 2001) also try to restrict search by analyzing domain structure. In their approach, they define the idea of the "shape" of a state. An algorithm is given to try to construct all the "shapes" for a particular domain instance. To extract a plan/solution, it looks for an action that connects a state in the starting "shape" to a state in the goal "shape". These shapes must be computed off-line for any particular domain instance.

The work of Haslum and Jonsson (Haslum & Jonsson 2000) shares a very similar goal: reduce the number of operators in order to reduce the branching factor and speed up search. They define the notion of redundant operator sets: intuitively, an operator is redundant to an existing sequence of operators if it does not add any new effects to the sequence. The set of redundant operators are computed before starting to plan; however, this is a computation that appears to be PSPACE-hard in general. An approximate algorithm is also given. Planning efficiency increases when these redundancies are found, but this kind of redundancy may not exist in all domains.

Additionally, Rintanen (Rintanen 2004) has looked at equivalence at the level of transition sequences for use in SAT-based planners.

The approach described in this paper, however, is intended to be a general method for reducing the action space that can be applied on-the-fly in a domain-independent manner. The equivalence classes of actions that are computed at each step produce an action set that can be used by any planning algorithm. We propose one such algorithm below.

Relational Envelope-based Planning

The Relational Envelope-based Planning algorithm (REBP) (Gardiol & Kaelbling 2004) is well-suited to address planning problems with large underlying spaces. It proceeds in two phases. First, given a domain theory and a problem instance, an initial plan of action is found quickly using classical planning techniques. Classical planning produces a focused search within high-probability sequences of actions, and yields an initial sequence called an *envelope* of states (Dean *et al.* 1995). Second, with additional time, this initial plan can be made more robust by considering deviations from the original envelope. Conditioned on a ground initial state, the number of states we expect to experience on the way to the goal is relatively small; thus, the effectiveness of REBP lies in limiting the state space in which policies searched for to an informative, reachable subset.

A fundamental step, however, is to produce the initial envelope efficiently. When the action space is large, however, this can be hard to do. In this case, a key insight gives us leverage: different ground action instances often produce qualitatively similar effects. For example, in a blocks world it often does not matter which block is picked up first as long as a stack of blocks is produced in the end. If it were possible to identify under what conditions actions produce equivalent kinds of effects, the planning problem could be simplified by considering a representative action (from each equivalence class) rather than the whole action space. The resulting reduction in branching factor can result in huge planning efficiency gains. Figure 1 shows an example.

Finding the initial envelope

Finding a trajectory of states with which to populate the initial envelope involves solving a planning problem from the ground initial state to a state satisfying a logical goal condition.

We represent planning domains in a subset of the PPDDL language.[1] A problem description contains the following elements: $\mathcal{P}$, a set of logical predicates, denoting the properties and relations that can hold among the finite set of domain objects, $\mathcal{O}$; $\mathcal{Z}$, a set of transition schemas; and $\mathcal{T}$, a set of object *types*. A schema $z \in \mathcal{Z}$, when applied in a state s, produces a *set* of ground actions, $z|_s$.

To find this plan, we execute heuristic-based search using the FF heuristic. (Hoffmann & Nebel 2001). The algorithm is shown in Figure 2.

Equivalence in relational domains

We need to properly define action equivalence in order to execute the steps b) and c) of the planning algorithm in Figure 2. To that end, we make the following crucial assumption:

Assumption 1 (Sufficiency of Object Properties)**.** *A domain object's function is determined only by its properties and relations to other objects, and not by its name.*[2]

[1] We do not consider conditional outcomes.

[2] What if we are in a setting in which a few objects' identities are in fact necessary? One could encode this information via supplementary properties, by adding a relation such as $\mathsf{block14}(X)$ that would only be true for block14. Obviously, if identity matters for a large number of objects, the approach described here would not be suitable.

```
1. Start with initial ground state, s and empty plan, P
2. Find state s', the best successor to s:
   a. calculate all ground actions applicable in s
   b. partition set of actions into equivalence classes
   c. apply a representative action a from each class,
      compute the most likely resulting state, s'
      evaluate s' using FF heuristic, h(s')
   d. if a unique state s' has the lowest h(s') value
        add the producing action, a, to P
   e. else,
        do breadth-first search until lowest h(s') found
      add the sequence of actions from s to s' to P
3. If s' is the goal, return the plan P.
4. Else, set s<=s', and return to step 2.
```

Figure 2: Planning algorithm. Note steps b) and c), which compute and make use of the reduced action space given by a partition over the actions.

For example, consider a blocks world in which the only two properties are the relation $\mathsf{on}()$ and the attribute $\mathsf{color}()$. Then if two blocks block14 and block37 are both red, are both on the table, and have nothing on them, they would be considered functionally equivalent. If block37 had another block on top of it, however, it would not be equivalent to block14. Intuitively, two objects are equivalent to each other if they are related in the same way to other objects that are, in turn, equivalent.

Here is the main contribution. We establish that a planning procedure that uses only equivalence-class representatives is complete whenever the original planning procedure, which had access to the whole action space, is complete. We need the following pieces: first, whenever goal is satisfied in a particular state s, then it must be satisfied by any state in s's equivalence class; second, equivalent actions taken from equivalent states produce equivalent successor states. These pieces let us construct an inductive argument to show that, from a given starting state, the successive substitution of one ground action by another in its equivalence class leads us to a state that still satisfies the goal.

Previous work on object equivalence, or symmetry, has used single, unary relations as a basis for computing similarity (Ellman 1993; Fox & Long 1999; 2002). However, we want to study object equivalence when more complex relationships are present. To aid our analysis, we view a relational state description as a graph, called the *state relation graph*. The nodes in the graph correspond to objects in the domain, and the binary relations between the objects correspond to the edges. For each pair of related nodes, we construct an edge representing the relation. In addition, nodes and edges are *labeled* with a string (or set of strings). Each node is labeled with the object's type, and each edge is labeled with the relation's name. If an object also participates in a unary relation, we augment its label set with that predicate's name. [3] Thus, we can establish equivalence between

[3] At present, we consider up to binary relations. In the case of relations with more than two arguments, we would have to consider

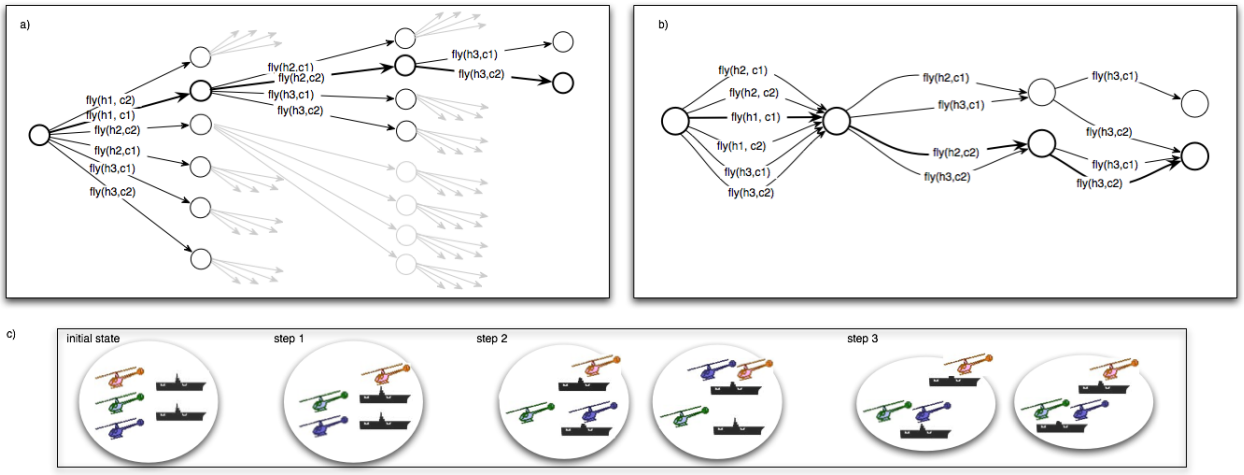


Figure 1: In this figure, we have an example domain in which the task is to fly each of the three helicopters onto one of two carriers. In a) is shown a picture of the search tree if we were to enumerate all the ground actions. However, there are only a few qualitatively different states, as seen on the bottom, in c). If we could eliminate distinguishing between actions that produce equivalent states, our search tree would be much more compact (b).

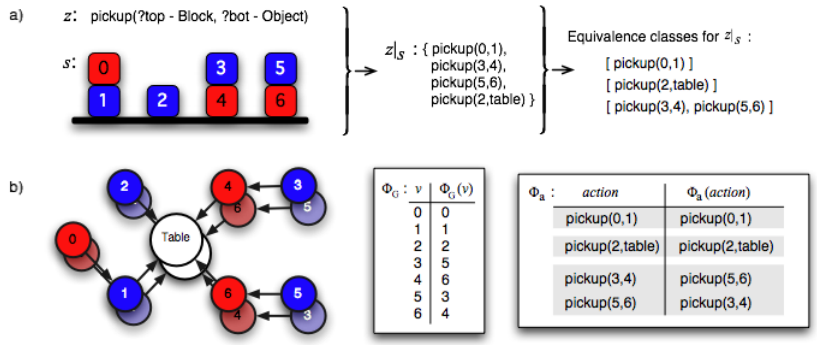


Figure 3: The steps involved in computing action equivalence. In part (a), the instantiation of the pickup operator z in a state s produces four ground actions. In part (b), the state relation graph for s shows we can map blocks 3 and 4 to blocks 5 and 6, respectively. This allows us to map the instantiation of `pickup(3,4)` to `pickup(5,6)`, and vice-versa. Thus, the four ground actions correspond to three equivalence classes.

two states by computing an isomorphism between the state relation graphs.

Definition 1 (State equivalence)**.** Two *states* are *equivalent*, written $s_1 \sim s_2$, if there exists an isomorphism, Φ, between the respective state relation graphs such that $\Phi(\mathcal{G}_{s_1}) = \mathcal{G}_{s_2}$.

Next, we need to define equivalence for actions. Intuitively, two actions should be considered equivalent if they produce equivalent states. However, this requires propagating a state through a transition rule for each calculation. A way to define action equivalence without doing such a propagation is to overload the notion of isomorphism to apply to sentences (of which actions are a special case).

Definition 2 (Action Equivalence)**.** The applications of action schema z in states s_1 and s_2 yield the sets of ground actions $z|_{s_1}$ and $z|_{s_2}$. Two ground actions $a_1 \in z|_{s_1}$ and $a_2 \in z|_{s_2}$ are equivalent if and only if there exists a Φ such that $\Phi(\mathcal{G}_{s_1}) = \mathcal{G}_{s_2}$ and $\Phi_a(a_1) = a_2$.

Essentially, we will be grouping two instances of an operator into the same equivalence class if there exists an *automorphism* between objects in the state that allows us to re-write one action instance as the other. Figure 3 shows an example of this computation.

Now we move to the next important step: we need to guarantee that if the goal condition, if satisfied in a particular state s, can be satisfied by any state equivalent to s. We prove that if a logical sentence is satisfied in a state s, then it is satisfied in any state $\tilde{s} \in [s]$, where $[s]$ is the equivalence class of s. We must be clear about the logical setting: we assume that an un-ground sentence (i.e., a goal condi-

a hypergraph representation to allow for edges of more than two nodes.

tion) contains *no* constants, and that a ground state is a fully ground list of facts (which we can treat as a conjunction or set of ground relations).

We provide one more definition for an important intermediate concept:

Definition 3 (Equivalent Planning Procedures)**.** Let P be a planning procedure such at at each state s, P selects an action a. Consider a planning procedure P' such that at each state $\tilde{s} \sim s$, P' chooses an action $\tilde{a} \sim a$. Then P and P' are defined to be *equivalent planning procedures*.

Theorem 1. *Let P be a complete planning procedure. Any planning procedure P'* equivalent *to P is also a complete planning procedure. That is,*[4]

$$\gamma(a_1, \ldots, a_n, s_0) \rightarrow g \Rightarrow \gamma(\tilde{a_1}, \ldots, \tilde{a_n}, s_0) \rightarrow g$$

Thus, any serial plan that exists in the full action space has an equivalent version in the partitioned space. (**?**)

Experimental Validation

As a check, we did a small study to illustrate the computational savings of planning with equivalence class sampling. Figures **??** shows these results. The experiments were done in the ICAPS 2004 blocks-world domain, varying the number of blocks from 2 to 7. In each case, the goal was to stack all of the blocks, and the starting state was with all blocks on the table. The x-axis of the graphs shows the plan step, and the y-axis shows the number of actions expanded in the search at that step. The top graph shows a linear y-axis, and the bottom graph shows it log-scale. Each curve corresponds to the performance of each algorithm in each size blocks world. The dashed lines correspond to the planning algorithm that uses all the actions, and solid lines correspond to the planning algorithm that uses a representative from each equivalence class.

With just five blocks in the domain, already the combinatorial growth in the branching factor is such that searching in the whole action space is hopeless. The equivalence-class based planner shows a consistently small branching factor even with six and seven blocks. The computational savings of computing the action classes is significant even in this small test domain. Further experiments are forthcoming in other domains from the ICAPS planning competition.

Conclusion

This is work explicitly attempts to define what it means for planning operators to be equivalent in the presence of complex relational structure. We formalize such a definition and illustrate the benefit of equivalence-class analysis for planning.

Taking advantage of structured action representations helps us ignore the distracting complexity and focus instead on the interesting complexity in a problem. We provide a formal basis for computing action equivalence classes that guarantees a complete planning procedure while significantly reducing the branching factor of the search. While our original motivation is the REBP algorithm, our findings are useful for efficient planning in general.

[4]Some notation: $\gamma(a_1, \ldots, a_n, s_0)$ denotes the *state* that results from executing the sequence of actions $a_1, \ldots, a_n$ starting from state s_0. The arrow denotes entailment.

References

Dean, T.; Kaelbling, L. P.; Kirman, J.; and Nicholson, A. 1995. Planning under time constraints in stochastic domains. *Artificial Intelligence* 76.

Ellman, T. 1993. Abstraction via approximate symmetry. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In *16th International Joint Conference on Artificial Intelligence*.

Fox, M., and Long, D. 2002. Extending the exploitation of symmetries in planning. In *AIPS*.

Fox, M.; Long, D.; and Porteous, J. 2005. Abstraction-based action ordering in planning. In *International Joint Conference on Artificial Intelligence*.

Gardiol, N. H., and Kaelbling, L. P. 2004. Envelope-based planning in relational MDPs. In *Advances in Neural Information Processing 16 (NIPS-2003)*.

Guere, E., and Alami, R. 2001. One action is enough to plan. In *International Joint Conference on Artificial Intelligence (IJCAI'01)*.

Haslum, P., and Jonsson, P. 2000. Planning with reduced operator sets. In *Artificial Intelligence Planning Systems*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14.

Rintanen, J. 2004. Symmetry reduction for SAT representations of transition systems. In *International Conference on Automated Planning and Scheduling*.

Concurrent Planning using Petri Net Unfoldings

Sarah Hickmott
National ICT Australia and
School of Electrical and Electronic Engineering University of Adelaide, Australia

Abstract

Those in the planning community previously exposed to the construct of Petri nets, have probably recognised a connection between the world modelled by these nets and a concurrent planning domain. Work to date however has failed to sufficiently capture and motivate the possible benefits of developing and utilising this connection, so the area has remained relatively undeveloped. We believe the factored state representation and concurrency semantics of Petri nets are so closely related to that of concurrent planning domains that cross cultivation between the areas will give theoretical insight and lead to the development of computationally efficient algorithms for concurrent planning. Our research seeks to develop the theory required to exploit the connection and create models and algorithms for optimally solving deterministic and probabilistic concurrent planning problems using low level Petri nets. Our approach casts the planning problem as a Petri net reachability problem. Unfolding is an attractive method for reachbility analysis because it utilises and maintains the concurrency and factored state structure of the Petri net. We capitalise the unfolding process for planning by guiding it with heuristics.

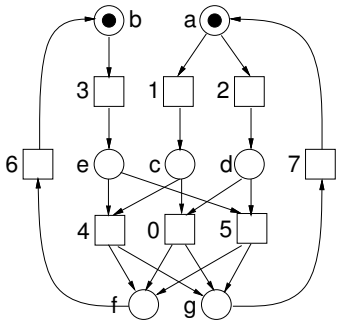
Petri nets are traditionally used for modeling and analysis of distributed systems. They can be used to exploit the structure of a planning domain in two beneficial ways. Firstly Petri nets, like STRIPS and PDDL operators, provide a non-flat representation of transition systems. They avoid explicitly enumerating the state space as it is implicit in their representation of variable -action relationships. This can be utilised in computation. Secondly Petri nets specifically represent concurrency and causal relations between actions. Consequently it is possible to avoid enforcing a total order on actions; this can have not only computational advantages via reduction of the policy space but also allows us to generate partially ordered plans.

We propose a translation from deterministic and probabilistic planning problems to Place-transition (PT) Petri nets. We then unfold the net in a specific manner. The unfolding of a Petri net maintains a partial ordering of actions based on causality, allowing us to search for an optimal solution plan without considering the unnecessary interleaving of actions or enumerating the state space entirely. For the deterministic case, we adapt the Esparza-Vogler-Romer (EVR) unfolding algorithm so that either the minimum cost partial plan is found during the unfolding procedure, or we identify that the goal is not achievable. We capitalise the unfolding by guiding it with planning heuristic. Note we are not restricted to unit-cost actions. For the probabilistic case, we unfold the Petri net up to a finite horizon. We propose a Bellman-like equation which then uses the unfolded net to determine the partial plan with the maximum probability of success, or expected reward, within the finite horizon. This later case does not require that all rewards be positive, and thus could be used for over-subscription planning.

Translation of a Planning Problem to a Petri net

A Place Transition (PT) net is a low level Petri net. A PT-net is a 5-tuple $PN = (P, T, F, W, M_0)$ where P and T are finite sets of places and transitions respectively; $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation; $W : F \rightarrow 1, 2..$ is the weight function; M_0 is the the initial marking; $P \cap T = \emptyset$ and $P \cup T \neq 0$. Figure 1 shows a simple PT net. The marking M of a Petri net is the mapping of tokens to places, and represents the state of the modelled world. The preset of a node x in the net, ${}^\bullet x$, is the set $\{y \in P \cup T | W(y, x) \geq 1\}$. The postset of a node, $x^\bullet$, is the set $\{y \in P \cup T | W(x, y) \geq 1\}$. A transition t is enabled if each of its input places p is marked with the weight of the arc connecting p to t. A particular marking M enables a transition t if $W(p, t) \leq M(p)\ \forall p \in P$. The occurrence of a transition absorbs the tokens in its input places and produces tokens in its output places thus moving the net from M to the new marking $M'(p) = M(p) - W(p, t) + W(t, p)\ \forall p \in P$. This corresponds to a state transition of the modelled system. A set of transitions T' is concurrently enabled at the marking M if it is possible for all $t \in T'$ to occur at once, viz. $\sum_{t \in T'} W(p, t) \leq M(p) \forall p \in P$. A more detailed review of Petri nets can be found in (Murata 1989).

A planning problem is a quadruple $\langle A, I, O, G \rangle$ where A is a set of state variables, $I : A \rightarrow \{0, 1\}$ is a state, O is a set of STRIPS operators, and G is a set of goal literals (Ghallab, Nau, & P.Traverso 2004). The set of literals over A is $L = \cup\{\neg a | a \in A\}$. The complement $\bar{l}$ of a literal $l \in L$ is defined by $\bar{a} = \neg a$ and $\bar{\neg a} = a$ for $a \in A$. A STRIPS operator $\langle p, e \rangle$ is 1-safe if $\{\neg l | l \in e\} \subseteq p$. A STRIPS operator $\langle p, e \rangle$ has a positive precondition if $p \subseteq A$. The first part of our translation involves mapping the planning problem to an equivalent one where every operator is 1-safe one, and there are no negative preconditions. 1-safety is established



Places are represented by circles, transitions by squares and token by black dots. Firing transitions 2 and 3 (in any order or concurrently) followed by firing transition 5, would result in a single token each in places f and g.

Figure 1: Example of a Petri net.

by replacing every operator by several 1-safe ones; we define $S(o)$ as set of operators obtained from some $o \in O$. Negative preconditions are eliminated by replacing each $\neg a$ by a corresponding positive precondition $\hat{a}$ and forcing $\hat{a}$ and a to always have opposite values. The reasons for this translation become clear soon.

Given a deterministic planning problem $R = \langle A, I, O, G\rangle$ we create a PT $pnet(R) = \langle P, T, F, W, M_o\rangle$ such that:

- the places are $P = A \cup \hat{A}$,
- the transitions are $T = \{S(o)|o \in O\}$,
- the set F of arcs is obtained from $t_0 = \langle p, e\rangle \in T$ as

$$\begin{aligned} & p \times \{t_0\} \\ & \cup\{\langle t_o, a\rangle | a \in A \cup \hat{A}, a \in p, \neg a \notin e\} \\ & \cup\{\langle t_o, a\rangle | a \in A, a \in e\} \\ & \cup\{\langle t_o, a\rangle | a \in A, \neg a \in e\} \end{aligned}$$

- $W(f) = 1$ for all arcs $f \in F$,
- for all $a \in A$, $M_0(a) = 1$ iff $I(a) = 1$ and $M_0(\hat{a}) = 1$ iff $I(a) = 0$, and for all $a \in A \cup \hat{A}$, $M_0(a) = 0$ or $M_0(a) = 1$.

The 1-safety of the STRIPS operators has allowed us to create 1-safe Petri nets, meaning it is not possible for more than one token to exist in a place. Without 1-safeness it would become complicated to maintain consistency in the net, as one would have to consider the semantics of multiple tokens in a place: if the respective literal becomes false all these tokens must be removed.

Petri net Unfolding

Unfolding is a method for reachability analysis which exploits and preserves the factored state representation and concurrency information in the Petri net. In the planning context this confers the ability to reason about partially ordered sets of actions directly, without having to consider their interleavings. It also enables the recognition and separate resolution of independent subproblems. During the planning process we can reason about the actions and cost required to assert some subset of state variables, and combine this information with that for another (thus-far) independent subset, at a future point when the two sets are no longer independent. The benefit of this 'divide-and-conquer' approach depends on the level of concurrency in the domain.

The unfolding of a PT-net $N = (P, T, F, W, M_0)$ produces an occurrence net $ON = (B, E, F')$, whose node sets are conditions B and events E. These nodes represent particular occurrences of the places and transitions, respectively, in possible runs of the original net from the initial marking. The unfolding achieves this by eliminating cycles and backward conflicts. Backward conflict is the case when two transitions output to the same place; by eliminating this we know exactly which transitions are involved in a particular marking. In the context of planning, this means that we know the exact set of actions that, when executed, lead to a state variable reaching a certain value at some point in the plan. The labelling function φ is a homomorphism from ON to N.

The main theoretical notions required to understand unfolding are that of a configuration and local configuration of an event. A configuration represents a possible partial run of the net. It is any set of events C such that: C is causally closed, e $\in C \Rightarrow e' \in C\ \forall e' \leq e$; and C contains no forward conflict, ${}^\bullet e_1 \cap {}^\bullet e_2 = 0\ \forall\ e_1, e_2 \in C,\ e_1 \neq e_2$. We can think about a configuration as a partially ordered plan. The local configuration of an event e, denoted $[e]$ is the minimal configuration containing event e. Consequently if we introduce a 'goal' transition t_g whose predecessors correspond to the set of goal literals G, then a solution plan is any partially ordered set of operators [e] such that $\varphi(e) = t_g$. A configuration C can be associated with a marking Mark(C) of the original net by identifying those conditions whose tokens are produced but not consumed after firing the events in C starting from the initial marking: Mark$(C) = \varphi((M_0 \cup C^\bullet)\backslash {}^\bullet C)$, where $C^\bullet = \{e^\bullet | e \in C\}$ and ${}^\bullet C = \{{}^\bullet e | e \in C\}$.

The unfolding process involves identifying which transitions are enabled by conditions currently in the occurence net that can be simultaneously marked. These transitions are referred to as the possible events. A new instance of each is added to occurence net, as are instances of the places in each of their postsets. The question of whether a set of conditions can be simultaneously marked is answered by determining whether the union of the local configurations of their presets forms a configuration. Figure 2 shows an example of unfolding.

Finite Complete Prefix of Unfolded net

In most cases, the unfolding β of a Petri-net is infinite. For this reason, we seek a complete finite prefix β' of β, one which contains as much information as β.

The key to obtaining a complete finite prefix is to identify those events at which we can cease unfolding without loss of information. Such events are referred to as *cut-off events* and are defined in terms of an *adequate order* on configurations (McMillan 1992; Esparza, Römer, & Vogler 2002):

Definition 1 *A partial order $\prec$ on the finite configurations of a branching process is an adequate order if*

1. *$\prec$ is well founded*
2. *$\prec$ refines $\subset$: $C_1 \subset C_2 \Rightarrow C_1 \prec C_2$*

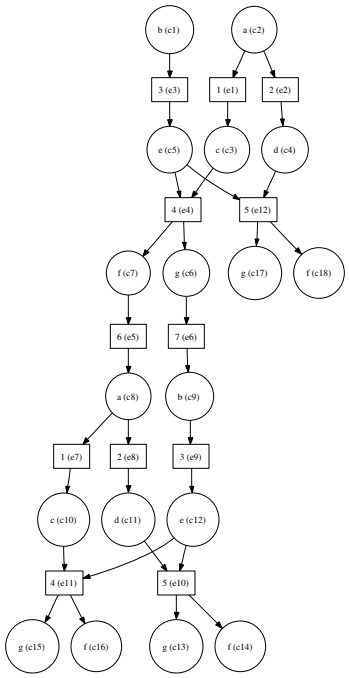


Figure 2: Prefix of an unfolding of the Petri net in Figure 1.

3. *$\prec$ is preserved by finite extensions: if $C_1 \prec C_2$ and Mark(C_1) = Mark(C_2) then $C_1 \oplus E \prec C_2 \oplus I_1^2(E)$, where $I_1^2(E)$ is an isomorphism mapping the finite extension E of C_1 onto the extension of C_2.*

Without loss of information, we can cease unfolding from an event e, if e takes the net to a marking which can be caused by some other event e' such that $[e'] \prec [e]$. This is because the events (and thus markings) which proceed from e will also proceed from e'. Relevant proofs can be found in (Esparza, Römer, & Vogler 2002):

MOLE (*http://www.fmi.uni-stuttgart.de/szs/tools/mole/*), is a free-ware unfolder which can be used for 1-safe PT-nets. It uses an adequate order on configurations, $\prec$, which is based primarily on comparing their cardinality. The prefix shown in Figure 2 is the complete finite prefix MOLE returns for the net in Figure 1.

Deterministic Concurrent Planning

Once the problem is translated to a PT-net, it is easy to let MOLE produce a partially ordered plan for that problem. It suffices to augment the STRIPS operator set with a dummy operator whose precondition is the goal, and to require mole to stop whenever an event labelled with the corresponding transition is added to the occurence net. The local configuration of this event is a partially ordered plan for the problem. Further, owing to the fact that MOLE orders events by increasing local configuration cardinality, this plan is minimal in the number of actions.

The cardinality-based ordering relation used by MOLE has a serious drawback for planning however, as it leads MOLE to perform a breadth-first search. If we were to swap the ordering to prefer events with larger local configurations to those with smaller ones, we would force MOLE into a depth-first search. However, since the resulting ordering is not adequate, there is no guarantee that the resulting planner would be complete. Checking for loops is not as straightforward in the unfolding framework as it is in a state-space search framework, since the markings of local configurations only have a partial view of the state. Indeed, the main purpose of an adequate order is to implement a form of loop detection. A natural idea is to change the ordering to provide MOLE with better guidance towards the goal, while preserving, and even generalising from the restricted notion of optimality currently in place. This rejoins the work on directed model-checking pionneered by Edelkamp et al. (Edelkamp, Lluch-Lafuente, & Leue 2001).

We propose that given an arbitrary *monotonic* heuristic, it is possible to build an adequate order which implements A*, letting the heuristic guide the unfolding towards optimal plans. Monotonic heuristics which, like h^m (Haslum & Geffner 2000), can be automatically generated from a planning problem description, are equally easily generated from PT-nets. In planning terms, let cost(o) be the (positive) cost of operator o, and res(o, s) be the result of applying o in state s. A Heuristic h (such that $h(s) \geq 0$ everywhere and $h(s) = 0$ at goal states) is monotonic iff $h(s) \leq h(\text{res}(o, s)) + \text{cost}(o)$ for all non-goal states s and operators o applicable in s. These definitions easily transfer to the PT-net case, by identifying each operator with the corresponding transition and considering a set of places P as the state in which all state variables but those in P are false. We define the following ordering on configurations:

Definition 2 ($\prec_h$) *Let h be a monotonic heuristic as defined above. For a configuration C, define $g(C) = \sum_{e \in C} cost(\varphi(e))$, and $f(C) = g(c) + h(Mark(C))$. Define $C \prec_h C'$ if and only if $f(C) < f(C')$ or $f(C) = f(C')$ and $|C| < |C'|$.*

Proof that this order is adequate will not be shown here. When running MOLE with this ordering for some monotonic heuristic h, we obtain a planner which generates partially ordered plans with the smallest total action cost. As far as we are aware, only the HSP* family of existing planners routinely optimise this metric (Haslum, Bonet, & Geffner 2005). In contrast, most state of the art planners optimise parallel plan length. It is possible our approach could be modified for concurrent temporal planning, but the full implications of this have not yet been considered.

Experimental Results

Our translation from propositional STRIPS operators to PT-nets is implemented in Standard ML within a program called Petrify. Petrify actually parses a large subset of PPDDL (Younes *et al.* 2005), and handles non-grounded domains, conditional, and probabilistic effects. We modified MOLE to implement a variety of search strategies and heuristics defined by their respective ordering relations. In Figure 3, we present results for PIPESWORLD and AIRPORT instances, for some of the variants as they provide a good illustration of the benefits and problems with our current implementation. Namely we show results for $h(s) = 0$, $h(s) = h^1_{max}$

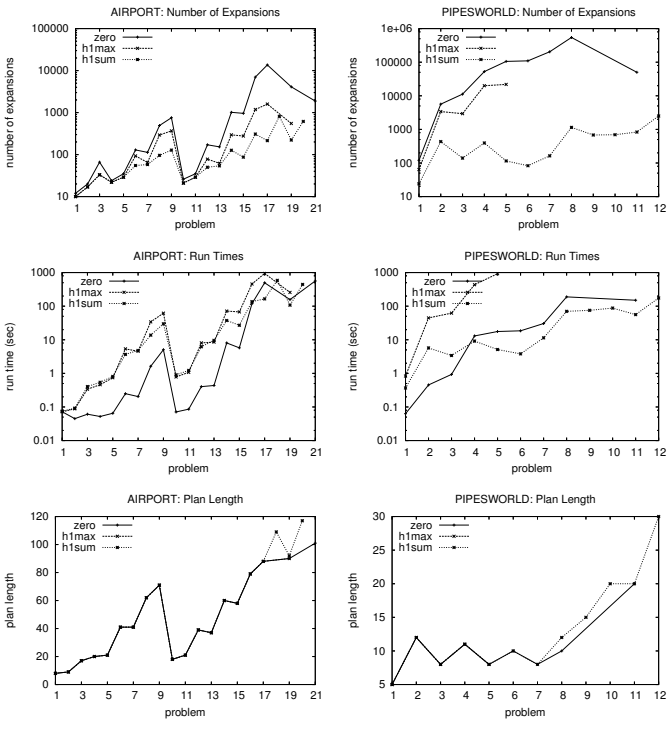


Figure 3: Experimental Result on PIPESWORLD and AIRPORT Instances

and $h(s) = h^1_{sum}$, where the later is the same as h^1_{max} but using the sum instead of the maximum. Out of these, only $\prec_0$ and $\prec_{h^1_{max}}$ are adequate orders and thus guarantee completeness.

As can be seen from the figures, the number of node expansions performed by MOLE significantly decreases (note the log scale) when switching from $h = 0$, to the monotonic h^1_{max} heuristic, and then to the non-admissible h^1_{sum}. This is particularly visible in PIPESWORLD where h^1_{max} and h^1_{sum} respectively expand over an order of magnitude and over three orders of magnitude fewer nodes than $h = 0$. However, this improvement in number of expansions does not carry over to run time. Without heuristic, our AIRPORT run times are as good as, and in a number of cases better than those obtained by the competition optimal planners, except SATPLAN. The h^1_{max} heuristic is always slower than breadth-first and h^1_{sum} only start yielding run-time gains when the improvement in number of expansions reaches two orders of magnitude. This is because, in a forward search, h^1 heuristics need to be recomputed at each expansion, and this computation has a complexity quadratic in the number of nodes of the PT-net. We expect to see a significant improvement in run time by switching to other automatically generated heuristics which can be pre-computed once and efficiently looked up during the search. Promising candidates include Pattern Database heuristics (Edelkamp 2002; Haslum, Bonet, & Geffner 2005).

Probabilistic Concurrent Planning

The translation for the probabilistic case is similar but somewhat more complicated and will not be formalised here. For probabilistic concurrent planning, we are currently focused on optimising the maximum expected reward, given a finite horizon. The Petri net is first unfolded completely, without generating any cut-off points, until the decided horizon. We propose a Bellman-like equation which uses the unfolding to answer the question: given this set of conditions are marked, what is the maximum expected reward from here and what event must be chosen to achieve this? This will not be discussed further here, due to space constraints.

Future Work

Experimental results are not presently entirely conclusive. They are only competitive with state of the art planners in some of the domains examined. Our immediate agenda is to implement pattern databases heuritics. Furthermore we will consider different translations to PT-nets and to higher level nets.

Acknowledgements

I would like to acknowledge Sylvie Thiébaux, Jussi Rintanen and Langford White for their ongoing contributions to this research.

References

Edelkamp, S.; Lluch-Lafuente, A.; and Leue, S. 2001. Directed explicit model checking with hsf-spin. In *SPIN*, 57–79.

Edelkamp, S. 2002. Symbolic pattern databases in heuristic search planning. In *AIPS*, 274–283.

Esparza, J.; Römer, S.; and Vogler, W. 2002. An improvement of mcmillan's unfolding algorithm. *Formal Methods in System Design* 20(3):285–310.

Ghallab, M.; Nau, D.; and P.Traverso. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *AIPS*, 140–149.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *AAAI*, 1163–1168.

McMillan, K. L. 1992. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *CAV*, 164–177.

Murata, T. 1989. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE* 77(4):541–580.

Younes, H. L. S.; Littman, M. L.; Weissman, D.; ; and Asmuth, J. 2005. The first probabilistic track of the international planning competition. *Journal of Artificial Intelligence Research* 24:851–887.

On the Smoothness of Linear Value Function Approximations

Branislav Kveton
Intelligent Systems Program
University of Pittsburgh
bkveton@cs.pitt.edu

Milos Hauskrecht
Department of Computer Science
University of Pittsburgh
milos@cs.pitt.edu

Abstract

Markov decision processes (MDPs) with discrete and continuous state and action components can be solved efficiently by hybrid approximate linear programming (HALP). The main idea of the approach is to approximate the optimal value function by a set of basis functions and optimize their weights by linear programming. It is known that the solution to this convex optimization problem minimizes the $\mathcal{L}_1$-norm distance in between the optimal value function and its approximation. In this paper, we relate this measure to the max-norm error of the same value function. We believe that this theoretical analysis may help to understand the quality of HALP approximations in continuous domains.

Introduction

Markov decision processes (MDPs) (Bellman 1957; Puterman 1994) provide an elegant mathematical framework for solving sequential decision problems in the presence of uncertainty. However, traditional techniques for solving MDPs are computationally infeasible in real-world domains, which are factored and represented by both discrete and continuous state and action variables. Approximate linear programming (ALP) (Schweitzer & Seidmann 1985) has recently emerged as a promising approach to address these challenges (Kveton & Hauskrecht 2006).

Our paper centers around hybrid ALP (HALP) (Guestrin, Hauskrecht, & Kveton 2004), which is an established framework for solving large factored MDPs with discrete and continuous state and action variables. The main idea of the approach is to approximate the optimal value function by a linear combination of basis functions and optimize it by linear programming (LP). The combination of factored reward and transition models with the linear value function approximation permits the scalability of the approach.

The quality of HALP solutions inherently depends on the choice of basis functions. Therefore, it is often assumed that these are provided as a part of the problem definition, which is unrealistic. The goal of this paper is to analyze the quality of HALP approximations. Based on the analysis, we provide a simple advice for selecting basis functions.

Hybrid factored MDPs

Discrete-state factored MDPs (Boutilier, Dearden, & Goldszmidt 1995) permit a compact representation of stochastic decision problems by exploiting their structure. In this work, we consider hybrid factored MDPs with exponential-family transition models (Kveton & Hauskrecht 2006). This model extends discrete-state factored MDPs to the domains of discrete and continuous state and action variables.

A *hybrid factored MDP with an exponential-family transition model (HMDP)* (Kveton & Hauskrecht 2006) is given by a 4-tuple $\mathcal{M} = (\mathbf{X}, \mathbf{A}, P, R)$, where $\mathbf{X} = \{X_1, \ldots, X_n\}$ is a state space characterized by a set of discrete and continuous variables, $\mathbf{A} = \{A_1, \ldots, A_m\}$ is an action space represented by action variables, $P(\mathbf{X}' \mid \mathbf{X}, \mathbf{A})$ is an exponential-family transition model of state dynamics conditioned on the preceding state and action choice, and R is a reward model assigning immediate payoffs to state-action configurations.[1] In the remainder of the paper, we assume that the quality of a policy is measured by the *infinite horizon discounted reward* $\mathrm{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$, where $\gamma \in [0, 1)$ is a *discount factor* and r_t is the reward obtained at the time step t.

Hybrid ALP

Value iteration, policy iteration, and linear programming are the most fundamental dynamic programming (DP) methods for solving MDPs (Puterman 1994; Bertsekas & Tsitsiklis 1996). Unfortunately, none of these methods is suitable for solving hybrid factored MDPs. First, their complexity grows exponentially in the number of state variables if the variables are discrete. Second, these methods assume a finite support for the optimal value function or policy, which may not exist if continuous variables are present. As a result, any feasible approach to solving arbitrary HMDPs is likely to be approximate. To compute these approximate solutions, Munos and Moore (2002) proposed an adaptive non-uniform discretization of continuous-state spaces and Feng *et al.* (2004) used DP backups of piecewise constant and piecewise linear value functions.

Linear value function model: Since a factored representation of an MDP may not guarantee a structure in the optimal value function or policy (Koller & Parr 1999), we resort to *linear value function approximation* (Bellman, Kalaba, &

[1] *General state and action space MDP* is an alternative name for a hybrid MDP. The term *hybrid* does not refer to the dynamics of the model, which is discrete-time.

Kotkin 1963; Van Roy 1998):

$$V^{\mathbf{w}}(\mathbf{x}) = \sum_i w_i f_i(\mathbf{x}). \quad (1)$$

This approximation restricts the form of the value function $V^{\mathbf{w}}$ to the linear combination of $|\mathbf{w}|$ basis functions $f_i(\mathbf{x})$, where $\mathbf{w}$ is a vector of tunable weights. Every basis function can be defined over the complete state space $\mathbf{X}$, but often is restricted to a subset of state variables $\mathbf{X}_i$ (Bellman, Kalaba, & Kotkin 1963; Koller & Parr 1999).

Similarly to discrete-state ALP (Schweitzer & Seidmann 1985), *hybrid ALP (HALP)* (Guestrin, Hauskrecht, & Kveton 2004) optimizes the linear value function approximation (Equation 1). Therefore, it transforms an initially intractable problem of estimating V^* in the hybrid state space $\mathbf{X}$ into a lower dimensional space $\mathbf{w}$. The HALP formulation is given by a linear program:

$$\begin{aligned} &\text{minimize}_{\mathbf{w}} \sum_i w_i \alpha_i \quad (2)\\ &\text{subject to: } \sum_i w_i F_i(\mathbf{x},\mathbf{a}) - R(\mathbf{x},\mathbf{a}) \geq 0 \quad \forall\, \mathbf{x},\mathbf{a}; \end{aligned}$$

where $\mathbf{w}$ represents the variables in the LP, α_i denotes *basis function relevance weight*:

$$\begin{aligned} \alpha_i &= \mathrm{E}_{\psi(\mathbf{x})}[f_i(\mathbf{x})] \quad (3)\\ &= \sum_{\mathbf{x}_D} \int_{\mathbf{x}_C} \psi(\mathbf{x}) f_i(\mathbf{x})\, \mathrm{d}\mathbf{x}_C, \end{aligned}$$

$\psi(\mathbf{x})$ is a *state relevance density function* weighting the approximation, and $F_i(\mathbf{x},\mathbf{a}) = f_i(\mathbf{x}) - \gamma g_i(\mathbf{x},\mathbf{a})$ is the difference between the basis function $f_i(\mathbf{x})$ and its discounted *backprojection*:

$$\begin{aligned} g_i(\mathbf{x},\mathbf{a}) &= \mathrm{E}_{P(\mathbf{x}'|\mathbf{x},\mathbf{a})}[f_i(\mathbf{x}')] \quad (4)\\ &= \sum_{\mathbf{x}'_D} \int_{\mathbf{x}'_C} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) f_i(\mathbf{x}')\, \mathrm{d}\mathbf{x}'_C. \end{aligned}$$

Vectors $\mathbf{x}_D$ ($\mathbf{x}'_D$) and $\mathbf{x}_C$ ($\mathbf{x}'_C$) are the discrete and continuous components of value assignments $\mathbf{x}$ ($\mathbf{x}'$) to all state variables $\mathbf{X}$ ($\mathbf{X}'$). The HALP formulation is feasible if the set of basis functions contains a constant function $f_0(\mathbf{x}) \equiv 1$. We assume that such a basis function is always present.

In the remainder of this paper, we analyze the quality of HALP approximations. Please refer to Hauskrecht and Kveton (2004), Guestrin *et al.* (2004), Kveton and Hauskrecht (2005), and Kveton and Hauskrecht (2006) for information on how to apply and solve HALP formulations.

Existing work

De Farias and Van Roy (2003) analyzed the quality of ALP. Based on their work, we may conclude that optimization of the objective function $\mathrm{E}_\psi[V^{\mathbf{w}}]$ in HALP is identical to minimizing the $\mathcal{L}_1$-norm error $\|V^* - V^{\mathbf{w}}\|_{1,\psi}$. This equivalence can be proved from the following proposition.

Proposition 1 *Let $\widetilde{\mathbf{w}}$ be a solution to the HALP formulation (2). Then $V^{\widetilde{\mathbf{w}}} \geq V^*$.*

Proof: The Bellman operator $\mathcal{T}^*$ is a contraction mapping. Based on its monotonicity, $V \geq \mathcal{T}^* V$ implies $V \geq \mathcal{T}^* V \geq \cdots \geq V^*$ for any value function V. Since constraints in the HALP formulation (2) enforce $V^{\widetilde{\mathbf{w}}} \geq \mathcal{T}^* V^{\widetilde{\mathbf{w}}}$, we conclude $V^{\widetilde{\mathbf{w}}} \geq V^*$. ■

Based on Proposition 1, we know that HALP optimizes the linear value function model with respect to the weighted $\mathcal{L}_1$-norm error $\|V^* - V^{\mathbf{w}}\|_{1,\psi}$. The following theorem bounds the quality of a greedy policy for the value function $V^{\widetilde{\mathbf{w}}}$.

Theorem 1 *Let $\widetilde{\mathbf{w}}$ be an optimal solution to the HALP formulation (2). Then the expected error of a greedy policy:*

$$u(\mathbf{x}) = \arg\sup_{\mathbf{a}} \left[R(\mathbf{x},\mathbf{a}) + \gamma \mathrm{E}_{P(\mathbf{x}'|\mathbf{x},\mathbf{a})} \left[V^{\widetilde{\mathbf{w}}}(\mathbf{x}') \right] \right]$$

can be bounded as:

$$\|V^* - V^u\|_{1,\nu} \leq \frac{1}{1-\gamma} \left\| V^* - V^{\widetilde{\mathbf{w}}} \right\|_{1,\mu_{u,\nu}},$$

where $\|\cdot\|_{1,\nu}$ and $\|\cdot\|_{1,\mu_{u,\nu}}$ are weighted $\mathcal{L}_1$-norms, V^u is a value function for the greedy policy u, and $\mu_{u,\nu}$ denotes the expected frequency of state visits generated by following the policy u given the initial state distribution ν.

Based on Theorem 1, the state relevance density function ψ should resemble the expected frequency of state visits $\mu_{u,\nu}$. Unfortunately, $\mu_{u,\nu}$ is unknown unless $V^{\widetilde{\mathbf{w}}}$ is known, which is optimized with respect to the unknown distribution $\mu_{u,\nu}$. To break this cycle, de Farias and Van Roy (2003) suggested an iterative scheme that resolves several LPs and adapts $\mu_{u,\nu}$ accordingly. Alternatively, real-world control problems often exhibit a lot of structure, which permits guessing of $\mu_{u,\nu}$.

Error bounds

This section demonstrates how to bound the max-norm error $\|V^* - V^{\mathbf{w}}\|_\infty$ of a linear approximation $V^{\mathbf{w}}$ in terms of its $\mathcal{L}_1$-norm error $\|V^* - V^{\mathbf{w}}\|_{1,\psi}$. This result is a step towards understanding the quality of HALP approximations. For instance, based on the work of Williams and Baird III (1993), we can bound the loss of acting greedily with respect to the value function $V^{\mathbf{w}}$ by its max-norm error $\|V^* - V^{\mathbf{w}}\|_\infty$. In combination with our work (Theorems 2 and 3), we can derive max-norm bounds on the quality of greedy policies for HALP approximations. Note that Theorem 1 only provides bounds on the $\mathcal{L}_1$-norm errors of greedy policies.

For discrete-state factored MDPs, we can easily prove the following proposition.

Proposition 2 *Let $\widetilde{\mathbf{w}}$ be an optimal solution to the HALP formulation (2) with discrete state variables. Then the max-norm error of $V^{\widetilde{\mathbf{w}}}$ can be bounded as:*

$$\left\| V^* - V^{\widetilde{\mathbf{w}}} \right\|_{\infty,\psi} \leq \left\| V^* - V^{\widetilde{\mathbf{w}}} \right\|_{1,\psi},$$

where $\|\cdot\|_{1,\psi}$ and $\|\cdot\|_{\infty,\psi}$ are $\mathcal{L}_1$ and infinity norms weighted by the state relevance density function ψ.

Proof: The claim directly follows from the definition of the norms $\|\cdot\|_{1,\psi}$ and $\|\cdot\|_{\infty,\psi}$. ■

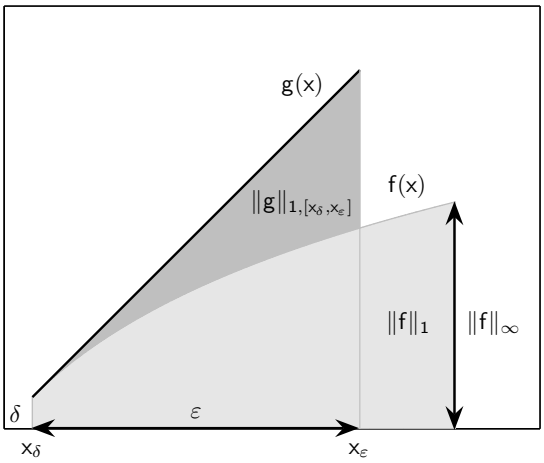


Figure 1: A graphical representation of the bound from Theorem 2 in a single dimension. Light and dark gray regions correspond to the integrals $\|f\|_1$ and $\|g\|_{1,[\mathbf{x}_\delta,\mathbf{x}_\varepsilon]}$.

Unfortunately, this bound is loose and not of much practical interest. In the remainder of this section, we prove a tighter bound for continuous-state spaces. The notion of continuity is captured through the Lipschitz condition.

Definition 1 *The function $f(\mathbf{x})$ is* Lipschitz continuous *if:*

$$|f(\mathbf{x}) - f(\mathbf{x}')| \leq K \|\mathbf{x} - \mathbf{x}'\|_\infty \quad \forall\, \mathbf{x}, \mathbf{x}'; \qquad (5)$$

where K is referred to as a Lipschitz constant*.*

In the rest of the paper, it is useful to think of the max-norm ($\mathcal{L}_1$-norm) as being the supremum (integral) of a function.

Theorem 2 *Let $\tilde{\mathbf{w}}$ be an optimal solution to the HALP formulation (2) with continuous state variables. If the function:*

$$f(\mathbf{x}) = \psi(\mathbf{x}) \left| V^*(\mathbf{x}) - V^{\tilde{\mathbf{w}}}(\mathbf{x}) \right|$$

is Lipschitz continuous, and there exists a state $\mathbf{x}_\delta$ such that $f(\mathbf{x}_\delta) \leq \delta$, the max-norm error of $V^{\tilde{\mathbf{w}}}$ can be bounded as:

$$\left\| V^* - V^{\tilde{\mathbf{w}}} \right\|_{\infty,\psi} \leq \delta + K \min \left\{ \sqrt[n]{\frac{\|V^* - V^{\tilde{\mathbf{w}}}\|_{1,\psi}}{\delta C}}, \sqrt[n+1]{\frac{2\,\|V^* - V^{\tilde{\mathbf{w}}}\|_{1,\psi}}{KC}} \right\},$$

where $\|\cdot\|_{1,\psi}$ and $\|\cdot\|_{\infty,\psi}$ are $\mathcal{L}_1$ and infinity norms weighted by the state relevance density function ψ, n is the number of state variables, K represents the Lipschitz constant of $f(\mathbf{x})$, and C is a problem-specific constant.

Proof: To prove the theorem, we define a function:

$$g(\mathbf{x}) = \delta + K \|\mathbf{x} - \mathbf{x}_\delta\|_1 . \qquad (6)$$

It follows that the function $g(\mathbf{x})$ is an upper bound on $f(\mathbf{x})$ because $f(\mathbf{x}_\delta) \leq \delta$, $\|\mathbf{x} - \mathbf{x}_\delta\|_1 \geq \|\mathbf{x} - \mathbf{x}_\delta\|_\infty$, and K is the Lipschitz constant of $f(\mathbf{x})$. Furthermore, $g(\mathbf{x})$ is increasing faster than $f(\mathbf{x})$ in every dimension. As a result, there exists a point $\mathbf{x}_\varepsilon$ such that $g(\mathbf{x}_\varepsilon) \geq \|f\|_\infty$, and the integral of $g(\mathbf{x})$ between $\mathbf{x}_\delta$ and $\mathbf{x}_\varepsilon$ is smaller or equal to $\|f\|_1$. A graphical interpretation of this situation in a single dimension is shown in Figure 1.

In general, the integral $\|g\|_{1,[\mathbf{x}_\delta,\mathbf{x}_\varepsilon]}$ can be computed as:

$$\|g\|_{1,[\mathbf{x}_\delta,\mathbf{x}_\varepsilon]} = \left[\delta + \frac{K}{2}\varepsilon\right] \prod_{i=1}^{n} \varepsilon_i,$$

where $\varepsilon_i = |x_{\varepsilon i} - x_{\delta i}|$, $\varepsilon = \|\mathbf{x}_\varepsilon - \mathbf{x}_\delta\|_1$, and n denotes the number of state variables $\mathbf{X}$. Since $\varepsilon = \sum_{i=1}^{n} \varepsilon_i$, we rewrite the equation as:

$$\|g\|_{1,[\mathbf{x}_\delta,\mathbf{x}_\varepsilon]} = C \left[\delta + \frac{K}{2}\varepsilon\right] \varepsilon^n,$$

where $C = \prod_{i=1}^{n} (\varepsilon_i/\varepsilon)$ is a problem-specific constant that guarantees $g(\mathbf{x}_\varepsilon) \geq \|f\|_\infty$. The constant C is bounded from above by n^{-n}. Finally, we recognize that C, δ, K, and ε are always nonnegative, which leads to the conclusion:

$$\varepsilon \leq \min \left\{ \sqrt[n]{\frac{\|f\|_1}{\delta C}}, \sqrt[n+1]{\frac{2\,\|f\|_1}{KC}} \right\} \qquad (7)$$

assuming $\|g\|_{1,[\mathbf{x}_\delta,\mathbf{x}_\varepsilon]} \leq \|f\|_1$. Direct combination of Equations 6 and 7 yields our final result. ■

To make the bound in Theorem 2 practical, we have to assure a low Lipschitz factor K and the existence of a state $\mathbf{x}_\delta$ such that $f(\mathbf{x}_\delta) \leq \delta$. We cannot guarantee the existence of such a state yet. However, we can affect the factor K by the choice of basis functions and state relevance densities. In particular, to achieve a low value K, we should use basis functions that yield close approximations to V^*. In practice, this condition cannot be guaranteed unless we know V^*. Furthermore, the Lipschitz factor of V^* may be large itself. To address these concerns, we generalize Theorem 2 to an arbitrary partitioning of the state space $\mathbf{X}$.

Theorem 3 *Let $\tilde{\mathbf{w}}$ be an optimal solution to the HALP formulation (2) with continuous state variables. If:*

$$\Omega = \{\omega_1, \ldots, \omega_{|\Omega|}\}$$

is a mutually-exclusive partitioning of the state space $\mathbf{X}$, the function:

$$f(\mathbf{x}) = \psi(\mathbf{x}) \left| V^*(\mathbf{x}) - V^{\tilde{\mathbf{w}}}(\mathbf{x}) \right|$$

is Lipschitz continuous on each partition, and there exists a state $\mathbf{x}_{\delta_\omega}$ for every ω such that $f(\mathbf{x}_{\delta_\omega}) \leq \delta_\omega$, the max-norm error of $V^{\tilde{\mathbf{w}}}$ can be bounded as:

$$\left\| V^* - V^{\tilde{\mathbf{w}}} \right\|_{\infty,\psi} \leq \max_{\omega \in \Omega} \left\{ \delta_\omega + K_\omega \min \left\{ \sqrt[n]{\frac{\|V^* - V^{\tilde{\mathbf{w}}}\|_{1,\psi_\omega}}{\delta_\omega C_\omega}}, \sqrt[n+1]{\frac{2\,\|V^* - V^{\tilde{\mathbf{w}}}\|_{1,\psi_\omega}}{K_\omega C_\omega}} \right\} \right\},$$

where the explanation of symbols is identical to Theorem 2. All subscripted symbols are partition-specific.

Proof: Based on the definition of the max-norm $\|\cdot\|_{\infty,\psi}$, we conclude:

$$\left\|V^* - V^{\tilde{\mathbf{w}}}\right\|_{\infty,\psi} = \max_{\omega\in\Omega} \left\|V^* - V^{\tilde{\mathbf{w}}}\right\|_{\infty,\psi_\omega},$$

where $\psi_\omega(\mathbf{x}) = \psi(\mathbf{x})\mathbf{1}_{\mathbf{x}\in\omega}(\mathbf{x})$, and $\mathbf{1}_{\mathbf{x}\in\omega}(\mathbf{x})$ is the indicator function of the partition ω. The final result is a consequence of bounding each $\left\|V^* - V^{\tilde{\mathbf{w}}}\right\|_{\infty,\psi_\omega}$ by Theorem 2. ■

Theorem 2 provides an insight into the relation between the $\mathcal{L}_1$-norm objective $\left\|V^* - V^{\tilde{\mathbf{w}}}\right\|_{1,\psi}$ and the max-norm error $\left\|V^* - V^{\tilde{\mathbf{w}}}\right\|_{\infty,\psi}$. The max-norm error can be minimized by lowering $\mathcal{L}_1$-norm errors $\left\|V^* - V^{\tilde{\mathbf{w}}}\right\|_{1,\psi_\omega}$ if the growth rate of K_ω and δ_ω is controlled. This result leads to an intuitive advice for choosing basis functions. If the shape of the value function V^* is not known, we should prefer smooth approximations. These are not likely to inflate Lipschitz constants K_ω where V^* is smooth.

Conclusions

Development of efficient methods for solving large factored MDPs is a challenging problem. In this paper, we analyzed the quality of linear approximations and bounded their max-norm error by the objective value in HALP. We believe that this analysis can help us to understand the quality of HALP approximations in continuous domains.

Acknowledgment

During the academic years 2004-06, the first author was supported by two Andrew Mellon Predoctoral Fellowships. The first author recognizes support from Intel Corporation in the summer 2005. This research was also partially supported by two National Science Foundation grants CMS-0416754 and ANI-0325353. We thank anonymous reviewers for providing comments that led to the improvement of the paper.

References

Bellman, R.; Kalaba, R.; and Kotkin, B. 1963. Polynomial approximation – a new computational technique in dynamic programming: Allocation processes. *Mathematics of Computation* 17(82):155–161.

Bellman, R. 1957. *Dynamic Programming*. Princeton, NJ: Princeton University Press.

Bertsekas, D., and Tsitsiklis, J. 1996. *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 1995. Exploiting structure in policy construction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1104–1111.

de Farias, D. P., and Van Roy, B. 2003. The linear programming approach to approximate dynamic programming. *Operations Research* 51(6):850–856.

Feng, Z.; Dearden, R.; Meuleau, N.; and Washington, R. 2004. Dynamic programming for structured continuous Markov decision problems. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, 154–161.

Guestrin, C.; Hauskrecht, M.; and Kveton, B. 2004. Solving factored MDPs with continuous and discrete variables. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, 235–242.

Hauskrecht, M., and Kveton, B. 2004. Linear program approximations for factored continuous-state Markov decision processes. In *Advances in Neural Information Processing Systems 16*, 895–902.

Koller, D., and Parr, R. 1999. Computing factored value functions for policies in structured MDPs. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 1332–1339.

Kveton, B., and Hauskrecht, M. 2005. An MCMC approach to solving hybrid factored MDPs. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, 1346–1351.

Kveton, B., and Hauskrecht, M. 2006. Solving factored MDPs with exponential-family transition models. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling*.

Munos, R., and Moore, A. 2002. Variable resolution discretization in optimal control. *Machine Learning* 49:291–323.

Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY: John Wiley & Sons.

Schweitzer, P., and Seidmann, A. 1985. Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications* 110:568–582.

Van Roy, B. 1998. *Planning Under Uncertainty in Complex Structured Environments*. Ph.D. Dissertation, Massachusetts Institute of Technology.

Williams, R., and Baird III, L. 1993. Tight performance bounds on greedy policies based on imperfect value functions. Technical Report NU-CCS-93-14, Northeastern University.

Memory-Efficient Graph Search in Planning and Model Checking

Peter Lamborn and **Eric A. Hansen**
Dept. of Computer Science and Engineering
Mississippi State University
Mississippi State, MS 39762
pcl16@msstate.edu

Abstract

Graph search plays a central role in both planning and model checking. We describe how to adapt an approach to memory-efficient graph search that has been used for domain-independent STRIPS planning in order to create a more memory-efficient domain-independent model checker. We discuss some changes to this approach that are required by the differences between planning and model checking, and report preliminary results that indicate its effectiveness.

Introduction

Model checking is a fully-automated formal technique for verifying that a property is satisfied, or modeled, by a transition system (Clarke, Grumberg, & Peled 2000). The transition system is often a protocol, such as a protocol for wireless communication or an embedded hardware device. A violation of the property is an error. For example, in a wireless protocol, an error could be starvation of a device. Model checking can find errors that are difficult to detect by testing and simulation, and is especially effective in finding errors with long traces or errors that occur after unusual sequences of events. Model checking can also verify that no errors exist, which is impossible using testing and simulation.

There are several important connections between research in model checking and research in automated planning and heuristic search. First of all, there is a large body of work on planning via model checking (Giunchiglia & Traverso 1999). In this work, planning domains are formalized as transition systems, goals are expressed as temporal formulas, and planning is done by using model checking to determine whether the temporal logic formula is true in the transition system. Another important connection between automated planning and explicit-state model checking is that both rely on state-space exploration using graph search. For both, scalability is limited by the state explosion problem – the size of the state space grows exponentially in the number of variables in its description. Finally, in both planning and model checking, there is an emphasis on domain-independent tools; there is a focus on developing domain-independent model checkers just as there is interest in developing domain-independent planners.

This short paper describes preliminary work on using an approach to memory-efficient graph-search originally developed for domain-independent planning in order to improve the scalability of a domain-independent model checker. The approach we adapt is called breadth-first heuristic search with layered duplicate detection (Zhou & Hansen 2006). Layered duplicate detection turns out to be a very effective approach to reducing the memory requirements of model checking because it can be easily implemented in a domain-independent way and works well in searching directed graphs; model checking always involves directed graphs. In our adaptation, we use breadth-first search without a heuristic to perform a complete search of a graph, in order to verify that a property always holds. We report preliminary empirical results that illustrate that this approach can significantly improve the range of models that can be verified.

Background

Graph search and model checking

In model checking, a transition system (e.g., for a protocol) is represented by a directed graph in which the nodes correspond to states of the system and the edges correspond to state transitions. For example, a state may indicate two packets in flight, a valid transition could be the reception of a packet, and the new state would contain just one packet in flight. The graph has an initial state, and the set of all possible paths in the graph represents the set of all possible behaviors of the protocol. Given this representation, a property can be verified by a complete search of the graph to make sure that no error states can be reached. If an error is found, a trace of the error (i.e., a path from the initial state to the error state) is returned and used to debug the protocol.

The scalability of model checking is limited by the size of the graph that must be searched. In explicit-state model checking, every generated state is stored in a hash table that is used for duplicate detection, which is the process of determining whether or not a newly-generated state is a duplicate of a previously-generated state. Since a complete search of the graph is needed to verify a system, the memory needed to store all generated states is the bottleneck of model checking. As a result, some approaches to model checking abandon verification and merely attempt to detect errors in a swift manner. Heuristic search algorithms such as A* have been used to efficiently find paths to error states, which are treated as goal states (Edelkamp, Lluch-Lafuente, & Leue 2001). Other approaches, such as randomized search, have

also been used (Jones *et al.* 2003).

Frontier search and layered duplicate detection

Frontier search is a memory-efficient approach to graph search that only stores the Open list, and saves memory by not storing the Closed list (Korf *et al.* 2005). Instead of the traceback method of solution recovery, it uses a divide-and-conquer method that involves finding an intermediate node along an optimal path and using it to divide the search problem into two subproblems – the problem of finding an optimal path from the start node to the intermediate node, and the problem of finding an optimal path from the intermediate node to the goal node. The subproblems are solved recursively by the same search algorithm until all nodes along an optimal solution path for the original problem are identified.

Since frontier search only stores nodes on the frontier, it needs to prevent already-explored nodes that are no longer in memory from being regenerated. In undirected graphs, Korf et al. (2005) use a technique called *used-operator bits* that is very effective. But directed graphs present more of a challenge. In the approach proposed by Korf et al., each time a node is expanded and its successors generated, all predecessors of these successors are also generated, even if no path has yet been found to these nodes. These *dummy nodes* are stored in the Open list with an g-cost of infinity until a path to them is found, at which point they acquire the g-cost of that path. Although this guarantees that no node is generated more than once, the dummy nodes would not be generated by a standard graph-search algorithm and there is no bound on the number of dummy nodes that can be generated. In some cases, the additional overhead for generating dummy nodes can make search performance worse.

Zhou and Hansen (2006) propose an alternative approach to duplicate detection that can be used in breadth-first search. In this approach, called layered duplicate detection, the Closed list is stored in layers, one for each g-cost, and earlier layers are deleted to recover memory. In undirected graphs, they point out that keeping just one previous layer in memory is sufficient to detect all duplicates. In directed graphs, they propose keeping one or more previous layers, where the number needed to prevent all duplicates depends on the structure of the graph. They also point out that even if no previous layers are kept in memory, the number of times a node can be regenerated is bounded by the depth of the search. In depth-first search, by contrast, the number of node regenerations can be exponential in the depth of the search.

Layered duplicate detection is easy to implement in a domain-independent way, in contrast to the approach to duplicate detection used in Korf et al.'s implementation of frontier search. Therefore, we use frontier breadth-first search with delayed duplicate detection to reduce the memory requirements of our domain-independent model checker.

Algorithm and preliminary results

We briefly describe our approach to memory-efficient model checking and present some preliminary experimental results.

Algorithm

The starting point for our implementation is the Murϕ model checker, a domain-independent tool that takes a description of a model as input and uses breadth-first search to verify that the model is correct (Dill 1996). If an error is found, it returns an error trace.

Since the graphs that are searched in model checking are directed graphs, we modify the breadth-first search algorithm to use layered duplicate detection. First of all, this requires indexing the hash table of stored states by layers, so that individual layers can be deleted as the search progresses. The breadth-first search proceeds as usual until it begins to run out of memory. At that point, it recovers memory by deleting layers of generated states from memory. The layers it deletes are those that are furthest from the frontier (i.e., the shallowest layers), since they are less likely to be useful in duplicate detection. Any closed nodes are eligible for deletion, if more memory is needed. Thus, as long as the Open list can fit in memory, the search continues. (If the Open list does not fit in memory, we could use beam search to continue to search for an error, but we do not consider this possibility in this paper.)

If an error is found, the error trace is recovered by using the traceback method to follow pointers backwards from the error state through as many layers as still reside in memory. If some layers are missing, the shallowest state in the error trace is treated as a goal state, and another search is conducted to find a path from the initial state to this state, in order to finish recovering the error trace. This is a modification of the divide-and-conquer technique for solution recovery.

Theoretical properties

Because the search is breadth-first, if an error is found, the error trace is guaranteed to be a shortest path to the error state. If no error is found, the search terminates when the Open list is empty and there are no more states of the graph to explore. If the search terminates in this way without finding an error, the model is verified.

However, when layers of the search graph are deleted to save memory, the search is no longer guaranteed to terminate with an empty Open list, even if the graph is finite. If some layers of the search graph are removed from memory, it is possible for duplicate nodes to be generated during the search. Zhou and Hansen (2006) give some conditions under which no duplicates will be generated, but these do not hold in general. In the worst case, the number of times a node can be regenerated is bounded by the depth of the search (d) and the number of of layers stored (l), and no state can be duplicated more than d/l times. In practice, duplicates are generated even less frequently than this. But the possibility of regenerating previously explored nodes means that there is no guarantee the search will terminate with an empty Open list, since the search can repeatedly regenerate and re-explore the same parts of the search graph.

If the model contains an error, however, breadth-first search is guaranteed to terminate by finding a shortest path to this error, no matter how many duplicates may be generated. This points to a second way to detect termination

Model	Peak Nodes in Memory	Total Nodes Generated	Number of Layers
cache32622	14,141	111,335	8
newlist8	3,930,856	24,714,307	44
arbiter13	1,521,655	11,545,717	17
arbiter14	1,429,446	11,255,853	14
adash1313e	684,946	13,480,127	15
ns22110	87,366	173,866	8
dynpart4	2,619,984	16,202,474	30

Table 1: Memory savings for seven different models.

and verify a model. If the diameter of the search graph is known, or can be bounded, and no error is found by the search algorithm in searching up to this depth, then no errors can be present and the model is verified. (If the search algorithm generates any nodes past this depth, they must be duplicates.) This method of detecting termination is similar to that used in bounded model checking, which uses satisfiability testing to verify that a model does not have any errors up to depth k (Clarke *et al.* 2001). If the model is verified for a depth that is equal to or greater than the diameter of the graph, called the *completeness threshold*, verification is complete. Various methods for determining a completeness threshold have been explored in the literature on bounded model checking, and can be applied in our approach to model checking.

Preliminary empirical results

Table 1 shows the performance of our modified algorithm in verifying seven different models. They include a cache coherency protocol with six caches and two disks (cache32622), a protocol for maintaining a linked list in a parallel environment with eight parallel nodes (newlist8), a mutual exclusion protocol with thirteen or fourteen threads and an arbiter in charge of who gets access (arbiter13 and arbiter14), the DASH communications protocol for a network of three nodes (adash1313e), the Needham-Schroeder protocol for secure communication for four participants with two intruders (ns22110), and an algorithm for dynamically partitioning a search with four processing nodes (dynpart4).

In model checking, the node data structure is typically very large because the state description includes many variables that can have a large range of possible values. For the cache coherency model, for example, the size of a single node is 167 bytes. Therefore, even though the models in Table 1 are relatively small, none can be verified using 1GB of memory without using our layered duplicate detection technique. The difference between the peak number of nodes stored and the number of nodes generated during the search gives a sense of how this approach improves scalability by allowing larger models to be verified using the same amount of memory. Note that because we use blind breadth-first search, the Open list it typically much smaller than the Closed list, and deleting the Closed list from memory saves a substantial amount of memory and significantly improves scalability.

Future Work

Besides testing this approach on larger and more varied models, we hope to eventually extend it in several ways. When the Open list does not fit in memory, we plan to use disk to continue the search. While disk provides much more storage than internal memory, it is also limited, and our approach will allow complete search of graphs that do not fit on disk. For example, Korf and Schultze (2005) describe a complete breadth-first search of the Fifteen Puzzle that requires 1.4 terabytes of disk just to store the search frontier. Their frontier search algorithm is for undirected graphs and specialized for the Fifteen Puzzle, whereas our approach will allow domain-independent search in directed graphs.

We will also consider parallel search algorithms. Our approach to both external-memory and parallel graph search will be based on *structured duplicate detection*, which has been shown to be very effective in domain-independent planning (Zhou & Hansen 2004). Thus, we will continue to leverage the close connections between graph search in domain-independent planning and model checking.

References

Clarke, E. M.; Biere, A.; Raimi, R.; and Zhu, Y. 2001. Bounded model checking using satisfiability solving. *Formal Methods in System Design* 19(1):7–34.

Clarke, E. M.; Grumberg, O.; and Peled, D. 2000. *Model Checking*. MIT Press.

Dill, D. 1996. The Murϕ verification system. In R. Alur, and T. Henzinger., eds., *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, volume 1102, 390–393. Springer Verlag.

Edelkamp, S.; Lluch-Lafuente, A.; and Leue, S. 2001. Directed explicit model checking with HSF-SPIN. In Dwyer, M. B., ed., *Proc. of the 8th Int. SPIN Workshop*, volume 2057 of *Lecture Notes in Computer Science*, 57–79.

Giunchiglia, F., and Traverso, P. 1999. Planning as model checking. In *Proceeding of the Fifth European Conference on Planning (ECP 1999)*, volume 1809 of *Lecture Notes in Computer Science*, 1–20. Springer.

Jones, M.; Mercer, E.; Bao, T.; Kumar, R.; and Lamborn, P. 2003. Benchmarking explicit state parallel model checkers. In *Proceedings of the 2nd International Workshop on Parallel and Distributed Methods in Verification*, 84–98.

Korf, R., and Schultze, P. 2005. Large-scale parallel breadth-first search. In *Proc. of the 20th National Conference on Artificial Intelligence (AAAI-05)*, 1380–1385.

Korf, R.; Zhang, W.; Thayer, I.; and Hohwald, H. 2005. Frontier search. *Journal of the ACM* 52(5):715–748.

Zhou, R., and Hansen, E. 2004. Structured duplicate detection in external-memory graph search. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-04)*, 683–688.

Zhou, R., and Hansen, E. 2006. Breadth-first heuristic search. *Artificial Intelligence* 170:385–408.

A Study of Process Languages for Planning Tasks

Stephen Lee-Urban, Héctor Muñoz-Avila

Department of Computer Science & Engineering, Lehigh University, Bethlehem, PA 18015-3084 USA. {sml3, hem4}@lehigh.edu

Abstract

We present the semantics of processes for planning tasks. We target Task-Method-Knowledge (TMK) process models, a widely used formalization of processes. These semantics are formalized by proving that Task-Method-Knowledge (TMK) process models can be translated into equivalent Hierarchical Task Networks (HTNs). We construct a Turing-computable translation function that maps TMK process model constructs to equivalent HTN constructs. We leverage well-known results from HTN planning to define soundness, completeness and complexity results for TMKs.

Introduction

Processes are an important family of knowledge constructs. Loosely speaking, a *process* is the means by which tasks are accomplished via a series of actions or operations. Process modeling is important when reasoning by concept reuse and modification; without such a model, reasoning takes place at the level of actions, such as STRIPS operators, and knowledge transfer to new problems is difficult. Process models ease knowledge transfer by capturing how to achieve a goal. This yields a flexible representation for problem solving.

Processes have been used in a wide range of applications. For example, they have been used to: represent web services integration (Curbera *et al*., 2002), build adaptive agents (Ulam *et al*,. 2005), build introspective agents that use models to identify failures in their reasoning (Murdock, 2001), and build a testbed for machine learning systems (Molineaux & Aha, 2005). For our analysis, we use the process language called *Task-Method-Knowledge process models* (Murdock, 2001), which was also used in the latter three applications mentioned above.

One drawback of Task-Method-Knowledge process models (TMKs) is that they lack clear semantics, in particular when viewed as a language for expressing planning problems as used in REM (Murdock, 2001) and TIELT (Molineaux & Aha, 2005). In this paper we define formal semantics for TMKs by demonstrating their equivalence with Hierarchical Task Networks (HTNs). This will allow us to state soundness and completeness results for planning with TMKs, show that planning with TMKs is NP-complete, and prove that they are strictly more expressive than STRIPS plans. Furthermore, we prove that a Turing-computable translation function exists from TMKs to HTNs that runs in polynomial time.

The significance of these results is threefold. First, the results imply that HTN reasoning capabilities that have clear semantics can be applied to the growing body of TMKs knowledge. Second, several advances involving HTNs, including results on their applicability (Wilkins, 1988) and learning them (Choi & Langley, 2005; Ilghami *et al*., 2005), can be applied to a wide range of real-world applications for which TMKs knowledge is readily available. Third, TMKs adopt common characteristics of process languages, including grouping of simple activities into more complex activities, iteration (e.g., *while* loops), variable assignment, and value return. Thus, our results should also apply to other process languages.

Hierarchical-Task-Network Planning

HTN planning is a form of planning that reasons at the level of high-level tasks instead of lower-level actions (Erol *et al*., 1994). In HTN planning, high-level tasks are repeatedly decomposed into simpler ones until all tasks have been reduced to primitive actions.

The main knowledge artifacts in HTN planning are called *methods*. A method encodes how to achieve a compound task. Methods consists of three elements: (1) the task being achieved, called the *head* of the method, (2) the set of *preconditions* indicating the conditions that must be fulfilled for the method to be applicable, and (3) the *subtasks* needed to achieve the head. The second set of knowledge artifacts are the *operators*. Operators in HTN planning have the same purpose as in STRIPS planning: they represent action schemes. Like STRIPS operators, an HTN operator consists of the primitive task it achieves and its effects, indicating how the world changes when it is applied. However, HTN operators have no preconditions because applicability conditions are determined in the methods.

Task-Method-Knowledge Planning

TMK planning is another form of planning that, like HTN planning, reasons at the level of high-level tasks instead of low-level actions. TMKs capture tasks (what an agent does), methods (how the agent works), and knowledge (the information used by the agent). Tasks are accomplished by methods, which are in turn further decomposed into lower level tasks as specified by the methods. A hierarchy is consequentially created where the leaves of the resulting TMK model tree are primitive tasks (not decomposable) that explicitly specify their effects, and the non-primitive

tasks are the internal nodes of the tree (Murdock, 2001). The set of plans representable TMK models is a superset of the set of plans representable as HTNs.

As with HTNs, *methods* are behavioral elements that encode how a piece of computation works; the overall function of a method is encoded in the task it addresses. Methods specify the means of accomplishing a task, and the applicability of a selected method to a particular task is determined by the method's preconditions and the current situation. A TMK method has three elements: *preconditions* that must be true in order to perform the method, *postconditions* that represent the consequences of performing it, and a reference to the first transition in a state-transition machine that implements the method's behavior. A method can be viewed as a state machine, which is convenient for describing the translation of statements (e.g., while, for, if). TMK operators have the same form as in STRIPS planning.

Translating TMKs into HTNs: A Sketch

Our translator selects each input TMK method and calls a recursive procedure that traverses its state-transition machine implementation. This state machine can be viewed as a sequence of statements s_1, .., s_n, where each statement s_i can be a complex statement consisting of its own statements. Our current implementation translates TMKs from TIELT (Molineaux, & Aha, 2005) into HTNs for the SHOP HTN planner (Nau *et al*., 1999).

The procedure HTNs() translates a list of TMK methods M_T into an equivalent collection M_T of HTN methods (Fig. 1). For every TMK method m_T we call the procedure toHTNs(), with the body of m_T and a dummy task that can always be accomplished (line 1-2). This procedure returns the collection M of HTN methods translating the body and a task t_{body} that is accomplished by methods in M (line 3). We construct a new method m, that decomposes t into t_{body} (line 4). All these are collected and returned (lines 5-6).

The procedure toHTNs() is also shown in Figure 2. It receives as input a sequence of TMKs statements E that must be executed before a task t_{next} (initially called with a dummy task). This procedure translates a finite state machine's statements in the reverse order (s_n, ..., s_1) in which they appear in the machine and each statement s_k is translated from the most to the least deeply nested. If E has only one statement s1 that has no body, we create a new task *nt* accomplished by a new method *m* that decomposes into the translation of that statement (lines 5-7). The task t_{next} is appended as the last subtask of the method (line 8). The pair (m,nt) is then returned (line 9). If E has only one statement s1 that has a body (i.e., one or more nested statements), then we return the result of a recursive call (lines 3 and 4). If E has more than one statement, these are translated in reverse order (lines 10-14). A recursive call is in line 12. We update the task t_{next} for the next iteration to be the task t returned by the previous recursive call (line 13). During each iteration, the resulting HTN methods are collected in M_H (line 14). Finally the pair (M_H,t) is returned (line 15).

Table 1 informally summarizes how the individual statements are translated in the algorithm toHTNs().

More compact translations are possible if HTN planners support expressions defined in the planning language PDDL (Long & Fox, 2002). PDDL is a STRIPS-based planning language, a subset of which is used in the annual AI planning competition. However, the mappings described here are the most direct and suffice to illustrate the equivalence between TMKs and HTNs.

Procedure HTNs(M_T)
Input: M_T is a list of TMKs methods
Output: A collection M_H of HTN methods translating M_T
1 $M_H \leftarrow \{\}$
2 **for** every TMKs method m_T =(t,body) in M_T **do**
3 (M,t_{body}) ← toHTNs(body,dummyTask())
4 m ← newMethod(t, <t_{body}>)
5 $M_H \leftarrow M_H \cup M \cup \{ m \}$
6 **return** M_H

Procedure toHTNs(E,t_{next})
Input: E is a sequence of TMKs statements that must be executed before t_{next}
Output: A collection M_H of HTN methods translating E and the task t that achieve the methods in M_H
1 **Let** E = (s_1, .., s_n)
2 **if** (n = 1) **then**
3 **if** (s_1 is complex) **then**
4 **return** toHTNs (s_1,t_{next})
5 **else**
6 nt ← newTask()
7 m ← newMethod(nt, translateAtomicStatement(s_1))
8 appendMethod(m, t_{next})
9 **return** ({m}, nt)
10 **else**
11 **for** k ← n to 1 do
12 (M,t) ← toHTNs(s_n, t_{next})
13 t_{next} ← t
14 $M_H \leftarrow M_H \cup M$
15 **return** (M_H,t)

Figure 1: Pseudo-code of the TMKs Translation Algorithm

TMKs	HTNs
Return (values from functions)	Use an unbound variable as a parameter in the caller's invocation; set same variable in called method's preconditions
If-then-else	Use HTN method syntax
While (iteration)	Use recursion
For (iteration)	Change to while, use recursion
Tasks with preconditions, effects	Add preconditions to methods and the effects as new tasks
Call	Subtask
Set (variable assignment)	Split into a new method and pass the variables by value

Table 1: Mapping from TMKs to HTNs

Semantics and Complexity of TMKs Planning

According to Erol *et al.* (1994) no well-established definition of planning language expressivity exists; expressivity has instead been defined using model-theoretic semantics, operational semantics, and the computational complexity of problems representable by the planning language.

One way to demonstrate the expressive equivalence of HTNs and TMKs is a presentation like that described in (Erol *et al.*, 1995), which states that the expressivity of two languages can be compared by demonstrating that a polynomial or Turing computable transformation exists between them. We presented a sketch of this transformation from TMKs to HTNs in the previous section. This sketch allows us to assert the following:

> **Theorem:** There exists a Turing-computable function ψ from the set of TMK planning problems to the set of HTN planning problems such that, for any TMK planning problem **P** and any plan σ, σ solves **P** iff σ solves the HTN planning problem $\psi(\mathbf{P}) = \mathbf{P_H}$, under the assumption that **P** and $\mathbf{P_H}$ have the same set of operators.

We define a TMK planning problem as a triple $\mathbf{P} = \langle d, I, \mathbf{T} \rangle$, where $\mathbf{T} = (M_T, O)$ is a TMK process model, I is the initial state, and d is the task network for which a plan is desired. Therefore, the translation function ψ defines a translation from M_T into M_H because d, I, and O are the same for both **P** and $\mathbf{P_H}$.

Given the existence of a transformation, both **P** and σ can apply to HTNs and therefore lend their well-defined semantics to TMKs. In particular:

- **TMK Planning is Sound and Complete**. This means that if a planning problem is solvable, a correct solution plan will be found. Informally, given a TMK planning problem $\mathbf{P} = \langle d, I, \mathbf{T} \rangle$, a plan that recursively achieves all tasks in d is called a correct plan of the planning problem (Nau *et al*., 1999). Given a TMK problem $\mathbf{P} = \langle d, I, \mathbf{T} \rangle$, where $\mathbf{T} = (M_T, O)$, the translation $\mathbf{H} = (\psi(M_T), O)$ defines an HTN planning domain. The solution plan for $\mathbf{P_H} = \langle d, I, \mathbf{H} \rangle$ is a sequence of actions, each an instance of an operator in O. This same sequence is also a solution for **P**.
- **TMKs are more expressive than STRIPS**. This means that, for every Turing-computable function ψ from the set of TMK planning problem instances to the set of STRIPS planning problem instances, there is a TMK planning problem instance **P**, for which either $\psi(\mathbf{P})$ is not defined or, if it is defined, σ does not solve $\psi(\mathbf{P})$. This result follows from HTNs being provably more expressive than STRIPS (Erol *et al*., 1994) and because TMKs can always be represented as equivalent HTNs.
- **TMK planning is NP-Complete**.

Conclusions and Future Work

In this paper we described formal semantics for Task-Method-Knowledge process models by constructing a Turing-computable translation function that maps TMKs into equivalent HTN constructs. This allowed us to state soundness, completeness, and complexity results for planning with TMK process models, and prove that TMK planning is strictly more expressive than STRIPS planning. Furthermore, we proved that this translation function runs in polynomial time.

In our future work, we wish to conduct experiments to confirm the theoretical results. We wish to empirically show that the overhead of plan generation running time with a translated domain is linearly correlated to an equivalent domain natively encoded in an HTN. We will explore solving project planning problems translated from TMKs into HTNs. We also wish to explore our conjecture that the translation works for other process languages.

References

Choi, D., & Langley, P. 2005. Learning teleoreactive logic programs from problem solving. *Proceedings of the Fifteenth International Conference on Inductive Logic Programming*. Bonn, Germany: Springer.

Curbera, F., Goland, Y., Klein, J., Leymann, F., Roller, D., Thatte, S., Weerawarana, S. 2002. Business Process Execution Language for Web Services, Version 1.0. *IBM technical report*.

Erol, K., Hendler, J., & Nau, D. 1994. HTN planning: Complexity and Expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence* (AAAI-94). AAAI Press

Erol, K., Hendler, J., & Nau, D. 1995. *Complexity results for HTN planning*. Annals of Mathematics and Artificial Intelligence. Kluwer.

Ilghami, O., Muñoz-Avila, H., Nau, D. S., and Aha, D. W. 2005. Learning approximate preconditions for methods in hierarchical plans. *Proceedings of the International Conference on Machine Learning* (ICML-05)

Fox, M., & Long, D. 2003. pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research.*.

Molineaux, M., & Aha, D.W. 2005. TIELT: A Testbed for Gaming Environments . In: Proceedings of the Twentieth National Conference on Artificial Intelligence. Pittsburgh, PA: AAAI Press.

Murdock, J. W. 2001. *Self-Improvement through Self-Understanding: Model-Based Reflection for Agent Adaptation*. Dissertation Georgia Institute of Technology.

Ulam, P., Goel, A., Jones, J. & Murdock, J.W. 2005. Using Model-Based Reflection to Guide Reinforcement Learning. *Proceedings of the IJCAI 2005 Workshop on Reasoning, Representation and Learning in Computer Games*. Edinburgh, UK.

Wilkins, D. E. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann Publishers Inc.

Software tool for the master production schedule conception based on the Capacitated Lot Sizing Problem

M. Gourgand[(1)], N. Grangeon[(2)], D. Lemoine[(2)], S. Norre[(2)]

LIMOS CNRS UMR 6158
[(1)] ISIMA, 24 avenue des Landais, 63000 Clermont Ferrand
[(2)] IUT de Montluçon, Avenue A. Briand, BP 2235, 03101 Montluçon
{gourgand,grangeon}@isima.fr
{lemoine,norre}@moniut.univ-bpclermont.fr

Introduction

Production management is a set of decisions that must reflect a compromise between customers' satisfaction and production criteria such as cost, delay and quality. The goal of production management is to ensure the continued success of the firm. Industrial planning plays a central part in this one. The latter can be divided into three hierarchical decisional levels:

- the strategic level deals with long-term decisions such as opening or closing of factories or determining building sites. (horizon is more than eighteen months),
- the tactical level which deals with conception of several plans such as Master Production Schedule, stock policy on a mid term horizon (six to eighteen months),
- the operational level which relates to the daily scheduling of the workshops.

Judging from the literature, the tactical planning is composed of two plans: The Sales and Operations Planning (S&OP) and the Master Production Schedule. The objective of S&OP is to obtain a compromise between sales objectives and production capacities. Therefore, it constrains the Master Production Schedule which determines, for each period, a balance between the capacity constraints and the customer's satisfaction while minimizing the production cost.

The mathematical model

Traditionally, the tactical planning models are based on Lot-Sizing models which determine the size of batches in order to minimize costs (setup cost, holding cost, production cost). Among those, there is a basic model: the "Capacitated Lot Sizing Problem" (CLSP) which elaborates the Master Production Schedule.

The CLSP can formally be described as a mixed-programming model:

$$Min \sum_{t=1}^{T}\left[\sum_{i=1}^{N}\left(s_i X_{it} + h_i I_{it} + r_i Q_{it}\right)\right] \qquad (Eq.1)$$

$$s.c:$$

$$\begin{cases} I_{i,t} = I_{i,t-1} + Q_{i,t} - d_{it}, & i \in \{1,..,N\}, \quad t \in \{1,..,T\} \quad (Eq.2) \\ \sum_{i=1}^{N} p_i Q_{it} \le C_t, & t \in \{1,..,T\} \quad (Eq.3) \\ Q_{it} \le C_t X_{it}, & i \in \{1,..,N\}, \quad t \in \{1,..,T\} \quad (Eq.4) \\ I_{it}, Q_{it} \in \mathbb{N}, & i \in \{1,..,N\}, \quad t \in \{1,..,T\} \quad (Eq.5) \\ Y_{it} \in \{0,1\}, & i \in \{1,..,N\}, \quad t \in \{1,..,T\} \quad (Eq.6) \end{cases}$$

Parameters for the CLSP:

N	Number of items.
T	Number of periods.
d_{it}	External demand for item i at period t.
C_t	Available capacity of the machine at period t.
p_i	Capacity request for producing one unit of item i.
s_i	Non-negative setup costs for item i.
h_i	Non-negative holding costs for item i.
r_i	Non-negative production costs for item i.
I_{i0}	Initial inventory for item i.

Decision variables for the CLSP:

Q_{it}	Production quantity of item i at period t.
I_{it}	Inventory of item i at the end of period t.
X_{it}	Binary variable which indicates whether a setup for item i occurs at period t ($X_{it}=1$) or not ($X_{it}=0$).

(Eq.1) is the objective function: it means the sum of the setup, the holding and the production costs that we seek to minimize.
(Eq.2) represents the inventory balances.
(Eq.3) represents the capacity constraint.
(Eq.4) represents the setup constraint: due to these restrictions, production of an item can only take place if the machine is set up for that particular item.
(Eq.5) are the non negativity conditions.
(Eq.6) the setup variables are defined as binary.

State of art

Solving CLSP is known as NP-Hard (Bitran et al. 1982). If positive setup times are added into the model, the feasibility problem is NP-complete (Trigeiro and al. 1989). In this case, the Eq 3 becomes

$$\sum_{i=1}^{N}\left(p_i Q_{it} + z_i X_{it}\right) \le C_t, \quad i \in \{1,..,N\}, \quad t \in \{1,..,T\} \quad (Eq.3')$$

where z_i is the setup time to set the machine up to produce the item i at period t.

Many researchers have developed solutions for CSLPs, including mathematical programming (Leung et al 1997, Eppen et al. 1987, Belvaux et al. 2001 …), heuristic solutions (Dogramaci et al. 1981, Trigeiro et al. 1989, Diaby et al. 1993, Kirca et al. 1995, Degraeve et al. 2003 …), and metaheuristics (Gopalakrishnan et al. 2001, Özdamar et al. 2002, Karimi et al. 2005 …)

Our contribution

Our proposal is divided into two successive axes:

- The first one concerns the computation of a feasible solution, i.e. the design of a tactical plan which respects the capacity constraint and the customer's demand, even if setup times are considered,
- The second one is the cost optimization of the feasible solution found before.

In each step, we propose a metaheuristic. We encode the solution by a matrix representing a production plan. For each metaheuristic, we define an objective function and neighbouring systems.

Computation of a feasible solution:

So as to, we use the kangaroo algorithm, a metaheuristic based on simulated annealing (Fleury, 1993). For this one, we propose:

- a new quadratic objective function which models the capacity overshooting, whose minimization allows the smoothing of the production in order to find a feasible plan.
- two types of neighbouring systems: the first one allows to move a quantity period by period if and only if the customer's demand and capacity constraint are respected. The second one, changes the current solution, neglects capacity but respects demand.
- We use customer demand as initial solution for this method. In most of the cases, this solution is not feasible.

The quadratic function is defined as:

$$k(Q) = \sum_{t=1}^{T}\left[Max\left(0, \left(\sum_{i=1}^{N} p_i Q_{it} + z_i X_{it}\right) - C_t\right)\right]^2$$

where Q is the proposed planning. As Shown in Fig.1, this latter allows the smoothing of the production.

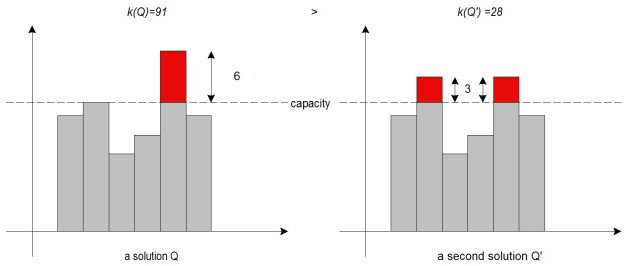


Fig1. Effect of the objective function

An initial solution Q^* is found if and only if $k(Q^*)=0$.

The first neighbouring system is summarised by the algorithm Algo1 :

Input: A solution Q
Output: A new solution Q^*
1. $Q^* \leftarrow Q$
2. Choose i randomly in {1,..,N}.
3. Choose randomly t_{start} and t_{target} in {1..T}
4. Compute K_{max}, the maximal quantity of item i to be shifted from t_{start} to t_{target} according to proposition 1 below.
5. Set α:=0.6, choose randomly $\beta \in [0,1]$
6. If $\beta \le \alpha$ then choose randomly K in $\{0,K_{max}\}$ else $K=K_{max}$.
7. $Q^*[i,t_{start}] := Q^*[i,t_{start}] - K$
8. $Q^*[i,t_{target}] := Q^*[i,t_{target}] + K$.

Algo1: Algorithm for the first neighbouring system.

α reflects distribution of two strategies: the $K=K_{max}$ strategy which tries to remove a maximum of setup time, and the other one which allows smaller adjustments.

Proposition1: Let Q a production planning which respects the demand. Q^* is another one if and only if:

- if target≤start:

$$K_{\max} = \min\left\{\left\lfloor\left(C_{target} - \sum_{j=1}^{N}\left(p_j Q_{jtarget} - z_j X_{jtarget}\right)\right)/\, p_i\right\rfloor, Q_{istart}\right\}$$

- if start>target:

$$K_{\max} = \min\left\{\left\lfloor\left(C_{target} - \sum_{j=1}^{N}\left(p_j Q_{jtarget} - z_j X_{jtarget}\right)\right)/\, p_i\right\rfloor, Q_{istart}, \min_{start \le t < target} I_{it}\right\}$$

The second neighbouring system uses the same algorithm but neglects capacity in the determination of K_{max}. By the way, we accept worse transitions in term of capacity.

Optimization of the feasible solution:

Concerning the optimization part, we use several metaheuristics based on simulated annealing algorithm:

- The objective function stands for the whole production cost for the plan as modelled in the CLSP (Eq 1),
- The neighbouring system allows the quantity moves period by period if and only if it respects the customer's demand and doesn't exceed the capacity of the target period. Moreover, it examines the inventory position in order to improve the computation of a new current solution while banishing bad tries.

The neighbouring system is based on the (Algo1). Indeed, proposition 1 ensures that if Q is feasible for the CLSP then Q^* deduced from (Ago1.) will keep that property. The major change is in the choice of α. Indeed, α is designed as a function of iteration count. Its goal is to support the choice of $K=K_{max}$ initially in order to remove a maximum of setup cost. This function converges gradually toward 0,6 because, after many tries, we have determined that it is a very good distribution for the two choices. Therefore

$$\alpha(iter)=0.6\text{-}1/\ln(1+10^3\times iter)$$

In order to measure the quality of the obtained solution, we have implemented a lagrangian relaxation to determine a lower bound. This one is inspired by Diaby's works (Diaby *et al.* 1993). Our relaxation deals with Capacities constraints (Eq.3). Indeed, Chen *et al.* 1990 have shown that it provides the best lower bound. The Wagner-Within algorithm (Wagner *et al.* 1958) is used to solve subproblems optimally.

Our method is tuned according to the general formula which guides lagrangian coefficients:

$$\lambda_t^{k+1} = Max\left[0, \lambda_t^k + \gamma_k \frac{(Z - Z_k)\sum_{i=1}^{N}\left(p_i Q_{it}^k + z_i X_{it}^k\right) - C_t}{\sum_{t=1}^{T}\left(\sum_{i=1}^{N}\left(p_i Q_{it}^k + z_i X_{it}^k\right) - C_t\right)^2}\right]$$

where, Z is an upper bound of the lagrangian relaxation, Z_k is the value of objective function for the lagrangian relaxation at iteration k: it could be updated during algorithm proceeding. $\left(Q_{it}^k, X_{it}^k\right)$ is the solution of the lagrangian relaxation problem at iteration k and $0 < \gamma_k \leq 2$.

Computational results:

Early results of testing our approach are promising. Indeed, we have tested these different methods on small instances. For each instance, we obtained an optimal solution. We are testing these different methods on Trigeiro's instances which are commonly used like benchmark in literature. We chose the ones selected by Wolsey in his lotesizelib (2000) because they appear to be among the most difficult ones (Belvaux *et al.* 2000)

Tab.1 shows some results about feasibility for these instances:

Name	N. Items	N. periods	Success	N. Iter
G30	6	15	Y	14994
G53	12	15	Y	115336
G57	24	15	Y	4475
G62	6	30	Y	4265
G69	12	30	Y	10650
G72	24	30	Y	9049

Tab.1: Feasibility tests

As we say, we obtain very good results for the feasibility problem. We also test our optimization metaheuristic on these same instances neglecting setup time, in a first time.

Tab.2 shows some results about it:

Name	Solver	Kangaroo			
		Sol.	N. Iter	LB	Gap
G30	**37156***	37267	1651164	36939,5	0,89 %
G53	72831*	**71599**	1434315	70717,73	1,25 %
G57	137762*	**137659**	384215	135938,96	1,27 %
G62	**62058***	62545	1375810	60626,80	3,17 %
G69	132097*	**131839**	1224781	129589,97	1,74 %
G72	295291*	**295209**	910361	287390,70	2,73 %

Tab.2: Optimization Tests

We use Cplex 9.1 as solver for our tests: (*) means that it stops with an "overflow error". We test our metaheuristic with 600s time limit. We calculate our Lower Bound (LB) according to our lagrangean relaxation. We can see that we obtain very good results even if metaheuristic time limit is very weak. However, we can observe that higher is the number of items, better is our Kangaroo, compared to the Solver. To conclude, this approach is promising. We are still testing it on all Trigeiro's instances.

The software tool:

In order to test all these methods, we have designed a software tool which incorporates all these techniques and which allows us to follow, in real time, the evolution of the metaheuristics proposed. This software allows us to import the Trigeiro's instances or to create ours by using a convivial interface where we can parameterize the production system (capacity, cost …) and the customer's demand.

The main part of it is the optimization tool in which are implemented all the optimization methods seen before. On the GUI, we can follow the metaheuristic behavior, the gap of the best solution found (with the lower bound determinated by our lagrangian relaxation), and the best plan found, at any time of the optimization.

Our prospect:

We have proposed a software tool for the deterministic CLSP. However, within the framework of industrial planning, the managers work with estimated demands, thus potentially subjected to strong fluctuations. Therefore, we wish to extend our research by taking into account the uncertainty in the demand and the production capacities (resources into breakdowns etc.) so as to incorporate a new dimension: the robustness, in the obtained solution. Moreover, the presented model is a single-level model which doesn't take into account the material requirement planning (MRP) for the end items' planning conception. Moreover, it doesn't integrate the multi-sites aspect in the current productions systems. Therefore we are considered multi-sites models of planning based on the lot-sizing model: Multi Level Capacitated Lot Sizing Problem (MLCLSP). The next step would be to propose a new tactical planning's model, multi-levels and multi-sites as well as resolution approaches which could provide us a robust production plan, under an uncertain industrial context.

References

Belvaux, G. and Wolsey, L.A. *Lot-sizing issues and a specialized branch-and-cut system BC-PROD*. Core discussion paper, 1998.

Bitran, G.R., and Yanasse, H.H. *Computational complexity of the capacitated lot size problem*. Management Science, 46(5):724–738, 1982.

Belvaux, G. and Wolsey, L. *Lot-sizing problems: modelling issues and a specialized branch-and-cut system BC-PROD.* Management Science, 46(5) :724–738, 2000

Chen, W.H., and Thizy, J.M. *Analysis of relaxation for the multi-item capacities lot-sizing problem*. Annal of Operations Research, 26:29–72, 1990.

Degraeve, Z. and Jans, R. *A new Dantzig-Wolfe reformulation and Branch-And-Price Algorithm for the Capacitated Lot Sizing Problem With Set up Times.* ERIM Report Series in Management ERS-2003-010-LIS, Erasmus University Rotterdam, The Netherlands.

Diaby, M., Bahl, M., Karwan, H.C., and Zionts, S. *Capacitated lot-sizing and scheduling by lagrangean relaxation*. European Journal of Operational Research, 59:444–458, 1992.

Dogramaci, A., Panayiotopoulos, J.C., and Adam, N.R. *The dynamic lot-sizing problem for multiple items under limited capacity*. AIIE Transactions, 13(4) :294–303, 1981.

Eppen, G.D., and Martin, R.K. *Solving multi-item lot-sizing problems using variable redefinition*. Operations Research, 35 :832-848, 1987.

Fleury, G. *Méthodes stochastiques et deterministes pour les problèmes NP-difficiles*. PhD thesis, Clermont-Ferrand II university.

Gopalakrishnan, M., Ding, K., Bourjolly, J.M. and Mohan, S. *A Tabu-search Heuristic for the Capacitated Lot-Sizing Problem with Set-up Carryover.* Management Science, 47(6): 851-863, 2001.

Karimi, B., Fatemi Ghomi, S.M.T and Wilson, J.M. *A tabu search heuristic for solving the CLSP with backlogging and set-up carry-over*. Journal of the Operational Research Society, 57(2):140-147, 2006.

Kirca, M., and Kökten, Ö. *A new heuristic approach for the multi-item dynamic lot sizing problem*. European Journal of Operational Research, 75 :332–341, 1994.

Leung, J.M., Magnanti, T.L., and Vachani. R., *Facets and algorithms for the capacitated lot sizing*. Mathematical programming, 45:331–359, 1989.

Özdamar, L., Birbil, S.I. and Portmann M.C. *Technical note: New results for the capacitated lot sizing problem with overtime decisions and setup times.* Production Planning & Control, 13(1):2-10, 2002.

Trigeiro, W.W., Thomas, L.J., and Mc Clain, J.O. *Capacitated lot sizing with setup times*. Management science, 35 :353–366, 1989.

Wagner, H.M., and Whitin, T.M. *Dynamic version of the economic lot size model*. Management science, 5 :89–96, 1958.

Woley, LotsizeLib, "Lot-Sizing Problems : A Library of Models and Matrices". http://www.core.ucl.ac.be/wolsey/lotsizel.htm, 2006

Towards Efficient Probabilistic Temporal Planning

Iain little
National ICT Australia & Computer Sciences Laboratory
The Australian National University
Canberra, ACT 0200, Australia
A related paper appears in ICAPS'06.

Introduction

Many real-world planning problems involve a combination of both time and uncertainty (Bresina *et al.* 2002). For instance, Aberdeen *et al.* (2004) investigate military operations planning problems that feature concurrent durative actions, probabilistic timed effects, resource consumption, and competing cost measures. It is the potential for such practical applications that motivates this research.

Probabilistic temporal planning combines concurrent durative actions with probabilistic effects. This unification of the disparate fields of probabilistic and temporal planning is relatively immature, and presents new challenges in efficiently managing an increased level of expressiveness. Some of our techniques for solving probabilistic temporal planning problems could be applied beyond the context they were developed in, and may prove useful in efficiently solving the simpler subproblems.

The most general probabilistic temporal planning framework considered in the literature is that of Younes and Simmons (2004). It is expressive enough to model generalised semi-Markov decision processes (GSMDPs), which allow for exogenous events, concurrency, continuous-time, and general delay distributions. This expressiveness comes at a cost: the solution methods proposed in (Younes & Simmons 2004) lack convergence guarantees and significantly depart from the traditional algorithms for both probabilistic and temporal planning. Concurrent Markov decision processes (CoMDPs) are a much less general model that simply allows instantaneous probabilistic actions to execute concurrently (Guestrin, Koller, & Parr 2001; Mausam & Weld 2004). Aberdeen *et al.* (2004) and Mausam and Weld (2005) have extended this model by assigning actions a fixed numeric duration. They solved the resulting probabilistic temporal planning problem by adapting existing MDP algorithms, and have devised heuristics to help manage the exponential blowup of the search space.

The ultimate goal of this research is to produce planners that are expressive enough to support: concurrent durative actions, probabilistic effects, metric resources, and cost functions; while being efficient enough to solve interesting-sized (real-world) problems.

We currently have two separate avenues of research with the aim of achieving this goal. The first approach is to combine a forward-chaining search with effective heuristics. We have developed a probabilistic temporal planner called `Prottle` using this approach (Little, Aberdeen, & Thiébaux 2005). `Prottle` uses a (deterministic) trial-based search algorithm with a heuristic that is based on an extension of the planning graph data structure.

Another approach to planning is the `Graphplan` framework (Blum & Furst 1997). While `Prottle` makes use of the planning graph—a data structure that originates from this framework—it does not use the framework's other key features; in particular, `Prottle` does not use a backward search. The `Graphplan` framework has previously been successfully applied to temporal planning (concurrent durative actions) (Smith & Weld 1999), but had not been successfully applied to probabilistic planning (actions with probabilistic effects) in its entirety. Extensions of this framework for probabilistic planning had been developed (Blum & Langford 1999), but either dispense with the techniques that enable concurrency to be efficiently managed, or are unable to produce optimal contingency plans.

As a way of investigating approaches to compressing the search space for probabilistic temporal planning, our other avenue of research has the goal of implementing a probabilistic temporal planning in the `Graphplan` framework. As the issues relating to probabilistic planning had not been adequately solved, and as a way of managing the complexity, we started by developing a (concurrent) probabilistic planner (Little & Thiébaux 2006). `Paragraph`, the resulting planner, is competitive with the state of the art, producing acyclic or cyclic plans that optionally exploit a problem's potential for concurrency. We are confident that this approach can be extended to the probabilistic temporal context.

This paper gives a brief overview of both `Prottle` and `Paragraph`, and concludes with remarks about our future research intentions. For more detailed descriptions and experimental results, please refer to the respective papers (Little, Aberdeen, & Thiébaux 2005; Little & Thiébaux 2006).

Prottle

`Prottle` is a probabilistic temporal planner that allows effects, the time at which they occur, and action durations to all be probabilistically determined. Its input language is the temporal STRIPS fragment of PDDL2.1 (Fox & Long 2003), but extended so that effects can be probabilistic, as

in PPDDL (Younes & Littman 2004). We also allow effects to occur at any time within an action's duration. The probabilistic and temporal language constructs interact to allow effect times and action durations to vary probabilistically. For clarity, each probabilistic alternative is given a descriptive label.

```
(:durative-action jump
    :parameters (?p - person ?c - parachute)
    :condition (and (at start (and (alive ?p)
                                   (on ?p plane)
                                   (flying plane)
                                   (wearing ?p ?c)))
                    (over all (wearing ?p ?c)))
    :effect (and (at start (not (on ?p plane)))
                 (at end (on ?p ground))
                 (at 5
                   (probabilistic
                     (parachute-opened 0.9 (at 42 (standing ?p)))
                     (parachute-failed 0.1
                       (at 13 (probabilistic
                                (soft-landing 0.1
                                  (at 14 (bruised ?p)))
                                (hard-landing 0.9
                                  (at 14 (not (alive ?p))))))))))
```

Figure 1: An example of an action to jump out of a plane.

Figure 1 shows an example action that represents a person jumping out of a plane with a parachute. After 5 units of time, the person makes an attempt to open the parachute. The case where this is successful has the label `parachute-opened`, and will occur 90% of the time; the person will gently glide to safety, eventually landing at time 42. However, if the parachute fails to open, then the person's survival becomes dependent on where they land. The landing site is apparent at time 13, with a 10% chance of it being soft enough for the person to survive. Alive or dead, the person then lands at time 14, 28 units of time sooner than if the parachute had opened. But regardless of the outcome, or how long it takes to achieve, the action ends with the person's body on the ground.

`Prottle`'s search space is defined in terms of an AND/OR graph. In the interpretation that we use, an AND node represents a *chance*, and an OR node a *choice*. We associate choice nodes with the selection of actions, and chance nodes with the probabilistic event alternatives.

Each node is used in one of two different ways: for *selection* or *advancement*. This is similar to what some temporal planners do, where states are partitioned between those that represent action selection, and those that represent time advancement (Bacchus & Ady 2001). This sort of optimisation allows forward-chaining planners to be better guided by heuristics, as action sets are structured into linear sequences.

The rules for node succession are defined by Figure 2. They can be summarised as: every successor of a node must either be a selection node of the same type, or an advancement node of the opposite type. Our choice of a search space structure is intended to be used with a 'phased' search, where action selection and outcome determination are kept separate. It might seem that it would be more efficient to have only a single selection phase, where an action's probabilistic branching is dealt with immediately after it is selected, but consider what this does to the problem: we would be assuming that an action's outcome is known as soon as

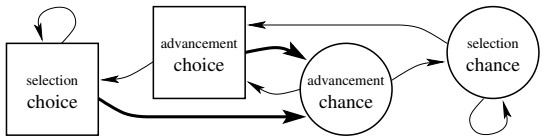


Figure 2: A state machine for valid node orderings. Time may only increase when traversing bold face arcs.

the action starts execution. In contrast, the phased approach allows the time at which this knowledge is available to be accurately represented, by deferring the branching until the appropriate time. This issue of knowledge becomes relevant when concurrency is combined with probabilistic effects. The conservative assumption — that we wait until actions terminate — breaks down when an action's duration can depend on its outcome.

Using the graph structure that we have established, we define a state of the search space as a node in an AND/OR graph that is identified by a *time*, *model* and *event queue*. The time of a state is generally the same as its predecessors, but may increase when advancing from choice to chance (see Figure 2). The model is the set of truth values for each of the propositions, and the event queue is a time-ordered list of pending events. An event can be an effect e.g. `(on ?p ground)`, a probabilistic event, or an action execution condition that needs to be checked. When the time is increased, it is to the next scheduled event time.

We associate states with both lower and upper cost bounds. As the search space is explored, the lower bounds will monotonically increase, the upper bounds monotonically decrease, and the actual cost is sandwiched within an ever-narrowing interval. We say that a state's cost has *converged* when, for a given $\epsilon \geq 0$: $U(s) - L(s) \leq \epsilon$ where U is the upper bound and L the lower bound of state s. A state's cost bounds are initially determined using a planning graph-based heuristic, and are updated by comparing its current values with those of its successors.

In addition to a cost, we also associate each state with a label of either *solved* or *unsolved*. A state is labelled as solved once the benefit of further exploration is considered negligible; for instance, once its cost has converged for a sufficiently small ϵ. The search algorithm ignores a state once it has been labelled as solved, and confines its exploration to the remaining unsolved states.

`Prottle` uses a search algorithm that combines a deterministic search with the convergence and labelling optimisations used by LRTDP (Bonet & Geffner 2003). As with previous probabilistic temporal planners (Aberdeen, Thiébaux, & Zhang 2004; Mausam & Weld 2005), this algorithm is trial-based, and explores the search space by performing repeated depth-first probes starting from the initial state.

Paragraph

`Paragraph` is a probabilistic planner that finds contingency plans that maximise the probability of reaching the goal within a given time horizon. These solutions are optimal in the non-concurrent case, and optimal for a re-

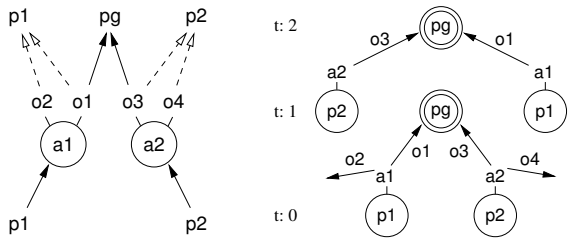


Figure 3: An action-outcome-proposition dependency graph and search space for an example problem.

stricted model of concurrency. A detailed description of this model—and of `Paragraph` in general—is given in (Little, Aberdeen, & Thiébaux 2005).

`Paragraph` extends the `Graphplan` framework to the probabilistic setting. To do this, it is necessary to extend the planning graph data structure to account for uncertainty. We do this by introducing a node for each of an action's possible outcomes, so that there are three different types of nodes in the graph: proposition, action, and outcome. Action nodes are then linked to their respective outcome nodes, and edges representing effects link outcome nodes to proposition nodes. Each persistence action has a single outcome with a single add effect. We refer to a persistence action's outcome as a *persistence outcome*. This extension is functionally equivalent to that described in (Blum & Langford 1999), except that we also adapt the planning graph's mutex propagation rules from the deterministic setting.

The solution extraction step of the `Graphplan` algorithm relies on a backward search through the structure of the planning graph. In classical planning, the goal is to find a subset of action nodes for each level such that the respective sequence of action sets constitutes a valid trajectory. The search starts from the final level of the graph, and attempts to extend partial trajectories one level at a time until a solution is found.

`Paragraph` uses this type of goal-regression search with an explicit representation of the expanded search space. This search is applied exhaustively, to find all trajectories that the `Graphplan` algorithm can find. An optimal contingency plan is formed by linking these trajectories together. This requires some additional computation, and involves using forward simulation through the search space to compute the possible world states at reachable search nodes.

As observed by Blum and Langford (1999), the difficulty with combining probabilistic planning with `Graphplan`-style regression is in correctly and efficiently combining the trajectories. Sometimes the trajectories will 'naturally' join together during the regression, which happens when search nodes share a predecessor through different 'joint outcomes' (sets of outcomes) of the same action set.

Unfortunately, the natural joins are not sufficient to find all contingencies. Consider the problem shown in Figure 3, which we define as:[1] the propositions $p1$, $p2$ and pg; $s_0 = \{p1, p2\}$; $G = \{pg\}$; the actions $a1$ and $a2$; and the outcomes $o1$ to $o4$. $a1$ has precondition $p1$ and outcomes $\{o1, o2\}$; $a2$ has precondition $p2$ and outcomes $\{o3, o4\}$. Both actions always delete their precondition; $o1$ and $o3$ both add pg. To simplify the example, we prohibit $a1$ and $a2$ from executing concurrently. The optimal plan for this example is to execute one of the actions; if the first action does not achieve the goal, then the other action is executed.

The backward search will correctly recognise that executing $a1$–$o1$ or $a2$–$o3$ will achieve the goal, but it fails to realise that $a1$–$o2, a2$–$o3$ and $a2$–$o4, a1$–$o1$ are also valid trajectories. The longer trajectories are not discovered because they contain a 'redundant' first step; there is no way of relating the effect of $o2$ and the precondition of $a2$, or the effect of $o4$ with the precondition of $a1$. While these undiscovered trajectories are not the most desirable execution sequences, they are necessary for an optimal contingency plan. In classical planning, it is actually a good thing that trajectories with this type of redundancy cannot be discovered, as redundant steps only hinder the search for a single shortest trajectory. Identifying the missing trajectories requires some additional computation beyond the goal regression search. We refer to trajectories that can be found using unadorned goal regression as *natural trajectories*.

The solution we have developed is based on constructing all 'non-redundant' contingency plans by linking together the trajectories that goal regression is able to find. This is sufficient to find an optimal solution, as there always exists at least one non-redundant optimal plan. `Paragraph` combines pairs of trajectories by linking a node in one trajectory to a node in the other. This can be done when a possible world state of the earlier node has a resulting world state that subsumes the goal set of the later node.

A detailed description of `Paragraph`'s acyclic search algorithm follows.[2] The first step is to generate a planning graph from the problem specification. This graph is expanded until all goal propositions are present and not mutex with each other, or until the graph levels off to prove that no solution exists. Assuming the former case, a depth-first goal regression search is performed from a goal node for the graph's final level. This search exhaustively finds all natural trajectories from the initial conditions to the goal. Once this search has completed, the possible world states for each trajectory node are computed by forward-propagation from time 0, and the node/state costs are updated by backward-propagation from the goal node. Potential trajectory joins are detected each time a new node is encountered during the backward search, and each time a new world state is computed during the forward state propagation. Unless a termination condition has been met, the planning graph is then expanded by a single level, and the backward search is performed from a new goal node that is added to the existing search space. This alternation between backward search, state simulation, cost propagation, and graph expansion continues until a termination condition is met. An optimal con-

[1] This problem was used by Blum and Langford (1999) to illustrate the difficulty of using goal-regression for probabilistic planning, and to explain their preference of a forward search in `PGraphplan`.

[2] We have another algorithm for extracting cyclic solutions.

Horizon	PRTTL Time	NA-PG Time	Cost
10	14.0	0.23	0.728
15	21.6	0.73	0.607
20	25.1	12.5	0.486
25	36.0	52.2	0.429
30	40.6	103	0.429

(a) g-tire

Horizon	PRTTL Time	CA-PG Time	PRTTL Cost	CA-PG Cost
5	4.38	0.08	0.272	0.204
6	14.9	0.13	0.204	0.193
7	168	0.26	0.178	0.156
8	554	0.71	0.151	0.149
15	–	613	–	0.078

(b) maze

tingency plan is then extracted from the search space by traversing the space in the forward direction using a greedy selection policy.

Example Results

We give a sample of our experimental results for `Prottle` and `Paragraph`. For more detailed comparative results, see (Little & Thiébaux 2006). Additional results for `Prottle` can be found in (Little, Aberdeen, & Thiébaux 2005). `Prottle` and `Paragraph` are implemented in Common Lisp, and were both compiled using `CMUCL` version 19c. These experiments were performed on a machine with a 3.2 GHz Intel processor and 2 GB of RAM.

Figure shows comparative results for two problems, g-tire and maze. Their PDDL definitions are available at `http://rsise.anu.edu.au/~thiebaux/benchmarks/pddl/`. The planner configurations used in these experiments are: `Prottle` with $\epsilon = 0$ and its cost-based planning graph heuristic (PRTTL), and `Paragraph` with its acyclic search using either the restricted concurrency model (CA-PG) or no concurrency (NA-PG).

The objective of the g-tire problem is to move a vehicle from one location to another, where each time the vehicle moves there is a chance of it getting a flat tire. There are spare tires at some of the locations, and these can be used to replace flat tires. This problem is not concurrent. The results compare `Prottle` to `Paragraph`'s acyclic search; `Paragraph` is faster for the earlier horizons, but `Prottle` scales better.

The maze problem involves a number of connected rooms and doors, some of which are locked and require a specific key to open. This problem has some potential for concurrency, although mostly of the type not allowed in composite contingency plans. None of the planner configurations fully expoit it. `Paragraph` scales much better than `Prottle` this time.

We have found that `Paragraph` usually out-performs `Prottle`. `Paragraph` has the best comparative performance on problems with a high forward branching factor and relatively few paths to the goal.

Conclusion and Future Work

In `Paragraph` and `Prottle`, we have made significant progress towards our goal of producing an efficient planner that can deal with all of: concurrent durative actions, probabilistic effects, metric resources, and cost functions. We believe that both planning approaches show promise, and have a strong potential for future improvement.

The most important future improvements for `Prottle` include: reducing the implementation's memory usage, devising ways of efficiently extracting a greater amount of heuristic information from the planning graph, and adding support for metric resources and cost functions. Another intriguing possibility is extending `Prottle`'s effect model (as a decision tree) to the more general graph. This might be an effective way of modelling exogenous processes.

We have many ideas for improving `Paragraph`'s performance, in particular by adapting optimisations developed for the `Graphplan` framework in the deterministic setting. For example, we have observed that a small amount of control knowledge in the form of mutex invariants can make a substantial impact on efficiency. This suggests that there would also be a benefit in investigating ways of strengthening the planning graph's mutex reasoning and in incorporating explanation-based learning. But the most important future direction of this research is extending `Paragraph` to the probabilistic temporal setting, which will allow us to compare our two approaches in the context of probabilistic temporal planning.

References

Aberdeen, D.; Thiébaux, S.; and Zhang, L. 2004. Decision-theoretic military operations planning. In *Proc. ICAPS*.

Bacchus, F., and Ady, M. 2001. Planning with resources and concurrency: A forward chaining approach. In *Proc. IJCAI*.

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90:281–300.

Blum, A., and Langford, J. 1999. Probabilistic planning in the Graphplan framework. In *Proc. ECP*.

Bonet, B., and Geffner, H. 2003. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proc. ICAPS*.

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

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

Guestrin, C.; Koller, D.; and Parr, R. 2001. Multiagent planning with factored MDPs. In *Proc. NIPS*.

Little, I., and Thiébaux, S. 2006. Concurrent probabilistic planning in the graphplan framework. In *Proc. ICAPS*.

Little, I.; Aberdeen, D.; and Thiébaux, S. 2005. Prottle: A probabilistic temporal planner. In *Proc. AAAI*.

Mausam, and Weld, D. 2004. Solving concurrent Markov decision processes. In *Proc. AAAI*.

Mausam, and Weld, D. 2005. Concurrent probabilistic temporal planning. In *Proc. ICAPS*.

Smith, D., and Weld, D. 1999. Temporal planning with mutual exclusion reasoning. In *Proc. IJCAI*.

Younes, H., and Littman, M. 2004. PPDDL1.0: The language for the probabilistic part of IPC-4. In *Proc. International Planning Competition*.

Younes, H. L. S., and Simmons, R. G. 2004. Policy generation for continuous-time stochastic domains with concurrency. In *Proc. ICAPS*.

Challenges for Temporal Planning with Uncertain Durations

Mausam and **Daniel S. Weld**
Dept. of Computer Science, University of Washington
Seattle, WA-98195
{mausam,weld}@cs.washington.edu
To appear as a short paper at ICAPS'06.

Abstract

We investigate the problem of temporal planning with concurrent actions having stochastic durations, especially in the context of extended-state-space based planners. The problem is challenging because stochastic durations lead to an explosion in the *space of possible decision-epochs*, which exacerbates the familiar challenge of growth in executable action combinations caused by concurrency. We present various observations and insights into different variations of this problem that form the basis of our future research.

Introduction

Recent progress in temporal planning (JAIR Special Issue 2003) raises hopes that this technology may soon apply to a wide range of real-world problems. However, concurrent actions with stochastic durations characterise many real-world domains. While both concurrency and duration uncertainty have independently received some attention by planning researchers, very few systems have addressed them in concert, and all of these systems have used an extended-state-space method (as opposed to a constraint-posting approach). In this paper we step back from specific algorithms and analyse the broader problem of concurrent temporal planning with actions having stochastic durations, especially in the context of extended-state-space planners.

We find that the problem is challenging in novel ways and opens interesting avenues for future research. The stochastic durations lead to an explosion in the *space of possible decision-epochs*, which exacerbates the familiar challenge of growth in executable action combinations caused by concurrency. The rate of decision-epoch growth increases with greater expressiveness in the action language, and we characterise the challenges along several dimensions, *e.g.*, the possible times for which action preconditions and effects may be specified, duration distributions of actions.

Expressiveness of Action Models

The action models handled by different temporal planners vary in complexity. Figure 1 lists different representations along two dimensions (ignoring continuous change). The simplest temporal model is used in TGP (Smith & Weld 1999). *TGP-style* actions require preconditions to be true throughout execution; the effects are guaranteed to be true only after termination; and actions may not execute concurrently if they clobber each other's preconditions or effects.

	Simple	Boundary	Metric
Deterministic duration	TGP	$\text{PDDL}_{2.1}$	Zeno
Prob. but independent	Prob. TGP	Prob. $\text{PDDL}_{2.1}$	
Joint distrib: dur ×effects			Prottle

Figure 1: *Action models* for temporal planning (ignoring continuous change). The horizontal axis varies the times at which preconditions and effects may be specified. The vertical axis varies the uncertainty in effects and its correlations with durations.

Along the horizontal axis, we vary the temporal expressiveness in the precondition and effect representations. $\text{PDDL}_{2.1}$ (Fox & Long 2003) is more expressive than TGP's representation as it can represent preconditions that are required to be true just at start, over whole action execution or just at the end. Where $\text{PDDL}_{2.1}$ allows effects to appear only at boundaries, Zeno's representation (Penberthy & Weld 1994) allows effects (preconditions) to appear at arbitrary intermediate points (and intervals).

Along the vertical axis, we vary the representation of uncertainty in the model. $\text{PDDL}_{2.1}$ doesn't support probabilistic action effects or durations. "Probabilistic $\text{PDDL}_{2.1}$" extends $\text{PDDL}_{2.1}$ along this direction, associating a distribution with each action duration; the distribution for durations is independent of that for effects. "Probabilistic TGP" extends the TGP action representation similarly. Even more expressive representations may use a single joint distribution — enabling action durations that are correlated with effects. Indeed, the representation language of Prottle (Little, Aberdeen, & Thiebaux 2005) contains all these features: effects at intermediate points, action durations correlated with probabilistic effects. Tempastic (Younes & Simmons 2004) uses probabilistic TGP-style actions, but because it also supports exogenous events, it is at least as expressive as Prottle. The blank entries in Figure 1 denote action languages that have not yet been discussed in the literature.

Planning with TGP-style Actions

We first study *TGP-style* actions in the context of uncertain durations (Smith & Weld 1999). TGP-style actions require preconditions to be true throughout execution; the effects are guaranteed to be true only after termination; and actions may not execute concurrently if they clobber each other's preconditions or effects.

We find that planning, even with these simplified action models, suffers significant computational blowup. All the

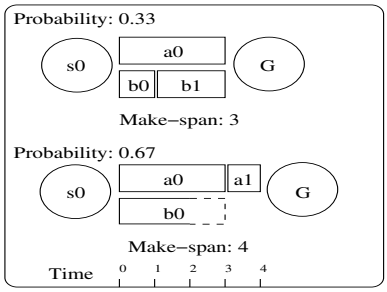


Figure 2: Planning with expected durations leads to a sub-optimal solution.

examples in this section apply regardless of whether effects are deterministic or stochastic. We investigate extensions to richer representations in the next sections.

We focus on problems whose objective is to achieve a goal state, while minimising total expected time (*make-span*), but our observations apply to cost functions that combine make-span and resource usage. This raises the question of *when* a goal counts as achieved. We require that all executing actions terminate before the goal is considered achieved.

A naive way to solve our problem is by ignoring duration distributions. We can assign each action a constant duration equal to the mean of its distribution, and then apply a deterministic-duration planner such as that of Mausam and Weld (2005). Unfortunately, this method may not produce an optimal policy, as the following example illustrates.

Example: Consider the planning domain in Figure 2, in which the goal can be reached in two independent ways — executing the plan $\langle a_0; a_1\rangle$, *i.e.*, a_0 followed by a_1, or the plan $\langle b_0; b_1\rangle$. Let a_0, a_1 and b_1 have constant durations 3, 1, and 2 respectively. Let b_0 have a uniform distribution between lengths 1, 2 and 3. It is clear that if we disregard b_0's duration distribution and replace it by the mean 2, then both these plans have an expected cost of 4. However, the truly optimal plan has duration 3.67 — start both a_0 and b_0; if b_0 finishes at time 1 (prob. 0.33) then start b_1, else (prob. 0.67) wait until a_0 finishes and execute a_1 to reach the goal. In this policy, the expected cost to reach the goal is 0.33×3 + 0.67×4 = 3.67. Thus for optimal solutions, we need to explicitly take duration uncertainty into account. □

Definition *Any time point when a new action is allowed to start execution is called a* decision epoch. *A* happening *is either 0 or a time when an action actually terminates.*

For TGP or probabilistic TGP-style actions with deterministic durations, restricting decision epochs to happenings suffices for optimal planning (Mausam & Weld 2005). Unfortunately, the same is not true for problems with duration uncertainty.

Temporal planners may be classified as having one of two architectures: constraint-posting approaches, in which the times of action execution are gradually constrained during planning (*e.g.*, Zeno and LPG (Penberthy & Weld 1994; Gerevini & Serina 2002)), and extended-state-space methods (*e.g.*, TP4 and SAPA (Haslum & Geffner 2001; Do & Kambhampati 2001)). The following example has important computational implications for state-space planners, because limiting attention to a subset of decision epochs can speed these planners.

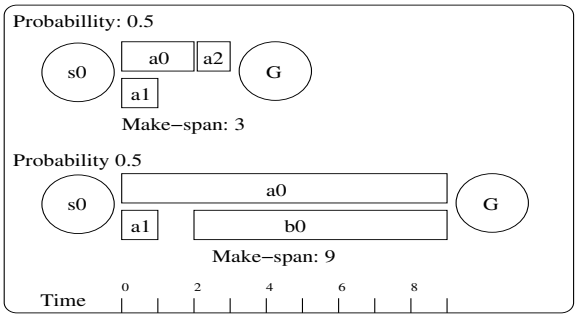


Figure 3: Intermediate decision epochs are necessary for optimal planning.

Example: Consider Figure 3, in which the goal can be reached in two independent ways — executing both $\{a_0, a_1\}$ followed by a_2 (*i.e.* effects of both a_0 and a_1 are preconditions to a_2); or by executing action b_0. Let a_1, a_2, and b_0 have constant durations 1, 1, and 7 respectively. Suppose that a_0 finishes in 2 time units with 0.5 probability and in 9 units the other half of the time. Furthermore, b_0 is mutex with a_1, but no other pairs of actions are mutex.

In such a domain, following the first plan, *i.e.*, $\langle\{a_0, a_1\}; a_2\rangle$, gives an expected cost of $6.5 = 0.5 \times 2 + 0.5 \times 9 + 1$. The second plan ($\langle b_0\rangle$) costs 7. The optimal solution, however, is to first start both a_0 and a_1 concurrently. When a_1 finishes at time 1, wait until time 2. If a_0 finishes, then follow it with a_2 (total length 3). If at time 2, a_0 doesn't finish, start b_0 (total length 9). The expected cost of this policy is $6 = 0.5 \times 3 + 0.5 \times 9$. □

Notice above that the optimal policy needs to start action b_0 at time 2, even when there is no happening at 2. Thus limiting the set of decision epochs to happenings does not suffice for optimal planning with uncertain durations. It is quite unfortunate that non-happenings are potentially necessary as decision epochs, because even if one assumes that time is discrete, there are many interior points during a long-running action; must a planner consider them all?

Definition *An action has* independent duration *if there is no correlation between its probabilistic effects and its duration. An action has* monotonic continuation *if the expected time until action termination is nonincreasing during execution.*

Actions without probabilistic effects have independent duration. Actions with monotonic continuations are common, *e.g.*, those with uniform, exponential, Gaussian, and many other duration distributions. However, actions with bimodal or multi-modal distributions don't have monotonic continuations.

We believe that if all actions have independent duration and monotonic continuation, then the set of decision epochs may be restricted to happenings without sacrificing optimality; this idea can be exploited to build a fast planner (Mausam & Weld 2006).

Timing Preconditions & Effects

Many domains require more flexibility concerning the times when preconditions and effects are in force: different effects of actions may apply at different times within the action's

```
:action a
   :duration 4
   :condition (over all P) (at end Q)
   :effect (at end Goal)
:action b
   :duration 2
   :effect (at start Q) (at end (not P))
```

Figure 4: A domain to illustrate that an expressive action model may require arbitrary decision epochs for a solution. In this example, b needs to start at 3 units after a's execution to reach $Goal$.

execution, preconditions may be required only to hold for part of execution, and executing two actions concurrently might lead to different results than executing them sequentially. Note that the decision epoch explosion is even more pronounced for such problems. Moreover, this not only affects optimality, but also affects the completeness of the algorithms. The following example with *deterministic* durations demonstrates this further.

Example: Consider the deterministic temporal planning domain in Figure 4 that uses PDDL$_{2.1}$ notation (Fox & Long 2003). If the initial state is P=true and Q=false, then the only way to reach $Goal$ is to start a at time 0, and b at time 3. Clearly, no action could terminate at 3, still it is a necessary decision epoch. □

Intuitively, two actions may require a certain relative alignment within them to achieve the goal. This alignment may force an action to start somewhere in the midst of the other's execution thus requiring a lot of decision epochs to be considered.

This example clearly shows that additional complexity in planning is incurred due to a more expressive action representation. It has important repercussions on existing planners. For instance, popular planners like SAPA and Prottle (Little, Aberdeen, & Thiebaux 2005) will not be able to solve this simple problem, because they consider only a restricted set of decision epochs. This shows that both these planners are incomplete (*i.e.*, problems may be incorrectly deemed unsolvable). Indeed, these planners can be naively modified by considering each time point as a decision epoch to obtain a complete algorithm. Unfortunately, such a modification is bound to be ineffective in scaling to any reasonable sized problem. Intelligent sampling of decision epochs is, thus, the key to finding a good balance between the two. Finding the exact modalities of such an algorithm is an important open research problem.

Continuous Action Durations

Previously, we assumed that an action's possible durations are taken from a discrete set. We now investigate the effects of dealing directly with continuous uncertainty. Let $f_i^T(t)dt$ be the probability of action a_i completing between times $t+T$ and $t+T+dt$, if we know that action a_i did not finish until time T. Similarly, define $F_i^T(t)$ to be the probability of the action finishing *after* time $t+T$.

Example: Consider the extended state $\langle X, \{(a_1, T)\}\rangle$, which denotes that action a_1 started T units ago in the world state X. Let a_2 be an applicable action that is started in this extended state. Define $M = \min(\Delta_M(a_1) - T, \Delta_M(a_2))$, where Δ_M denotes the maximum possible duration of execution for each action. Intuitively, M is the time by which at least one action will complete. Also, let J_n and Q_n denote the n^{th} revision to the expected cost to reach a goal starting from a state or a state-action pair respectively (Mausam & Weld 2005). Q_n may be computed as follows:

$$Q_{n+1}\left(\langle X, \{(a_1, T)\}\rangle, a_2\right) = \int_0^M f_1^T(t) F_2^0(t) \left[t + J_n\left(\langle X_1, \{a_2, t\}\rangle\right)\right] dt + \int_0^M F_1^T(t) f_2^0(t) \left[t + J_n\left(\langle X_2, \{a_1, t+T\}\rangle\right)\right] dt \quad (1)$$

Here X_1 and X_2 are world states obtained by applying the deterministic actions a_1 and a_2 respectively on X. Recall that $J_{n+1}(s) = \min_a Q_{n+1}(s, a)$. For a fixed point computation of this form, we desire that J_{n+1} and J_n have the same functional form[1]. Going by the equation above this seems very difficult to achieve, except perhaps for very specific action distributions in some special planning problems. For example, if all distributions are constant or if there is no concurrency in the domain, then these equations are easily solvable. But for anything mildly interesting, solving these equations is a challenging open question.

Non-Monotonic Duration Distributions

Dealing with continuous multi-modal distributions worsens the decision epochs explosion. We illustrate this below.

Example: Consider the domain of Figure 3 except that let action a_0 have a bi-modal distribution, the two modes being uniform between 0-1 and 9-10 respectively as shown in Figure 5(a). Also let a_1 have a very small duration. Figure 5(b) shows the expected remaining termination times if a_0 terminates at time 10. Notice that due to bi-modality, this time increases between 0 and 1. The expected time to reach the goal using plan $\langle\{a_0, a_1\}; a_2\rangle$ is shown in the third graph.

Now suppose that, we have started $\{a_0, a_1\}$, and we need to choose the next decision epoch. It is easy to see that the optimal decision epoch could be any point between 0 and 1 and would depend on the alternative routes to the goal. *E.g.*, if duration of b_0 is 7.75, then the optimal time-point to start the alternative route is 0.5 (right after the expected time to reach the goal using first plan exceeds 7.75). □

We have shown that the choice of decision epochs depends on the expected durations of the alternative routes. But these values are not known in advance, in fact these are the ones being calculated in the planning phase. Therefore, choosing decision epochs ahead of time does not seem possible. This makes the optimal continuous multi-modal distribution planning problem mostly intractable for any reasonable sized problem.

Correlated Durations and Effects

When actions' durations are correlated with the effects, then failure to terminate provides additional information regarding an action's effects. For example, non-termination at a

[1]This idea has been exploited in order to plan with continuous resources (Feng *et al.* 2004).

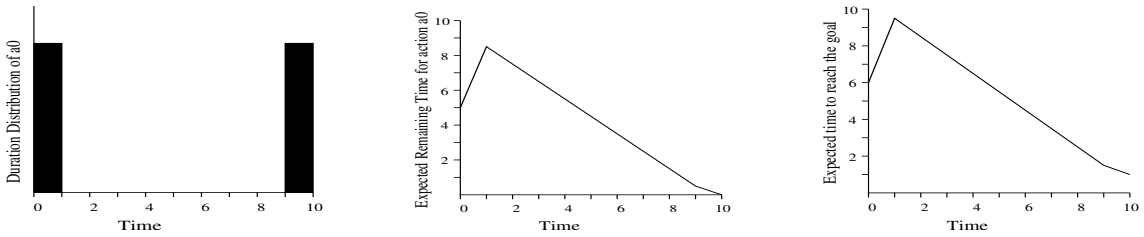


Figure 5: If durations are continuous (real-valued) rather than discrete, there may be an infinite number of potentially important decision epochs. In this domain, a crucial decision epoch could be required at any time in $(0, 1]$ — depending on the length of possible alternate plans.

point may change the probability of the action's eventual effects, and this may prompt new actions to be started. Thus, these points need to be considered for decision epochs, and cannot be omitted, even with TGP-style actions.

Notion of Goal Satisfaction

Different problems may require slightly different notions of when a goal is reached. For example, we have assumed thus far that a goal is not "officially achieved" until all executed actions have terminated. Alternatively, one might consider a goal to be achieved if a satisfactory world state is reached, even though some actions may be in the midst of execution. There are intermediate possibilities in which a goal requires some *specific* actions to necessarily end.

Interruptible Actions

We have assumed that, once started, an action cannot be terminated. However, a richer model may allow preemptions, as well as the continuation of an interrupted action. The problems in which all actions could be interrupted at will have a significantly different flavour. To a large extent, planning with such actions is similar to finding different concurrent paths to the goal and starting all of them together, since one can always interrupt all the executing paths as soon as the goal is reached. For instance, example in Figure 3 no longer holds, since b_0 can be started at time 1, and later terminated as needed to shorten the make-span.

Conclusions

This paper investigates planning problems with concurrent actions having stochastic durations, focussed primarily on extended-state-space planners. We identify the explosion in the number of decision epochs as the main cause of computational blowup. No longer can a planner limit action-initiation times to points when a different action has terminated. The rate of decision-epoch growth increases with greater expressiveness in the action language, and we characterise the challenges along several dimensions.

Even with simple probabilistic TGP-style actions, many more decision epochs must be considered to achieve optimality. However, if all durations are unimodal and uncorrelated with effects, we conjecture that one can bound the decision epochs in terms of times of action terminations.

We show that for $\text{PDDL}_{2.1}$ and richer action representations, the currently employed extended state space based methods are incomplete, and the straightforward ways to ensure completeness are highly inefficient. Developing an algorithm that achieves the best of both worlds is an important research question.

Additionally, we discuss the challenges posed by continuous time, observing that techniques employing piecewise constant/linear representations, which are popular in dealing with functions involving continuous variables, may be ineffective for our problem. These techniques rely on the same functional forms for successive approximations of the value function — and this does not hold in our case. Other potent directions for future research include multi-modal distributions, interruptibility, and correlated durations and effects. We develop algorithms to handle some of these issues in (Mausam & Weld 2006).

Acknowledgments

We thank Sumit Sanghai for theorem proving skills and advice. We also thank Jiun-Hung Chen, Nilesh Dalvi, Maria Fox, Jeremy Frank, Subbarao Kambhampati, and Håkan Younes for helpful suggestions. Raphael Hoffman, Daniel Lowd, Tina Loucks, Alicen Smith and the anonymous reviewers gave useful comments on prior drafts. This work was supported by NSF grant IIS-0307906, ONR grants N00014-02-1-0932, N00014-06-1-0147 and the WRF / TJ Cable Professorship.

References

Do, M. B., and Kambhampati, S. 2001. Sapa: A domain-independent heuristic metric temporal planner. In *ECP'01*.

Feng, Z.; Dearden, R.; Meuleau, N.; and Washington, R. 2004. Dynamic programming for structured continuous Markov decision processes. In *UAI'04*, 154.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR Special Issue on 3rd International Planning Competition* 20:61–124.

Gerevini, A., and Serina, I. 2002. LPG: A planner based on local search for planning graphs with action graphs. In *AIPS'02*, 281.

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.

Little, I.; Aberdeen, D.; and Thiebaux, S. 2005. Prottle: A probabilistic temporal planner. In *AAAI'05*.

Mausam, and Weld, D. 2005. Concurrent probabilistic temporal planning. In *ICAPS'05*, 120–129.

Mausam, and Weld, D. 2006. Probabilistic temporal planning with uncertain durations. In *AAAI'06*.

Penberthy, J., and Weld, D. 1994. Temporal planning with continuous change. In *AAAI'94*, 1010.

Smith, D., and Weld, D. 1999. Temporal graphplan with mutual exclusion reasoning. In *IJCAI'99*, 326–333.

Younes, H. L. S., and Simmons, R. G. 2004. Policy generation for continuous-time stochastic domains with concurrency. In *ICAPS'04*.

Efficient and Expressive Extensions of Constraint-Based Temporal Reasoning

Michael D. Moffitt
Department of Electrical Engineering and Computer Science
University of Michigan
mmoffitt@eecs.umich.edu

Abstract

In this extended abstract, I present a brief overview of several proposed extensions to the field of constraint-based temporal reasoning. Combined, these extensions allow one to reason efficiently and simultaneously about overconstrained problems, preferences, finite-domain constraints, and uncertain situations within the context of Temporal CSPs. I also describe a particularly exciting application of these techniques to an area of research outside the usual scope of temporal reasoning.

Introduction

In the field of artificial intelligence, a great deal of effort has been extended toward improving existing methods for *temporal constraint satisfaction*. Temporal Constraint Satisfaction Problems (TCSPs) (Dechter *et al.* 1991) are especially suited to express constraints regarding the time, order, and duration of events, and as a result, it is common to find TCSPs applied in problems relating to planning and scheduling. Recent work has begun to extend the TCSP to handle uncontrollable events (Vidal & Fargier 1999; Morris & Muscettola 2005; Venable & Yorke-Smith 2005) and preferences (Khatib *et al.* 2001; Peintner & Pollack 2004).

However, there are certain situations where existing temporal representations and reasoning systems remain inadequate. First, it may be the case that a given TCSP is overconstrained, and thus admits no solution. If one instead desires a partial solution, where as many constraints can be satisfied as possible, traditional DTP solving algorithms are insufficient. Second, there are some scenarios in which the constraints of a given CSP contain a mixture of both finite-domain and temporal constraints. The problem of constructing such hybrid representations and algorithms has, until recently, been largely overlooked. Finally, there may be cases where the constraints of the problem are themselves uncertain. When such decisions lie outside the control of the constraint engine, it may be valuable to model the manner in which this information becomes known in an online environment, or to efficiently precompute a set of potential solutions in advance.

The objective of my thesis is to extend the particularly expressive Disjunctive Temporal Problem (DTP) along with traditional meta-CSP algorithms in order to cope with overconstrained problems, preferences, finite-domain constraints, and uncertain situations. To achieve these goals while maintaining efficiency requires both the creation of novel methods as well as the integration of well-established techniques that have proven effective in prior literature. An additional goal is to expand the range of applications to which TCSPs can be applied, demonstrating their usefulness outside the typical planning and scheduling domains that have been the focus of previous work.

Background

A *Disjunctive Temporal Problem* (DTP) (Stergiou & Koubarakis 1998) is a type of TCSP defined by a pair $\langle X, C \rangle$, where each element $X_i \in X$ designates a time point, and each element $C_i \in C$ is a constraint of the form:

$$c_{i1} \vee c_{i2} \vee ... \vee c_{in}$$

where in turn, each c_{ij} is of the form:

$$a_{ij} \leq x_{ij} - y_{ij} \leq b_{ij}$$

with $x_{ij}, y_{ij} \in X$ and $a_{ij}, b_{ij} \in \Re$. A solution to a DTP can be defined in one of two ways. The first of these is as an object-level assignment of a numeric value to each of the time points in X, such that all the constraints in C are satisfied. A second type of solution is a *meta-CSP* assignment. Here, instead of directly considering assignments to the time points in X, a meta-variable C_i is created for each constraint in the DTP. The domain $D(C_i)$ is simply the set $\{c_{i1}, c_{i2}, ..., c_{in}\}$, representing the various disjuncts one can choose to satisfy that disjunctive constraint. A meta-CSP solution is thus a selection of a single disjunct for each meta-variable such that the resulting set of inequalities is consistent.

Temporal Constraint Relaxation

A significant portion of my thesis deals with the problem of constraint relaxation in Disjunctive Temporal Problems. In this section I describe both systematic and approximate methods for handling overconstrained problems, and suggest how these can be applied to the more interesting issue of temporal preference optimization.

```
Partially-Solve-DTP(A, U, cost, upperbound)
If (cost ≥ upperbound) return
If (U = ∅)
    best-solution-so-far ← A
    upperbound ← cost
    return
EndIf
C_i ← select-variable(U), U' ← U − {C_i}
For each disjunct c_ij of D(C_i)
    A' ← A ∪ {C_i ← c_ij}
    If (consistent(A'))
        Partially-Solve-DTP(A', U', cost, upperbound)
    EndIf
EndFor
A' ← A ∪ {C_i ← ε}
Partially-Solve-DTP(A', U', cost + 1, upperbound)
```

Figure 1: A PCS algorithm for DTPs

Partial Constraint Satisfaction of DTPs

Existing packages for solving DTPs, such as Epilitis (Tsamardinos & Pollack 2003) and TSAT++ (Armando *et al.* 2004), are sufficient for problems that admit one or more consistent solutions. However, in the event that a given DTP is overconstrained, these solvers are unable to provide anything other than a notice of failure. In some situations, one may instead desire a partial solution, in which as many constraints are satisfied as possible.

My thesis work extends traditional meta-CSP-based search algorithms in order to achieve *partial constraint satisfaction* of DTPs (Moffitt & Pollack 2005a). To accomplish this, the domain of each meta-variable (or constraint) in the DTP must be implicitly augmented with an empty disjunct, labeled 'ϵ'. This mechanism allows constraints to be violated explicitly during the meta-CSP search, a nuance that sets the algorithm apart from previous applications of partial constraint satisfaction to classical CSPs (Freuder & Wallace 1992). A solver, named Maxilitis, applies a branch-and-bound search (outlined in Figure 1) to minimize the total number of so-called ϵ*-relaxations*.

Applying Local Search to DTPs

One drawback to the systematic algorithm is that it can become rather expensive for extremely overconstrained problems that require a large number of constraint violations. Although Maxilitis has the anytime property (meaning that it can be interrupted at any time to extract a suboptimal solution), one may wonder whether there are more efficient ways of obtaining such solutions.

To address this question, my thesis work includes an application of local search to overconstrained temporal problems (Moffitt & Pollack 2005b). In contrast to previous work on DTPs, the approach works within a total assignment space at the object-level, and thus abandons the meta-CSP and corresponding graph-based consistency algorithms that have been employed in prior DTP literature. This particular search space presents several interesting challenges, such as the presence of infinitely many neighbors at each search node.

Revisiting Temporal Preference Optimization

One of the more active subjects in recent TCSP literature is the problem of *preferential optimization*. In this line of research, traditional temporal constraints (Dechter *et al.* 1991) are augmented with local preference functions that express how well a particular assignment satisfies the corresponding constraint. Early versions of this research focused on the problem of maximizing the minimum such preference value (Khatib *et al.* 2003; Peintner & Pollack 2004), although later developments have begun to address the more challenging problem of utilitarian optimization (Morris *et al.* 2004), where the sum of the individual preference values is maximized. Unfortunately, existing CSP-based methods for this objective (Peintner & Pollack 2005) have been shown to suffer in performance compared to more general SAT-based approaches (Sheini *et al.* 2005).

My thesis work explores a new means of obtaining utilitarian optimal solutions to Disjunctive Temporal Problems with Preferences (DTPPs) (Moffitt & Pollack 2006a). I depart from the SAT encoding and instead introduce the Valued DTP (VDTP). In contrast to the traditional semiring-based formalism (Bistarelli, Montanari, & Rossi 1997) that annotates legal object-level tuples of a constraint with preferences, the framework I develop instead assigns elementary costs to the constraints themselves, as is commonly done in finite-domain Valued CSP literature (Schiex *et al.* 1995). While this reformulation provides no increase in expressive power, it simplifies some of the computational difficulties related to temporal optimization, since (as mentioned earlier) search strategies for DTP solving rarely invoke object-level assignments directly. After proving that the VDTP can express the same set of utilitarian optimal solutions as the DTPP with piecewise-constant preference functions, I develop a method for achieving *weighted constraint satisfaction* within the meta-CSP search space that has traditionally been used to solve DTPs without preferences. This allows the application of well-established strategies that have proven effective in previous literature on both temporal reasoning and constraint optimization. As shown in Figure 2, empirical results suggest that an implementation of this approach (named WEIGHTWATCHER) consistently outperforms prior DTPP solvers – including GAPD (Peintner 2005) and the SAT-based solver ARIO (Sheini *et al.* 2005) – by several orders of magnitude.

Temporal/Finite-Domain Hybrid CSPs

There are some cases where the constraints of a given problem contain a mixture of both finite-domain and temporal components. For instance, consider the task of scheduling a set of meetings, where each meeting must be held in one of finitely-many locations. Temporal CSPs can quite easily capture temporal aspects of the problem such as start and end times, but a finite-domain network may be needed to reason about the locations. If these separate constraint networks exhibit any degree of interaction (e.g., if the physical locations of two meetings have an effect on their pairwise temporal relationship), then some kind of hybrid approach is required.

My thesis work considers the problem of constructing a

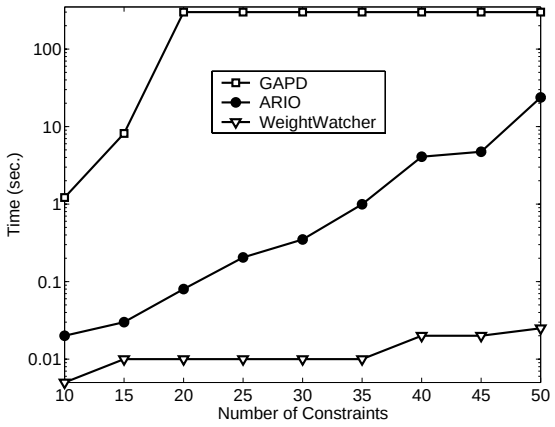


Figure 2: Median running times for GAPD, ARIO, and WEIGHTWATCHER for DTPPs of varying sizes (Timeout set at 300s)

hybrid constraint system capable of managing both finite-domain CSPs and temporal constraints (Moffitt *et al.* 2005), an endeavor that poses two formidable challenges. The first of these is that a more flexible representation is required to express both the individual constraint systems and their interaction. The second is that a new algorithm for establishing consistency of the hybrid problem is needed. In particular, I am in the process of developing a *least-commitment* algorithm especially suited for cases in which the finite-domain network is large but relatively underconstrained.

Dealing with Uncertainty

Uncertainty is a common element in many real-world scenarios. Within the context of temporal reasoning, prior work on uncertainty has focused on the presence of uncontrollable events (Vidal & Fargier 1999; Morris & Muscettola 2005; Venable & Yorke-Smith 2005), where the values of some subset of time points are decided on by nature. The problem is then no longer one of consistency, but rather one of controllability.

My thesis work examines a different dimension of uncertainty; specifically, how to deal with situations in which the constraints of the problem are themselves uncertain. For instance, if it is unknown whether a pair of activities must share the same resource, there may or may not exist a non-overlap constraint between them. Such uncertainty could exist even if the the object-level temporal variables are themselves fully controllable. In the presence of such uncertainty, several options are available. First, one can attempt to model the manner in which these constraints become known in an online environment, allowing the various notions of controllability to be generalized. Second, one can alternatively reason about the possible individual *realizations* of the problem, where each of these corresponds to a single DTP whose constraints are fixed. As an example, a precomputed set of potential solutions to the original problem, known as a *covering set closure* in classical CSP literature (Yorke-Smith & Gervet 2003), could be constructed. This might be done in a sequential enumerative fashion; or, since these realizations will likely share a significant amount of structure, some kind of parallelized approach may prove more efficient. Optimization variants exist as well, where a single solution is generated that maximizes the likelihood of feasibility. This work is largely in development, and I am still in the process of comparing my approaches to a wide body of related literature.

An Application to Optimal Rectangle Packing

So far, we have explored ways in which temporal representations and reasoning methods can be extended in order to handle overconstrained problems, preferences, finite-domain constraints, and uncertainty with respect to the constraints of the problem. While no single application has been used exclusively to motivate these extensions, one can imagine how the domains of planning and scheduling would benefit most directly, as they are popular areas to which temporal reasoning has traditionally been applied.

However, there are other problems that have attracted recent interest where TCSP techniques have yet to be considered. For instance, consider the topic of rectangle packing, a problem that has drawn attention from several diverse fields of computer science (e.g., VLSI/CAD) in addition to some areas of operations research. The current state-of-the-art (Korf 2003; 2004) has cast optimal rectangle packing as a CSP in which a variable is created for each rectangle, whose legal values are the positions that rectangle could occupy without exceeding the boundaries of the enclosing space. In addition, there is a binary constraint between each pair of rectangles, requiring that they do not overlap. To solve this CSP, Korf developed a backtracking algorithm, where each partial assignment is defined to be the fixed placement of a subset of rectangles. By obtaining lower bounds on the amount of wasted space at each node in the search, an algorithm was constructed that is the fastest known for optimal rectangle packing.

My thesis work addresses the problem of optimal rectangle packing (Moffitt & Pollack 2006b) in a way that departs from the aforementioned search space. Specifically, I cast the problem of optimally packing a set of rectangles with fixed orientations as a meta-CSP, in which a meta-variable is created for each pair of rectangles, whose values are the four pairwise relationships (i.e., *above*, *below*, *left of*, *right of*) that prevent that pair from overlapping. As such, commitment to the exact placement of any rectangle is not established until a consistent solution has been generated. I show how to apply several powerful DTP-solving techniques to this problem, and also develop a suite of new methods that exploit both the symmetry and geometry present in this particular domain. Despite its many differences with the fixed-placement formulation, the meta-CSP algorithm is shown to be quite competitive in performance, as evidenced in Figure 3 on a set of benchmarks fully explained in (Moffitt & Pollack 2006b).

Motivation for Other Extensions

The domain of rectangle packing proves to be an extremely interesting application of not only existing TCSP methods,

N	Opt. Dimen.	Korf '04	Moffitt '06
14	23×45	0	0
15	23×55	1	1
16	27×56	2	3
17	39×46	10	10
18	31×69	1:08	1:29
19	47×53	8:15	4:11
20	34×85	13:32	15:03
21	38×88	1:35:08	1:32:01
22	39×98	6:46:15	4:51:23
23	64×68	36:54:50	29:03:49
24	56×88	213:33:00	146:38:48

Figure 3: Experimental results for minimum-area rectangles than contain all consecutive squares from 1×1 up to $N\times N$. Runtime is reported in hours, minutes, and seconds.

but also of the other extensions proposed in this thesis. For instance, although the current formulation cannot represent rotatable rectangles, these could be handled by encoding the rectangles' orientations as finite-domain variables, and exploiting the hybrid representation discussed earlier. In addition, I have collaborated with researchers in VLSI to develop the *floorplan repair* problem, and proposed a means to solve it using a variation of the ϵ-relaxation (Moffitt *et al.* 2006).

Integration of Techniques

Within each of the extensions described in this paper, there are a number of issues that remain to be addressed. However, one of the more challenging tasks is to combine these extensions into a single, unified framework that elegantly integrates all techniques. The development of a unified framework is crucial for handling complicated real-world scenarios, such as calendar management and meeting scheduling, that require each of these extensions to some degree. A system based on this framework will be implemented and compared in both design and performance to other related recent developments.

References

Armando, A.; Castellini, C.; Giunchiglia, E.; and Maratea, M. 2004. A SAT-based decision procedure for the boolean combination of difference constraints. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT-2004)*.

Bistarelli, S.; Montanari, U.; and Rossi, F. 1997. Semiring-based constraint satisfaction and optimization. *Journal of the ACM* 44(2):201–236.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1-3):61–95.

Freuder, E. C., and Wallace, R. J. 1992. Partial constraint satisfaction. *Artificial Intelligence* 58(1-3):21–70.

Khatib, L.; Morris, P.; Morris, R.; and Rossi, F. 2001. Temporal constraint reasoning with preferences. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-2001)*, 322–327.

Khatib, L.; Morris, P.; Morris, R.; and Venable, K. B. 2003. Tractable Pareto optimal optimization of temporal preferences. In *Proceedings of the 18th International Joint Conference on Artificial Intelligece (IJCAI-2003)*, 1289–1294.

Korf, R. E. 2003. Optimal rectangle packing: Initial results. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS-2003)*, 287–295.

Korf, R. E. 2004. Optimal rectangle packing: New results. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-2004)*, 142–149.

Moffitt, M. D.; Ng, A. N.; Markov, I. L.; and Pollack, M. E. 2006. Constraint-Driven Floorplan Repair. To appear in *Proceedings of the 43rd Design Automation Conference (DAC-2006)*.

Moffitt, M. D.; Peintner, B.; and Pollack, M. E. 2005. Augmenting disjunctive temporal problems with finite-domain constraints. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005)*, 1187–1192.

Moffitt, M. D., and Pollack, M. E. 2005a. Partial constraint satisfaction of disjunctive temporal problems. In *Proceedings of the 18th International Florida Artificial Intelligence Research Society Conference (FLAIRS-2005)*, 715–720.

Moffitt, M. D., and Pollack, M. E. 2005b. Applying local search to disjunctive temporal problems. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-2005)*, 242–247.

Moffitt, M. D., and Pollack, M. E. 2006a. Temporal preference optimization as weighted constraint satisfaction. To appear in *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-2006)*.

Moffitt, M. D., and Pollack, M. E. 2006b. Optimal rectangle packing: a meta-CSP approach. To appear in *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS-2006)*.

Morris, P. H., and Muscettola, N. 2005. Temporal dynamic controllability revisited. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005)*, 1193–1198.

Morris, P.; Morris, R.; Khatib, L.; Ramakrishnan, S.; and Bachmann, A. 2004. Strategies for global optimization of temporal preferences. In *Proceedings of the 10th International Conference on Principles and Practices of Constraint Programming*, 408–422.

Peintner, B., and Pollack, M. E. 2004. Low-cost addition of preferences to DTPs and TCSPs. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-2004)*, 723–728.

Peintner, B., and Pollack, M. E. 2005. Anytime, complete algorithm for finding utilitarian optimal solutions to STPPs. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005)*, 443–448.

Peintner, B. M. 2005. *Algorithms For Constraint-Based Temporal Reasoning With Preferences*. Ph.D. Dissertation, University of Michigan.

Schiex, T.; Fargier, H.; and Verfaillie, G. 1995. Valued constraint satisfaction problems: hard and easy problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-1995)*, 631–639.

Sheini, H. M.; Peintner, B.; Sakallah, K. A.; and Pollack, M. E. 2005. On solving soft temporal constraints using SAT techniques. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP-2005)*, 607–621.

Stergiou, K., and Koubarakis, M. 1998. Backtracking algorithms for disjunctions of temporal constraints. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, 248–253.

Tsamardinos, I., and Pollack, M. E. 2003. Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence* 151(1-2):43–90.

Venable, K. B., and Yorke-Smith, N. 2005. Disjunctive temporal planning with uncertainty. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-2005)*, 1721–1722.

Vidal, T., and Fargier, H. 1999. Handling contingency in temporal constraint networks: From consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence* 11(1):23–45.

Yorke-Smith, N., and Gervet, C. 2003. Certainty closure: A framework for reliable constraint reasoning with uncertainty. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP-2003)*, 769–783.

Path Planning Under Uncertainty: Complexity and Algorithms (ICAPS Doctoral Consortium: Thesis Abstract)

Evdokia Nikolova
MIT Computer Science & AI Lab
Cambridge MA 02139, USA
enikolova@csail.mit.edu

Introduction

Finding shortest paths is a classic and fundamental problem in theoretical computer science which has influenced a wide array of other elds. Finding stochastic shortest paths has also been extensively studied though it has proven harder to formalize and yield classic results. What is the right way to de ne stochastic shortest paths, when we only know random distributions for the edge weights? Is it shortest paths on average, or shortest paths minimizing a combination of mean and variance, or minimizing some other speci ed criterion? Are they found adaptively or non-adaptively? A variety of problem variants have appeared in the literature, and most have ended up minimizing the expected length of paths, or a combination of expected lengths and expected costs such as bicriterion problems (Pallottino & Scutella 1997). Adaptive formulations have prevailed, perhaps because a non-adaptive minimization of the expected path length trivially reduces to the deterministic shortest path problem.

Few researchers have considered optimizing a nonlinear function of the path length. Some notable work includes that of Loui (Loui 1983) who de nes a decision-theoretic framework, where the optimal path maximizes the expected utility of the user for a class of monotonically increasing utility functions. Fan *et al.* (Fan, Kalaba & Moore) present an adaptive heuristic for paths that maximize the probability of arriving on time. Formulations of this type with nonlinear objective, though perhaps most useful in practice, have been sparse, because the hardness of the problem arises and accumulates from many levels: combinatorial, distributional, analytic, functional, to list a few. We elaborate on these sources below.

We focus on stochastic shortest paths models which can effectively factor the sources of dif culty and whose solution draws from a variety of areas underlying the problem. In addition our solutions contain techniques that may be useful in solving other combinatorial problems and more generally, a number of nonconvex optimization problems.

Related Work

A lot of the related work on shortest paths in stochastic networks has focused on the notion of shortest paths in expectation, e.g., (Bertsekas & Tsitsiklis 1991). Other models have added costs on the edges in addition to travel times (Chabini 2002), (Miller-Hooks & Mahmassani 2000) where the costs depend on the realized travel times and in this way can capture a measure of uncertainty.

Finding the path of smallest expected length trivially reduces to a deterministic shortest path problems and does not take into account risk in predicting the optimal route. Since most real world applications care about a tradeoff between risk and expectation, we consider nonlinear objectives that capture more information about the edge distributions. Closest to this model, Loui (Loui 1983) considered a decision analytic framework for optimal paths under uncertainty, however he only studied monotone increasing cost functions and his algorithm has running time $O(n^n)$ in the worst case. Mirchandani and Soroush (Mirchandani & Soroush 1985) extended his work to a quadratic cost function of the path length, however their algorithm is also an exhaustive search over all potentially optimal paths, and thus exponential in the worst case.

Another branch of the stochastic shortest path literature has focused on adaptive algorithms (Fan, Kalaba & Moore), (Gao & Chabini 2002), (Boyan & Mitzenmacher 2001), which compute the optimal next edge in light of lengths or travel times already realized en route to the current node. Another direction has been to give approximations and heuristics for expected shortest paths in stochastic networks with nonstationary (time-varying) edge length distributions (Miller-Hooks & Mahmassani 2000), (Fu & Rilett 1998), (Hall 1986), to list a few. In this proposal, we only consider stationary edge length distributions, that do not change with time; time-varying distributions will be the subject of future work.

Problem Statement

The offline stochastic shortest path problem takes as input a graph and independent probability distributions for all its edge weights. It asks for the optimal path between a given source and a destination, which minimizes the expectation of a speci ed objective function. The term offline is used to emphasize that we seek a nonadaptive algorithm for an optimal path, before we observe any of the realized edge weights. When the cost function is linear, the problem becomes equivalent to a deterministic shortest path with edge weights equal to the expectations of their corresponding random variables. Thus, the challenge is when the objective is non-linear, which is also the case that most often occurs in practical applications.

Notation. We denote the graph $G = (V, E)$, with $|V| = n$ and $|E| = m$. Let the source and destination be S and T respectively. Denote the random weight of edge e by X_e. The objective function is $C : \mathcal{R} \rightarrow \mathcal{R}$. Strictly speaking, $C(X)$ is a function of the random path length $X = \sum X_e$. Thus, our problem is to solve

$$\min_{\pi} \mathbf{E}[C(\sum_{e \in \pi} X_e)] \quad (1)$$

for paths π between the source and destination.

The meaning of a non-linear cost function of the path length is not as intuitive as the notion of penalty for being late. Thus, we provide an equivalent formulation of the objective function, by including the extra parameter t for the clock time relative to a deadline at time 0. The penalty for arriving at the destination at time t is $\tilde{C}(t)$ (t is negative for early arrivals and positive for late arrivals). The expected cost of a path is then $\int_0^\infty f(x)\tilde{C}(t+x)dx$ where $f(.)$ is the probability density of the length X of the path and t is the departure time. For a xed departure time t, the cost of the path is $C_t(X) = \tilde{C}(t+X)$, simply a horizontal shift by t units, and the minimization of its expectation over the set of paths is equivalent to the problem (1). When it is clear from the context, the parameter dependence $C_t(X)$ will be suppressed and we will write $C(X)$. This richer framework allows us to solve an additional problem: what is the optimal path and the optimal departure time t? This question is well de ned for non-monotone cost functions with a global minimum.

We sometimes distinguish the cost functions by calling $\tilde{C}(t)$ the penalty function (since it explicitly speci es a penalty for being late), and $\mathbf{E}[C(X)]$ the objective function.

Note that we may not expect to solve the problem (1) in full generality for several reasons.

- Combinatorial dif culty. Even in the absence of randomness, when the edge weights are fully deterministic, a wide class of cost functions reduce to nding the longest path in the graph, which is NP-hard and inapproximable within a reasonable factor (Karger, Motwani & Ramkumar 1997).
- Distributional dif culty. The distributional assumptions on the edge lengths may bring a dif culty on their own, to the extent that we cannot even compute the distribution of the total length of a path $X = \sum X_e$, let alone evaluate the function $\mathbf{E}[C(X)]$ and minimize it. For example, Kleinberg *et al.* show that computing the distribution of the sum of n non-identical Bernoulli random variables is #P-hard (Kleinberg, Rabani & Tardos).
- Analytic dif culty. Even with additive edge length distributions such as the Normal distribution, with which we can readily compute the sum $X = \sum X_e$, we might not be able to get a closed analytic form of the objective function $\mathbf{E}[C(X)] = \int f(x)C(x)dx$ and thus cannot optimize it ef ciently. This is a common problem in decision theory and related elds, which therefore focus attention on conjugate pairs of function and distribution families (more precisely, conjugate priors), *i.e.,* function-distribution pairs for which the integral can be computed in a closed form and the Expected Cost function lies in the same family as the original Cost function $C(X)$. For example, standard conjugate pairs are (Beta, Binomial) and (Gamma, Exponential).
- Functional dif culty. Having computed the distribution of the path length X and a closed form expression for the objective function $\mathbf{E}[C(X)]$, we are left with an integer optimization problem, to minimize a function over the collection of ST-paths of graph G. Relaxing the integer constraint, we have to optimize the function $\mathbf{E}[C(X)]$ over the path polytope in $\mathcal{R}^m$. The path polytope likely does not have any nice description with fewer than exponentially many linear constraints. Thanks to the separability of a linear objective into the graph edges, the deterministic shortest path problem has an ef cient combinatorial solution. However, other than the linear and exponential objectives, no other cost function is separable into the edges (Loui 1983) and thus we might not hope to nd exact optimal solutions in the general case. In special cases, convex and quasi-convex objective functions may admit greedy approaches that are equivalent to gradient descent on the path polytope, or they may admit ef cient enumeration of a small set of candidate paths, which would contain the optimum. Non-convex functions in the relaxed problem may achieve an optimum anywhere in the path polytope, and as there are no general ef cient methods for non-convex programming, it might not be tractable to nd the relaxed optimum, nor approximate the integer optimum.

In addition to looking for ef cient and approximation algorithms, we would like to understand the degree of

dif culty each factor above contributes with.

For the case of general objective, we prove hardness and inapproximability results for objectives with a global minimum. We then describe approximations based on a combination of problem substructure and discretization. This method applies to non-separable objectives which have a separable term, and the solution idea is similar to partial minimization of a multivariate function.

We also study several speci c but fundamental cost functions together with several different distributions, and offer hardness results, exact and approximation algorithms based on a variety of techniques.

References

Bertsekas, D. P. and Tsitsiklis, J. N. 1991. An analysis of stochastic shortest path problems. *Math. Oper. Res.* **16** (3): 580-595.

Boyan, J. and Mitzenmacher, M. 2001. Improved Results for Route Planning in Stochastic Transportation Networks. *ACM-SIAM Symposium on Discrete Algorithms*.

Bruno, J.L.; Downey, P.J.; and Fredrickson, G.N. 1981. Sequencing tasks with exponential service time to minimize the expected flowtime or makespan. *Journal of the ACM*, **28**: 100-113.

Chabini, I. 2002. Algorithms for k-Shortest Paths and Other Routing Problems in Time-Dependent Networks. *Transportation Research* Part B: Methodological.

Chvatal, V. 1980. Hard Knapsack Problems. *Operations Research*, **28**: 1402-1411.

Fan, Y.; Kalaba, R.; and Moore, II, J. E. Arriving on Time. *Journal of Optimization Theory and Applications*, forthcoming.

Fu, L. and Rilett, L. R. 1998. Expected Shortest Paths in Dynamic and Stochastic Traf c Networks. *Transportation Research*, Part B, **32**: 499-516.

Gao, S. and Chabini, I. 2002. Optimal Routing Policy Problems in Stochastic Time-Dependent Networks. *Proceedings of the IEEE 5th International Conference on Intelligent, Transportation Systems*, Singapore, pp. 549-559.

Hall, R. W. 1986. The Fastest Path Through a Network with Random Time-Dependent Travel Times, *Transportation Science*, **20**: 182-188.

Karger, D.; Motwani, R.; and Ramkumar, G. D. S. 1997. On approximating the longest path in a graph. *Algorithmica*, **18**: 82-98.

Kleinberg, J.; Rabani. Y.; and Tardos, E. 2000. Allocating Bandwidth for Bursty Connections *SIAM Journal on Computing*, **30**: 191-21.

Loui, R. P. 1983. Optimal Paths in Graphs with Stochastic or Multidimentional Weights. *Communications of the ACM*, **26**: 670-676.

Miller-Hooks, E. D. and Mahmassani, H. S. 2000. Least Expected Time Paths in Stochastic, Time-Varying Transportation Networks. *Transportation Science*, **34**: 198-215.

Mirchandani, P. and Soroush, H. 1985. Optimal paths in probabilistic networks: a case with temporary preferences. *Computers and Operations Research,* **12** (4): 365-381.

Nikolova, E. 2005. Stochastic Shortest Paths. Manuscript.

Pallottino, S. and Scutella M. G. 1997. Shortest Path Algorithms in Transportation Models: Classical and Innovative Aspects. TR-97-06. Universita di Pisa Dipartimento di Informatica, Pisa, Italy.

Papadimitriou, C.H. and Yannakakis, M. 1991. Shortest paths without a map. *Theoretical Computer Science* **84**: 127-150.

Papadimitriou, C. H. and Steiglitz, K. 1998. *Combinatorial optimization: Algorithms and complexity.* Dover Publications, Inc., N.Y.

Vazirani, V. V. 2001. *Approximation Algorithms.* Springer-Verlag Berlin Heidelberg.

Easy and Hard Conformant Planning

Héctor Palacios
Departamento de Tecnología
Universitat Pompeu Fabra
08003 Barcelona, SPAIN
hector.palacios@upf.edu

Advisor: Héctor Geffner
Departamento de Tecnología
ICREA & Universitat Pompeu Fabra
08003 Barcelona, SPAIN
hector.geffner@upf.edu

Abstract

Even under polynomial restrictions on plan length, conformant planning remains a very hard computational problem as plan verification itself can take exponential time. This heavy price cannot be avoided in general although in many cases conformant plans are verifiable efficiently by means of simple forms of disjunctive inference. We report an efficient but incomplete planner capable of solving non-trivial problems quickly. In this work, we show that this is possible by mapping conformant into classical problems that are then solved by an off-the-shelf classical planner. The formulation is sound as the classical plans obtained are all conformant, but it is incomplete as the inverse relation does not always hold. Atoms L/X_i that represent conditional beliefs 'if X_i then L' are introduced in the classical encoding and combined with suitable actions when certain invariants are verified. Empirical results over a wide variety of problems illustrate the power of the approach. We propose extensions to this formulation.

Introduction

Conformant planning is a form of planning where a goal is to be achieved when the initial situation is not fully known and actions may have non-deterministic effects (Goldman & Boddy 1996). Conformant planning is computationally harder than classical planning, as even under polynomial restrictions on plan length, plan verification remains hard (Turner 2002). This additional complexity cannot be avoided in general. This difference in complexity explains why it is still very easy to come up with simple and small conformant problems that no general domain-independent planner can solve, while the same is no longer true for classical planners. The main motivation of this work is to narrow this gap by developing an approach that targets 'simple' conformant problems effectively. The approach will not be complete but it will provide solutions to *some non-trivial conformant planning problems* by translating them intro *classical planning problems*(Palacios & Geffner 2006). New problems are fed into a classical planner. The translation is sound as the classical plans are all conformant, but it is incomplete as the converse does not always hold. The translation scheme accommodates 'reasoning by cases' by means of an 'split-protect-and-merge' strategy; namely, atoms L/X_i that represent conditional beliefs 'if X_i then L' are introduced in the classical encoding that are then combined by suitable actions when certain invariants in the plan are verified.

While several effective but incomplete formulations of conformant planning have been formulated before, like 0-approximation (Baral & Son 1997), none, as far as we know, can represent these types of plans, while those planners that can *represent* them (Cimatti, Roveri, & Bertoli 2004; Brafman & Hoffmann 2004), are not able to *compute* them except for very small problems.

Conformant Planning

For a conformant planning problem, if the number m of possible initial states $s_0 \in Init$ is bounded and actions are deterministic, the conformant planning problem P with a fixed horizon n can be mapped in the SAT problem over the formula (Palacios & Geffner 2005)

$$\bigwedge_{s_0 \in Init} T^{s0}(P, n) \qquad (1)$$

where if $T(P, n)$ is the propositional theory that encodes the problem P with horizon n. $T^{s0}(P, n)$ is $T(P, n)$ with two modifications: first, fluent literals L_0 (L at time 0) are replaced by true/false iff L is true/false in the (complete) state s_0, and second, fluent literals L_i, $i > 0$, are replaced by 'fresh' literals L_i^{s0}, one for each $s_0 \in Init$.

Eq. 1 can be thought as expressing m 'classical planning problems', one for each possible initial state $s_0 \in Init$, that are *coupled* in the sense that they all share the same set of actions; namely, the action variables are the only variables shared across the different subtheories $T^{s0}(P, n)$ for $s_0 \in Init$.

For bounded m, the resulting class of conformant planning problems with a fixed horizon can be mapped polynomially into SAT, generalizing the SAT encoding of classical planning problems which corresponds to $m = 1$ (Kautz & Selman 1996). Also, for a sufficiently large horizon, this formulation is *complete*. In other words, for this interesting class of problems, the formulation of Eq 1 takes advantage of the reduced complexity without restricting the inferences at all. However, expressivity and complexity, however, are not the only problems; *efficiency* or control is the other. A planner using Eq. 1 naively will not scale.

We have already proposed two approaches to optimal classical conformant planning based on logical formulations (Palacios *et al.* 2005; Palacios & Geffner 2005). Both of them translate the problem into CNF, and obtain a plan by

doing logical operations and search. The logical approach has been very importan on optimal classical planning (Kautz & Selman 1996), where they map it into SAT. In vplan (Palacios *et al.* 2005) we presented a complete optimal planner that rejects plan candidate by checking through model counting that it does not work for some initial state. In cf2sat (Palacios & Geffner 2005) (for *conformant2sat*) we construct a new propositional formula by doing logical operations as forgetting (Lin & Reiter 1994) and conditioning. The models of these new formula are all the possible plan. We feed that formula into a SAT solver to obtain a plan. Logical operations in both planners became feasible by compiling the propositional theory into d–DNNF (Darwiche 2002), a formal norm akin to OBDD. We obtained good results on some very complex domains but failed to scale in more simple problems.

One way to trade off completeness for efficiency in conformant planning results from approximating belief states (Bonet & Geffner 2000). For example, the 0-approximation introduced in (Baral & Son 1997) represents belief states bel by means of two sets: the set of literals that are true in bel, and the set of literals that are false in bel. Variables which do not appear in either set are unknown.

Conformant planning under the 0-approximation is thus no more complex, theoretically, than classical planning. The problem however is that the 0-approximation is strongly incomplete, as it does not capture any non-trivial form of disjunctive inference. For example, given a disjunction $p \vee q$ and an action a that maps either p or q into r, the semantics will not validate a as a conformant plan for r. Indeed, disjunctions that are not tautologies are thrown away.

Translation

The translation scheme maps a conformant planning problems P into a classical planning problems $K(P)$. We describe the contents of $K(P)$ in two parts, starting with the basic core $K_0(P)$. We assume that P is given by tuples of the form $\langle F, O, I, G\rangle$ where F stands for the fluent symbols in the problem, O stands for a set of actions a, I is a set of clauses over F defining the initial situation, and G is a set of literals over F defining the goal. In addition, every action a has a precondition given by a set of fluent literals, and a set of conditional effects $C \rightarrow L$ where C is a set of fluent literals and L is a literal. We assume that actions are all *deterministic* and hence that all uncertainty lies in the initial situation. We will usually refer to the conditional effects $C \rightarrow L$ of an action a as the *rules* associated with a, and sometimes write them as $a : C \rightarrow L$. Also, we use the expression $C \wedge X \rightarrow L$ to refer to rules with literal X in their bodies. In both cases, C may be empty. Last, when L is a literal, we take $\neg L$ to denote the complement of L.

Definition 1 (Core Translation) [1] *The* core translation *maps the conformant problem P into the classical problem $K_0(P) = \langle F', O', I', G'\rangle$ where*

- $F' = \{KL, K\neg L \mid L \in F\}$
- $I' = \{KL, \neg K\neg L \mid L \in I\} \cup \{\neg KL', \neg K\neg L' \mid L' \notin I\}$
- $G' = \{KL \mid L \in G\}$
- $O' = O$ *but with conditional effect* $a : C \rightarrow L$ *replaced by* $a : KC \rightarrow KL$ *and* $a : \neg K\neg C \rightarrow \neg K\neg L$.

For any literal L in P, KL denotes its 'epistemic' counterpart in $K_0(P)$ whose meaning is that L is known. We write KC for $C = L_1 \wedge L_2 \ldots$ as an abbreviation for $KL_1 \wedge KL_2 \ldots$, and $\neg K\neg C$ for $\neg K\neg L_1 \wedge \neg K\neg L_2 \ldots$.

The intuition behind the translation is simple: first, complementary literals L and $\neg L$ whose status is not known in the initial situation in P are 'negated', by mapping them into the literals $\neg KL$ and $\neg K\neg L$ that are jointly consistent. This mapping removes all uncertainty from $K_0(P)$. In addition, to ensure soundness, each conditional effect $a : C \rightarrow L$ in P maps, not only into the 'supporting' rule $a : KC \rightarrow KL$ but also into the 'cancellation' rule $a : \neg K\neg C \rightarrow \neg K\neg L$ that guarantees that literal $K\neg L$ is deleted (prevented to persist) when action a is applied except when C is known to be false.

We extend the translation further so that the disjunctions in P are taken into account in a form that is similar to the Disjunction Elimination inference rule used in Logic

$$\text{If } X_1 \vee \cdots \vee X_n,\ X_1 \supset L, \ldots, \text{and } X_n \supset L \text{ then } L \quad (2)$$

For this, we will create new atoms in $K(P)$, written L/X_i, that aim to capture the conditional beliefs $X_i \supset L$. Then, the resulting classical encoding will be such that once these atoms are 'achieved' for each $i = 1, \ldots, n$, and when they are suitably 'protected', the literal L will be made 'achievable' by an extra 'dummy' action with conditional effect similar to (2). In principle, any rule $a : C \wedge X_i \rightarrow L$ in P with X_i uncertain can be used to produce a rule $a : KC \rightarrow L/X_i$ in $K(P)$, meaning that if KC is known and a is applied, then if X_i was true, L will become true.

Rule 2 (Split) *For each rule $a : C \wedge X_i \rightarrow L$ in P where X_i is a literal that appears in a disjunction $X : X_1 \vee \cdots \vee X_n$, then add to $K(P)$ the atoms L/X_j, $j = 1, \ldots, n$, all initialized to* ***false****, and the rules $a : KC \rightarrow L/X_i$.* [2]

The combinations of the conditional beliefs represented by the atoms L/X_i is achieved by means of extra actions added to the classical encoding $K(P)$ that generalize (2) slightly, allowing some of the cases X_i to be disproved:[3]

Rule 3 (Merge) *For each disjunction $X : X_1 \vee \cdots \vee X_n$ and atom L in P such that L/X_i is an atom in $K(P)$, add to $K(P)$ a new action $a_{X,L}$ with conditional effect*

$$(L/X_1 \vee K\neg X_1) \wedge \cdots \wedge (L/X_n \vee K\neg X_n) \wedge FLAG_{X,L} \rightarrow L$$

where $FLAG_{X,L}$ is a fluent initialized to ***true****. If $L = X_i$ for some $i \in [1, n]$, remove the conjunct $(L/X_i \vee K\neg X_i)$ from the rule body.*

[1] We will present a simplified subset of the transformation rules due to lack of space. In particular, we will assume that every action only has one rule and no preconditions. The general translation appears in (Palacios & Geffner 2006).

[2] If we want L/X_i to mean exactly that '*right after the action* a, if X_i is true, then L is true', some additional care is needed about the other rules of the action. Details in the full paper.

[3] When using the classical plans obtained from $K(P)$ as conformant plans in P, such 'dummy' actions must be removed.

Problem	cf2cs(ff) Time	cf2cs(ff) Length	CFF Time	CFF Length
Bomb-100-1	0,84	199	96,2	199
Bomb-100-60	9.64	140	23,53	140
Cube-7-Ctr	0,02	24	38,2	39
Cube-9-Ctr	0,05	33	—-	—-
Cube-11-Ctr	0,09	42	—-	—-
Sqr-8-Ctr	0,03	22	140,5	50
Sqr-12-Ctr	0,04	32	—-	—-
Sqr-64-Ctr	9,66	188	—-	—-
Grid-4-4	0,06	25	0,11	25
Grid-4-5	0,05	30	0,14	30
Safe-50	0,05	50	134,4	50
Safe-70	0,08	70	561,8	70
Safe-100	0,28	100	—-	—-

Table 1: Plan times and lengths obtained by a classical planner (FF) over $K(P)$ translation (`cf2cs(ff)`) in relation to Conformant FF for various conformant problems P. Times in seconds. The symbol '—-' means cutoff exceeded (30 mins or 800Mb)

A key distinction from Logic is that the disjunction $X_1 \vee \cdots \vee X_n$ and the conditional beliefs 'if X_i then L' represented by the atoms L/X_i need all be **preserved** until they are combined together to yield L. This is the purpose of the boolean $FLAG_{X,L}$ that is initially set to true, but which is deleted when an action is taken in a context where it is not possible to prove that 1) L is preserved (if true), 2) the disjunction $X \vee L$ is preserved, and 3) the conditional beliefs represented by the atoms L/X_i achieved are preserved. This is accomplished by extending $K(P)$ with the rules that delete $FLAG_{X,L}$ when it is necessary.

These rules more detailed and other rules can be read in (Palacios & Geffner 2006). They yield expressivity without sacrificing efficiency, as they manage to *accommodate non-trivial forms of disjunctive inference in a classical theory without having to carry disjunctions explicitly in the belief state:* some disjunctions are represented by atoms like L/X_i, and others are maintained as *invariants* enforced by the resulting encoding.

Theorem 2 (Soundness $K(P)$) *Any plan that achieves the literal KL in $K(P)$ is a plan that achieves L in the conformant problem P.*

Experimental Results

We have implemented the translation program `cf2cs` that takes a conformant planning problem P as input and outputs a classical problem $K(P)$. Table 1 shows the plan times and lengths obtained by Conformant FF (Brafman & Hoffmann 2004) vs. `cf2cs(ff)` (FF planner fed with the problem generated by cf2cs). Translations only require a few seconds. Among the existing benchmarks, not included in the table, there are three domains, Sorting-Nets, (Incomplete) Blocks, and Ring, which cannot be handled by our translation scheme.

Discussion & Future Work

In vplan (Palacios *et al.* 2005) we presented a complete optimal planner that reject plans candidate that does not work for some initial state. In cf2sat (Palacios & Geffner 2005) we proposed to generate a propositional formula that encodes all the possible conformant planners, and called a SAT solver over it. In both cases we require an exponential process step of compiling into d–DNNF.

We have introduced a translation scheme that enables a wide class of conformant planning problems to be solved by an off-the-shelf classical planner. The translation accounts for a limited form of 'reasoning by cases' by means of an 'split-protect-and-merge' strategy; namely, atoms L/X_i that represent conditional beliefs 'if X_i then L' are introduced, and when certain invariants are verified, they are combined. This translation is incomplete because it is equivalent to a transformation like cf2sat, but considering only simple disjunctions of fluents instead of every initial state.

We want to explore allowing combinations of disjunctions by introducing atoms L/X_iY_j. For rules $a : C \wedge L \rightarrow M$, we can add $a : KC \wedge L/X_i \rightarrow M/X_i$, but in many cases it can lead to an exponential number of added atoms. However, we hope that some domains such as the Rings can be solved by a combination of these new rules, even when the new transformation will not be complete. We want to detect whether a problem is suitable for doing those additional transformations by using causal graphs. It will allow us, for instance, to see how many labels we need to consider for a variable.

The results presented here suggest to look for new propositional theories similar to Eq. 1. We can split on atoms that really need to be considered for solving the problem. We also can combine cf2cs, for easy problems, with cf2sat, for more complex problems, and obtain an hybrid planner that scales in a broader set of benchmarks. Moreover, as the plans obtained by cf2cs do not appear to be suboptimal, we want to identify when optimality holds and guarantee that. We also want to look for similar rules that allow transformations of other kinds of non-probabilistic uncertain planning, such as contingent planning.

Related Work

We did not compare the performance of cf2cs with many of the planners available because our goal is to map *some* conformant problems into classical planning. We compare with CFF (Brafman & Hoffmann 2004) as a way to show that our results are encouraging with respect to the state of the art. Most of them try to give a suboptimal solution to any conformant problem (Brafman & Hoffmann 2004; Cimatti, Roveri, & Bertoli 2004; Ferraris & Giunchiglia 2000). FragPlan (Kurien, Nayak, & Smith 2002) try to solve the general problem, but it can be used in more realistic environments where a partially conformant plan are needed.

Acknowledgements

We thank Blai Bonet for the PDDL parser and Joerg Hoffmann for providing FF and CFF.

References

Baral, C., and Son, T. C. 1997. Approximate reasoning about actions in presence of sensing and incomplete information. In *Proc. ILPS 1997*, 387–401.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proc. of AIPS-2000*, 52–61. AAAI Press.

Brafman, R., and Hoffmann, J. 2004. Conformant planning via heuristic forward search: A new approach. In *Proc. ICAPS-04*.

Cimatti, A.; Roveri, M.; and Bertoli, P. 2004. Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence* 159:127–206.

Darwiche, A. 2002. On the tractable counting of theory models and its applications to belief revision and truth maintenance. *J. of Applied Non-Classical Logics*.

Ferraris, P., and Giunchiglia, E. 2000. Planning as satisfiability in nondeterministic domains. In *Proceedings AAAI-2000*, 748–753.

Goldman, R. P., and Boddy, M. S. 1996. Expressive planning and explicit knowledge. In *Proc. AIPS-1996*.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of AAAI-96*, 1194–1201. AAAI Press / MIT Press.

Kurien, J.; Nayak, P.; and Smith, D. 2002. Fragment-based conformant planning. In *Proc. 13th Int. Conf. on Automated Planning and Scheduling (ICAPS-2002)*.

Lin, F., and Reiter, R. 1994. Forget it! In *Working Notes, AAAI Fall Symposium on Relevance*, 154–159. American Association for Artificial Intelligence.

Palacios, H., and Geffner, H. 2005. Mapping conformant planning to sat through compilation and projection. In *1st Workshop on QCSP - CP-2005*.

Palacios, H., and Geffner, H. 2006. Compiling uncertainty away: Solving conformant planning problems using a classical planner (sometimes). In *Proc. of (AAAI-06)*.

Palacios, H.; Bonet, B.; Darwiche, A.; and Geffner, H. 2005. Pruning conformant plans by counting models on compiled d-DNNF representations. In *Proc. of the 15th Int. Conf. on Planning and Scheduling (ICAPS-05)*. AAAI Press.

Turner, H. 2002. Polynomial-length planning spans the polynomial hierarchy. In *JELIA '02: Proc. of the European Conference on Logics in AI*, 111–124. Springer-Verlag.

Predictive Planning for Supply Chain Management: Adapting to Competitor Behavior*

David Pardoe
Department of Computer Sciences
The University of Texas at Austin
dpardoe@cs.utexas.edu

Introduction

In today's industrial world, supply chains are ubiquitous in the manufacturing of many complex products. Traditionally, supply chains have been created through the interactions of human representatives of the various companies involved. However, recent advances in planning, scheduling, and autonomous agent technologies have sparked an interest, both in academia and in industry, in automating the process (Kumar 2001).

From a planning and scheduling perspective, supply chain management simultaneously requires long-range inventory management, mid-range customer negotiations, and short-term factory scheduling, all of which interact closely.

One barrier to supply chain management research is that it can be difficult to benchmark automated strategies in a live business environment, both due to the proprietary nature of the systems and due to the high cost of errors. The Trading Agent Competition Supply Chain Management (TAC SCM) scenario provides a unique testbed for studying and prototyping supply chain management agents by providing a competitive environment in which independently created agents can be tested against each other over the course of many simulations in an open academic setting (Arunachalam & Sadeh 2005). In a TAC SCM game, each agent acts as an independent computer manufacturer in a simulated economy. The agent must procure components such as CPUs and memory; decide what types of computers to manufacture from these components as constrained by its factory resources; bid for sales contracts with customers; and decide which computers to deliver to whom and by when.

One crucial challenge in supply chain management is that decisions must often be made in the face of considerable uncertainty. For instance, purchases of production resources may need to be negotiated long before accurate information about customer preferences becomes available. This challenge is particularly evident in TAC SCM, where sources of uncertainty include the capacity of suppliers to deliver components, the nature of customer demand, and the actions of other agents as they compete for components and customers.

I have designed an agent to compete in TAC SCM, TacTex-05 (winner of the 2005 competition), that addresses this uncertainty by taking a predictive approach to its many planning and scheduling decisions. In particular, TacTex-05 makes predictions concerning the types and quantities of computers that will be requested by customers, the capacities of component suppliers and the prices they are likely to offer, and the probability that an offer to a customer will be accepted at a particular price. Planning and scheduling takes place using these predictions.

In this abstract, I will first provide details on the TAC SCM scenario and give an overview of the design of TacTex-05. Then I will describe my current work, which focuses on learning to adapt to the behavior of competing agents.

The TAC Supply Chain Management Scenario

In this section, I provide a brief summary of the TAC SCM scenario. Full details are available in the official specification document (Collins *et al.* 2005).

In a TAC SCM game, six agents act as computer manufacturers in a simulated economy managed by a game server. The length of a game is 220 simulated days, with each day lasting 15 seconds of real time. The game can be divided into three parts: i) component procurement, ii) computer sales, and iii) production and delivery, as expanded on in the remainder of this section and illustrated in Figure 1.

Component Procurement

The computers are made from four components: CPUs, motherboards, memory, and hard drives, each of which come in multiple varieties. From these components, 16 different computer configurations can be made. Agents must purchase these components from a set of suppliers managed by the game server.

Agents wanting to purchase components send requests for quotes (RFQs) to suppliers indicating the type and quan-

*This abstract is largely based on a paper in the ICAPS 2006 techical program (Pardoe & Stone 2006) that contains both a complete description of the TacTex-05 agent and a number of experimental results. I would like to thank my advisor, Peter Stone, for assistance with this work.

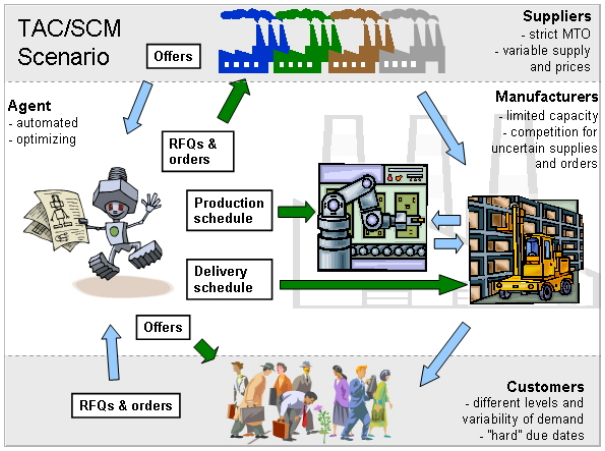


Figure 1: The TAC SCM Scenario (figure taken from (Collins *et al.* 2005)).

tity of components desired, the date on which they should be delivered, and a reserve price stating the maximum amount the agent is willing to pay. Suppliers respond to RFQs the next day by offering a price for the requested components if the request can be satisfied. Agents may then accept or reject the offers.

Suppliers have a limited capacity for producing components; this capacity varies throughout the game according to a random walk. The price offered in response to an RFQ depends on the fraction of the supplier's capacity that is free before the requested due date.

Computer Sales

Customers wishing to buy computers send the agents RFQs consisting of the type and quantity of computer desired, the due date, a reserve price indicating the maximum amount the customer is willing to pay per computer, and a penalty that must be paid for each day the delivery is late. Agents respond to the RFQs by bidding in a first-price auction: the agent offering the lowest price on each RFQ wins the order. The number of RFQs sent by customers each day depends on the level of customer demand, which fluctuates throughout the game.

Production and Delivery

Each agent manages a factory where computers are assembled. Factory operation is constrained by both the components in inventory and assembly cycles. Each day an agent must send a production schedule and a delivery schedule to the server indicating its actions for the next day. The production schedule specifies how many of each computer will be assembled by the factory, while the delivery schedule indicates which customer orders will be filled from the completed computers in inventory. Agents are required to pay a small daily storage fee for all components in inventory at the factory.

Overview of TacTex-05

Given the detail and complexity of the TAC SCM scenario, creating an effective agent requires the development of tightly coupled modules for interacting with suppliers, customers, and the factory. TacTex-05 is a fully implemented agent that operates within the TAC SCM scenario. In this section, I present a high-level overview of the agent.

Agent Components

Figure 2 illustrates the basic components of TacTex-05 and their interaction. There are five basic tasks a TAC SCM agent must perform:

1. Sending RFQs to suppliers to request components
2. Deciding which offers from suppliers to accept
3. Bidding on RFQs from customers requesting computers
4. Sending the daily production schedule to the factory
5. Delivering completed computers

The first two tasks are assigned to a *Supply Manager* module, and the last three to a *Demand Manager* module. The Supply Manager handles all planning related to component inventories and purchases, and requires no information about computer production except for a projection of future component use, which is provided by the Demand Manager. The Demand Manager, in turn, handles all planning related to computer sales and production. The only information about components required by the Demand Manager is a projection of the current inventory and future component deliveries, along with an estimated replacement cost for each component used. This information is provided by the Supply Manager.

The tasks to be performed by these two managers can be viewed as optimization tasks: the Supply Manager tries to minimize the cost of obtaining the components required by the Demand Manager, while the Demand Manager seeks to maximize the profits from computer sales subject to the information provided by the Supply Manager. In order to perform these tasks, the two managers need to be able to make predictions about the results of their actions and the future of the economy. TacTex-05 uses three predictive models to assist the managers with these predictions: a predictive *Supplier Model*, a predictive *Demand Model*, and an *Offer Acceptance Predictor*.

The Supplier Model keeps track of all information available about each supplier, such as TacTex-05 's outstanding orders and the prices that have been offered in response to RFQs. Using this information, the Supplier Model can assist the Supply Manager by making predictions concerning future component availability and prices.

The Demand Model tracks the customer demand in each of the three market segments, and tries to estimate the underlying demand parameters in each segment. With these estimates, it is possible to predict the number of RFQs that will be received on any future day. The Demand Manager can then use these predictions to plan for future production.

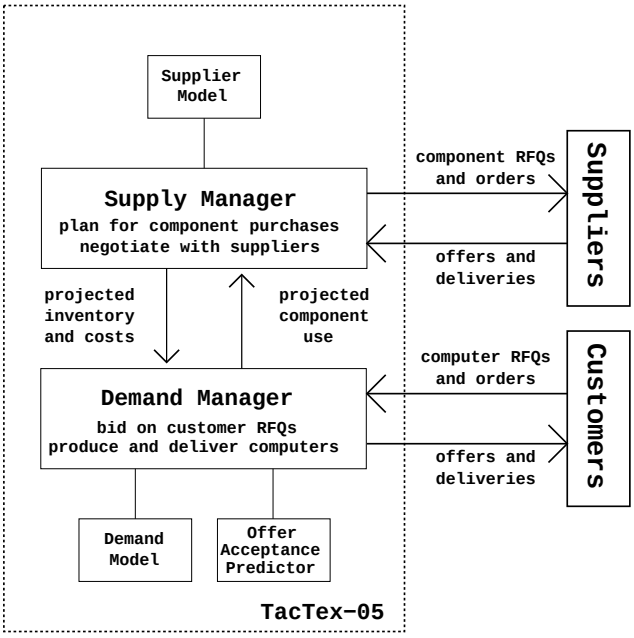


Figure 2: An overview of the main agent components

When deciding what bids to make in response to customer RFQs, the Demand Manager needs to be able to estimate the probability of a particular bid being accepted (which depends on the bidding behavior of the other agents). This prediction is handled by the Offer Acceptance Predictor. Based on past bidding results, the Offer Acceptance Predictor produces a function for each RFQ that maps bid prices to the predicted probability of winning the order.

Current Work: Adapting to Competing Agents

The TAC SCM competition consists of a series of rounds. During each round an agent faces the same five opponents in a number of games. When analyzing competition results, it quickly becomes apparent that the nature of the economy within a game depends heavily on the agents participating. An agent that consistently achieves a high profit against one set of opponents may lose a large amount of money against a different set of opponents in a different round. This fact suggests the potential value of designing an agent that can adapt to the behavior of whatever opponents it happens to be facing during a particular round. Enabling TacTex-05 to adapt in such a fashion is the primary focus of my current work. (The general development of such adaptive agents in agent-based economies will be the focus of my thesis; in addition to studying the TAC SCM domain, I have explored auction domains in which seller agents adapt the parameters of auction mechanisms in response to the observed behavior of bidding agents (Pardoe *et al.* 2005).)

The primary means by which TacTex-05 can be made more adaptive is through improvements to the predictive modules described previously. In particular, I would like to improve long-term predictions of computer prices and component prices, both of which can vary considerably based on opponent behavior. Currently, the predictions made by the predictive modules are based primarily on observations from the current game. Another source of information that could be useful in making predictions is the events of past games, made available in log files kept by the game server.

The potential benefit from basing predictions on the results of these past games is illustrated by the one form of adaptation used by TacTex-05 during the 2005 TAC SCM competition. At the beginning of each game, many agents place relatively large component orders (when compared to the rest of the game) to ensure that they will be able to produce computers during the early part of the game. Prices for some components may also be lower on the first day than they will be afterwards, depending on the due date requested. Determining the optimal initial orders to place is difficult, because no information is made available on the first day of the game. As a result, many agents use the same hard-coded initial orders in each game. TacTex-05 takes advantage of this fact by basing its predictions of early-game component prices on the prices observed in past games. An analysis of the final round of competition (Pardoe, Stone, & VanMiddlesworth 2006) showed that first-day prices were unusually attractive due to the purchasing patterns of the agents. As a result of its adaptivity, TacTex-05 recognized this opportunity and purchased significantly more components on the first day of each game than its competitors. The savings on component costs accounted for much of TacTex-05's winning margin. Although this example illustrates the value of adaptation, it is admittedly ad hoc. One goal of my current work is to identify additional opportunities for adaptation automatically, through techniques that will generalize to other domains.

One possible approach is the use of machine learning techniques to develop more accurate predictive models. In fact, I explored this possibility in past work (Pardoe & Stone 2004), finding that learned predictors could indeed improve agent performance. There is one primary drawback to this approach, however: it requires that an agent be able to draw training data from a large number of games against the same opponents. A single round of competition consists of a relatively small number of games, at most 16, raising the question of how a machine learning approach could successfully be applied. In particular, during the first game of a round, there would be no data from which the agent could learn.

Thus, I am currently exploring means by which TacTex-05 can begin a round of competition with fairly general predictive modules, and then revise them based on data as it becomes available, rather than starting tabula rasa. One valuable resource in my work is the TAC Agent Repository,[1] a collection of agents made available by competition participants for research purposes. By simulating rounds of competition with various combinations of these agents,

[1] http://www.sics.se/tac/showagents.php

along with variations of TacTex-05 designed to exhibit particular behaviors, I can observe a wide range of different economies. Using these simulations I hope to answer the following questions:

- What properties remain the same from one set of opponents to another? (e.g., component prices tend to decrease over the course of a game)
- What properties are highly dependent on the set of opponents? (e.g., how quickly computer prices rise when customer demand increases)
- What fixed predictive models result in the best performance across a wide range of opponent sets?
- As additional data becomes available, how can these predictive models successfully be revised?

Answering the last question presents the largest challenge from a learning perspective. Approaches I am currently investigating include the use of online learning methods for combining expert advice (where each expert represents a predictive model learned for a particular opponent set) and metalearning methods (in which the performance of a learning system is improved through experience with a family of related tasks – in this case various opponent sets).

References

Arunachalam, R., and Sadeh, N. 2005. The supply chain trading agent competition. *Electronic Commerce Research and Applications* 4:63–81.

Collins, J.; Arunachalam, R.; Sadeh, N.; Eriksson, J.; Finne, N.; and Janson, S. 2005. The supply chain management game for the 2006 trading agent competition. Technical report. Available from http://www.sics.se/tac/tac06scmspec_v16.pdf.

Kumar, K. 2001. Technology for supporting supply-chain management. *Communications of the ACM* 44(6):58–61.

Pardoe, D., and Stone, P. 2004. Bidding for customer orders in TAC SCM. In *AAMAS 2004 Workshop on Agent Mediated Electronic Commerce VI: Theories for and Engineering of Distributed Mechanisms and Systems*.

Pardoe, D., and Stone, P. 2006. Predictive planning for supply chain management. In *Sixteenth International Conference on Automated Planning and Scheduling*.

Pardoe, D.; Stone, P.; Saar-Tsechansky, M.; and Tomak, K. 2005. Adaptive auctions: Learning to adjust to bidders. In *The Fifteenth Annual Workshop on Information Technologies and Systems*.

Pardoe, D.; Stone, P.; and VanMiddlesworth, M. 2006. TacTex-05: An adaptive agent for TAC SCM. In *AAMAS 2006 Workshop on Trading Agent Design and Analysis / Agent Mediated Electronic Commerce*.

Integration of Constraint-Based Off-line and On-line Approaches to Project Scheduling

Riccardo Rasconi*

Institute for Cognitive Science and Technology
Italian National Research Council

riccardo.rasconi@istc.cnr.it

Abstract

Solving a scheduling problem involves considerations of twofold nature: on one hand, the solution must be searched according to specific optimization needs (off-line requirements), while on the other hand, great effort should be employed into producing solutions which can be safely put into execution in unpredictable environments (on-line requirements). In many cases, these two classes of necessities hide mutually conflicting aspects. Aim of my work is to assess the possible integration of off-line and on-line procedures in project scheduling in order to find the best balance between the two, in view of the inherently dynamical utilization of each produced scheduling solution.

Introduction

Traditionally, planning and scheduling communities have tackled the scheduling problem according to one of the two following mainstreams. On one side, much effort has been put into the development of methodologies producing solutions which are characterized by a certain degree of robustness, therefore retaining the ability to absorb the effects of exogenous events (proactive approach). On the other side, the *buffer* that protects the solution against possible disruptions is inherently limited, and the need to device mechanisms to reactively counteract circumstances that fall beyond its boundaries (reactive approach), is not eliminated.

The present work introduces a schedule management schema which tends to integrate the off-line and on-line approaches: according to this schema, the task of the scheduler is not limited to the production of a sequence of activities, as well as the process of controlling schedule executability is not exclusively played on the ground of on-line reaction and activity dispatchment. This work is inspired by the following considerations: (a) regardless the proactive approach employed to produce the baseline schedule, a dynamic analysis on the actual behaviour of the schedule execution is necessary in order to prove, from the operational standpoint, both the efficacy of the choices made and the soundness of the arguments which led to those choices; (b) merely counting on the effectiveness of schedule adjustments at execution time is prone to fostering myopic decisions which may readily result in a complete schedule disruption. Analyzing how the proactive phase may influence (and possibly guide) the reactive phase at execution time is in my opinion as important as assessing the best baseline schedule production strategy on the base of the schedule's particular dynamic behavior. The information that can be extracted from the two phases may reveal mutually useful in order to find an optimal strategy combination, as well as the reasons behind its optimality. For example, one may find an initial solution which is optimal with respect to some criteria of interest (e.g. the makespan), and may wish to know what are the online rescheduling strategies which preserve optimality as much as possible, given some knowledge about the types of exogenous events that may occur during execution.

Aim of the present abstract is to describe the steps I have taken in order to pursue the previous goal. The scheduling problem we specifically focus upon is the project scheduling problem (Brucker *et al.* 1999). These problems are characterized by a rich internal structure. They are based on a network of activities, among which it is possible to identify complex temporal relations that can be used to model a number of variably rigid causal links which normally constrain the tasks in a project. As a further source of complexity, several heterogeneous resources with different capacities serve the activities according to complex modalities.

As a first step, I have implemented an experimental framework which allows to compare different approaches to schedule synthesis and execution in a fair and controlled way. This empirical platform can be used to carry on a set of *reproducible* experiments by (1) simulating the execution of a number of baseline schedules produced with different proactive methods, (2) disturbing their execution with predefined exogenous events, and (3) assessing their behavior by using separate reactive scheduling policies. Different off-line and on-line solving procedures can be compared independently from each other, given the generality of the devised architecture.

Great attention was paid in order to grant measureability and reproducibility of the experimentations. A Reactive Scheduling Problem Benchmark Generator (RSP-BG) has been produced, which, given a project scheduling problem $\mathcal{P}$, returns a number of *exogenous events* (or *disturbs*) that might be fired during the execution of the solution [*baseline($\mathcal{P}$)*]. The production of such events must be strictly related to the structure of each scheduling problem in order to maximize the probability of event acceptance during execution. In order to guarantee experimental fairness, for each initial problem, the RSP-BG allows to produce a specified number of disturbs characterized by a given overall difficulty.

Following the production of a reactive scheduling prob-

*Also affiliated with the Dipartimento di Informatica, Sistemistica e Telematica (DIST) of the University of Genova, Italy

lem benchmarks, a number of explorative experiments have been performed by simulating the execution of baseline schedules synthesized by different proactive techniques and therefore characterized by a different degree of initial "temporal flexibility" (Cheng & Smith 1994).

The Reactive Scheduling Problem Benchmark Generator

Real world uncertainty can be reasonably singled out in the following points: activity delays, growth of activity processing times, lowerings of resource availability, variations in the number of activities, changes in the mutual ordering of the activities.

Currently I have focused our attention on the temporal changes which normally characterize the physical environments, such as delays of the activities start times and/or modifications of activity processing times:

– *delay of the activity start time*: activity a_i undergoes a delay of Δ_{st} time units, at $t = t_{aware}$ ($e_{delay} = \langle a_i, \Delta_{st}, t_{aware} \rangle$);
– *change of activity processing time*: activity a_i's processing time p_i is extended by Δ_p time units, at t_{aware} ($e_p = \langle a_i, \Delta_p, t_{aware} \rangle$).

In order to evaluate the difficulty of each benchmark, proper metrics are introduced to evaluate the structure of a scheduling problem as a set of unexpected events $\mathcal{E} = \{e_1, \ldots e_n\}$ is introduced. For instance, let us consider a scheduling problem $\mathcal{P}'$ obtained by adding to the original problem $\mathcal{P}$ an event e_k; given a metric $\mu()$, it is then possible to compare the structures of the problems $\mathcal{P}'$ and $\mathcal{P}$ by considering the μ variation value: $\Delta_\mu = |\mu(\mathcal{P}) - \mu(\mathcal{P}')|$.

One of the possible metrics to use for this purpose is the following, defined as the average width, relative to a given temporal horizon H, of the temporal slack associated with each pair of activities (a_i, a_j):

$$fldt_H = \sum_{i=1}^{n} \sum_{j=1 \wedge j \neq i}^{n} \frac{slack(a_i, a_j)}{H \times n \times (n-1)} \times 100 \quad (1)$$

where $slack(a_i, a_j)$ is the width of the allowed distance interval between the end time of activity a_i and the start time of activity a_j. This metric characterizes the *fluidity* of a solution (Cesta, Oddi, & Smith 1998), i.e., the ability to use flexibility to absorb temporal variation in the execution of activities: it is a measure of the possibility that a temporal variation concerning an activity is absorbed by the temporal flexibility of the solution instead than generating a deleterious domino effect (the higher the value of $fldt_H$, the less the risk, i.e., the higher the probability of localized changes).

The Schedule Execution Simulation Monitor

The idea behind the open schedule management framework used in this work is simple: an off-line solver produces the baseline solution and delivers it to an on-line module which takes care of assessing its dynamic characteristics by stressing it in a variety of ways.

The overall framework is composed of three modules: the *off-line solver* and the *real world simulator* work off-line and have the job of, respectively, computing the initial solution and generating the exogenous events, intended to disturb the schedule execution; the third module, the *on-line solver*, works on-line and is responsible to complete a simulated execution of the initial solution (the baseline schedule). A number of disturbing events synthesized by the RSP-BG are injected during the simulated execution at the times specified within each event, and their effects are counteracted by the on-line module, which is endowed with a portfolio of *rescheduling* algorithms to the aim of restoring schedule consistency whenever necessary.

Testing Schedule Flexibility

The particular problem I focus upon is the Resource-Constrained Project Scheduling Problem with minimum and maximum time lags, or RCPSP/max. This is a particular project scheduling problem which presents constraints that define the minimum and maximum distance between the execution of two activities[1].

Each baseline solution is computed according to different procedures: the result is the production of initial solutions retaining different degree of temporal flexibility. A temporally flexible solution can be described as a network of activities whose start times (and end times) are associated with a set of feasible values (feasibility intervals). Underlying the activity network there exists a Temporal Constraint Network (TCN (Dechter, Meiri, & Pearl 1991)), composed of all the start and end points of each activity (time points), bound to one another through specific values which limit their mutual distances (activity on the arc representation). The search approaches used in our schema focus on decision variables which represent conflicts in the use of the available resources; the solving process proceeds by ordering pairs of activities until all conflicts in the current problem representation are removed. This approach is usually referred to as Precedence Constraint Posting (PCP (Cheng & Smith 1994)), because it revolves around imposing precedence constraints (the *solution constraints*) on the TCN in order to solve the resource conflicts, rather than fixing rigid values to the start times.

In (Cesta, Oddi, & Smith 1998) it is shown that the previous schedule representation inherently provides a certain level of resilience at execution time (i.e. it produces *Flexible Schedules*), even though temporal and resource consistency are guaranteed only if the lower bounds (or upper bounds) from the feasibility intervals are chosen for the time points. In order to overcome the limitation imposed by the flexible schedule, i.e. having only one consistent solution, a generalization of the TCN produced by a PCP phase is proposed in works such as (Cesta, Oddi, & Smith 1998; Policella *et al.* 2004), in which methods for defining a set of both time and resource feasible solutions are presented. This new representation is called *Partial Order Schedule* ($\mathcal{POS}$), a special case of a flexible solution which is obtained by replacing the solution constraints with a new set of constraints that impose a stronger condition on the TCN (*chaining constraints*). A $\mathcal{POS}$ is a flexible solution such that any possible temporal solution is also a resource-consistent assignment. As a third type of solution, fixed time solutions have been tested against Flexible schedules and $\mathcal{POS}$s.

The execution of each baseline schedule has been simulated, in the face of an increasing number of exogenous

[1] RCPSP/max is recognized as a quite complex problem; in fact, even the feasibility version of the problem is NP-hard. The reason for the NP-hardness lies in the presence of maximum time-lags, which inevitably imply the satisfaction of deadline constraints.

Algorithm 1: Solve a scheduling problem P and Execute one of its solution S

```
Input: problem P, policies parameter retract and pos
Output: Execution report
// off-line phase
S ← offlineScheduler(P)
if S does not exist then
    STOP (SOLVER FAILURE)
if pos then
    S ← createPOS(S)
// on-line phase
while a disturb E exists do
    if retract then
        if propagation(E, S) fails ∨ S is not resource consistent then
            S ← removeChoice (S)
            if propagation(E,S) fails then
                STOP (EXECUTION FAILURE)
            S ← onlineScheduler(S)
            if S does not exist then
                STOP (EXECUTION FAILURE)
            if pos then
                S ← createPOS*(S)
    else
        if propagation(E,S) fails then
            STOP (EXECUTION FAILURE)
        if S is not resource consistent then
            S ← onlineScheduler(S)
            if S does not exist then
                STOP (EXECUTION FAILURE)
```

events, according to the Algorithm 1.

The algorithm is divided in an off-line and an on-line section; in the former, the initial solution can be computed by the `offlineScheduler()` either as a flexible schedule (case FS) or as a $\mathcal{POS}$ (case POS) through the `createPOS()` procedure, depending on the value of the flag `pos`. In the latter, and regardless how the initial solution is produced, it is put into execution according to different modalities, depending on the value of the flag `retract`. At each step of the execution cycle, the environment is sensed for possible disturbs. Afterwards, if `retract` $= true$, the execution algorithm firstly removes all the constraints imposed in the previous solving process (`removeChoice()`), and secondly looks for a new solution (`onlineScheduler()`), possibly creating a new $\mathcal{POS}$. If `retract` $= false$, a new solution is searched leaving the previously imposed solution constraints untouched. In both cases, the algorithm initially checks for temporal consistency after each disturb is acknowledged through the `propagation()` procedure.

Preliminary Experiments and Results

Table 1 shows some preliminary results of our investigation (refer to (Rasconi, Policella, & Cesta 2006) for a more detailed description). As explained earlier, we evaluate the different combinations of off-line/on-line policies — POS-R ($\mathcal{POS}$ + retraction), POS-NR ($\mathcal{POS}$ + no retraction), FS-R (Flexible Schedule + retraction), and FS-NR (Flexible Schedule + no retraction). To make the comparison more complete, we add a further execution mode based on the use of fixed time solution where each activity is assigned a single start time instead of a set of alternatives.

For each entry in the tables, we take into account the following aspects: the number of unexpected events (*number of disturbs*) injected during each single execution, the percentage (with respect to the number of initially solved problems) of the schedules which successfully completed the execution (*% executed*), the execution failure percentage due to the inability to find an alternative solution (*% failed resch.*), the execution failure percentage due to the impossibility to accept the exogenous event on behalf of the TCN (*% refused events*), the average makespan of the solutions at the end of the execution (*mk*), the average difference between the initial and the final makespan (Δ *mk*), the percentage of the performed rescheduling actions with respect to the number of the injected disturbs (*% rescheduling*)[2], the average CPU time, in msecs, to compute the initial solution (*CPU Off-line*), the average CPU time spent to perform all reschedulings during the execution (*CPU On-line*), the *sensitivity* of activity start time w.r.t. the execution process (ψ).

For a fair comparison of the different policies, the data presented in the rightmost part of the table are computed on the basis of the problem instances commonly executed with all the execution strategies.

One of the most striking results that we observe regards the different abilities in preserving the executability of a solution. The outcome shows that the use of partial order schedules tends to lower the success rate in terms of completed executions (*% executed* column). As the table presents, this is mainly due to the dramatic increase in the number of rejected disturbs (*refused events* column). This apparent anomaly can be explained as follows: the creation of a $\mathcal{POS}$ inherently involves a higher level of "constrainedness" in the TCN, in order to guarantee a resource conflict-free solution. This circumstance inevitably makes the TCN more reluctant in accepting new contraints, in the specific case, the constraints which model the exogenous events. Also, note how this effect gets worse as the number of the exogenous events increases (86,87% in the POS-NR case with 1 event, against 56,66% with 5 events).

The rightmost part of Table 1 offers different yet interesting results. One of the most important characteristic to be observed is the extremely low rate of necessary reschedulings exhibited by the POS-R/POS-NR policies (*% resched.* column): this result is all but surprising and confirms the theoretical expectations which motivated the study on the $\mathcal{POS}$. As shown, the need for schedule revision in case of $\mathcal{POS}$ utilization roughly decreases by more than 50% in case of 5 disturbs.

A maybe misleading results is given by the comparison of the final makespan (mk) obtained respectively by using the Retraction and the No-Retraction strategies. In fact, one would expect the R strategies (which allow a greater re-shuffling) to return better makespan values with respect to NR strategies. This is not our case because, as described before, the rescheduling actions are performed by using a less specialized makespan-optimizing procedure (`onlineScheduler` in Algorithm 1) which tends to spoil

[2] We recall that we have a rescheduling action each time the on-line solver is invoked.

	number of disturbs	% executed	failed resch.	refused events	mk	Δmk	% resched.	CPU off-line	CPU on-line	ψ
FS-R		91,04%	2,08%	6,88%	424,60	9,02	24,38%	36242,48	766,15	5,44
POS-R	**1**	87,29%	2,08%	10,63%	419,88	5,07	11,58%	36287,86	303,97	3,00
FS-NR		91,87%	1,25%	6,87%	419,06	3,48	24,14%	36242,48	130,74	1,48
POS-NR		86,87%	2,50%	10,62%	417,11	2,31	11,58%	36287,86	54,59	1,05
fixed time		89,79%	3,75%	6,45%	437,36	21,78	99,75%	36242,48	3035,68	15,16
FS-R		85,21%	3,13%	10,66%	435,54	13,95	23,04%	30259,15	874,70	9,48
POS-R	**2**	76,46%	2,29%	21,25%	429,22	8,41	10,03%	32371,82	674,86	5,17
FS-NR		85,62%	2,71%	11,66%	427,90	6,30	22,88%	30259,15	258,50	3,07
POS-NR		73,95%	5,00%	21,04%	424,74	3,93	9,56%	32371,82	97,46	2,02
fixed time		81,25%	8,54%	10,20%	446,62	25,02	99,53%	30259,15	2768,37	17,73
FS-R		79,17%	3,96%	16,87%	449,84	19,40	20,27%	26318,37	963,16	11,83
POS-R	**3**	69,58%	2,92%	27,50%	441,49	11,93	9,41%	28406,98	675,18	6,96
FS-NR		80,00%	2,50%	17,50%	439,66	9,23	22,37%	26318,37	371,03	4,08
POS-NR		67,71%	5,41%	26,87%	436,24	6,68	9,63%	28406,98	151,26	3,16
fixed time		77,50%	6,87%	15,62%	458,32	27,89	99,22%	26318,37	3716,15	19,77
FS-R		70,21%	4,17%	25,63%	464,12	28,85	22,45%	25682,40	2391,92	17,36
POS-R	**5**	60,42%	3,13%	36,46%	455,56	21,07	10,57%	27544,98	1748,56	13,12
FS-NR		70,41%	3,33%	26,25%	447,42	12,15	21,66%	25682,40	646,90	5,80
POS-NR		56,66%	7,08%	36,25%	444,68	10,18	10,48%	27544,98	289,17	4,81
fixed time		67,08%	8,33%	24,58%	465,56	30,29	98,43%	25682,40	6721,48	19,74

Table 1: Summarizing data for each execution strategy (the values in the last six columns are computed on the intersection set of all successfully executed j100 problems)

the makespan quality. On the other hand, the NR strategy that tries to maintain the schedule continuity is also able to obtain a preservation of makespan values.

Another interesting aspect can be observed by comparing the *CPU on-line* values between the *Retraction* and *No Retraction* strategies. In general, the Retraction methods require a higher CPU on-line load because the removal of the solution constraints inevitably re-introduces some resource conflicts that must be solved by rescheduling. But the intriguing result lies in the fact that this difference in the CPU on-line rates stands *despite the comparable amount of performed reschedulings*. Let us look at the difference between the FS-R and FS-NR rates: it can be seen that, in the 5 events case, we have 2392 ms. (FS-R) against 647 ms. (FS-NR), although the number of performed reschedulings is practically the same ($\approx 21\%$)! The same effect can be observed between the POS-R and POS-NR cases: 1748 ms. against 289 ms, notwithstanding the same ($\approx 10.5\%$) number of reschedulings. This circumstance can be explained as follows: NR execution modes retain all the temporal constraints of the previous solution: hence, the rescheduler is bound to work on a smaller search space, finding the next solution almost immediately.

Ongoing Work

The analysis being performed on the considered scheduling benchmarks is returning several interesting information. Some results confirm the expectations while other require a certain level of analysis in order to be correctly understood. For instance, the rigid behavior exhibited by the fixed time schedules when confronted with dynamically variable environments is totally confirmed, as confirmed is the behavior of schedules characterized by a more flexible nature.

However, among several other aspects, the experiments revealed a scarce capability in accepting exogenous events on behalf of the $\mathcal{POS}$: post-experimental analysis about this unexpected effect has shown that this drawback is due to an increase of constrainedness in the TCN, necessary to guarantee a resource conflict-free solution at all times. This has suggested several research lines which are the object of ongoing work, such as the production of a different class of $\mathcal{POS}$s, through the development of alternative chaining procedures aimed at minimizing the inevitable constrainedness increase in the TCN. Moreover, the observed dynamic behavior of the schedules suggests to study the introduction of different reactive techniques. For instance, a possible approach under current development is based on *informed retraction* procedures, where the constraint removal strategy is preceded by a search phase to determine the constraints which are to be retracted, depending on the particular dynamic requirements.

Acknowledgments. I would like to thank Amedeo Cesta and Nicola Policella for their continuous and invaluable support in this research, as well as all my colleagues at ISTC-CNR.

References

Brucker, P.; Drexl, A.; Mohring, R.; Neumann, K.; and Pesch, E. 1999. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operational Research* 112:3–41.

Cesta, A.; Oddi, A.; and Smith, S. F. 1998. Profile Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. In *Proceedings, AIPS-98*, 214–223. AAAI Press.

Cheng, C., and Smith, S. F. 1994. Generating Feasible Schedules under Complex Metric Constraints. In *Proceedings, AAAI-94*, 1086–1091. AAAI Press.

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

Policella, N.; Oddi, A.; Smith, S. F.; and Cesta, A. 2004. Generating Robust Partial Order Schedules. In *CP 2004*, volume 3258.

Rasconi, R.; Policella, N.; and Cesta, A. 2006. Fix the schedule or solve again: Comparing constraint-based approaches to schedule execution. In *Proceedings of Constraint Satisfaction Techniques for Planning and Scheduling Problems Workshop ICAPS 2006*. To appear.

Improving Planning Techniques for Web Services

Francisco Carlos Palao Reinés
Dept. of Computer Science and Artificial Intelligence
University of Granada, SPAIN
palao@decsai.ugr.es

Abstract

The new trend on software development is oriented to web services running in collaborative environments. Intelligent planning techniques are very useful to compose complex calls to these web services. However, there are still some issues that need to be improved to use planning and scheduling techniques in dynamic and collaborative contexts like the web service environment. This paper proposes some extensions to our planning system for using it to compose web service calls.

Introduction

This research extends the SIADEX environment, which is a planning system oriented to assist the command technical staff for decision support in forest fire fighting operations. The system is composed of different components communicating each other and working together through the Internet. These components are implemented as web services: they are pieces of software that make themselves available over the Internet and use standard XML messaging system. The World Wide Web is turning into a new paradigm called the Collaborative Web (Pallot, Prinz, & Schaffers 2005) where not only documents are connected through the network but collaborative services as well. The SIADEX project has some different web services working together for a common goal (Figure 1). One web service to store the knowledge of the problem; another to make the planning process; and a third monitors the plan execution. The planning web service uses the SIADEX planner that has been developed by us and is a forward state-based HTN temporal planner (Castillo *et al.* 2006). Moreover there is a user interface where technical staff can introduce the problem to solve it, see the plan generated by the planner and follow the execution.

To achieve a correct system operation we need to call the different web services in the correct order and time. In order to do that we have developed a central server that synchronizes all the component of the architecture. This central server has been called the InfoCenter. The InfoCenter is based on a publish/subscribe architecture (Carzaniga, Rosenblum, & Wolf 2001) (PSA) that works as follows. Each web service or user interface can publish information in the central server (InfoCenter) and also it may subscribe to the information (published by others web services or clients) that they want. For instance, when the technical staff publishes the problem, the InfoCenter sends it to the planner web service that is subscribed to it.

At the present, our architecture is very simple because only a web service of each kind is available and no external web services can connect to the InfoCenter to make the system more complete. Therefore the InfoCenter can easily compose calls to the web services to achieve the requests, the execution and the monitoring of the plans.

Now, we want to extend the system not only to assist the technical staff for decision support in forest fire fighting but also for e-business, e-tourism and workflow applications among others. Therefore we need to extend the system with new web services, some of them different than the current ones and another with similar capabilities. However, the current PSA presents some lacks that impede the extension of the system. For instance, the InfoCenter can not choose what web service is the correct one (or optimal one) when there are more than one web service that offer the same functionality. Furthermore, the PSA is purely reactive because it only make web service calls upon receptio nof a publication and it is not able to compose sequences of web services with a longer time horizon.

In order to extend the system architecture we are going to use planning techniques into the InfoCenter, as well some frameworks to use planning techniques for web services composition have been purposed (Madhusudan & Uttamsingh 2006; Mithum, desJardins, & Finin 2003) but there are still a lot of issues to solve to fulfill our InfoCenter requirements. We are thinking of using our own planner, SIADEX, to use it inside the InfoCenter. However, we need to improve it for some reasons. Firstly, the web services environment is a very dynamic context. Therefore, the state of web services (the domain) and the requests of the users (the goal) can change during the execution of calls to web services. And our planner can not check domain and state changes during planning time. Secondly, there could be a large number of web services with similar capabilities and the InfoCenter has to evaluate them to decide which one is the best to achieve its goals. And thirdly, we are thinking of an architecture oriented to the Collaborative Web, so our intelligent InfoCenter need to communicate with others intelligent servers in order to access trough them to resources that are not directly connected to it. So, the planner needs to be extended

with distributed planning skills and to be able to understand web services standard languages.

In this work we present our system SIADEX as a framework that will be extended with novel ideas to give solutions to all these problems.

The SIADEX architecture

SIADEX is a system being developed under a research contract with the Andalusian Regional Ministry of Environment. Its objective is to assist the command technical staff in the design, dispatching and progress of forest fire fighting plans. It is composed of different, domain independent web services (Figure 1), that offers different services, that are distributed and communicate with each other using XML-RPC standard protocols.

- SIADEX Planner: Is a planning web service that can be called by XML-RPC protocol. SIADEX is a forward state-based HTN temporal planner (Castillo *et al.* 2006). It uses its own hierarchical extension of PDDL 2.2 level 3 language, that makes it very expressive. It also has the capability to include embedded Python scripts in the domain definition, that allows us to implement external calls at planning time.
- BACAREX: Is an ontology web service that stores the knowledge related to the planning domain. In our case its stores information about the forest fire fighting domain in Andalusia (Spain). BACAREX is also capable of generating domain and problem files that are processed by our planning web service.
- Monitor service: This web service splits the plan into several pieces and sends every piece to the person in charge of executing it. These parts of the plan will be presented to the user using any portable electronic device. The monitor controls the plan executions attending the dependences between tasks and their possible delays (Castillo *et al.* 2006).
- User interfaces: We have provided GUI capabilities to the planning system for the expert. The GUI is built on top of the ArcView GIS tool (ESRI). This GUI is totally domain dependent and oriented toward the interaction with the forest fire technical staff. We have also developed a web interface to monitor the execution of the plan with any available web browser.
- InfoCenter: It is the central component of our architecture. All the aforementioned web services are connected to the InfoCenter and collaborate each other by passing messages through it. The InfoCenter has been developed as a publish/subscribe architecture (PSA) in which the others web services can subscribe to the information that they want and publish the information that they have to share with others web services. The PSA works correctly in small environments like this with a few web services but it is purely reactive. And we want to extend this architecture to larger and more dynamic environments where we would need a deliberative server able to compound sequences of calls to web services. To achieve this we are thinking on extending the InfoCenter with planning techniques that we need to develop and are explained below.

Composing Web Services and SIADEX

There is an interchange of information between all the web services described above during a planning episode to assist the technical staff for decision support. In this section we show how this interchange of information is done at the present and how would be done in the future supporting larger environments of web services.

Present operation between web services

The InfoCenter is the Broker Server in our PSA . Each web service or user interface can publish information in the central server (InfoCenter) and also can subscribe to the information (published by others web services or clients) that they want. The basic cycle of the present architecture is:

1. The user interface publishes the goal of the planning problem defined by the technical staff and the InfoCenter sends it to the ontology web service that is subscribed to all new information about the world state and the new goals.
2. The ontology web service BACAREX publishes the domain and the problem translated into PDDL from the ontology knowledge and the InfoCenter sends it to the SIADEX Planner that is subscribed to all the domains and problems in PDDL generated.
3. The SIADEX Planner publishes the plan generated and the broker server sends it to the Monitor because it is subscribed to new plans.
4. The Monitor publishes the actions that have to be executed at each time and the InfoCenter sends it to the technical staff in charge of doing it.
5. Until the plan is completely executed, the technical staff send (public) confirmations about actions completed to the InfoCenter and the Monitor, that is subscribed to new events of the actions, publishes new actions. BACAREX is also subscribed to the new events of the actions in order to update the world state in the ontology.

The cycle shown above can be carried out with the present PSA that implements the InfoCenter. However, if we had more web services connected and some of them offer the same or similar functionalities, we would need to make more complex compositions of web services that we can not make now. At the moment, the InfoCenter can not choose what web service is the correct one (or optimal one) when there are more web services that offer the same functionality. Furthermore, the PSA is purely reactive because it only make web service calls when receive a publication and it is not able to compose sequences of web services with a longer time horizon. Note that it is not only a selection problem to pick the best web service, we need to make a sequence of calls to web services to know if the goal can be achieved with the available web services. We want to keep the easy

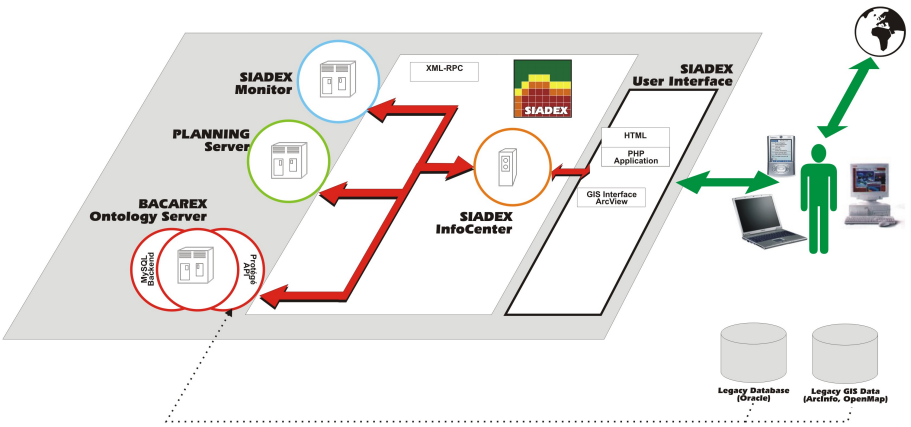


Figure 1: General overview of SIADEX Architecture.

connectivity and scalability of our current PSA but in a deliberative way. In order to do that, we will extend the InfoCenter or Broker Server with intelligent planning techniques like we describe below.

Future operation between web services

In the last section we have seen that we need to compose complex calls to web services in dynamic and larger environments. In order to do that we need to improve some features of our own SIADEX planner to use it in the InfoCenter. The features with which we need to extend our planner are shown in this section.

Continuous revision of the state and the goal. A great advantage of our PSA is the high response capabilities to the environment changes. So, it has to be supported by the planner. The set of services available could be constantly changing as online or offline status while we are executing the sequence of calls to web services. In addition the user could change his goals at execution time. Therefore the planning process needs to be continually checking for changes in the domain, in the state or in the goals (Giunchiglia & Traverso 1999; Madhusudan & Uttamsingh 2006) during the execution of web services. We can achieve this using the embedded Python scripts in the domain definition to implement external calls at planning time. There are two kinds of calls to extern web services. Firstly, calls to ensure a complete solution by checking that the web service is available. Secondly, calls to ensure a sound solution by checking that the web service behavior is the correct one.

Automated generation of domains and heuristics. As new web services are connected or disconnected to the InfoCenter the domain knowledge changes and the planner need to know it to compose the sequences of calls to web services (plans). We need to make an ontology inside the InfoCenter that stores the web service information (the domain) and define an automatic process that translates the ontology knowledge into the PDDL readable by the planner (Sirin *et al.* 2004). In addition, this web service has to be evaluated to make optimal plans. We need to implement automated heuristics generations (Zimmerman & Kambhampati 2003). To judge the optimality of the plans we need to evaluate the web services by using metrics about the network behavior (response time, transfer rating) and the final user preferences (it could be the price of a product, the duration of a trip, etc).

Distributed planning capabilities. As we have said in the Introduction section, Internet is turning into a new paradigm named the "Collaborative Web" where services collaborate between them. Therefore the InfoCenter has to be able to share with other intelligent servers its plans (or part of them) or to ask others intelligent servers for plans (or part of them). In order to do that we need to consider all the work done in collaborative planning environments (desJardins & Wolverton 1999). Furthermore, the InfoCenter has to be able to understand standard web service description languages such as WSDL (Christensen *et al.* 2001) and to generate plan outputs as web service flows with standard specifications such as BEPEL4WS (Curbera 2002) or OWL-S (Coalition 2003).

Concluding remarks

We have described a new approach in the SIADEX planning system architecture in order to prepare it for the new trends on web services environments. The main challenges faced at planning time are in dynamic conditions and the need to collaborate with others web services architectures. We sketch an extension of our planner to check the completeness and soundness of the solutions in these environments and to be able to communicate with others web services architectures. All the changes proposed are about the central component of the architecture: the InfoCenter. That is the one in charge to compose the sequences of calls to web services with the new planning techniques that will be developed. These sequences of calls to web services have to be formulated in standards specifications such as BPEL4WS or OWL-S.

Acknowledgements

This research was supported by the contract NET033957 with the Andalusian Regional Ministry of Environment for

the assisted design of forest fighting plans.

References

Carzaniga, A.; Rosenblum, D.; and Wolf, A. 2001. Design and evaluation of a widearea event notification service. *ACM Transactions on Computer Systems* 19:332–383.

Castillo, L.; Fdez-Olivares, J.; Garcia-Perez, O.; and Palao, F. 2006. Efficiently handling temporal knowledge in an htn planner. In *International Conference on Automated Planning & Scheduling*.

Christensen, E.; Curbera, F.; Meredith, G.; and Weerawarana, S. 2001. The web services descriptio nlanguage wsdl. http://www-4.ibm.com/software/solutions/webservices/resources.html.

Coalition, O. S. 2003. Owl-s: Semantic markup for web services. OWL-S White Paper http://www.daml.org/services/owl-s/0.9/owl-s.pdf.

Curbera, F. e. a. 2002. Business process execution language for web services. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel.

desJardins, M., and Wolverton, M. 1999. Coordinating planning activity and information flow in a distributed planning system. *AI Magazine* 20:45–53.

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

Giunchiglia, F., and Traverso, P. 1999. Planning as model checking. In *In Proc. 5th European Conference on Planning*.

Madhusudan, T., and Uttamsingh, N. 2006. A declarative approach to composing web services in dynamic environments. In *Decision Support System 41 (2) 325-357*.

Mithum, S.; desJardins, M.; and Finin, T. 2003. A planner for composing services described in daml-s. In *ICAPS2003 Workshop on Planning for Web Services*.

Pallot, M.; Prinz, W.; and Schaffers, H. 2005. Future workplaces, towards the 'collaborative web'. In *1st AMI@WORK Communities Forum Day*.

Sirin, E.; Parsia, B.; Wu, D.; Hendler, J.; and Nau, D. 2004. A declarative approach to composing web services in dynamic environments. In *J. Web Sem 1(4): 377-396*.

Zimmerman, T., and Kambhampati, S. 2003. Learning-assisted automated planning: Looking back, taking stock, going forward. *AAI Magazine* 24:(2) 7396.

Exploiting Portfolio Strategy to Explore the Interaction of Problems and Algorithms in AI Planning *

Mark Roberts
Computer Science Department
Colorado State University
Fort Collins, Colorado 80523
mroberts@cs.colostate.edu

It is well known that a problem-speci c approach can lead to an algorithm that does well on one problem but fails on other problems with a markedly different structure ((Wolpert & Macready 1997) consider the discrete case). Portfolios are a general way to overcome algorithm bias and maintain robustness across a range of problems. A portfolio controls the run time of a suite of algorithms with a strategy consisting of: *selecting* which algorithms to run; *ranking* the selected algorithms; and *allocating* computational time to them. An ideal strategy maximizes success and minimizes total computation. This means the strategy must be accurate and fast.

Previous portfolios used a variety of strategies. One of the rst formulations calculated the risk of selecting an algorithm (Huberman, Lukose, & Hogg 1997). Gomes and Selman (1997) explicitly computed the value of multiple restarts of the same algorithm. The Bus meta-planner (Howe *et al.* 1999) used a round robin allocation based on a trade-off between a simple model of success and expected cost. Other portfolio approaches (e.g., (Gratch & Chien 1996; Minton 1996; Fink 1998; Baptista & Silva 2000; Leyton-Brown *et al.* 2003; Lagoudakis & Littman 2000)) used easily extracted features of the problems and solution progress and rely, for their portfolio strategy, on models generated off-line from problem instances. Most approaches also leverage statistical run-time distribution information (e.g., (Horvitz *et al.* 2001; Beck & Freuder 2004)) while some, such as (Guerri & Milano 2004), focus on extracting features that uncover *structure* of the problem instance.

The goodness or utility of a portfolio strategy can be measured in different ways. For example, we can examine the raw number of problems it solves and examine its robustness on unseen problems. We can compare portfolio performance against the individual or aggregate performance of the suite of algorithms. We can examine the selection, ranking, and allocation strength of the portfolio strategy against the best it could have done. Finally, *we can study the portfolio strategy for clues about* why *one algorithm is favored over another and what led the portfolio to make the distinction.* Ultimately, it is this last measure that will foster deeper insight and explanation.

*This research was sponsored by the National Science Foundation under grant number IIS-0138690.

My research analyzes portfolio performance on planning problems from the International Planning Competitions to uncover algorithmic and problem structure dependencies through:

1. Modeling planner performance of 23 planners across almost 4000 benchmark problems.
2. Constructing portfolio strategies using a principled methodology based on analysis of and learning from previous off-line performance.
3. Measuring portfolio strategies to test speci c hypotheses about what constitutes effective selection, ranking, and allocation.
4. Examining planner performance, models of the performance, features used to build those models, domain information, and dependencies between these sets to develop and test speci c hypotheses leading to stronger explanations of search performance in planning.

What follows are highlights of some of my ndings with regard to questions (1) and (2) that form the core of a recently submitted AAAI workshop paper. My dissertation research will address all four of these questions.

Modeling Planner Performance

We study 23 classical planners, and for each planner, we construct two models: *success* and *time*. Success estimates $P(\text{solution found}|\text{problem, planner})$. Time predicts computation time needed for a given planner to complete a given problem.

Each problem instance is de ned by 57 features that can be automatically extracted from problem and domain de nitions. The set starts with features from (Howe *et al.* 1999) and (Hoffmann 2001) and adds others. We divide the features into four categories of increasing knowledge and computational cost: domain speci c, instance speci c, action interaction and Hoffmann's state space topology[1] (Hoffmann 2001). Based on the amount of computation time to compute features, we designate the domain and instance-speci c as 'fast' features and the action interaction and topological features as 'expensive' features.

We run all planners on 3959 STRIPS PDDL problems from 77 domains. The problems are taken from Hoffmann's dataset (Hoffmann 2004), the UCPOP Strict benchmark,

[1]We thank Jörg Hoffmann for supplying the code.

IPC sets (IPC1, IPC2, IPC3 Easy Typed and IPC4 Strict Typed) and 37 other problems from two domains (Sodor and Stek) that have been made publicly available. Each planner is allowed 30 minutes and 768 Meg. We used 22 identically con gured Pentium 4 3.4Ghz computers.

We use the WEKA data mining package (Witten & Frank 2005) to build the models. We tried several different models from WEKA; to begin with, we focused our work on two simple models that worked well: OneR and J48. OneR selects the single feature that yields the highest prediction value on the training set, while J48 is a simple decision tree based on Quinlan's C4.5. By default, we use 10-fold cross validation.

We distinguish our models based on the data we use to build them: *all* data and *old* data (all but IPC4). The *time3* model divides 30 minutes of possible time into 10 equal sized (three minute) bins. The *logTime* model uses 5 bins of time based on the division (in seconds): $\{1, 10, 100, 1000, 10000\}$.

Predicting Success The distribution of results in the raw data values tends to be skewed. 35.4% of the runs succeed over all planners and problems. 97.1% (successful) and 96.4% (failure) of runs complete in under 3 minutes. The distributions do vary across planners from a range of 5%-70% for success and 65.5%-100% for % successful runs under 3 minutes and 72.6%-100% for % failed runs under 3 minutes. We found that J48 predicts success for *old* with 96.7% average accuracy (sd of 3.2) and for *all* with 96.8% average accuracy (sd of 2.12).

Predicting Time The run-time distributions (RTDs) are heavily skewed for each planner. Over all planners, 77.8% of the runs nish in less than one second and 6.9% nished in greater than 1000 seconds[2]. Given these heavy tails, we found that *logTime* predictions are much more accurate than *time3* predictions. The average prediction accuracy using J48 with *binned data* was 93.52% (sd of 6.55) for *logTime*.

Which features are informative? To examine if speci c features are informative we constructed OneR models, which rely on a *single* feature for classi cation. For success on the old problems, the `average number of negations in effects` was the best predictor for nine of the planners; the `predicate arity` was best for another four. The rst feature may indicate where the often used h^+ heuristic may have trouble; the second roughly influences branching in the search space.

When we examined the features used for the *old* data, we found that fast features had been selected for predicting success for 16 of the 23 planners and for predicting time for 5 of the 12 planners. All of the expensive features selected were from Hoffmann's set. The mean accuracy for the success models was 91.8% using fast features and 93.6% using expensive features. The mean accuracy for time models was 96.2% using fast features and 78.2% using expensive features; however, the skews on the distribution for models using the expensive features tended to be more extreme.

[2]We did not construct time models for nine planners that either always finished in less than 1 second or that too rarely succeeded in less than 30 minutes.

Do we need all of the features? We noted a considerable (100 to 1000-fold) difference in computing the feature cost for the fast features and the expensive features. We wondered if we could use only the fast features without sacri cing accuracy. On a subset of the data for which we had all feature information, a paired sample T-test of using or removing the expensive features was not signi cant ($p < 0.61$ for success and $p < 0.49$ for *time3*); it appears that expensive features are not necessary for accurate models.

Do models for older problems generalize? To test this question, we trained models using *old* and tested with cross-validation as well as with IPC4. A paired sample T-test comparing accuracy of each of the planner models when tested on *old* versus IPC4 is highly signi cant ($P < .0001$); the models are statistically signi cantly more accurate for the old problems. This suggests that the IPC4 problems must be classi ed differently than the older problems.

Portfolio Construction

We construct our portfolio in a principled manner from the model analyses. To date, we have tried several variants that are increasingly informed by deeper insight.

Pruning with static allocation We developed and tested a simple model of portfolio allocation by applying the RTDs. We noted that the highest median planner time was 5 seconds. Our rst strategy selected only those planners to those that were predicted to succeed, ranked that pruned set by probability of success, and allocated ten seconds (5 seconds rounded to the next largest bin) to each planner. We found that this strategy was not very robust, though it did signi cantly outperform the average planner performance. In hindsight, a simple explanation is that the planners were stopped too soon; we effectively stopped some planners somewhere between their 50th and 80th percentiles for time-to-success.

Using unique planners We rst examined how we might reduce the number of algorithms that the portfolio uses. Let $\mathcal{A}$ be the entire set of algorithms under consideration. In general, we seek to nd the minimum covering of unique planners, $\mathcal{U} \subset \mathcal{A}$, such that all problems solved by $\mathcal{A}$ remain solved by $\mathcal{U}$. A minimum covering is equivalent to a set covering, which is $\mathcal{NP}$-Complete, but has a known polynomial algorithm, Greedy-Set-Cover, that has an approximation ratio $\rho(n) = \ln|\mathcal{A}|+1$ (Cormen *et al.* 2003). We implemented Greedy-Set-Cover and found that at least 14 planners could be removed.

Quiting with confidence A closer look at the RTDs showed us that we could set a reasonable pausing time for the planners on each iteration of a round robin portfolio. In this way, we start with a higher con dence than the median, and we gradually increase the con dence in each pass of the algorithm. For the unique planners, all but one achieve the 80th percentile at 10 seconds. At 100 seconds, one planner achieves each of the 89th, 94th, and 97th percentiles, ve reach the 98th percentile, and three achieve the 99th percentile. At 200 seconds, six have reached the 99th percentile. This observation suggests a stepped approach for allocating time.

The current portfolio begins with the set of planners under consideration (either $\mathcal{A}$ or $\mathcal{U}$); it does not itself perform any

selection. For these planners, the portfolio ranks the planners according to the learned models in decreasing probability of success then in increasing probability of failure. This ensures that the portfolio tries the planner most likely to succeed first and the planner most likely to fail last.

The portfolio allocates time to the planners in a series of round robin stages. The first stage tries the first five planners for ten seconds each; we chose the first five because it is half of $|\mathcal{U}|$. The second stage starts at the top of the ranking and runs all planners up to 100 seconds. Every stage thereafter adds 100 seconds. The portfolio stops when 1) a planner succeeds, 2) no planners are alive to run, or 3) max-time is exceeded.

We trained the models for this portfolio using a random sampling of 90% of the problems and tested it on the remaining 10% (394). Of this 394 problems, 371 (94.2%) were solved by at least one planner. The best single planner, SGPlan-04 solved 291 (73.9%) of these problems. To examine the impact of using the culled set, we compare the performance of the portfolio using all planners ($\mathcal{A}_{port}$) against the portfolio using the unique planners ($\mathcal{U}_{port}$). In terms of robustness, $\mathcal{A}_{port}$ solved 307 (77.9%) problems while $\mathcal{U}_{port}$ solved 325 (82.5%). Both portfolios significantly perform faster than the average planner run time (by about 6 seconds) according to a paired sample T-test.

Future Work

Incorporating learning from prior experience into a portfolio yielded some promising results. But there still remains much work toward the primary goal of *linking* the search bias of various planners with their performance on specific problems. In this section, we present several key points of continuing work that we hope will reveal deeper insight into understanding specific planning approaches.

Features To date, we have examined three basic feature types: domain/problem instance, action interaction, and topological. The OneR and J48 models provide some evidence linking particular features to performance prediction. Already, we have begun to examine other features for impact on performance prediction. We expect to identify new features that help explain less direct action interactions as we perform richer domain analysis. Further, we hope to examine the feature set for overlap and uniqueness similar to what we have done for planners.

Planners We began our research with the widest set of planners possible. We plan to examine the data for interactions based on planner type (such as SAT-based, POCL-base, Graphplan, Relaxed Graphplan, or Hybrid). We use the default settings of all planners; our work can easily be extended to include some limited parameter tuning of the planners by viewing each change in parameters as a new planner. Most of the planners we used came from the IPCs. This coming IPC5 will provide another set of planners to extend our work. Attending ICAPS during an IPC year will also provide direct interaction (and possible collaboration with) the authors of these planners to aid in gaining deeper insight.

Problems This work is based in the classical planning paradigm. Recent extensions to PDDL were intended to push the community toward more realistic problems (Hoffmann & Edelkamp 2005); our work indicates that there is a significant difference between the old and new problems. My hope is to fully develop a principled methodology for portfolio construction and then extend it to the newer Temporal and Probabilistic tracks of the IPC. Another potential area for understanding the planner behavior is through the problem generators from the IPCs - these could be used to test specific hypotheses about the dependencies we notice.

Classifiers We mined our data with two simple classifiers to predict success and run time. We conjecture that there is a wealth of information beyond this utilitarian approach, so we hope to extend these analyses by incorporating more sophisticated classification techniques (such as Bayesian or k-nearest neighbor clustering) to mine relationships between the features, the problems, and the algorithms.

Unified Model Work by (Kambhampati, Knoblock, & Yang 1995) and (Kautz & Selman 1999) unifying planning approaches under the same framework provides a foundation for exploring deeper analysis of planners.

On-line learning Our current system uses prior, off-line learning to inform the portfolio strategy. It seems reasonable to consider that we could extend the models into an on-line learning paradigm.

Computational Trade-offs Our preliminary research indicates we may be able to use only fast features to predict success and time. But we need to examine this hypothesis in light of newer domains and problems (especially since the models built on old models didn't generalize). If we find that expensive features are necessary, we expect to examine in more detail the question of determining the appropriate level of computational effort for a strategy. Classifiers that incorporate feature cost into classification would be helpful determining an appropriate trade-off point.

The focus of incorporating these extensions is to identify dependencies between the domains, heuristics, algorithms, and run time dynamics in classical planning. The main directions of this research are 1) in providing a principled methodology for portfolio construction and 2) in providing a framework for exploiting the portfolio strategy to refine our understanding of planning approaches.

Acknowledgments

I want to thank Adele Howe, my dissertation adviser, and the reviewer of this paper who both provided feedback and ideas for further directions for this work. Finally, a thanks to all the authors who make their planners available for study.

References

Baptista, L., and Silva, J. P. M. 2000. Using randomization and learning to solve hard real-world instances of satis ability. In *Principles and Practice of Constraint Programming*, 489–494.

Beck, J. C., and Freuder, E. C. 2004. Simple rules for low-knowledge algorithm selection. In *Proc. of 1st CPAIOR*.

Cormen, T.; Leiserson, C.; Rivest, R.; and Stein, C. 2003. *Introduction to Algorithms*. MIT press, Cambridge, MA, second edition.

Fink, E. 1998. How to solve it automatically: Selection among problem solving methods. In *Proc. of 4th AIPS*, 128–136.

Gomes, C. P., and Selman, B. 1997. Algorithm portfolio design: Theory vs. practice. In *Proc. of 13th UAI*. Linz, Austria.: Morgan Kaufman.

Gratch, J., and Chien, S. 1996. Adaptive problem-solving for large-scale scheduling problems: A case study. *JAIR* 4:365–396.

Guerri, A., and Milano, M. 2004. Learning techniques for automatic algorithm portfolio selection. In *Proc. of 16th ECAI*, 475–479.

Hoffmann, J., and Edelkamp, S. 2005. The deterministic part of IPC-4: An overview. *JAIR* 24:519–579.

Hoffmann, J. 2001. Local search topology in planning benchmarks: An empirical analysis. In *Proc. of 17th IJCAI*, 453–458.

Hoffmann, J. 2004. *Utilizing Problem Structure in Planning: A local Search Approach*. Berlin, New York: Springer-Verlag.

Horvitz, E.; Ruan, Y.; Gomes, C. P.; Kautz, H.; Selman, B.; and Chickering, D. M. 2001. A bayesian approach to tackling hard computational problems. In *Proc. of 17th UAI*, 235–244.

Howe, A. E.; Dahlman, E.; Hansen, C.; von Mayrhauser, A.; and Scheetz, M. 1999. Exploiting competitive planner performance. In *Proc. of 5th ECP*.

Huberman, B. A.; Lukose, R. M.; and Hogg, T. 1997. An economics approach to hard combinatorial problems. *Science* 275:51–54.

Kambhampati, S.; Knoblock, C. A.; and Yang, Q. 1995. Planning as re nement search: A uni ed framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence* 76(1-2):167–238.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and graph-based planning. In *Proc. IJCAI-99*.

Lagoudakis, M. G., and Littman, M. L. 2000. Algorithm selection using reinforcement learning. In *Proc. 17th ICML*, 511–518.

Leyton-Brown, K.; Nudelman, E.; Andrew, G.; McFadden, J.; and Shoham, Y. 2003. A portfolio approach to algorithm selection. In *Proc. of 18th IJCAI*.

Minton, S. 1996. Automatically con guring constraint satisfaction programs: A case study. *Constraints* 1(1/2):7–43.

Witten, I. H., and Frank, E. 2005. *Data Mining: Practical machine learning tools and techniques*. Number ISBN 0-12-088407-0. San Francisco: Morgan Kaufmann, 2nd edition.

Wolpert, D. H., and Macready, W. G. 1997. No free lunch theorems for optimization. *IEEE Trans./ on Evolutionary Comp.* 1(1):67–82.

Infeasible Search Analysis for Oversubscribed Scheduling Problems

Mark F. Rogers
Computer Science Dept., Colorado State University
Ft. Collins, CO 80523 USA
rogersma@cs.colostate.edu

Introduction

Researchers have expended considerable effort developing local search neighborhoods and heuristics that restrict search to feasible space. However, some have discovered that a variety of problems may be solved efficiently if an algorithm includes both feasible and infeasible solutions in its search. In a procedure called *strategic oscillations* (Glover 1989), Glover defined a local search strategy that alternates a search between feasible and infeasible states and uses a *span* to control the number of moves the search makes in either region. An algorithm may increase or decrease its span periodically to vary the search intensity in feasible and infeasible regions.

Strategic oscillation algorithms appear to hold two advantages over feasible-only search: first, the oscillations tend to focus a search around the feasible-infeasible frontier, or *boundary region*, where optimal solutions reside for many constrained optimization problems (Glover 1989; Schoenauer & Michalewicz 1996). Second, some researchers claim that infeasible search opens new routes to optima and permits an algorithm to exploit short-cuts in the space (Glover 1989; LeRiche, Knopf-Lenoir, & Haftka 1995; Kelly, Golden, & Assad 1993; Michalewicz 1996).

The boundary region is defined as feasible states that have infeasible neighbors, or infeasible states that have feasible neighbors. When a search reaches a boundary-region state, its next move may introduce or resolve constraint violations as the search moves into the opposite region. Thus boundary-region search is motivated by the conjecture that optima for constrained optimization problems frequently include *binding* or *active* constraints. A constraint becomes active whenever a variable reaches its minimum or maximum allowed value. Research has verified that boundary region search finds good solutions for domains such as laminate design (LeRiche & Haftka 1994), numerical optimization (Schoenauer & Michalewicz 1996) and single-machine scheduling (Hurink & Keuchel 2001).

That a search may benefit from short-cuts is an appealing concept, but until now we have seen no direct established evidence that infeasible search finds paths that could not be reached as directly using feasible solutions. In addition, the concept of short-cuts appears to conflict with the goal of boundary-region search; to uncover efficient paths through infeasible space, a search must move away from the boundary region, effectively postponing boundary exploration until the search returns from infeasible space. We have confirmed that a strategic oscillation algorithm finds short-cuts through infeasible space, but it is difficult to justify extensive infeasible search.

We have experimented with two satellite scheduling domains that each consist of a single *oversubscribed* satellite: each problem instance has many more requests than the satellite can accommodate. The first domain is the 2003 ROADEF Challenge problems: satellite scheduling problems designed for a competition (Cung 2003). We have applied a strategic oscillation algorithm, $Tabu_{CL}$ (Cordeau & Laporte 2003) to the ROADEF problems and to a synthetic earth observing satellite (EOS) domain. In both domains the best solutions are likely to contain tasks with active constraints. The search spaces are large: ROADEF has $n! \cdot 2^n$ possible states in a problem with n tasks; for EOS the number of states is bounded by $n! \cdot m^n$, where n is the number of tasks and m is the maximum number of time windows allocated to a task. Both domains include time window constraints which we relax in order to introduce infeasible solutions into a search.

Infeasible Path Segments

We define a *path segment* as any sequence of schedules explored during a search. A *feasible* segment is one in which each schedule is feasible. An *infeasible* segment is one where each schedule in the segment contains at least one constraint violation except for the start and end schedules, which are feasible. This restriction on the start and end schedules helps us divide infeasible search into distinct infeasible segments.

We further distinguish between infeasible segments that stay within the boundary region and those that move outside the boundary region. We define a *boundary-region* segment as an infeasible segment whose states all reside along the boundary. When at least one infeasible state has no feasible neighbors, we refer to its segment as a *deep* infeasible segment. These distinctions allow us to evaluate the relative merits of large and small oscillation span values, and provide insights into how likely infeasible search is to yield significant benefits.

To assess the efficiency gained through infeasible search, we consider three phenomena: cycles, detours and short-

cuts. When a segment's start and finish state are identical, then the segment is a *cycle*. If there is a cycle contained within an infeasible segment from one infeasible state to another, then we identify the segment as a *detour*. Cycles and detours degrade search performance by wasting moves. The final infeasible segment we consider is the *short-cut*: an infeasible segment whose length is shorter than any feasible segment between two feasible states.

We have conducted experiments to test the hypothesis that infeasible solutions facilitate efficient search by opening short-cuts in a search space. For our oversubscribed scheduling problems, we have found that short-cuts do exist for segments that leave the boundary region, but their efficiency may be offset by infeasible segments that yield little or no benefit to a search.

Tabu Search

The $Tabu_{CL}$ implementation uses three neighborhood operators: `insert` that inserts an image into the schedule; `remove` that removes an image from the schedule, and `replace` that changes the image that will be acquired. The `insert` operator finds the first possible location for an image by examining each possible slot in the schedule. Thus a schedule is biased towards placing tasks in the earliest possible slot. This strategy ignores possible time window violations for images other than those immediately adjacent to the candidate image, so an `insert` operation may generate infeasible schedules.

The `replace` operator attempts to remove an image from the schedule and to replace it with an equivalent image from the same request. Thus the `replace` operator may also introduce time window violations whenever the transition times or the time window for an alternate image differ from those of the original.

$Tabu_{CL}$ strategic oscillations with a simple linear penalty function and a scale factor, α. For a schedule s, a profit value $p(s)$, and time window violation count $w(s)$, the evaluation function $f(s)$ is given by:

$$f(s) = p(s) - \alpha w(s) \quad (1)$$

As a search progresses, the value α controls oscillations about the boundary region. As a search moves through feasible space, α decreases stochastically, thus encouraging the search to admit infeasible schedules. Once the search enters infeasible space, α begins to increase and eventually forces the search back toward feasible schedules.

We applied a modified version of this algorithm to the EOS problems using the same neighborhood operators. The EOS problems use a different objective function than ROADEF, but the strategic oscillation behavior is the same.

To account for the boundary region's influence, we implemented a restricted version of the tabu search ($Tabu_{BR}$) that prevents a search from moving away from the boundary. We compared $Tabu_{CL}$ with $Tabu_{BR}$, to see how much a search benefits from boundary-region search and how much it gains from traversals away from the boundary.

Infeasible Segment Analysis

During each run, whenever a search transitioned from a feasible state to an infeasible state, we recorded the feasible state and each move in the infeasible path segment. When the search transitioned again back to feasible space, we recorded the ending feasible state and allowed the search to continue until it started another infeasible segment.

We conducted infeasible path analysis using a separate program to assess each infeasible segment recorded during a search. To categorize infeasible segments, we implemented an algorithm that detects the characteristic infeasible segment types defined in the previous section. In addition to categorizing infeasible paths we also differentiate between boundary region traversals and deep traversals that move beyond the boundary region.

Results

We have confirmed the existence of short-cuts for algorithms that include infeasible states. Our infeasible path statistics confirm that short-cuts, cycles and detours all occur during infeasible search. However, the value of deep infeasible trajectories varies between problems.

When we use strategic oscillations to allow a search to probe deeply into infeasible regions, we find that in most cases, deep traversals generate a higher proportion of improving moves than boundary-region search. At the same time, they tend to enter fewer cycles, and they uncover a higher proportion of short-cuts in the space. Each of these attributes implies that deep infeasible traversals have the potential to make a search highly efficient.

However, when we compare $Tabu_{BR}$ with $Tabu_{CL}$ on the ROADEF problems we do not find consistent performance improvements. Possibly this is because short-cuts still comprise a minority of deep traversals (roughly 20-40%), while deep traversals themselves consume more iterations than boundary-region traversals. Thus while deep traversals may occasionally yield promising short-cuts, this benefit is offset by the number of iterations squandered on non-improving deep path segments. To address this issue, we have begun exploring ways to eliminate unproductive segments.

Infeasible Tabu List

Our first approach was to augment $Tabu_{CL}$ with a tabu list for infeasible paths. With this enhancement we have been able to eliminate most of the cycles and detours found in infeasible traversals. We hoped that an infeasible tabu list would also force a search to explore a more diverse selection of infeasible paths than the original search and thus exploit more shortcuts. Although the enhanced algorithm nearly doubled the proportion of shortcuts for the ROADEF problems, we saw little improvement for most problems and the new tabu list tended to degrade performance overall.

For the EOS problems we found that an infeasible tabu list made the search more efficient than the original search in some cases, but made little difference in others. Evidently it is not sufficient merely to increase the proportion of short-

cuts: in addition we need to consider where a shortcut begins in a space, and whether it finds an improving path.

Future Work

Infeasible path metrics should give us new ways to assess strategic oscillation search. By creating tools that enable us to measure infeasible search attributes, we obtain more comprehensive algorithm performance metrics than we would by merely charting an objective function during a run. By examining the relationships between algorithm features and infeasible search metrics, we should be able to improve these algorithms further or demonstrate that improvements are unlikely. Ultimately our goal is to identify a "best" algorithm for a given problem domain, but there are still questions we want to answer regarding the ROADEF and EOS domains.

Although an infeasible tabu list yielded good results for some problems, it provides little insight into search behavior that leads to unimproving shortcuts. We want to examine the states where we find cycles, detours and unproductive shortcuts to see if there are trends common to these phenomena that will allow us to eliminate a proportion of wasted moves. As our results show, the difference between boundary region search and deep infeasible search can be subtle; eliminating even a small number of detrimental moves could improve search performance.

"Jump" Tabu

Our $Tabu_{CL}$ ROADEF results revealed that for many infeasible paths longer than two moves, the best moves in each neighborhood changed little as the search proceeded into infeasible space. We are investigating the possibility we could make $Tabu_{CL}$ more efficient simply by eliminating the neighborhood searches for some initial infeasible steps. By sorting the best moves in the initial infeasible step and then applying several of them at once, we could "jump" ahead in the search by generating the same infeasible path using fewer evaluations.

A naïve approach is simply to apply k of these moves at the start of each infeasible path to make a jump. The average infeasible path length for ROADEF problems ranges from 2 to 5 moves and of these, approximately half will be `insert` or `replace` moves at the start of an infeasible segment. Thus the simplest implementation applies the first $k = 2$ moves from an initial infeasible neighborhood, skipping one set of evaluations. A more sophisticated approach will attempt to predict an optimal number of moves to make in a single jump as a function of the penalty parameter value, the current gain and the violation counts for the best infeasible moves.

The "jump" approaches may allow us to reduce wasted evaluations by applying multiple moves at once. We also wish to determine whether it is possible to predict when an infeasible traversal is likely to produce a productive shortcut. At this point, our infeasible path statistics do not reveal any obvious trends. Currently we are investigating whether a machine learning tool such as a C4.5 classifier could identify subtle indications that lead to productive infeasible paths. If we can construct such a classifier, then we may gain further insight into what causes unproductive infeasible paths, or incorporate the classifier directly into a search.

Neighborhoods and Algorithms

To date, our work has focused on $Tabu_{CL}$ and its performance with two oversubscribed scheduling problems. Our results hint that an algorithm's neighborhood operators may dictate shortcuts' frequency and influence during local search. Thus we want to expand our investigation to examine shortcut behavior with different neighborhood operators and algorithms.

In $Tabu_{CL}$, neighborhood operators are not symmetric: it may not be possible to undo a move in a single step. For example, if we `remove` a task and immediately `insert` the same task, the resulting schedule may change. The algorithm greedily schedules tasks at the earliest possible time, so a task that is removed and reinserted may move to an earlier slot in the schedule. We hypothesize that this kind of asymmetry may explain the large number of shortcuts we find with $Tabu_{CL}$. Thus we would like to know how relevant symmetry is for finding shortcuts whether algorithms that use symmetric neighborhoods can find shortcuts as well.

If we are able to identify search characteristics that lead to productive infeasible search, a related goal will be to apply our results to other search algorithms such as simulated annealing or genetic algorithms. Research has shown that strategic oscillation strategies are compatible with temperature schedules in simulated annealing (Anagnostopoulos *et al.* 2003). In addition, researchers have applied strategic oscillations successfully to genetic algorithms in a number of domains (for example, (Joines & Houck 1994; Coit, Smith, & Tate 1996; Eiben & Ruttkay 1996; Bean & Hadj-Alouane 1997)).

Strategic oscillations can be an effective method for solving constrained optimization problems by relaxing problem constraints to introduce infeasible solutions into a search. Infeasible space provides access to both sides of the boundary region and allows a search to focus its attention on good solutions that may reside there. However if we want to develop efficient search algorithms, then we need to understand what characteristics make infeasible search successful. By studying infeasible path statistics, we hope to uncover the infeasible search features that yield insights into how a search behaves and may thus allow us to exact the benefits while avoiding the unproductive forays.

References

Anagnostopoulos, A.; Michel, L.; Hentenryck, P. V.; and Vergados, Y. 2003. A simulated annealing approach to the traveling tournament problem. In *Proceedings of CPAIOR 2003*.

Bean, J. C., and Hadj-Alouane, A. B. 1997. A genetic algorithm for the multiple-choice integer program. *Operations Research* 45(1):92–101.

Coit, D. W.; Smith, A. E.; and Tate, D. M. 1996. Adaptive penalty methods for genetic optimization of constrained combinatorial problems. *INFORMS Journal on Computing* 8(2):173–182.

Cordeau, J. F., and Laporte, G. 2003. Maximizing the value of an earth observation satellite orbit. Technical Report CRT-2003-27, Centre de recherche sur les transports.

Cung, V.-D. 2003. ROADEF'2003: Results of the final stage (base X) of the challenge. `http://www.prism.uvsq.fr/~vdc/ROADEF/CHALLENGES/2003/results030203_final.html`.

Eiben, A. E., and Ruttkay, Z. 1996. Self-adaptivity for constraint satisfaction: Learning penalty functions. In *International Conference on Evolutionary Computation*, 258–261.

Glover, F. 1989. Tabu search–Part I. *ORSA Journal on Computing* 1(3):190–206.

Hurink, J. L., and Keuchel, J. 2001. Local search algorithms for a single-machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics* 112(1-3):179–197.

Joines, J. A., and Houck, C. R. 1994. On the use of nonstationary penalty functions to solve nonlinear constrained optimization problems with GA's. In *International Conference on Evolutionary Computation*, 579–584.

Kelly, J. P.; Golden, B. L.; and Assad, A. A. 1993. Large-scale controlled rounding using tabu search with strategic oscillation. *Annals of Operations Research* 41:69–84.

LeRiche, R., and Haftka, R. T. 1994. Improved genetic algorithm for minimum thickness composite laminate design. In *Proceedings of the International Conference on Composite Engineering, Aug 28–31*.

LeRiche, R.; Knopf-Lenoir, C.; and Haftka, R. T. 1995. A segregated genetic algorithm for constrained structural optimization. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, 558–565. Morgan Kaufmann Publishers Inc.

Michalewicz, Z. 1996. *Genetic Algorithms + Data Structures = Evolution Programs*. London: Springer.

Schoenauer, M., and Michalewicz, Z. 1996. Evolutionary computation at the edge of feasibility. In *Parallel Problem Solving from Nature IV*, 245–254. Berlin: Springer.

Thesis Summary: First-order Decision-Theoretic Planning

Scott Sanner
Department of Computer Science
University of Toronto
ssanner@cs.toronto.edu

1 Overview

MDPs have become the *de facto* standard for modelling decision-theoretic planning problems. Recent work on MDPs has focused primarily on two research areas:

1. *Language extensions for MDP models.* The language used to specify an MDP determines how succinctly a given domain can be described. In addition, the structure used in succinct model specification often has a direct impact on the design of algorithms that can efficiently exploit this structure.
2. *Exploiting MDP structure for efficient solution algorithms.* There are many types of MDP structure that can be exploited by solution algorithms to avoid full state and action enumeration. By exploiting this structure, solution algorithms can scale to MDPs that would be otherwise unsolvable with full state and action enumeration.

In recent years, *first-order MDPs (FOMDPs)* (Boutilier, Reiter, & Price 2001) have become a popular formalism for modelling decision-theoretic planning problems, owing to their ability to succinctly represent planning problems stated as stochastic variants of STRIPS (Fikes & Nilsson 1971) or PDDL (McDermott *et al.* 1998). In this thesis summary, I outline current work and future directions for research in the areas of language extensions and exploitation of structure to efficiently represent and solve first-order decision-theoretic planning problems.

2 Previous and Current Research

To date, previous and current research has focused primarily on exploiting various types of structure that naturally occurs in decision-theoretic planning problems.

2.1 Exploiting Independence for Exact Solutions

Previous work (Boutilier, Dearden, & Goldszmidt 2000; Hoey *et al.* 1999) for solving propositionally factored MDPs has used tree and ADD (R.I. Bahar *et al.* 1993) data structures to exploit context-specific independence (Boutilier *et al.* 1996). Very recent work on affine extensions of ADDs (Sanner & McAllester 2005) has provided a data structure that extends ADDs to compactly exploit additive, multiplicative, and context-specific independence in MDP inference. Empirical results suggest that the AADD outperforms both traditional tabular representations and ADDs on a variety of problems containing additive and multiplicative structure.

In another vein, current research on *first-order* ADDs (FOADDs) has examined methods for generalizing the ADD data structure from propositional to first-order representations. This work allows decision diagram nodes to consist of full first-order formulae that are lexicographically ordered with respect to a fixed relation ordering. Then, ordered resolution can be efficiently applied during a generalization of the standard ADD $Reduce(\cdot)$ and $Apply(\cdot)$ operations to prune out inconsistent nodes in the resulting decision diagram. With the FOADD data structure and algorithms properly defined, it is then straightforward to extend them to a first-order affine ADD (FOAADD), thus allowing FOMDP solution algorithms to exploit context-specific, additive, and multiplicative structure. Initial empirical results with the FOADD approach for value iteration have been promising, showing a marked increase in performance over a simple enumerated case representation. However, more comprehensive experimentation with both FOADDs and FOAADDs is needed to fully evaluate the impact of each of these data structures on the efficiency of FOMDP inference algorithms.

2.2 Exploiting Basis Function Representations for Approximate Solutions

Given the complexity of solving first-order MDPs, approximation techniques offer an efficient alternative to finding an exact solution. One popular method for approximating MDP solutions has been to represent the value function as a linear combination of weighted basis functions. Previous work (Guestrin *et al.* 2002; Schuurmans & Patrascu 2001) has provided efficient solution techniques for finding good sets of basis function weights by exploiting the structure of propositionally factored MDPs.

More recent work on first-order approximate linear programming (FOALP) solution approaches (Sanner & Boutilier 2005) has extended these basis function techniques to efficiently solve for approximate value functions in FOMDPs. This work represents the value function as a weighted set of first-order basis functions. It then uses a first-order generalization of cost network maximization in

tandem with constrain generation techniques to efficiently solve for settings of these weights. Empirical results have shown that these techniques are relatively efficient in practice and yield policies that outperform both handcoded heuristics and myopically optimal policies. In addition, these solutions yield error bounds on policy quality that apply equally to all domain instantiations of a problem – a novel result for the relational and first-order MDP literature.

However, despite the successes of the FOALP work, there are many future refinements to this work that could further improve results. One interesting question for future work is whether the uniform relevance weighting of partitions currently used in the FOALP objective is the best approach. It would be informative to explore alternative FOALP objective specifications and to evaluate their impact on value function quality over a variety of domains. In addition, it is an interesting question as to whether dynamic reweighting schemes could improve solution quality by identifying state partitions with large error and adjusting their relevance weights so they receive more emphasis on the next LP iteration. Altogether, such improvements to FOALP could bolster an already promising approach for efficiently and compactly approximating FOMDP value functions.

3 Future Research Directions

Future research directions build on current research directions and additionally look at first-order MDP modelling language extensions and the generalization of solution algorithms to handle such extensions.

3.1 Modelling Language Extensions

Perhaps one of the most important frontiers for FOMDP and general decision-theoretic planning research is that of providing the user with a rich set of language features that enable them to naturally model real-world problems. Following are a number of modelling language extensions that were motivated by actual planning problems along with a brief discussion of the modifications to solution algorithms to handle such extensions.

Sum and Counting Aggregators In first-order domains, it is often very natural to predicate transition function and reward dependencies on the count of objects satisfying some criteria. For example, in a logistics domain, the probability that a delivery truck leaves on time may depend on the number of packages being loaded on the truck, and the reward might be the count of packages successfully delivered. While counting and inequalities can be stated for *specific* values within first-order logic, *generic* counting for potentially infinite domains cannot be done without augmentations of the FOMDP specification language. For example, using a count aggregator $\#_p\ \phi(p)$ which counts the number of instantiations of p that make $\phi(p)$ true, one can easily state a reward that scales with the number of packages successfully delivered: $\#_p\ [Package(p) \wedge Delivered(p)]$. However, there is no finite representation of such a reward for potentially infinite domains when using only standard first-order quantifiers; Such a specification would need to provide a condition and corresponding reward value for every possible count of packages.

The specification of sum and counting aggregators and the necessary extension of the regression operator is straightforward for FOMDPs. Thus, a general research approach for solving FOMDPs with sum and count aggregators would rely on defining the general backup operators and a full dynamic programming (DP) solution algorithm. However, solving such extensions of FOMDPs with a full DP algorithm presents a number of technical complications. Specifically, *simplification* and *consistency checking* are needed by full DP solution algorithms and the complex interaction between quantifiers and sum aggregators makes both of these operations very difficult.

Consequently, once the full DP solution to FOMDPs with sum and count aggregators has been defined, this definition could be applied to a more tractable approximate solution approach using basis functions. Basis function solution techniques would pose an elegant and tractable solution approach since they do not require that formulae be simplified and since sampling techniques can be used to reduce the sum aggregators to first-order formulae for which consistency checking is straightforward. Although sampling and the avoidance of simplification considerably blowup the representation, these solution methods discard this representational blowup and project the value function down to a set of basis function weights, thus maintaining compactness.

Handling Quantity Sum and counting aggregators are useful for counting domain objects satisfying some criteria, but discrete and continuous quantities can also be represented directly as relational attributes, e.g. $hasPackages(Paris, 5)$. While this specification does not allow one to model the specific properties of individual elements contributing to the quantity, it is a commonly used construct in planning domains and is simple to formalize in a FOMDP using the $+$ and $-$ arithmetic functions and equality/inequality predicates. The primary difficulty with reasoning in such domains is introducing relevant rules of inference for performing tractable inconsistency detection with the intended interpretation of these additional language elements.

Topological Structure Topological structure occurs commonly in a number of planning domains, especially those with underlying location constraints. For example, logistics problems may require that trucks can only travel on certain roads to reach different cities, and that planes can only fly to certain airports in certain cities. While these domains can be formalized in FOMDPs for *arbitrary* underlying topologies, the solutions to such FOMDPs are often intractable as they have to take into account every possible topology for a potentially infinite number of locations.

When considering problems with an underlying topology, it is reasonable to assume that the topology is fixed and to solve the FOMDP with respect to that specific topology. In doing this, one can then make use of efficient graph algorithms in place of first-order reasoning during the FOMDP solution process. Such techniques have the capacity to yield efficient solutions for problems with underlying topologies

– solutions which would otherwise prove difficult with first-order reasoning techniques alone.

Concurrent Actions Many real-world planning domains allow multiple non-interfering actions to be executed simulataneously. While this can currently be done within the FOMDP framework, it involves specifying primitive actions corresponding to all possible joint action combinations that could take place. This approach is inefficient in that it requires an inordinately (if not infinitely) large number of joint actions, but also the specification of positive and negative effects for each of these actions. What is needed for efficient reasoning with concurrent actions is an efficient method for factoring both the effects and the value of simultaneously executed primitive actions in order to tractably deal with the combinatorial explosion of potential action executions. Two potential sources of ideas for this work are ConGolog (De Giacomo, Lesperance, & Levesque 2000) which specifies transition semantics for situation calculus domains with concurrent and exogenous actions, and work in MDPs (Meuleau *et al.* 1998) that deals with weakly coupled MDPs, each MDP having its own set of actions and local effects.

Program Constraints Quite often, one has a good idea of the general sequence of actions that an agent should follow and there are simply a few choice points which should be left to the agent to decide according to some decision-theoretic criterion. In this case, it is useful to extend FOMDP solution techniques to handle program constraints such as those specified by DT-GOLOG (Boutilier *et al.* 2000). The primary approach to solving a FOMDP under such constraints should be a relatively straightforward extension of the hierarchical abstract machine (HAM) framework (Parr & Russell 1998; Andre & Russell 2001; 2002) for solving MDPs. However, modifications will be required to generalize this technique to first-order state spaces.

3.2 Efficient Approximation Algorithms

There appear to be two distinct approaches taken by FOMDP solution algorithms: extensions of value and policy iteration for MDPs and extensions of linear programming techniques for basis function approximations of MDP value functions. Each of these techniques has its own advantages and disadvantages and thus it is worth examining possible extensions that can be made to both classes of algorithms.

Approximation Techniques that Exploit Independence Value iteration techniques based on FOADDs and FOAADDs hold the promise of yielding efficient exact solutions to FOMDPs. However, it is also interesting to look at approximation extensions of these algorithms in the flavor of the APRICODD (St-Aubin, Hoey, & Boutilier 2000) extension of SPUDD (Hoey *et al.* 1999). In brief, APRICODD approximates an MDP value function by interleaving SPUDD value iteration steps with an approximation step that prunes nodes from the value function ADD in order to maintain a tractable representation. If the approximation is carried out so that it keeps track of the minimum and maximum bounds for the value function, then it is still possible to achieve convergence of the approximated value function under certain conditions. Following this work, APRICODD-style extensions to SPUDD could likewise be generalized to FOMDP value iteration algorithms based on FOADDs and FOAADDs, thus creating a new class of FOMDP approximate solution algorithms.

There are two main research questions for such extensions that have not been considered yet. First, while efficient ADD-based value function approximation techniques have been explored in APRICODD, approximation with AADDs is still an open research area. AADDs pose a number of difficulties for approximation techniques, namely that the properties of the data structure do not permit the direct extension of methods used for approximation with ADDs. Second, while efficient MDP value function approximation techniques have been explored for propositional MDPs, it is an open question as to whether these same techniques will apply to FOMDPs or whether modifications must be made that take into consideration the first-order structure found in FOADDs and FOAADDs. Both of these questions will need to be resolved for efficient FOMDP approximate solution techniques based on FOAADDs.

Alternate Basis Function Approximation Approaches Currently, only the first-order approximate linear programming (ALP) (Sanner & Boutilier 2005) approach to approximating the FOMDP value function as a linear combination of basis functions has been considered. However there are two alternate approaches to finding basis function weights that have also been considered in the propositional MDP literature. These two methods are approximate value iteration (AVI) and approximate policy iteration (API), the latter having been identified as typically offering higher quality solutions than ALP.

While it is relatively straightforward to define the first-order extensions of the propositional versions of AVI and API in the spirit of the extension for ALP, this leaves a number of computational issues which have posed problems for such extensions. Both AVI and API require that a policy be derived at each step for their respective weight projection tasks.[1] However, naive methods for deriving a policy from a set of Q-functions have turned out to yield extremely large policies and have proved intractable to work with in practice. Before AVI and API can be effective techniques for approximate FOMDP solutions, a compact method for representing the policy and computing it must be derived. For now, one promising approach appears to involve structuring the policy using a FOADD or FOAADD to avoid unnecessary redundancy and to provide as compact a representation of the policy as possible.

References

Andre, D., and Russell, S. 2001. Programmable reinforcement learning agents. In *In Advances in Neural Informa-*

[1] While the fact that a policy needs to be derived for API is obvious, this may not be the case for AVI. However it turns out that given the infinite action space of FOMDPs, deriving the $\max$ over the Q-functions inherently requires carrying out an operation equivalent to finding the policy for a given value function.

tion Processing Systems, volume 13.

Andre, D., and Russell, S. 2002. State abstraction for programmable reinforcement learning agents. In *In Proc. AAAI-02*. Edmonton, Alberta: AAAI Press.

Boutilier, C.; Friedman, N.; Goldszmidt, M.; and Koller, D. 1996. Context-specific independence in Bayesian networks. In *UAI 96*, 115–123.

Boutilier, C.; Reiter, R.; Soutchanski, M.; and Thrun, S. 2000. Decision-theoretic, high-level agent programming in the situation calculus. In *AAAI 00*, 355–362.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 2000. Stochastic dynamic programming with factored representations. *Artificial Intelligence* 121:49–107.

Boutilier, C.; Reiter, R.; and Price, B. 2001. Symbolic dynamic programming for first-order MDPs. In *IJCAI 01*, 690–697.

De Giacomo, G.; Lesperance, Y.; and Levesque, H. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121(1–2):109–169.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *AI Journal* 2:189–208.

Guestrin, C.; Koller, D.; Parr, R.; and Venktaraman, S. 2002. Efficient solution methods for factored MDPs. *JAIR*.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. In *UAI 99*, 279–288.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL—The planning domain definition language.

Meuleau, N.; Hauskrecht, M.; Kim, K.-E.; Peshkin, L.; Kaelbling, L. P.; Dean, T.; and Boutilier, C. 1998. Solving very large weakly coupled Markov decision processes. In *AAAI 98*, 165–172.

Parr, R., and Russell, S. 1998. Reinforcement learning with hierarchies of machines. In M. Jordan, M. K., and Solla, S., eds., *Advances in Neural Information Processing Systems 10*. Cambridge: MIT Press. 1043–1049.

R.I. Bahar; E.A. Frohm; C.M. Gaona; G.D. Hachtel; E. Macii; A. Pardo; and F. Somenzi. 1993. Algebraic Decision Diagrams and Their Applications. In *IEEE /ACM International Conference on CAD*.

Sanner, S., and Boutilier, C. 2005. Approximate linear programming for first-order mdps. In *UAI 2005*.

Sanner, S., and McAllester, D. 2005. Affine algebraic decision diagrams (aadds) and their application to structured probabilistic inference. In *IJCAI 2005*.

Schuurmans, D., and Patrascu, R. 2001. Direct value approximation for factored MDPs. In *Advances in Neural Information Processing Systems 14 (NIPS-2001)*. to appear.

St-Aubin, R.; Hoey, J.; and Boutilier, C. 2000. APRICODD: Approximate policy construction using decision diagrams. In *Advances in Neural Information Processing Systems 13 (NIPS-2000)*, 1089–1095.

Say "No" to Grounding: An Inference Algorithm for First-Order MDPs

Olga Skvortsova
International Center for Computational Logic
Technische Universität Dresden
skvortsova@iccl.tu-dresden.de

Abstract

We propose an algorithm, referred to as ALLTHETA, for performing efficient domain-independent symbolic reasoning in a planning system FLUCAP that solves first-order MDPs. The computation is done avoiding vicious grounding.

Introduction

Markov Decision Processes (MDPs) are de facto standard representational and computational model for decision-theoretic planning problems. Recently, several compact representations for propositionally-factored MDPs have been proposed, including dynamic Bayesian networks (Boutilier, Dean, & Hanks 1999) and algebraic decision diagrams (Hoey *et al.* 1999). For instance, the SPUDD algorithm (Hoey *et al.* 1999) has been used to solve MDPs with hundreds of millions of states optimally, producing logical descriptions of value functions that involve only hundreds of distinct values.

Meanwhile, many realistic planning domains are best specified in first-order terms. However, most existing implemented solutions for first-order MDPs (FOMDPs) rely on grounding, i.e., eliminate all variables at the outset of a solution attempt by instantiating terms with all possible combinations of domain objects, e.g., (2002). This technique is very impractical because the number of propositions grows considerably with the number of domain objects and relations. This has a dramatic impact on the complexity of the algorithms that depends directly on the number of propositions. Moreover, as soon as the universe of objects is infinite, these algorithms cannot be made to work. Finally, systems for solving FOMDPs that rely on state grounding also perform action grounding which is problematic in first-order domains, because the number of ground actions also grows drastically with domain size.

To address these difficulties, we have recently proposed a first-order generalization of LAO* algorithm (Karabaev & Skvortsova 2005), referred to as FOLAO*, in which our contribution was to show how to perform heuristic search for FOMDPs, circumventing their grounding. In order to ensure first-order reasoning without descending to the propositional level, a planning system should be equipped with highly-optimized domain-independent inference algorithms that compute sets of successor and predecessor states of a given state wrt. a given action. Such inference algorithms rely on non-trivial symbolic computations as, e.g., unification or subsumption problem under some equational theory between two states specified as first-order terms.

In this paper, we develop an algorithm, referred to as ALLTHETA, that solves the subsumption problem under AC1[1] equational theory and delivers all possible substitutions. The computation is done avoiding aggressive grounding. ALLTHETA has been recently integrated into the planning system FLUCAP (Hölldobler, Karabaev, & Skvortsova 2006).

First-order Representation of MDPs

First, we propose a concise representation of FOMDPs within Probabilistic Fluent Calculus ($\mathcal{PFC}$). $\mathcal{PFC}$ is a logical approach to modelling dynamically changing and uncertain environments based on first-order logic (Hölldobler, Karabaev, & Skvortsova 2006).

MDPs An MDP is a tuple $(\mathcal{Z}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{C})$, where $\mathcal{Z}$ and $\mathcal{A}$ are finite sets of states and actions, resp.; $\mathcal{P} : \mathcal{Z} \times \mathcal{Z} \times \mathcal{A} \to [0, 1]$, written $\mathcal{P}(z'|z, a)$, specifies transition probabilities of reaching a state z' by executing a in z. $\mathcal{R} : \mathcal{Z} \to \Re$ is a real-valued reward function associating with each state z its immediate utility $\mathcal{R}(z)$. $\mathcal{C} : \mathcal{A} \to \Re$ is a real-valued cost function associating a cost $\mathcal{C}(a)$ to each action a. A solution of an MDP is a policy $\pi : \mathcal{Z} \to \mathcal{A}$ that maximizes the total expected discounted reward received when executing the policy π over an infinite horizon. The value of a state z with respect to a policy π is defined recursively as:

$$V_\pi(z) = \mathcal{R}(z) + \mathcal{C}(\pi(z)) + \gamma \sum_{z' \in \mathcal{Z}} \mathcal{P}(z'|z, \pi(z)) V_\pi(z'),$$

where $0 \leq \gamma < 1$ is a discount factor.

Probabilistic Fluent Calculus: Formally, let Σ denote a set of function symbols. We distinguish two function symbols in Σ, namely $\circ/2$ which is associative (A), commutative (C), and admits the unit element, and a constant 1. Let $\Sigma_{-} = \Sigma \setminus \{\circ, 1\}$. Non-variable Σ_{-}-terms are called fluents.

[1] A - associative, C - commutative, 1 - unit element.

Let $\mathcal{F}$ denote the set of fluents. Fluent terms are defined inductively as follows: 1 is a fluent term; each fluent is a fluent term; $F \circ G$ is a fluent term, if F and G are fluent terms.

A *state* is a fluent term. We assume that each fluent may occur at most once in a state, i.e., states of the form $euro \circ euro$ are disallowed. For example, a state $Z = on(X', Y') \circ on(Y', t) \circ cl(X') \circ e$ denotes that some clear block X' is on the block Y', which is on the table, the gripper is empty and something else might be also true. We note that the negation can be effortlessly included in the language (Hölldobler, Karabaev, & Skvortsova 2006). The interpretation over $\mathcal{F}$, denoted as $\mathcal{I}$, is the pair $(\Delta, \cdot^{\mathcal{I}})$, where the domain Δ is a set of all finite sets of ground fluents from $\mathcal{F}$; and an interpretation function $\cdot^{\mathcal{I}}$ which assigns to each state Z a set $Z^{\mathcal{I}} = \{d \in \Delta | \exists\theta.(Z \circ U)\theta =_{AC1} d\}$, where θ is a substitution and U is a new AC1-variable. Thus, states in $\mathcal{PFC}$ represent clusters of individual states. In this way, they embody a form of state space abstraction, referred to as first-order state abstraction, and, hence, can be treated as abstract states. E.g, the state $z_1 = on(b, c) \circ on(c, t) \circ cl(b) \circ e \circ cl(f)$, where t stands for table and b, c and f are blocks, is represented by the abstract state Z above; whereas $z_2 = on(b, c)$ is not, since other three 'mandatory' fluents of Z are missing in z_2. In essence, abstract states are defined under incomplete semantics, viz., other fluents that are not explicitly present in the state description might also hold, as e.g., $cl(f)$ appears in the state $z_1 \in Z^{\mathcal{I}}$.

Actions are first-order terms leading with an action function symbol. For example, the action of picking up some block X from another block Y might be denoted as *pickup* (X, Y). Stochastic actions are described via decomposition into deterministic primitives under nature's control, referred to as nature's choices. E.g., action *pickup* (X, Y) can be defined by means of successful *pickupS* (X, Y) and failure *pickupF* (X, Y) nature's choices. Preconditions and effects of an action a, denoted as *Pre* (a) and *Eff* (a), respectively, are abstract states. E.g., for preconditions and effects of the action *pickupS* (X, Y), we have: *Pre* $(\textit{pickupS}(X, Y)) := on(X, Y) \circ cl(X) \circ e$ and *Eff* $(\textit{pickupS}(X, Y)) := h(X)$, where $h(X)$ stands for the fact of holding a block X. Probabilities of each nature's choice, rewards and action costs can be defined in an obvious way.

An Inference Algorithm for FOMDPs

Systems for solving FOMDPs that rely on state grounding also perform action grounding which is problematic in first-order domains, because the number of ground actions grows drastically with domain size. Herein, we show how to perform inferences, i.e., compute successors and predecessors of a given abstract state, with action schemata directly, avoiding unnecessary grounding.

For this, an inference problem of finding all a-successors (all a-predecessors) of an abstract state Z is represented in terms of the AC1-unification problem[2], referred to as AC1-UNIFY(Z_1, Z_2), where Z_1 represents the preconditions (effects) of a and $Z_2 = Z$. AC1-UNIFY(Z_1, Z_2) is defined by: $\exists\theta.\ (Z_1 \circ U)\theta =_{AC1} (Z_2 \circ W)\theta$, where U and W are new AC1-variables.

Intuitively, an action a is applicable to an abstract state Z iff it is applicable to *all* individual states that constitute $Z^{\mathcal{I}}$. In order to determine all fragments of Z, an action a is applicable to, we compute all solutions for the following AC1-unification problem: $(\textit{Pre}\,(a) \circ U)\theta =_{AC1} (Z \circ W)\theta$. In this way, the bindings for W define the fragments $Z^i = (Z \circ W)\theta$ of Z, an action a is applicable to. Moreover, the bindings for U allow us to construct the successors of Z^i, i.e., $Z^i_{succ} := (\textit{Eff}\,(a) \circ U)\theta$. In essence, in order to compute the set of all a-successors of all fragments of Z, a is applicable to, it is enough to find all solutions θ for the above AC1-unification problem.

In this work, we present a restricted case of AC1-unification, i.e., AC1-subsumption, referred to as AC1-SUBSUME(Z_1, Z_2), where $(Z_2 \circ W)\theta = Z_2$:

$$\exists\theta.\ (Z_1 \circ U)\theta =_{AC1} Z_2\ .$$

There are at least two applications of AC1-SUBSUME(Z_1, Z_2) in the FOLAO* algorithm. First, for detecting a more specific abstract state between Z_1 and Z_2, that can be removed from the state space thereafter. Second, for computing a set of *all* states that are reachable from an initial state wrt. all actions.

In the following, we exploit the fact that the AC1-subsumption problem is a specialization of the θ-subsumption problem on general clauses, since abstract states are Horn clauses with empty head (Scheffer, Herbrich, & Wysotzki 1996). The θ-subsumption problem for clauses C and D is a problem of whether there exists a substitution θ such that $C\theta \subseteq D$ (or, in our terms, $(C \circ U)\theta =_{AC1} D$).

In general, θ-subsumption is NP-complete (Scheffer, Herbrich, & Wysotzki 1996). It is known that deterministic subsumption, i.e., when there exists an ordering of fluents, such that in each step there is a fluent which has exactly one match that is consistent with the previously matched fluents, can be solved in polynomial time. Unfortunately, in general, there are only few, or none at all, fluents in a state that can be matched deterministically.

Following (Scheffer, Herbrich, & Wysotzki 1996), we have developed two approaches to reduce the complexity of non-deterministic θ-subsumption, and hence, AC1-subsumption. Both approaches have been reconciled in an algorithm, referred to as ALLTHETA, that returns all solutions for the AC1-subsumption problem.

Phase one: context-based subsumption. One approach is context-based matching candidate elimination. In general, a fluent f in an abstract state Z_1 can be matched with several fluents in an abstract state Z_2, that are referred to as matching candidates of f. The approach is based on the idea that fluents in Z_1 can be only matched to those fluents in Z_2, the context of which include the context of the fluents in Z_1. The context is given by occurrences of identical variables or chains of such occurrences and is defined up to some fixed depth. In effect, matching candidates that do not meet the above context condition can be effortlessly pruned. In most cases, such pruning results in deterministic

[2] AC1-unification problem is a unification problem under the equational theory AC1.

subsumption, thereby considerably extending the tractable class of abstract states. Deterministic subsumption that exploits the context information is referred to as context-based deterministic subsumption.

For example, two abstract states $Z_1 = on(X,Y) \circ on(Y,t)$ and $Z_2 = on(a,b) \circ on(b,c) \circ on(c,t) \circ on(d,t)$ cannot be subsumed deterministically because each fluent in Z_1 has more than one matching candidate in Z_2. However, exploiting the context information already at depth 1 enables us to conclude that Z_1 subsumes Z_2. At depth 1, the context of $on(X,Y)$ contains the path $on \cdot 2 \rightarrow 1 \cdot on$, i.e., a variable Y appears at position 2 in $on(X,Y)$ and at position 1 in $on(Y,t)$. The context of $on(Y,t)$ contains the path $on \cdot 1 \rightarrow 2 \cdot on$, i.e., the variable Y appears at position 2 in $on(X,Y)$ and at position 1 in $on(Y,t)$. The contexts of the fluents in Z_2 are $\{on \cdot 2 \rightarrow 1 \cdot on\}$, $\{on \cdot 1 \rightarrow 2 \cdot on, on \cdot 2 \rightarrow 1 \cdot on\}$, $\{on \cdot 1 \rightarrow 2 \cdot on, on \cdot 2 \rightarrow 2 \cdot on\}$ and $\{on \cdot 2 \rightarrow 2 \cdot on\}$, resp. The fluent $on(Y,t)$ has two matching candidates, viz., $on(c,t)$ and $on(d,t)$. Since the context of $on(Y,t)$ can only be embedded in the context of $on(c,t)$, the matching candidate $on(d,t)$ is excluded and $on(Y,t)$ can be matched deterministically. Then, the matching substitution $\mu_1 = \{Y \mapsto c\}$ is applied to Z_1. As a result, the fluent $on(X,Y)\mu_1 = on(X,c)$ can be matched deterministically to $on(b,c)$ with $\mu_2 = \{X \mapsto b\}$. Hence, both fluents can be matched deterministically and the substitution $\theta = \mu_1\mu_2$ was found without backtracking.

There is a well-known tradeoff. The deeper inside the abstract state we look, thus devoting the considerable effort for computing the context itself, the higher the pruning rate is. Alternatively, if the depth value is underestimated, we save time and space for constructing the context but end up with a larger search space. Very often, the optimal depth has the value of 2.

Phase two: ALL-CLIQUES**.** In some cases, however, after performing the context-based deterministic subsumption, there still remain some fluents that cannot be matched deterministically. Thus, a remaining space of matching candidates has to be searched for a substitution. For this, a second approach that reduces the complexity of non-deterministic AC1-subsumption, referred to as ALL-CLIQUES, has been developed. ALL-CLIQUES is a modified version of its ancestor CLIQUE (Scheffer, Herbrich, & Wysotzki 1996), where all cliques are computed and additional pruning techniques have been developed in order to alleviate the search for substitutions.

ALL-CLIQUES exploits a well-known correspondance between the AC1-subsumption problem and the clique problem, i.e., a problem of finding a clique[3] of the fixed size in a graph. More precisely, an abstract state Z_1 subsumes an abstract state Z_2 iff there is a clique of size $|Z_1|$ in the space of matching candidates for fluents in Z_1. By the size $|Z|$, we mean the number of fluents comprising Z. The candidates that do not form a clique can be effortlessly excluded from the search space.

We start with constructing a substitution graph (V,E) for abstract states Z_1 and Z_2 with nodes $v = (\mu, i) \in V$, where

[3]A clique in a graph is a set of pairwise adjacent nodes.

Function `findPath(`V, E, *Paths*, v, *currPath*, i`)`

```
1  if valid(v) then
2      currPath:=currPath∪{v}
3      if i = |Z1| then
4          Paths := Paths ∪ {currPath}
5      else
6          foreach u = (μ', i+1) ∈ V with (v,u) ∈ E do
7              if clique(u, currPath) then
8                  findPath(V, E, Paths, u, currPath,
                   i+1)
9  else V := V \ {v}
10 return Paths
```

μ matches some fluent at position i in Z_1 to some fluent in Z_2 and $i \geq 1$ is referred to as a layer of v. Two nodes (μ_1, i_1) and (μ_2, i_2) are connected with an edge iff $\mu_1\mu_2 = \mu_2\mu_1$ and $i_1 \neq i_2$.

ALL-CLIQUES returns all paths *Paths* in the graph (V,E) that start at the first layer and form a clique of size $|Z_1|$. Its core is the function FINDPATH. If `valid(`v`)` is true, i.e., v has at least one edge to each layer, v is added to the current path *currPath*. If v is located at the last layer then *Paths* is updated with the *currPath*. Otherwise, if a next-layer neighbour u of v forms a clique with the nodes in *currPath*, i.e., `clique(`u, *currPath*`)` holds in line 7, then `findPath` is called recursively for u. The removal of invalid nodes in line 9 is a distinct feature of ALL-CLIQUES, which was not introduced before. Another important pruning technique, employed in ALL-CLIQUES, relies on the idea of a layered substitution graph. In contrast to (Scheffer, Herbrich, & Wysotzki 1996), we organize a substitution graph in layers, i.e., each node $v = (\mu, i) \in V$ belongs to a layer i. The layers should be visited in the order of their appearance. The layered architecture of the substitution graph is a natural way to avoid duplicate occurrences of the same clique in the set of all cliques. In effect, context-based determinacy and ALL-CLIQUES are combined into an algorithm, referred to as ALLTHETA, that delivers all substitutions for the AC1-SUBSUME(Z_1, Z_2) problem.

Experimental Evaluation

We demonstrate the advantages of using the context information for efficient domain-independent symbolic reasoning in FOMDPs on a system, referred to as ALLTHETA. ALLTHETA has been recently integrated as a module into the FLUCAP 1.1 planning system, that is a successor of FLUCAP 1.0 (Hölldobler, Karabaev, & Skvortsova 2006) that has entered the probabilistic track of the International Planning Competition IPC'2004. The experimental results were all obtained using a Linux RedHat machine running at 2.4 GHz Intel Celeron with 1 Gb of RAM.

Table 1 presents the comparison results of ALLTHETA with the system FASTTHETA (Ferilli *et al.* 2003) on the CBW dataset. CBW stems from the colored Blocksworld scenario that was first introduced during the IPC'2004. CBW is, currently, one of a few probabilistic scenarios that are represented in first-order terms and, hence, enable to make use

B	C	Total time, sec.						
			AllTheta					
		FTheta	d=0	d=1	d=2	d=3	d=4	d=5
5	3	0.5	2.9	0.4	0.3	0.3	0.4	1.0
	4	0.4	2.0	0.3	0.2	0.2	0.3	0.6
	5	0.4	1.7	1.3	0.2	0.2	0.2	0.5
10	3	1.5	44.7	1.1	0.5	0.5	1.0	4.3
	4	1.1	22.4	1.1	0.4	0.4	0.5	1.4
	5	0.9	13.5	1.0	0.5	0.5	0.8	3.1
15	3	3.9	n/a	2.3	0.9	0.9	1.7	7.7
	4	3.5	243.3	2.4	0.8	0.9	2.0	10.6
	5	2.8	84.7	2.0	0.7	0.7	1.2	4.9
20	3	8.7	n/a	10.1	4.6	3.1	4.2	15.7
	4	9.2	n/a	3.3	1.1	1.0	1.8	8.5
	5	7.3	n/a	3.0	1.0	1.1	2.1	11.6
25	3	16.5	n/a	7.2	2.0	1.8	4.1	28.3
	4	17.1	n/a	7.8	1.8	1.7	4.2	30.7
	5	15.7	n/a	7.3	1.7	1.8	4.2	34.0
50	3	164.9	n/a	n/a	38.8	29.5	28.6	52.2
	4	201.1	n/a	186.8	33.0	26.0	27.9	42.7
	5	175.1	n/a	140.4	30.8	26.3	29.1	57.7
75	5	702.5	n/a	240.8	58.0	47.2	52.3	121.8
100	5	n/a	n/a	452.6	96.7	78.1	74.0	155.0

Table 1: Performance comparison of ALLTHETA (denoted as AllTheta) with FASTTHETA (denoted as FTheta) on the CBW dataset.

of symbolic reasoning. CBW differs from the classical case in that, along with the unique identifier, each block is assigned a specific color. A goal formula, specified in first-order terms, provides an arrangement of colors instead of an arrangement of blocks. FASTTHETA, that is motivated by the field of Inductive Logic Programming (ILP), can be applied to compute all solutions of the AC1-subsumption problem.

In the following, we motivate the importance of the context depth parameter. Altogether, there are 100 abstract states that lead to 10000 subsumption tests. The column labelled Total time presents the time needed to solve all of 10000 subsumption tests. A 30-mins slot is allocated for each problem. The cells marked with 'n/a' mean that the limit was exceeded. Each CBW problem is defined by a number B of blocks and a number C of colors.

In CBW case, on small problems of size up to 25 blocks, the depth parameter d posesses the optimal value of 2. Whereas, on larger problems, this value grows. This reflects the necessity to store an additional context information about the fluents in an abstract state. The special case of d=0 means that no context information is considered. In comparison to ALLTHETA, the runtime of FASTTHETA grows considerably faster in the size of a problem. For example, at depth of 2, for the five-colored 15, 25 and 75 problems, FASTTHETA is by factor of 4, 8 and 15 slower. As a result, it could scale to problems up to the size of 75 blocks only. Whereas, the limit of ALLTHETA comprises 360 blocks.

Neither FASTTHETA nor ALLTHETA are sensitive to the number of colors in a problem. In contrast, grounding-based reasoners are severely affected by this parameter. The timing results for a special case of d=0 demonstrate the dramatic loss in runtime in comparison even with the case of d=1, where the context information about the direct neighbours of a fluent is counted.

Most importantly, present results indicate that the domain-independent inference algorithm ALLTHETA performs symbolic reasoning for first-order MDPs in about the same time as the domain-specific subsumption solver that was integrated in FLUCAP 1.0. We note that the latter reduces the AC1-subsumption problem to a quadratic variant of the subset problem. Whereas, the former solves the general case, which is NP-complete. For example, for a single subsumption test at depth of 2 in the problem of 15 blocks and 3 colors, ALLTHETA requires of about 92 microseconds. Whereas, for its domain-specific counterpart, the runtime comprises 85 microseconds. Finally, FASTTHETA has outperformed ALLTHETA by a factor of four, on the Mutagenesis dataset that is a classical ILP testbed.

Conclusions, Related and Future Work

We have proposed an algorithm, referred to as ALLTHETA, for performing automated domain-independent symbolic reasoning in FOMDPs. The construction is done avoiding grounding. In comparison to FASTTHETA, our approach scales better on larger FOMDPs. Some related approaches are known. For example, Django (Maloberti & Sebag 2004) is, nowadays, the fastest θ-subsumption checker that is based on the constraint satisfaction. Yet, it returns a binary answer 'yes/no' only and provides no solutions, even in the positive case. In (Kersting, van Otterlo, & de Raedt 2004), authors employ a generalized AC1-subsumption framework in the ReBel algorithm. ReBel treats abstract states as sets of fluents. Whereas, ALLTHETA can potentially work with multisets. We plan to incorporate these and disequalities into our setting. It is also important to extend our results towards the case of the AC1-unification problem.

Acknowledgements

We thank reviewers for their comments. Many thanks to Eldar Karabaev and Georg Rammé for fruitful discussions. This work is supported by the grant GRK 334 under auspices of DFG.

References

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural Assumptions and Computational Leverage. *JAIR* 11.

Feng, Z., and Hansen, E. 2002. Symbolic heuristic search for factored markov decision processes. In *AAAI*.

Ferilli, S.; Di Mauro, N.; Basile, T.; and Esposito, F. 2003. A complete subsumption algorithm. In *AI*IA*.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic Planning using Decision Diagrams. In *UAI*.

Hölldobler, S.; Karabaev, E.; and Skvortsova, O. 2006. FLUCAP: A heuristic search planner for first-order MDPs. *JAIR*. To appear.

Karabaev, E., and Skvortsova, O. 2005. A Heuristic Search Algorithm for Solving First-Order MDPs. In *UAI*.

Kersting, K.; van Otterlo, M.; and de Raedt, L. 2004. Bellman goes relational. In *ICML*.

Maloberti, J., and Sebag, M. 2004. Fast theta-subsumption with constraint satisfaction algorithms. *ML* 55(2).

Scheffer, T.; Herbrich, R.; and Wysotzki, F. 1996. Efficient θ-subsumption based on graph algorithms. In *ILP Workshop*.

Extending the Use of Plateau-Escaping Macro-Actions in Planning

Amanda Smith
Department of Computer and Information Sciences,
University of Strathclyde,
26 Richmond Street,
Glasgow, G1 1XH
email: amanda.smith@cis.strath.ac.uk

Abstract

Many fully automated planning systems use a single, domain-independent heuristic to guide search and no other problem-specific guidance. While these systems exhibit excellent performance, they are often out-performed by systems which are either given extra human-encoded search information, or spend time learning additional search control information offline. The benefit of systems which do not require human intervention is that they are much closer to the ideal of autonomy. This document discusses a system which learns additional control knowledge, in the form of macro-actions, during planning, without the additional time required for an online learning step. The results of various techniques for managing the collection of macro-actions generated are also discussed. Finally, an explanation of the extension of the techniques to other planning systems is presented.

Introduction

This document introduces the core ideas in the investigation into the management, and extension of use, of plateau-escaping macro-actions in planning. The ideas are explored in a planner called Marvin (Coles & Smith 2004), which competed in the Fourth International Planning Competition (IPC 4). In the competition Marvin generated *plateau-escaping macro-actions* and used them later in the search process to solve each problem. This investigation extends this approach to allow the caching of these macro-actions for use again in solving future problems, and the issues of library management this entails. Further ideas related to widening the applicability of plateau-escaping macro-actions to other heuristics and other planning technologies are also explored. Many previous approaches to macro-action generation and management require an offline learning step (Botea, Muller, & Schaeffer 2004) (or human intervention (Nau *et al.* 2003)) to generate a set of macro-actions which are then used, unchanged, in solving the remainder of the problem. This approach differs from existing approaches in that it does not require additional offline learning time to generate and filter macro-actions: they are filtered and selected dynamically.

Plateau Escaping Macro-Actions

The underlying search strategy used by Marvin is based on enforced hill-climbing as used in FF (Hoffmann & Nebel 2001). In the process of using EHC to perform forward-chaining heuristic search, guided by the RPG 'h+' heuristic, plateaux are encountered. Plateaux occur when a local minimum in the search space has been reached and all successor steps require either a sideways move (not changing the current heuristic value) or an uphill move (increasing the current heuristic value). It is these plateaux that are the core difficulty encountered when planning: it is relatively easy to make progress towards the goal when the heuristic is being informative; however, the exhaustive search performed to escape a plateau is expensive. On inspection of the steps required to escape plateaux in a given domain, it is often the case that the same sequence of actions is used to escape many plateaux, but with different parameter bindings.

As exhaustive search is required to escape from a plateau, construction of plateau-escaping action sequences is computationally expensive. Since plateau-escaping sequences often have similar structure, it is clear that memoising these action sequences for later use—when plateaux are once-again encountered—can potentially reduce planning time. Plateau-escaping action sequences are used to construct *plateau-escaping macro-actions*; these can be applied by the planner upon reaching later plateaux. The extraction of plateau-escaping macro-actions from plateau-escaping sequences is computationally inexpensive: the planner simply notes the sequence of actions applied since the start of the plateau. Generation of plateau-escaping sequences is expensive but since the search to find these sequences is being done anyway, to solve the planning problem, no additional search must be done to generate the macro-actions. The impact on planner performance caused by these macro-actions has been evaluated (Coles & Smith 2005).

Inferring Plateau-Escaping Macro-Actions

When the start of a plateau is detected—that is, when no successor state with a strictly-better heuristic value can be found—best-first search commences from the current state. During best-first search, each successor state stores the actions that have been applied to reach it since the start of the plateau: when a strictly-better state is eventually found, this list of actions is the plan segment that forms the basis of the plateau-escaping macro-action.

In order to make the macro-actions produced as useful and reusable as possible the plan segment is processed before

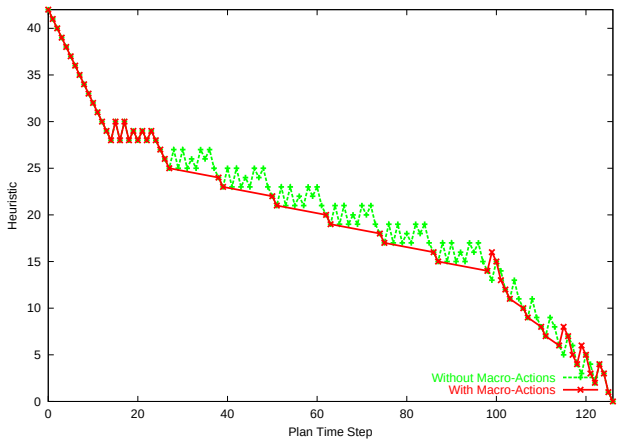


Figure 1: Heuristic Landscape over Makespan, With and Without Macro-Actions

being made into a macro-action. Any independent threads of execution that exist in the plan are separated to produce macro-actions involving as few entities as possible.

Caching Macro-Actions

Having solved one problem in a given domain it would be wasteful to disregard everything that has been learnt in doing so when going on to solve another problem in the same domain. Many of the plateau-escaping macro-actions provide a useful insight into the structure of the search landscape, under the RPG heuristic, across many problems in a given domain not just for a single problem. For example, figure 1 shows the heuristic profile across the solution plan, with and without plateau-escaping macro-actions, to a problem in the philosophers domain taken from IPC 4. The structure is common to all of the problems in the philosophers domain. It can clearly be seen that the macro-actions allow the planner to skip across plateaux exploiting the knowledge gained during previous search to solve a given problem. It would be useful to store macro-actions for use in solving future problems in the same domain. In doing this, however, the issue of increasing the branching factor in the search space by adding macro-actions becomes even more critical.

A large library of macro-actions is generated; a good pruning strategy is therefore essential. If the planner were to store all macro actions generated during the process of solving all problems, and then consider their application at every point during the search, the performance of the planner would almost certainly degrade significantly. One macro-action caching strategy was shown to improve planning performance over a configuration using no macro-actions and over the configuration generating macro-actions on a per-problem basis (Coles & Smith 2005).

Managing a Library of Cached Macro-Actions

Many machine learning techniques work by solving the problems with and without using a given feature, in this case macro-action, and compare the results obtained from both tests to decide whether or not that feature is beneficial. In this approach, however, since the goal is to do the learning online, without the need to solve additional problems, a different strategy is required. Online learning has two major benefits, the first being the removal of the requirement to solve additional problems; the second is that learning can be done not only on small instances (which may not accurately represent the larger, more interesting, problems) but also on the larger instances giving information that is potentially more useful.

Each problem is solved only once, this means that the conventional supervised learning techniques used by many systems cannot be used. Due to the nature of the macro-actions used, it is possible to do learning without the need to test the performance of the system with and without each macro-action. The only information available to the system is the number of times the macro-action has been used and the number of problems that have been solved since the macro-action was last used. The strategy for pruning will therefore be based on the usage statistics. If a macro-action is going to be rated based solely on usage statistics there must be a strong reason to believe that use of such a macro-action will improve planner performance. In the general case it may be that a macro-action is used frequently but does not greatly improve search performance: the heuristic may have led search quickly in the appropriate direction without the need for the macro-action. Plateau-escaping macro-actions, however, avoid exhaustive search; it is therefore far more reasonable to use the premise that the application of a plateau-escaping macro-action implies a time saving as a basis for this caching strategy. A similar observation was made by Minton (Minton 1985). The evaluation of the caching strategy itself will determine whether or not this is a reasonable premise on which to base a caching strategy.

The information available from the usage statistics of macro-actions is stored with each macro-action in the library: that is, the number of times the given macro-action has been used and the number of problems solved since the macro-action was last used. Three different approaches to caching macro-actions have been considered, each with various different parameters.

Search-Time Pruning

The most obvious approach to library management is simply to keep all of the macro actions ever generated. The advantage of this is that no useful macro-actions will ever be pruned as a result of an overly aggressive pruning strategy. The disadvantage is the the library of stored macro-actions will grow indefinitely and macro-actions which are of no use will be kept and may have to be considered during search. This has the potential to greatly increase the branching factor and make search to find a solution to the problem considerably more difficult.

Time-Out Pruning

The motivation for time-out pruning is to reduce the size of the library of macro-actions by removing those macro-actions which have not been useful in solving recent problems. The basis of this strategy is to remove macro-actions

that have not been used in solving the last n problems from the library. Some macro-actions are never re-used after being discovered once: such actions will be removed from the library reasonably quickly, thus only being present increasing the branching factor in a few problems.

The success of this strategy will clearly depend on the decided value of n, the *caching interval*. If the caching interval is very short, i.e. n is small, then potentially useful macro-actions may be discarded too hastily; if the caching interval is too large then it is likely that the planner will have to deal with large numbers of non-reusable macro-actions. The value of n that gives the best results varies on a domain-dependent basis; however, in order to create a fully automated system a single value of n must be decided upon. Investigations so far show that value giving the best perfomance considering all of the domains evaluated is $n = 2$.

Survival of the Fittest

The pruning strategy employed in this approach is to keep only the n most used macro-actions in the library. The larger the value of n is, the more the branching factor is potentially increased; the smaller the value of n, the greater risk of discarding useful macro-actions. The advantage of this approach over the time-out approach is that macro-actions that are useful in many problems will not be deleted from the library simply because they are not used in a (potentially short) run of adjacent problems. This approach also imposes an upper limit on the size of the library meaning that the increase in branching factor is more tightly controlled but a number of the best macro-actions can still be kept.

Investigating the Properties of Useful Macro-Actions

Many systems using macro-actions impose arbitrary limits on both the number of macro-actions to be used by the system at any given time and the maximum length of macro-actions. A flexible system that can dynamically select which macro-actions to use can, however, allow a thorough investigation of the characteristics of useful macro-actions without disregarding certain classes of macro-actions.

Figure 2 shows the mean number of uses of macro-actions of various lengths across a range of domains, most of which are taken from IPC 3 and IPC 4. Many previous approaches have been based on the idea that only macro-actions with short lengths should be used, without necessarily fully investigating usage statistics. It can be seen from the results that plateau-escaping macro-actions of length 2 are used the most frequently but it is interesting to note that macro-actions of other lengths are quite often used. Furthermore logic suggests that use of a longer plateau-escaping sequence will result in a greater performance improvement as more exhaustive search is potentially avoided. Of note is a very large macro-action, of length 71, that was generated in the blocksworld-4ops domain. Further experiments have shown that the use of this macro-action allows three extra problems to be solved within the 30 minute time limit. It is worth noting though that, despite this, performance is degraded slightly on some problems.

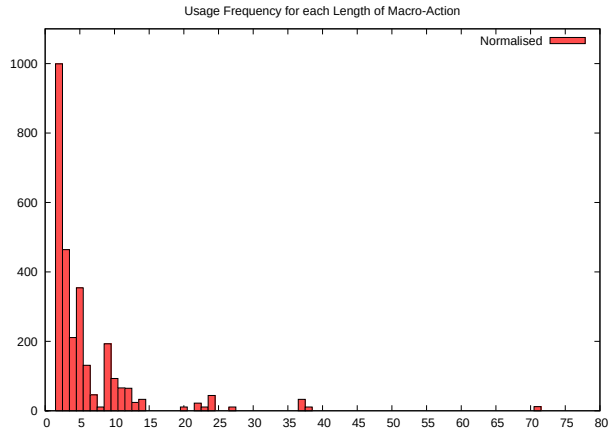


Figure 2: Normalised Usage Statistics for Differing Lengths of Macro-Actions

Generation and Use of Plateau-Escaping Macro-Actions Under Other Heuristics

Although this technique has been developed and discussed primarily using enforced hill-climbing under the RPG heuristic the idea can extend to other types of planning technology using different heuristics.

Generating Plateau-Escaping Macro-Actions for the Downward Heuristic

The idea of plateau-escaping macro-actions can be easily mapped to other heuristics used in forward chaining search. Different heuristics give rise to different search landscape profiles; it is, however, the case that all heuristics will have a weakness somewhere[1]. In forward chaining search this can often be characterised by the plateau-like situations observed in FF.

Experiments suggest that the generation of macro-actions can improve the performance of a planner using the downward heuristic to perform enforced hill-climbing. It is however often the case, contrary to the results generated using the RPG heuristic, that a version of the planner generating macro-actions on a per-problem basis generally performs better than a version caching these macro-actions for future use. This is due to the different nature of the two heuristics: the relaxation formed by the RPG heuristic is fixed and will model problems in the same domain in a similar manner. During the calculation of the downward heuristic, however, a step that breaks links based on dependencies is introduced, and the number of dependencies on a given graph-link varies between problems. It is therefore, often the case, that the downward heuristic will give a more different landscape between problems in the same domain, rendering caching of plateau-escaping macro-actions less effective.

[1]That is, of course unless a solution to the problem is to be used as a heuristic which is, of course, not useful: if a solution to the plan construction problem is found it is no longer necessary to search for one.

Macro-actions generated on the same problem, however, appear to give a greater performance improvement than using the equivalent technique under the RPG heuristic.

Further Work

The investigation is to be extended further to explore the wider applicability of plateau-escaping macro-actions. This includes use of plateau-escaping macro-actions, generated under the RPG heuristic, in planners that do not use this heuristic. The motivation for this is that the core hard part of a problem exists when solving it using any approach, and that part does not lie where the relaxed planning graph can accurately model the problem polynomially. Other work to be pursued in conjunction with Adi Botea and Andrew Coles is to investigate the observation that some macro-actions appear to improve search performance due to their role in improving accuracy of the heuristic estimate rather than their application during search.

The extraction of plateau-escaping macro-actions from solution plans for use in various planners is to be considered. Extraction of macro-actions from optimal plans is an extension of this approach: such macro-actions have the potential to reduce the makespan of plans by encouraging the planner to take a route that has previously resulted in an optimal solution.

References

Botea, A.; Muller, M.; and Schaeffer, J. 2004. Using component abstraction for automatic generation of macro-actions. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*, 181–190.

Coles, A., and Smith, A. 2004. Marvin: Macro-actions from reduced versions of the instance. IPC4 Booklet, Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004). Extended Abstract.

Coles, A. I., and Smith, A. J. 2005. On the inference and management of macro-actions in forward-chaining planning. In Tuson, A., ed., *Proceedings of the 24th UK Planning and Scheduling SIG*.

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

Minton, S. 1985. Selectively generalizing plans for problem-solving. In *In Proceedings of the Ninth International Joint Conference on Artificial Intelligence, IJCAI '85)*, 596–599.

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

Selecting Among Heuristics by Solving Thresholded k-Armed Bandit Problems

Matthew J. Streeter[1] and **Stephen F. Smith**[2]
Computer Science Department
and Center for the Neural Basis of Cognition[1] and
The Robotics Institute[2]
Carnegie Mellon University
Pittsburgh, PA 15213
{matts, sfs}@cs.cmu.edu

Abstract

Suppose we are given k randomized heuristics to use in solving a combinatorial problem. Each heuristic, when run, produces a solution with an associated quality or value. Given a budget of n runs, our goal is to allocate runs to the heuristics so as to maximize the number of sampled solutions whose value exceeds a specified threshold. For this special case of the classical k-armed bandit problem, we present a strategy with $O(\sqrt{np^*k \ln n})$ additive regret, where p^* is the probability of sampling an above-threshold solution using the best single heuristic. We demonstrate the usefulness of our algorithm by using it to select among priority dispatching rules for the resource-constrained project scheduling problem with maximal time lags (RCPSP/max).

1. Introduction

Suppose we are given a set of k randomized heuristics to solve a combinatorial optimization problem. Running the i^{th} heuristic produces a solution with an associated quality or value. With (unknown) probability $p_i(t)$, the value is $> t$. The value of $p_i(t)$ is instance-dependent, and the heuristics are black boxes whose only observable behavior is the value of the solutions they return. We would like to solve the following problem:

Problem 1: Given a budget of n runs, allocate runs among the heuristics so as to maximize the probability that a solution with value $> t_1$ is obtained.

Unfortunately, when solving Problem 1 we cannot do better than to select heuristics at random (no information about the probabilities p_i is gained until an acceptable solution has been found, at which point the information is useless). We will instead focus on the following related problem:

Problem 2: Given a budget of n runs, allocate runs among the heuristics so as to maximize the expected number of solutions with value $> t_2$.

The idea is that $t_2 < t_1$, that $n \cdot \max_i p_i(t_1)$ is prohibitively small, and that $n \cdot \max_i p_i(t_2)$ is small but not prohibitively so. If $\arg\max_i p_i(t_1) = \arg\max_i p_i(t_2)$ (i.e., the heuristic that is most likely to generate a nearly-acceptable solution is also most likely to generate an acceptable solution), then in solving Problem 2 we will also solve Problem 1. In practice, we have found that sets of heuristics for real combinatorial problems often have this property.

1.1. A Taxonomy of k-Armed Bandit Problems

In a k-armed bandit problem, we are faced with a set of k slot machines ("one-armed bandits"), each with a single arm. Each arm, when pulled, returns a payoff drawn from a fixed distribution over the interval $[0, 1]$. Given a budget of n pulls, we wish to allocate pulls so as to maximize some objective. We consider three variants of the problem, as summarized in the following table. The "thresholded" variant is new to this paper.

Problem	Objective to maximize
Classical	Total payoff
Max	Maximum payoff (from any single pull)
Thresholded	Number of payoffs that exceed a fixed threshold t.

Note that the thresholded k-armed bandit problem is a special case of the classical k-armed bandit problem where payoffs are drawn from $\{0, 1\}$.

We denote the mean payoff of the i^{th} arm by μ_i, and define $\mu^* = \max_i \mu_i$. For thresholded k-armed bandit problems, we denote by p_i the probability that a payoff from the i^{th} arm exceed the specified threshold, and define $p^* = \max_i p_i$.

1.2. Contributions

The contributions of this paper are twofold. First, we present an algorithm for the classical k-armed bandit problem with additive regret $O(\sqrt{n\mu^*k \ln n})$. When applied to the thresholded k armed bandit problem, our algorithm has additive regret $O(\sqrt{np^*k \ln n})$. Our algorithm has better regret than the algorithm of Auer, Cesa-Bianchi, and Fischer (2002b), which has additive regret $O(\sqrt{nk \ln n})$. Regrettably, our bound is slightly worse than that of the algorithm of Auer

et al. (2002a), which addresses a more general problem that the one considered here. We hope to address this discrepancy in future work.

Second, we demonstrate that an algorithm for the thresholded k-armed bandit problem can be profitably applied to the problem of selecting among priority dispatching rules for the RCPSP/max.

1.3. Related Work

The classical k-armed bandit problem was first studied by Robbins (1952) and has since been the subject of numerous papers; see Berry and Fristedt (1986) and Kaelbling (1993) for overviews. As discussed in §1.2, the algorithms of Auer et al. (2002a) and Auer, Cesa-Bianchi, and Fischer (2002b) are the most relevant to this paper.

Studies of the max k-armed bandit problem have much the same objectives as ours (Cicirello & Smith 2004; 2005; Streeter & Smith 2006). These works use ideas from *extreme value theory* to justify assumptions about the payoff distributions of each arm. In constrast, our work takes a non-parametric approach. We compare our algorithm to the max k-armed bandit algorithm of Cicirello and Smith (2005) in §4.

3. An Interval Estimation Algorithm

We will analyze the following procedure for solving (thresholded) k-armed bandit problems.

Procedure **IntervalEstimation** (n, δ):
1. Initialize $x_i \leftarrow 0$, $n_i \leftarrow 0$, and $u^i \leftarrow \infty$ for all $i \in \{1, 2, \ldots, k\}$.
2. Repeat n times:
   (a) $i^* \leftarrow \arg\max_i u^i$.
   (b) Pull arm i^*; increment x_i by the payoff received; and increment n_i.
   (c) $u^i \leftarrow U(x_i, n_i, \delta)$.

The function $U(x_i, n_i, \delta)$ returns a $1 - \delta$ upper confidence interval for p_i. More formally, for any parameter p_i (as well as any n_i, and δ), we are guaranteed that

$$\mathbb{P}_{p_i}[U(x_i, n_i, \delta) < p_i] \leq \delta \,. \tag{1}$$

The tightest possible upper bound U can be computed exactly using the binomial distribution. For the purposes of our analysis, it is easier to consider a weaker upper bound defined using Chernoff's inequality.

Chernoff's inequality. *Let $X = \sum_{i=1}^{n} X_i$ be the sum of n i.i.d. variables with $X_i \in \{0, 1\}$ and $\mathbb{P}[X_i = 1] = p$. Then for $\beta > 0$,*

$$\mathbb{P}\left[\frac{X}{n} > (1+\beta)p\right] < \exp\left(-\frac{\mu\beta^2}{3}\right)$$

and

$$\mathbb{P}\left[\frac{X}{n} < (1-\beta)p\right] < \exp\left(-\frac{\mu\beta^2}{2}\right).$$

Lemma 1. *The function U defined by*

$$U(x_i, n_i, \delta) = \max\{p_i | f(p_i, x_i, n_i) > \delta\}$$

where

$$f(p_i, x_i, n_i) = \exp\left(-\frac{1}{2} n_i p_i \left(1 - \frac{x_i}{n_i p_i}\right)^2\right)$$

satisfies condition (1).

Proof. Omitted. □

We first establish a bound on the number of times a suboptimal arm will be sampled.

Lemma 2. *With probability at least $1 - n\delta$, each arm $i \in \{1, 2, 3, \ldots, k\}$ will be sampled at most $\frac{6}{p^*}(1 - \sqrt{\alpha_i})^{-2} \ln(\frac{1}{\delta})$ times, where $\alpha_i = \frac{p_i}{p^*}$.*

Proof. Omitted. □

Theorem 1. *Running interval estimation for n trials with parameter $\delta = \frac{1}{n^3}$ yields at least*

$$s - 6\sqrt{2s(k-1)\ln(n)} - \frac{1}{n}$$

above-threshold payoffs in expectation, where $s = np^*$.

Proof. We consider only the special case $k = 2$. The proof for general k is similar.

Assume $p_1 = p^*$ and let $p_2 = \alpha p_1$, where $\alpha < 1$. By Lemma 2, we sample arm 2 at most $\min\{n, \frac{6}{p_1}(1 - \sqrt{\alpha})^{-2} \ln(\frac{1}{\delta})\}$ times, so (with probability at least $1 - n\delta$) expected regret is at most

$$p_1(1-\alpha) \min\left\{n, \frac{6}{p_1}(1-\sqrt{\alpha})^{-2} \ln\left(\frac{1}{\delta}\right)\right\} .$$

For $\alpha < 1$, we have

$$\begin{aligned} \tfrac{1-\alpha}{(1-\sqrt{\alpha})^2} &= \tfrac{1-\alpha}{(1-\sqrt{\alpha})^2} \cdot \tfrac{(1+\sqrt{\alpha})^2}{(1+\sqrt{\alpha})^2} \\ &= \tfrac{(1+\sqrt{\alpha})^2}{1-\alpha} \\ &< \tfrac{4}{1-\alpha} \,. \end{aligned}$$

Thus the expected regret is at most

$$\min\left\{s\Delta, \frac{24}{\Delta} \ln\left(\frac{1}{\delta}\right)\right\}$$

where we define $\Delta = 1 - \alpha$. Solving the equation $s\Delta = \frac{24}{\Delta} \ln(\frac{1}{\delta})$ gives $2\sqrt{\frac{6}{s} \ln(\frac{1}{\delta})}$ as the value of Δ that maximizes expected regret. So the expected regret is at most $2\sqrt{6s \ln(\frac{1}{\delta})} = 6\sqrt{2s \ln(n)}$.

With probability $n\delta = \frac{1}{n^2}$, Lemma 2 cannot be applied. Because regret can never exceed n, this increases expected regret by at most $\frac{1}{n}$. □

4. Experimental Evaluation

To evaluate the practical value of our interval estimation algorithm, we use it to select among randomized priority dispatching rules for the resource-constrained project scheduling problem with maximal time lags (RCPSP/max). Briefly, in the RCPSP/max one must assign start times to each of a number of activities in such a way that certain temporal and resource constraints are satisfied. Such an assignment of start times is called a *feasible schedule*. The objective is to find a feasible schedule whose makespan is minimal, where makespan is defined as the maximum completion time of any activity.

Even without maximal time lags (which make the problem more difficult), the RCPSP is NP-hard and is "one of the most intractable problems in operations research" (Möhring *et al.* 2003). When maximal time lags are included, the feasibility problem (i.e., deciding whether a feasible schedule exists) as well as the optimization problem is NP-hard.

4.2. Heuristics

We consider six randomized priority dispatching rules for the RCPSP/max. An approach that selects among randomized priority dispatching rules has been shown to give competitive performance on benchmark instances of the problem (Cicirello & Smith 2005). We consider the six randomized priority dispatching rules in the set $\mathcal{H} = \{LPF, LST, MST, MTS, RMS, Random\}$; see Cicirello and Smith (2004; 2005) for a more complete description of these heuristics.

4.3. Methodology

We evaluate our approach on a set $\mathcal{I}$ of 540 RCPSP/max instances from the ProGen/max library (Schwindt 1996). For each RCPSP/max instance $I \in \mathcal{I}$, we ran each heuristic $h \in \mathcal{H}$ 10,000 times, storing the results in a file. Using this data, we created a set $\mathcal{K}$ of 540 6-armed bandit problems (each of the six heuristics $h \in \mathcal{H}$ represents an arm). For each instance $K \in \mathcal{K}$, we ran three algorithms with a budget of $n = 10,000$ pulls: our interval estimation algorithm, the QD-BEACON algorithm of Cicirello and Smith (2005), and a straw man algorithm that simply sampled the arms in a round-robin fashion. When running our interval estimation algorithm, we use a thresholded version of K. We calculated (offline) the highest threshold such that, for some heuristic $h \in \mathcal{H}$, at least 5% of the schedules had quality (equal to -1 times makespan) in excess of the threshold.

4.5. Results

We first evaluate the three algorithms in terms of the number of above-threshold schedules that were obtained. For each algorithm $\mathcal{A}$ and each instance $\mathcal{K}$, we computed the ratio of the number of above-threshold schedules sampled by $\mathcal{A}$ to the number that would have been sampled using the single best heuristic. The table below shows the minimum and average value of this ratio over all 540 instances. In addition, the table shows p_{best}, the probability that the arm sampled most often by $\mathcal{A}$ was an arm that would yield a maximum-quality solution if sampled for all n trials.

Heuristic	**Min. ratio**	**Avg. ratio**	p_{best}
Interval estimation	0.80	0.95	0.93
QD-BEACON	0	0.81	0.88
Round-robin	0.16	0.49	NA

The above table shows that interval estimation outperforms the other two algorithms in terms of its ability to obtain above-threshold schedules.

We additionally computed the fraction of instances where the best schedule generated by interval estimation was better than the best schedule generated by QD-BEACON (resp. round-robin). Ignoring ties, the best schedule from interval estimation was superior to that from QD-BEACON in 84% of the time, and superior to that from round-robin 85% of the time.

Future Work

Below we outline two areas for potential future work.

Threshold selection. In the experiments reported in §4, we calculated (offline) the highest threshold such that, for some heuristic $h \in \mathcal{H}$, at least 5% of the schedules had quality in excess of the threshold. In a real application, the thresholds instead must be determined online, and may be adjusted dynamically over time. We are currently investigating approaches to this problem.

Variable run lengths and restarts. In this work, we have assumed that each run of each heuristic has (approximately) the same computational cost. It is desirable to effectively handle a set of heuristics where the run time varies significantly across heuristics and across multiple runs of a single heuristic. In this scenario it is also desirable to restart a heuristic if it appears unlikely to produce an acceptable solution within a reasonable amount of time. Specifically, it is desirable to learn online, for each heuristic, a time bound after which the heuristic should be restarted.

References

Auer, P.; Cesa-Bianchi, N.; Freund, Y.; and Schapire, R. E. 2002a. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing* 32(1):48–77.

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002b. Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47:235–256.

Berry, D. A., and Fristedt, B. 1986. *Bandit Problems: Sequential Allocation of Experiments*. London: Chapman and Hall.

Cicirello, V. A., and Smith, S. F. 2004. Heuristic selection for stochastic search optimization: Modeling solution quality by extreme value theory. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming*, 197–211.

Cicirello, V. A., and Smith, S. F. 2005. The max k-armed bandit: A new model of exploration applied to search

heuristic selection. In *Proceedings of AAAI 2005*, 1355–1361.

Kaelbling, L. P. 1993. *Learning in Embedded Systems*. Cambridge, MA: The MIT Press.

Möhring, R. H.; Schulz, A. S.; Stork, F.; and Uetz, M. 2003. Solving project scheduling problems by minimum cut computations. *Management Science* 49(3):330–350.

Neumann, K.; Schwindt, C.; and Zimmerman, J. 2002. *Project Scheduling with Time Windows and Scarce Resources*. Springer-Verlag.

Robbins, H. 1952. Some aspects of sequential design of experiments. *Bulletin of the American Mathematical Society* 58:527–535.

Schwindt, C. 1996. Generation of resource–constrained project scheduling problems with minimal and maximal time lags. Technical Report WIOR-489, Universität Karlsruhe.

Streeter, M. J., and Smith, S. F. 2006. An asymptotically optimal algorithm for the max k-armed bandit problem. Technical Report CMU-CS-06-110, Department of Computer Science, Carnegie Mellon University.

Collaborative Authoring of Plan-Based Interactive Narrative

James M. Thomas

Department of Computer Science
North Carolina State University
Raleigh, NC 27695 USA
jmthoma5@ncsu.edu

Abstract

My research provides an interface for non-technical authors to collaborate with a planning system to create interactive narrative. I describe a domain metatheory to allow for qualitative elaborations of narrative domains. A graphical user interface that exploits this metatheory is used to specify authorial preferences. These preferences are employed to enhance the qualitative reasoning of the planning system.

Research Problem

"Interactive narratives" are the stories that develop within virtual worlds in which human users interact with one or more computer controlled agents. A persistent challenge in these systems is the narrative paradox: "how to reconcile the needs of the user who is now potentially a participant rather than a spectator with the idea of narrative coherence." (Aylett 2000).

My research builds on an approach for solving this paradox first described as the Mimesis system (Riedl, Saretto, and Young 2003). Mimesis generates plans for actions of story world characters based on hierarchical task decompositions and discrete causal requirements. Although Mimesis simultaneously solves for plot coherence and character believability, the authors acknowledge (Riedl and Young 2004) that a primary limitation is the lack of a search space heuristic that would allow the system to judge the relative "goodness" of one plan over another. In other words, there is no mechanism to ensure that particular narrative qualities such as "suspense", "surprise" or "romance" will be produced in resulting plans.

One might attempt to define a generalized heuristic function in terms of universally accepted narrative ideals, but most planners lack a sufficiently powerful model to make associations between such generalized ideals and the semantics of a specific problem domain and plan space. Even if that bridge were built, there is no consensus view of the ideals that guarantee "good" narrative in the first place. As author Somerset Maugham quipped, "There are three rules for writing the novel. Unfortunately, no one knows what they are".

Research Strategy

My plan is to define heuristic functions for each interactive narrative based on the author's preferences of setting and story. For the system to capture these preferences and report them to the planner, it must have an integrated understanding of the definitions of actions and entities in the problem domain (the setting) and the effects that the constraints on those actions have in defining the topology of the plan space (story experiences). To best create that integrated understanding, my research asks a patient author to remain "in the loop" throughout the plan construction process.

A major challenge of this strategy is how to best conserve and apply the limited time and attention of the author. One of the artifacts of my research strategy is a GUI tool called Bowman that allows the plan author to:

1. Describe the domain.
2. Describe the goals for the story.
3. Describe selection criteria (heuristics).
4. Request possible story plans.
5. View graphs of story plans.
6. Compare story plans.
7. Refine and reiterate.

Bowman is part of the Zócalo suite of planning tools available at NCSU at http://zocalo.csc.ncsu.edu. Like Mimesis, the planning component of Zócalo is based on Longbow, a decompositional (HTN) partial order causal link planner described by Young, Pollack and Moore (1994). To allow the author to "request story plans" as shown above, Bowman passes an XML representation of the planning problem to a planning web-service to generate plans. The Bowman user may specify an arbitrary URL at which the web service resides, or the user may direct plan request to an on-board instance of the planner bundled with Bowman. The planner interface supports several styles of plan requests. The user may request ask for the next complete plan, or for the next N plans, or for as many plans as can be generated by in N seconds. The Bowman user can explore the details of individual plans as well as the

entire plan space through scalable vector graphics (SVGs) that can be navigated through mouse over and mouse click. For example, as shown in figure 1, each step in the plan is represented by a collection of rounded rectangles and labeled arrows between steps represent causal and ordering links.

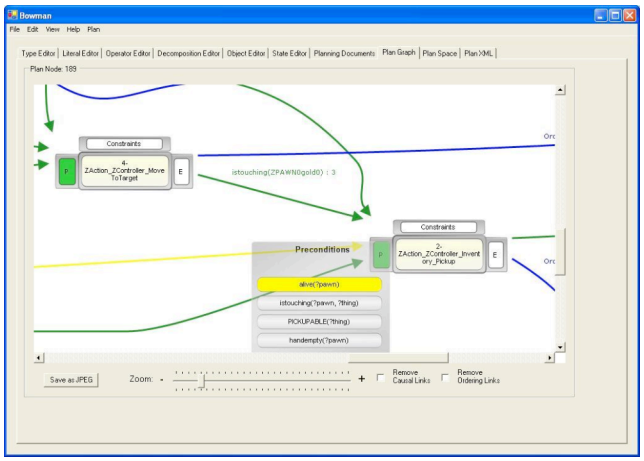


Figure 1: Bowman - Plan Node View

The smaller rectangle on the left side of each step labeled with the letter "P" contains the preconditions of the step. The precondition bubble for a step is filled in with a green colored background if all the preconditions are satisfied for the plan. If the user moves the mouse over the precondition bubble, a semi-transparent window pops up containing each of the preconditions in the plan. As the user moves the mouse down over each precondition, it is highlighted in yellow, as is the causal link which establishes that condition.

Figure 2 contains a Bowman depiction of the plan space as a tree of nodes, where each node is a partial plan.

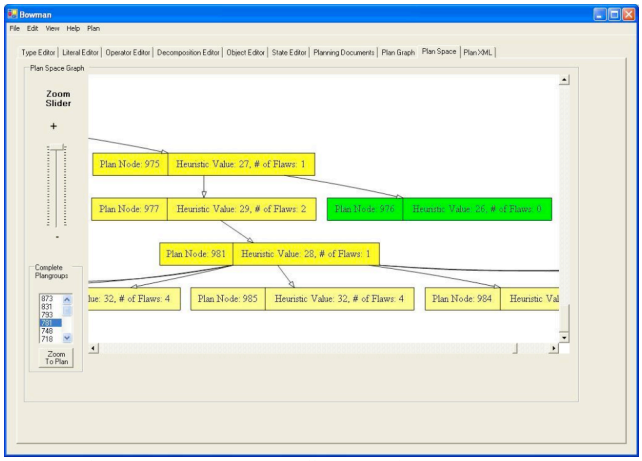


Figure 2: Bowman - Plan Space View

Plan nodes are colored and labeled according to the number of plan flaws they contain. A plan flaw is an open precondition, a threatened causal link, or a flawed decomposition. Plan nodes with zero flaws are shown in green and plans with one or more flaws are shown in progressively lighter shades of yellow. The author can move from the plan space view to view a particular plan node by clicking on it.

Although a modern graphical interface to the planning system certainly affords some efficiency and expressivity advantages, exploration of any reasonably interesting planning problem will quickly exhaust the patience of even the most patient and computer-friendly human author. The core of my research is knowledge representation and elicitation strategies that optimize the use of the human author's creativity. A key requirement is to provide reasonable default strategies for elements of the knowledge representation so that the author need not explicitly express neutral opinions and may focus efforts on those areas for which they are most opinionated. To make this problem more tractable, these strategies are informed by the requirements of planning for interactive narrative.

Interactive narrative domains occupy a promising intermediate level of complexity between the "blocks world" and the real world. Because interactive narrative takes place in a virtual world, its domains are both fully knowable and fully malleable. An advantage for planning research is that these domains may be amended or contracted to suit the requirements of the planning problem. But interactive narrative also introduces special challenges for planning. For example, it is not enough to find a single complete and consistent plan. Authors are interested in understanding how unplanned user actions may affect story goals. This in turn raises issues about the variability of narrative experiences that are possible with each construction and how those possibilities shift as authors make changes.

In fact, the plan author may be responsible not only for the story, but is likely responsible for the domain representation as well. Furthermore, the author may be involved in the creation of the virtual world that is the planning domain. As interactive narrative planning is a component within this larger creative process, there are possibilities and requirements for experimentation and exploration than are not found working with real world domains. This affords researchers new ways to investigate relationships between domains, their representations, planning problems, and the resulting plan spaces. The most fervent hope I hold out for my research is that it leads to new insights into these relationships.

Increasing Domain Knowledge in Planning

Traditional automated planners are not designed specifically to facilitate iterative collaboration with the plan author. Research into collaborative planning methodologies has generally been referred to as advisable

or mixed-initiative planning. Advisable planning (Myers 1996) attempts to shape the behavior of the planner by adding additional information to the definition of the planning problem prior to planner invocation. Mixed-initiative planners allow for the iterative and incremental construction of the plan with both the user and the planner capable of proposing or initiating requests to change aspects of the problem or solution. In essence, advisable planning is a special case of mixed-initiative planning, where the initiative is first taken by the plan author, then by the planning system. As such, the success of advisable planning is strongly tied to the knowledge representation it employs to describe the domain. Myers has demonstrated the value of a "domain metatheory" (Myers 2000) that describes the planning domain in terms an author can use to evaluate resulting plans (Myers and Lee 1999). Further research in this area has shown methods for prioritizing the decision choices made available to the plan author to maximize the impact on plan quality (Wolverton 2004). This is a promising method for conserving the limited time and attention of the human author in a mixed-initiative system.

Myers' domain metatheory serves two masters in that it is meant to be intelligible and relevant to human plan authors but also serves as a basis for automated reasoning about plans. My research tool, Bowman, translates abstract domain metatheoretic constructs presented to human authors into a more compact representation for use by the planner. I call this representation the Domain Elaboration Framework, or DEF. The basis of DEF is a STRIPS-style (Fikes and Nilsson 1971) planning domain characterized by objects, conditions and operators. Where the domain metatheory introduced by Myers relies on a description logic of roles, role-fills, features, and measures, DEF uses an alternate grammar of **types**, **dimensions**, **weights**, and **measurements**.

A type is a symbolic name of a node in a global hierarchy of author-defined types with a unique root node named "***anyThing***". Every operator, parameter, and object instance has at least one associated type, and zero or more associated **measurements**. A **measurement** consists of a **dimension** and a **weight**. A **dimension** is a symbolic name selected from a global list of unique author-defined dimensions. A **weight** specifies a relative intensity of the dimension on a normalized interval. Thus, DEF provides a very simple and general elaboration of planning domains at a fine level of granularity. An application employing DEF must provide greater expressive power at the level of the user interface.

Application of DEF to Interactive Narrative

A series of conventions, mostly enforceable through the Bowman GUI, facilitate the application of DEF to the domain of interactive narrative.

Agent Types

A mechanism is needed to distinguish between user-controlled agents and system-controlled agents. System controlled agents are often referred to as bots, or as NPCs - Non Player-Controlled characters. This distinction can be realized through a convention applied to the population of the type hierarchy of DEF. For interactive narrative domains, the type hierarchy can be rooted with "agent" and "non-agent". "Agent" can be further subdivided into "NPC" and "User". Bowman can ensure that all operators contain explicit representations of the types of agent capable of invoking the action.

Mediation Strategies

As described in by Riedl, Saretto, and Young (2003) the planner is responsible for detecting user actions that could threaten the story plan. For each of these *exceptional* actions, the system must determine if changing part of the unexecuted portion of the plan can *accommodate* the action or if an *intervention* is required. An *intervention* requires that the requested action does not execute. Instead an instance of a non-threatening action, called a *failure mode* is substituted for the requested action in real-time.

Bowman can expand the depiction of complete plans to include the application of all available mediation strategies. The plan author can use Bowman to compare these expanded complete plans to see how resilient each is to user action. Authors may be interested in ensuring that the alternative narrative paths dictated by alternative user actions are different or similar based on various qualitative criteria.

Bowman may also help the author in the definition of failure modes. Since a failure mode is simply a list of operators, Bowman can easily highlight the subset of operators from the current library that are good candidates for a particular failure mode, or it may prompt the author to invent suitable operators by describing their characteristics. For example, there may be a need for a failure mode that causes a *shoot(?shooter, ?gun, ?target)* action to fail such that *^alive(?target)* is **not** an effect of its invocation.

Custom Heuristics

Bowman allows the plan author to construct heuristics based on any of the attributes described in DEF to apply relative weights on different flaws or features of a plan. Thus, the author can encode arbitrary narrative preferences and use iterative refinement of the plan space to ensure that optimal levels of "kissing" are in each story, ensure that the possible execution paths have the desired level of conformity or diversity, or simply understand the shortest and longest success paths through a narrative.

Narrative Goal Conflicts

As authors build more narrative goals into their planning problems, it may become more difficult to find complete plans, if these preferences are treated as universally hard constraints. In the narrative domain, it is likely that authors would prefer a sub-optimal plan to having no solution at all. Bowman uses the DEF vocabulary to enable the plan author to specify degrees of "softness" of lower criticality to goals. Methods are being explored to iteratively resolve conflicts between narrative goals and the rules that describe the problem domain. This resolution must be in the form of an 'anytime' algorithm, as the size of the decision space likely exceeds the patience of any human author.

Research Status

Currently, Bowman can be used to define planning domains and planning problems using classical constructs of objects, conditions and operators. The only DEF construct to be realized in Bowman to date is "type". Figure 3 shows how Bowman allows the preconditions of an operator (in this case "LaunchSpaceship") to be edited. Note the use of "type" to color-code the parameters of the literals and the object hierarchy. This aids the author in dragging and dropping properly typed object instances or operator parameters to fill these literal definitions. As shown earlier, Bowman can send these problem definitions to a planner and navigate the resulting plan space.

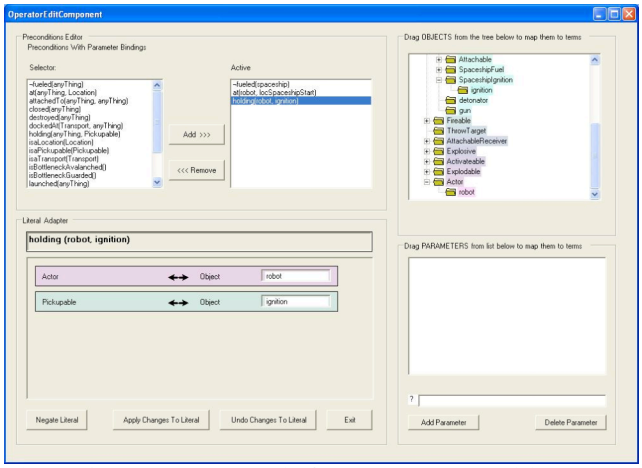


Figure 3: Bowman – Edit Operator Preconditions

Still to be implemented are the remaining DEF constructs, narrative mediation strategies, custom heuristics and higher-level abstractions for managing conflicting hard and soft goals. Finally an evaluation of the expressivity and usefulness of Bowman and DEF must be undertaken. Given the lack of models to which this can be easily compared, its usefulness is likely best gauged through an ablative study to show gauge incremental changes in effectiveness as features are added or removed. I hope to finish this work and defend it in a dissertation within twelve months.

Acknowledgements

This is material is based upon work supported under a National Science Foundation Graduate Research Fellowship and National Science CAREER award 0092586.

References

Aylett, R.S. 2000. Emergent Narrative, Social Immersion And "Storification" *Proceedings, Narrative Interaction for Learning Environments*, Edinburgh.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intellligence,* 2, pp. 189-208.

Myers, K. 2000. Domain Metatheories: Enabling User-Centric Planning. In *Proceedings of the AAAI Workshop on Representational Issues for Real-World Planning Systems*, K. Myers and Y. Gil, Eds.

Myers, K., and Lee, T. J. 1999. Generating Qualitatively Different Plans Through Metatheoretic Biases. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence,* AAAI Press.

Myers, K. L. 1996. Advisable Planning Systems. In *Advanced Planning Technology*, A. Tate, Ed. AAAI Press, Menlo Park, CA.

Riedl, M., Saretto, C., and Young, R. M. 2003. Managing Interaction Between Users And Agents In A Multiagent Storytelling Environment. *In Proceedings of the Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-03).*

Riedl, M., and Young, R. M. 2004. An Intent-Driven Planner For Multi-Agent Story Generation. *In Proceedings of the Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-04).*

Wolverton, M. 2004. Prioritizing Planning Decisions In Real-World Plan Authoring. In *Proceedings of the ICAPS04 Workshop on Connecting Planning Theory with Practice.*

Young, R.M., Pollack, M.E., and Moore, J.D. (1994). Decomposition and causality in partial-order planning. In *Proceedings of the Second International Conference on AI and Planning Systems*, 188-193, Chicago, IL, 1994.

Planning with Preferences and Trajectory Constraints by Integer Programming

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 automated planning. This extended abstract provides a brief overview of the paper that will be presented at the ICAPS Workshop on Planning with Preferences and Trajectory Constraints. A synopsis of some of my future research plans is given at the end.

Introduction

Given the recent success of integer programming approaches to automated planning (van den Briel, Vossen, & Kambhampati 2005), I believe that these approaches are a good avenue to explore further both because of the recent improvements, and the fact that with preferences, planning becomes an optimization problem, which integer programming is naturally equipped to handle.

Preferences and trajectory constraints are two new language features in PDDL3.0 that can be used to express hard and soft constraints on plan trajectories, and that can be used to differentiate between hard and soft goals. Hard constraints and goals define a set of conditions that must be satisfied by any solution plan, while soft constraints and goals define a set of conditions that merely affect solution quality.

In particular, preferences assume a choice between alternatives and the possibility to rank or order these alternatives. In PDDL3.0, preferences can be defined on states, on action preconditions, on trajectory constraints, or on some combination of these. Since preferences may or may not be satisfied for a plan to be valid they impose soft constraints or goals on the planning problem. Trajectory constraints, on the other hand, define a set of conditions that must be met throughout the execution of the plan. They can be used to express control knowledge or simply describe restrictions of the planning domain. Since trajectory constraints define necessary conditions for a plan to be valid (except in the case where the trajectory constraint is a preference) they impose hard constraints or goals on the planning problem.

Neither preferences nor trajectory constraints have yet gotten a lot of attention from the planning community, but the importance of solution quality and the efficient handling of hard and soft constraints and goals has increasingly been addressed by some recent works.

Planning with preferences is closely related to oversubscription planning. In oversubscription planning goals are treated as soft goals as there are not enough resources to satisfy all of them. This problem has been investigated by Smith (2004) and further investigated by several other works.

Preferences, however, are more general than soft goals as they also include soft constraints. Son and Pontelli (2004) describe a language for specifying preferences in planning problems using logic programming. Their language can express a wide variety of preferences, including both soft goals and soft constraints, but it seems that it has not been used for testing yet. Empirical results for planning with preferences are provided by Rabideau, Engelhardt and Chien (2000) and Brafman and Chernyavsky (2005). Rabideau, Engelhardt and Chien describe an optimization framework for the ASPEN planning system, and Brafman and Chernyavsky describe a constraint based approach for the GP-CSP planning system.

Planning with trajectory constraints is closely related to reasoning about temporal control knowledge and temporally extended goals, which are discussed by Ghallab Laruelle (1994) and Muscettola 1994. Edelkamp (2005) handles trajectory constraints by converting a PDDL3.0 description into a PDDL2.2 description and then using a heuristic search planner.

In this extended abstract I will show a few examples of how to express preferences and trajectory constraints by linear constraints over 0-1 variables. These constraints are then to be added to the integer programming formulation of the planning problem after which the model is solved. Currently, I'm in the process of incorporating these constraints in the integer programming formulations described in van den Briel, Vossen, & Kambhampati (2005).

Simple Preferences

Simple preferences are preferences that appear in the goal or that appear in the preconditions of an action. Goal preferences can be violated at most once (at the

end of the plan), whereas precondition preferences can be violated multiple times (each time the corresponding action is executed).

For each goal preference in the planning problem we introduce a 0-1 variable p, where $p = 1$, if the goal preference is violated and, $p = 0$ if the goal preference is satisfied. Similarly, for each precondition preference for action a at step t ($1 \leq t \leq T$) we introduce a 0-1 variable $p_{a,t}$, where $p_{a,t} = 1$, if the precondition preference is violated for action a at step t and, $p_{a,t} = 0$ if the precondition preference is satisfied for action a at step t. This way all violations can be counted for separately and given different costs in the objective function of the formulation.

Constraints for goal and precondition preferences are easily modeled by integer programming. There are only finitely many operators in PDDL3.0, including some standard operators like `or`, `and`, and `imply`, which can all be represented by one or more linear constraints.

Examples

In the examples we will use variables $x_{a,t}$ to denote the execution of an action a at step t, and use variables $y_{f,t}$ to denote the truth value of a fluent f at step t. This is slightly different from the notation and variables used in the formulations by van den Briel, Vossen, and Kambhampati 2005, but for explanation purposes we think it is more obvious this way.

In PDDL3.0, the goal preference p_1 "We would like that person1 is at city2" is expressed as follows.

```
(:goal (and (preference p1
  (at person1 city2))))
```

The inequality corresponding to preference p_1 is given by:

$$p_1 \geq 1 - y_{\text{at person1 city2},T} \quad (1)$$

Thus preference p_1 is violated ($p_1 = 1$) if person1 is not at city2 at the end of the plan ($y_{\text{at person1 city2},T} = 0$).

The goal preference p_2 "We would like that person1 or person2 is at city2" is expressed as follows.

```
(:goal (and (preference p2 (or
  (at person1 city2) (at person2 city2)))))
```

The inequality corresponding to preference p_2 is given by:

$$p_2 \geq 1 - y_{\text{at person1 city2},T} - y_{\text{at person2 city2},T} \quad (2)$$

Now, preference p_2 is violated if neither person1 nor person2 is at city2 at the end of the plan. Preference p_2 is satisfied when either or both person1 and person2 are at city2 at the end of the plan.

The goal preference p_3 "We would like that person2 is at city1 if person1 is at city1" is expressed as follows.

```
(:goal (and (preference p3 (imply
  (at person1 city1) (at person2 city1)))))
```

The inequality corresponding to preference p_3 is given by:

$$p_3 \geq y_{\text{at person1 city1},T} - y_{\text{at person2 city1},T} \quad (3)$$

So preference p_3 is violated if person2 is not at city1 while person1 is.

Preferences over preconditions are different from goal preferences as they depend on both the execution of an action and on the state of the precondition of that action. Moreover, a precondition preference is defined for each plan step t, where $1 \leq t \leq T$. In PDDL3.0, the precondition preference $p_{4,\text{fly}?a?c1?c2,t}$ "We would like that some person is in the aircraft" whenever we fly aircraft $?a$ from city $?c1$ to city $?c2$ is expressed as follows:

```
(:action fly
 :parameters (?a - aircraft ?c1 ?c2 - city)
 :precondition (and (at ?a ?c1)
   (preference p4
     (exists (?p - person) (in ?p ?a))))
 :effect (and (not (at ?a ?c1))
   (at ?a ?c2)))
```

The inequalities corresponding to each ground fly ?a ?c1 ?c2 action is given by:

$$p_{4,\text{fly}?a?c1?c2,t} \geq x_{\text{fly ?a ?c1 ?c2,t}} - \sum_{?p} y_{\text{in ?p ?a},t} \quad \forall 1 \leq t \leq T \quad (4)$$

Thus, preference $p_{4,\text{fly}?a?c1?c2,t}$ is violated at step t if we fly aircraft $?a$ from city $?c1$ to city $?c2$ at step t ($x_{\text{fly ?a ?c1 ?c2,t}} = 1$) without having any passenger $?p$ onboard at step t ($y_{\text{in ?p ?a},t} = 0$, for each $?p$).

Qualitative Preferences

In propositional planning, qualitative preferences include trajectory constraints and preferences over trajectory constraints none of which involve numbers. Given the space limitations we will mainly concentrate on the trajectory constraints here that use the new modal operators of PDDL3.0 in this section.

There is a general rule of thumb for the operators `forall` and `always`. `forall` indicates that the trajectory constraint must hold for each object to which it is referring to. For example, `forall (?b block)` means that the trajectory must hold for each instantiation of $?b$, thus we generate the trajectory constraint for all blocks $?b$. `always` in propositional planning is equivalent to saying for all t, thus we generate the trajectory constraint for all t where $1 \leq t \leq T$.

Constraints for trajectories are easily modeled by integer programming through observing the different operators carefully. It is often the case, that the trajectory constraint simply represent one of the standard relationships described earlier in this paper.

Examples

In PDDL3.0 the trajectory constraint "A fragile block can never have something above it" is expressed as

follows.

```
(:constraints (and (always (forall (?b block)
  (implies (fragile ?b) (clear ?b))))))
```

The inequality corresponding to this trajectory constraint corresponds to the relation that fragile *implies* clear for all blocks $?b$, for all steps t, where $1 \leq t \leq T$. It is given by:

$$y_{\text{fragile ?b},t} - y_{\text{clear ?b},t} \leq 0 \quad \forall ?b, 1 \leq t \leq T \qquad (5)$$

The trajectory constraint "Each block should be picked up at most once" which is expressed as follows.

```
(:constraints (and (forall (?b block)
  (at-most-once (holding ?b)))))
```

It translates to an *at most once* relation for all blocks $?b$ and is given by:

$$y_{\text{holding ?b},0} + \sum_{a \in A, 1 \leq t \leq T : \text{holding ?b} \in ADD(a)} x_t^a \leq 1 \quad \forall ?b \qquad (6)$$

Likewise the trajectory constraint "Each block should be picked up at least once" is expressed as follows.

```
(:constraints (and (forall (?b block)
  (sometime (holding ?b)))))
```

This translates to a *sometime* relation for all blocks $?b$ and is given by:

$$\sum_t y_{\text{holding ?b},t} \geq 1 \quad \forall ?b \qquad (7)$$

Continuing in the same way, the trajectory constraint "A truck can visit city1 only if it has visited city2 sometime before" is expressed in PDDL3.0 as follows.

```
(:constraints (and (forall (?t truck)
  (sometime-before
    (at ?t city1) (at ?t city2)))))
```

The corresponding inequality describes a *sometime-before* relationship for all trucks $?t$ and is given by:

$$\sum_{1 \leq s < t} y_{\text{at ?t city2},s} \geq y_{\text{at ?t city1},t} \quad \forall ?t, 1 \leq t \leq T \qquad (8)$$

More examples can be presented, but it seems enough to bring the point across that integer programming provides a natural framework for modeling propositional planning with preferences and trajectory constraints.

Conclusions and Future Work

This extended abstract shows a few examples of how to model preferences and trajectory constraints by integer programming manually. The main challenge is to automatically generate these constraints and add them to the integer programming formulation of the planning problem. Especially, generating constraints for complicated instances of preferences and trajectory constraints that contain nested expressions can be tricky, but is feasible.

An interesting analysis for future work would be to compare the performance of the integer programming formulations that use preferences and trajectory constraints as side constraints (as shown in the above examples) with integer programming formulations that handle preferences and trajectory constraints which are compiled down into PDDL2.2.

The general focus of my future research is to extend and improve the current integer programming formulations for automated planning, and to apply integer programming techniques to a broader range of planning problems, including resource planning and temporal planning. Techniques like branch-and-cut and branch-and-price will be at the base of most of these extensions.

References

Brafman, R., and Chernyavsky, Y. 2005. Planning with goal preferences and constraints. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*, 182–191.

Edelkamp, S. 2005. Efficient planning with state trajectory constraints. In Sauer, J., ed., *Proceedings Workshop Planen, Scheduling und Konfigurieren / Entwerfen*, 89–99.

Ghallab, M., and Laruelle, H. 1994. Representation and control in IxTeT, a temporal planner. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, 62–67.

Muscettola, N. 1994. *Intelligent Scheduling*. Morgan Kaufmann. chapter HSTS: Integrating planning and scheduling, 169–212.

Rabideau, G.; Engelhardt, B.; and Chien, S. 2000. Using generic preferences to incrementally improve plan quality. In *Proceedings of the 2nd NASA International Workshop on Planning and Scheduling for Space*, 11–16.

Smith, D. 2004. Choosing objectives in over-subscription planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS)*, 393–401.

Son, T., and Pontelli, E. 2004. Planning with preferences unsing logic programming. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 247–260.

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 15th International Conference on Automated Planning and Scheduling (ICAPS)*, 310–319.

Scheduling with uncertain durations: generating β-robust schedules using constraint programming

Christine Wei Wu and **Kenneth N. Brown**
Cork Constraint Computation Center,
Department of Computer Science,
University College Cork, Ireland
{cww1, k.brown}@cs.ucc.ie

J. Christopher Beck
Toronto Intelligent Decision Engineering
Laboratory, Dept. of Mechanical and
Industrial Engineering,University of Toronto,
Canada. jcb@mie.utoronto.ca

Abstract

Many real-world scheduling problems are subject to change, and scheduling solutions should be robust to those changes. We consider a single-machine scheduling problem where the processing time of each activity is characterized by a normally-distributed random variable, and we attempt to minimize flowtime. We develop an initial constraint model for generating the β-robust schedule - the schedule that has highest probability of producing a flowtime less than a stated bound. Experiments with this initial model show that a constraint-based approach is feasible, but that better propagation methods will be required.

Introduction

Many real-world scheduling problems are subject to change: new jobs arrive, resources fail, or tasks take longer than expected. If these changes are significant, it may be better to generate solutions that are robust to them. A number of approaches have been proposed to handle uncertain scheduling problems. Redundancy-based Scheduling generates schedules with temporal slack so that unexpected events during execution can be handled by using that reserved slack (Davenport, Gefflot, & Beck 2001; Gao 1995). Contingent scheduling anticipates likely disruptive events and generates multiple schedules which optimally respond to the anticipated events (Drummond, Bresina, & Swanson 1994; Fowler & Brown 2003). Probabilistic scheduling uses probabilities over possible events, and searches for schedules which optimize the expected value of some performance measure (Daniels & Carrillo 1997; Walsh 2002; Beck & Wilson 2004; 2005). A number of approaches use sampling and scenarios, in order to produce robust decisions (Bent & Hentenryck 2004; Beck & Wilson 2004).

In particular, Daniels and Carrillo (Daniels & Carrillo 1997) introduced the concept of the β-robust schedule for a single machine scheduling problem with processing time uncertainty, which aims to maximize the probability of achieving a given performance level. They considered flowtime (the amount of time the tasks remain in the system) as the performance metric. They solved the problem by a branch-and-bound method with dominance rules, and heuristics for branch selection.

Constraint-based methods have proven to be very effective in a wide range of industrial scheduling problems. The advantage comes from the flexibility of the modeling language, and the ability of the solvers to deliver effective performance despite the presence of a wide range of different constraints and objectives. For these reasons, we develop an initial constraint model for solving the β-robust scheduling problem. For any given schedule, we will measure the probability of the total flowtime being less than a target level. We will then generate a schedule which maximizes the probability. Also, we will show the benefits of using standard constraint programming techniques. We introduce another objective, which is to find a schedule which optimizes the target level that can be achieved with a given probability. In this paper, we present a primal CP model to solver those problems.

The flowtime of a schedule

Before we show the primal model of the β-robust scheduling problem, we need to go through series of formal definitions and mathematical calculations. Those mathematical formulas will then give us a clear indication of any necessary variables and values for modeling the problem as a constraint satisfaction problem (CSP). There are several criteria of measuring schedules. In this report, we use the total flowtime to judge the solutions.

In a single machine scheduling problem, in which each job consists of a single task, a machine can only process one job at a time, and a job cannot be interrupted once started, a solution is a sequence of the jobs, and we assume the jobs are executed in sequence with no delay between them. Suppose we have a sequence $J_1, J_2, \ldots J_n$. Each job J_i has an arrival time A_i (its earliest possible start time), a start time ST_i, a duration d_i, and an end time E_i. We assume that each job is available for processing at time 0 (i.e. $A_i = 0$). We note the following simple relations: $E_i = ST_i + d_i$, $ST_1 = 0$, $ST_i = E_{i-1}$, and hence $E_i = \sum_{j=1}^{i} d_j$.

The *flowtime* is the total time the jobs are in the system: $TFT = \sum_{i=1}^{n}(E_i - A_i)$. Because we assume $Ai = 0$, we can rewrite the equation for total flowtime as follows:

$$TFT = \sum_{i=1}^{n} E_i = \sum_{i=1}^{n}\sum_{j=1}^{i} d_j = \sum_{i=1}^{n}(n+1-i) * d_i \quad (1)$$

We now assume that each job J_i's duration is an independent normally distributed random variable $d_i \sim N(\mu_i, \sigma_i^2)$.

We assume that the jobs will still be executed in the given sequence, regardless of the actual values of the durations. We note that for any two independent random variables $X \sim N(\mu_x, \sigma_x^2)$ and $Y \sim N(\mu_y, \sigma_y^2)$, and two constants a and b, the sum $aX + bY$ is also a normally distributed random variable, such that $aX+bY \sim N(a\mu_x+b\mu_y, a^2\sigma_x^2+b^2\sigma_y^2)$.

Since the activity durations are independent normally distributed random variables, and flowtime is a linear combination of durations, then for any particular sequence of jobs, the flowtime is also a normal random variable. From (1): $TFT \sim N(\sum_{i=1}^{n}(n-i+1)\mu_i, \sum_{i=1}^{n}(n-i+1)^2\sigma_i^2)$.

β-robust schedules

The β-robust schedule is useful because rather than gambling on the expected performance (or the average actual performance over a number of runs), it gives a lower limit on the performance, and to state the confidence in being able to achieve that level.

For example, consider the simple problem consisting of three jobs $\{x, y, z\}$, with uncertain durations $\{d_x \sim N(9,2), d_y \sim N(5,1), d_z \sim N(8,7)\}$. The sequence $s_e = \langle y, z, x\rangle$ has a flowtime which is distributed as $N(40, 39)$. 40 is, in fact, the smallest expected flowtime possible for this problem. An alternative sequence, $s_\beta = \langle y, x, z\rangle$, has flowtime $\sim N(41, 24)$, and thus has a higher expected flowtime. However, suppose we now introduce a desired maximum flowtime of (for example) 51: the scheduler will incur a penalty if the actual schedule has a flow time greater than 51. Sequence s_e has a probability of 0.04 of producing a flowtime greater than 51, while s_β has a probability of just 0.02 of delivering a flowtime greater than 51, and thus s_β is likely to be less expensive. s_β is the β-*robust* (Daniels & Carrillo 1997) schedule for the maximum flowtime of 51 - that is, it has the highest probability of delivering a flowtime no greater than 51. In addition, for the confidence level of 0.98, s_β also delivers the minimal flowtime limit (51).

Definition 1. *For the single machine scheduling problem with n jobs, with normally-distributed uncertain durations, and with a flowtime limit S, the β-robust scheduling problem is to find the sequence, s, which maximizes the probability of the flowtime being less than S. That is, find the s that maximizes $Prob(flowtime(s) \leq S)$ (Daniels & Carrillo 1997).*

First, we show how to compute $Prob(\ flowtime \leq S\ )$ for an arbitrary sequence of the n jobs. Since the random variables in the problem are normally distributed, we can use the formula below to compute the probability of $flowtime \leq S$, where μ is the mean flowtime of the sequence, and σ^2 is its variance: $\phi(x \leq X) = 1/\sigma\sqrt{2\pi}\ \int_{-\infty}^{X} e^{\frac{-(x-\mu)^2}{2\sigma^2}}\, dx$. An arbitrary normal distribution can be converted to a standard normal distribution by changing variables to $z = (x-\mu)/\sigma$, so the normal distribution function becomes:

$$\phi(x \leq X) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{\frac{-t^2}{2}}\, dt = \frac{1}{2} + \phi(z)$$

where $\phi(z) = 1/\sqrt{2\pi}\ \int_{0}^{z} e^{\frac{-t^2}{2}}\, dt$. Hence, the probability of $flowtime \leq S$ can be computed by

$$Prob(\ flowtime \leq S\ ) = \frac{1}{2} + \phi(z) \qquad (2)$$

where

$$z = \frac{S - mean(flowtime)}{\sqrt{var(flowtime)}}. \qquad (3)$$

For each possible schedule, we can compute the mean and variance of the flowtime by $mean(flowtime) = \sum_{i=1}^{n}(n-i+1)\mu_i$ and $var(flowtime) = \sum_{i=1}^{n}(n-i+1)^2\sigma_i^2$ as in equation (1). Then, $\phi(z)$ can be obtained by checking z in the standard normal distribution table (Z-table).

Alternatively, there is a simple approximation of $\phi(z)$ which is good to two decimal places (Weisstein 2006), given by

$$\phi(z) \approx \varphi(z) \begin{cases} 0.1z(4.4-z) & (0 \leq z \leq 2.2) \\ 0.49 & (2.2 < z < 2.6) \\ 0.50 & (z \geq 2.6) \end{cases} \qquad (4)$$

The β-robust schedule is one of those alternative sequences of the jobs, such that it has the maximum probability of $flowtime \leq S$. To find a β-robust schedule, we need to have an objective function to maximize the probability. We use the approximation of $\phi(z)$ to compute the probability, because $\varphi(z)$ increases on $[0, +\infty)$ that simplifies the calculation (the proof is straightforward and omitted here). By using that simplification, maximizing the probability of $flowtime \leq S$ is the same as maximizing z.

$$\begin{aligned} objective &= max(\ \frac{1}{2} + \phi(z)\ ) \\ &\approx \frac{1}{2} + max(\ \varphi(z)\ ) = \frac{1}{2} + \varphi(\ max(z)\ ). \end{aligned}$$

With above analysis and calculations, we are ready to introduce our constraint models for the β-robust scheduling problem.

Primal model

The primal model is shown in Figure 1. We assume a set $\{J_1, J_2, \ldots J_n\}$ of jobs, each with a normally-distributed random variable duration $D_i \sim N(\mu_i, \sigma_i^2)$. Differing from the previous sections, we now do not assume that the jobs are scheduled in the given sequence. With each job J_i, we associate a position variable, Pos_i, with domain $\{1, 2, \ldots, n\}$. The position variable Pos_i represents the position of J_i in the sequence: for instance, $Pos_2 = 3$ states that J_2 is scheduled to be the third job to start on the machine. Besides position variables, we also introduce additional variables for computing flowtime mean and variance and then the probability. The formula (1) indicates that flow time can be viewed as the sum of the contributions from all jobs. We define flowtime contribution of J_i as $FTContrib_i = (n - Pos_i + 1)D_i$. $meanFTContrib_i$ and $varFTContrib_i$ are the mean and variance of the flowtime contributions from J_i. The former has an integer value in $[\mu_i, n\mu_i]$, and the latter has a value in $[\sigma_i^2, n^2\sigma_i^2]$. The goal is to sequence those jobs, i.e. assign a distinct value to each Pos_i, such that the likelihood of the sequence (schedule) to

Figure 1: Primal Model

Variables:
Job positions: $Pos_1, ..., Pos_n$
Job mean flowtime contributions: $meanFTContrib_1, ..., meanFTContrib_n$
Job variance flowtime contributions: $varFTContrib_1, ..., varFTContrib_n$
Constraints:
`allDifferent`(Job positions)
$meanFTContrib_i = (n - Pos_i + 1)\mu_i$
$varFTContrib_i = (n - Pos_i + 1)^2\sigma_i^2$
$mean(flowtime) = \sum_{i=1}^{n} meanFTContrib_i$
$var(flowtime) = \sum_{i=1}^{n} varFTContrib_i$
Dominance constraints:
for $0 < i < j \leq n$,
$\mu_i \leq \mu_j$ and $\sigma_i^2 \leq \sigma_j^2 \Rightarrow Pos_i < Pos_j$
$\mu_i \geq \mu_j$ and $\sigma_i^2 > \sigma_j^2 \Rightarrow Pos_i > Pos_j$
$\mu_i > \mu_j$ and $\sigma_i^2 = \sigma_j^2 \Rightarrow Pos_i > Pos_j$

$$objective = max(z) = max(\ \frac{S - mean(flowtime)}{\sqrt{var(flowtime)}}\ )$$

achieve a fixed system performance level S is optimized, i.e. $max(\ probability(\ flowtime \leq S\ )\ )$.

Firstly, we have a permutation constraint that ensures each job takes a different position in the sequence. This can be implemented as a global all-different constraint on all the Pos_i. Also if we consider the flowtime as a sum of contributions from each job, from formula (1), we have

$$\begin{aligned} mean(flowtime) &= \sum_{i=1}^{n} meanFTContrib_i \\ &= \sum_{i=1}^{n} (n - Pos_i + 1)\mu_i, \\ var(flowtime) &= \sum_{i=1}^{n} varFTContrib_i \\ &= \sum_{i=1}^{n} (n - Pos_i + 1)^2\sigma_i^2. \end{aligned}$$

With those additional variables, we can use formula (2), (3) and (4) to compute the probability of a schedule's actual flowtime being less than S.

We are also able to impose some dominance constraints as in figure 1, using the properties of the β-robust schedule.

Theorem 1. *In a β-robust schedule, if job i with $D_i \sim N(\mu_i, \sigma_i^2)$ precedes job j with $D_j \sim N(\mu_j, \sigma_j^2)$, then either the mean duration of job i, μ_i, is no greater than the mean duration of job j, μ_j, or the duration variance of job i, σ_i^2, is no greater than the duration variance of job j, σ_j^2, that is $\mu_i \leq \mu_j$ or $\sigma_i^2 \leq \sigma_j^2$. (see Appendix for the proof)*

With this property, we post further constraints: for job i and job j ($0 < i < j \leq n$), if $\mu_i \leq \mu_j$ and $\sigma_i^2 \leq \sigma_j^2$, then $Pos_i < Pos_j$; if $\mu_i \geq \mu_j$ and $\sigma_i^2 > \sigma_j^2$ then $Pos_i > Pos_j$; if $\mu_i > \mu_j$ and $\sigma_i^2 = \sigma_j^2$ then $Pos_i > Pos_j$. Note that for the jobs have the same duration mean and the same duration variance, we take the lexicographical order on their indexes, i.e. if $\mu_i = \mu_j$ and $\sigma_i^2 = \sigma_j^2$ and $i < j$, then $Pos_i < Pos_j$.

As stated before, modeling the β-robust scheduling problem as a standard CSP enable us to change the objectives easily for different purposes. In particular, we can generate a schedule which optimizes the target level that can be achieved with a given probability.

Instead of maximizing the probability with a given system performance S, we might want to minimize S such that there exists a schedule that can achieve S with a fixed probability. That is $Min(\ S\ )$ such that $Probability(\ X \leq S\ ) \geq C$, where C is the fixed probability. Using the same primal model, we can get z value from formula (2) and (4) $z = \varphi^{-1}(C - \frac{1}{2})$. Then, from formula (3), we have a new objective function $min(S) = min(\ z * \sqrt{var(flowtime)} + mean(flowtime)\ )$. Note that φ^{-1} is not a one-to-one correspondence function (i.e. there are more than one z values for each value of C) at $C = 0.99$ and $C = 1.0$. For not over estimating S, we select the smallest z from all possible values. That is we set $z = 2.21$ when $C = 0.99$ and $z = 2.6$ when $C = 1.0$.

Besides those constraints we discussed above, we also implement a variable ordering heuristic to guide search. From formula (3), we can see that the β-robust schedule has the optimized combination of $mean(flowtime)$ and $var(flowtime)$. In order to find the β-robust schedule more quickly, we prefer to first schedule a job i, which has shorter mean processing time μ_i and smaller variance σ_i^2. We use a family of variable ordering heuristics, ordering the jobs by increasing $\mu_i + q * \sigma_i^2$, selecting a value for q based on the problem characteristics (probability levels). For higher probabilities, we expect the variance to be more significant, and so we choose higher values of q which give increasing weight to the duration variance in the variable ordering. Note that this variable ordering heuristic does not improve the total solving speed (i.e. the time of finding the schedule and proving it is the optimal), but does shorten the time to find the optimal solution.

Experimental results

We implemented the β-robust scheduling problem as a constraint satisfaction problem using ILOG Scheduler and Solver 6.0. Our first aim is to verify our initial constraint model, and so we expect to see the same pattern of results as obtained by (Daniels & Carrillo 1997). Secondly, we want to determine whether or not a constraint model is feasible for such problems, and so we hope to see runtimes of a similar order of magnitude. If we succeed in both aims, we will then investigate more sophisticated constraint models.

We consider problems with either 10 or 15 jobs, using the same experimental setup as (Daniels & Carrillo 1997). Table 1 contains the results for our constraint methods and the corresponding figures taking directly from (Daniels & Carrillo 1997). The CPU is the computation time for finding and proving the β-robust schedule. It also shows the differences (in average and in maximum deviation) between the mean

Table 1: Computational performance of β-robust solution procedure.

		Constraint model			Branch-and-bound		
total jobs	prob. level	CPU (sec.)	Avg. abv. SEPT (%)	Max. abv. SEPT (%)	CPU (sec.)	Avg. abv. SEPT (%)	Max. abv. SEPT (%)
10	0.85	0.1	0.1	0.4	0.2	0.1	0.8
	0.95	0.1	0.3	1.7	0.2	0.3	1.9
	0.99	0.1	0.5	1.9	0.3	0.6	2.5
15	0.85	2.3	0.1	0.3	1.0	0.1	0.5
	0.95	2.4	0.2	0.7	1.7	0.2	1.0
	0.99	3.0	0.3	1.5	2.1	0.4	1.9

processing time of the β-robust schedule and the shortest expected processing time (SEPT). Table 1 indicates that we do have a similar pattern in term of the mean flowtime of the β-robust schedule compared to the SEPT schedule. In addition, our CPU time is comparable for the smaller problems, but is poorer for the larger problems. This indicates that a constraint-based approach may be feasible, but that a more sophisticated model with better propagation will be required. Moreover, we set up a further experiment to determine the effort require to prove that the solution is optimal. The results shows that the program takes little time (e.g. 7% of total) to find the best solution but usually a long time (e.g. 93% of total) to prove if it is the β-robust schedule. We believe that a problem is hard for our model if it has many jobs with similar duration mean and variance. The program is able to do little propagation, and thus spends a lot of time trying different permutations of the jobs for no benefit.

With the general model, we can also give the minimum system performance S for a problem, so that the jobs in the problem can be scheduled to achieve the minimized S with a desired probability level. The details of experimental results have been abridged.

Future work

In constraint programming, it is sometimes very useful to change view points to study the same problem. Particularly, for a permutation problem (a constraint satisfaction problem in which each decision variable takes an unique value), we can transpose the roles of the variables and the values in presenting the underlying problem to give a new dual model which is also a permutation problem (Hnich, Smith, & Walsh 2004).

We are currently working on the dual model of the primal model, and a third model which channels between the primal model and its dual. We believe using the combined model will help us to improve the solving speed. We also plan to investigate better bounds for pruning branches at the top of the search tree, better heuristics to guide the search, and the construction of a global constraint for achieving β-robustness. We are also conducting an investigation into the characteristics of the problems which make some of them much harder to solve than others. Finally, we plan to extend this work to consider problems with multiple machines and with non-zero arrival times, for which the probability calculations reported here will not apply.

Conclusion

In this paper, we presented a general constraint model for the β-robust scheduling problem, which allows us to produce schedules which are robust to uncertainty in the durations of tasks. With flowtime as the performance measure, we can optimize the probability and find a most promising schedule to satisfy the system performance requirement; or we can optimize the performance level for a fixed probability. Our initial model demonstrates that a constraint-based approach is feasible for this problem, but that more more sophisticated models are required for good performance.

References

Beck, J. C., and Wilson, N. 2004. Job shop scheduling with probabilistic durations. *Proceedings of the Sixteenth European Conference on Artificial Intelligence* 652–656.

Beck, J. C., and Wilson, N. 2005. Proactive algorithms for scheduling with probabilistic durations. *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence* 1201–1206.

Bent, R., and Hentenryck, P. V. 2004. Online stochastic and robust optimization. *Ninth Asian Computing Science Conference* 286–300.

Daniels, R. L., and Carrillo, J. E. 1997. Beta-robust scheduling for single-machine systems with uncertain processing times. *IIE Transactions* 29:977–985.

Davenport, A. J.; Gefflot, C.; and Beck, J. C. 2001. Slack-based techniques for robust schedules. *Proceedings of the Sixth European Conference on Planning* 7–18.

Drummond, M.; Bresina, J. L.; and Swanson, K. 1994. Just-in-case scheduling. *Proceedings of the 12thNational Conference on Artificial Intelligence (AAAI)* 1098–1104.

Fowler, D. W., and Brown, K. N. 2003. Branching constraint satisfaction problems and markov decision problems compared. *Annals of Operations Research* 118:85–100.

Gao, H. 1995. Building robust schedules using temporal protectionan empirical study of constraint-based scheduling under machine failure uncertainty. *Masters thesis, Department of Industrial Engineering, University of Toronto, Toronto, Canada*.

Hnich, B.; Smith, B.; and Walsh, T. 2004. Dual modelling of permutation and injection problems. *Journal of Artificial Intelligence Research* 21:357–391.

Walsh, T. 2002. Stochastic constraint programming. *Proceedings of 15th European Conference on Artificial Intelligence* 111–115.

Weisstein, E. W. 2006. Normal distribution function. *Mathworld, Wolfram Research, Inc.* http:// mathworld. wolfram. com/ Normal Distribution Function. html.

Discrepancy Search with Reactive Policies for Planning

SungWook Yoon
Electrical & Computer Engineering
Purdue University
West Lafayette, IN 47907
sy@purdue.edu

Abstract

We consider a novel use of mostly-correct reactive policies. In classical planning, reactive policy learning approaches could find good policies from solved trajectories of small problems and such policies have been successfully applied to larger problems. Often, due to the inductive nature, the learned reactive policies are mostly correct but commit errors on some portion of the states, rendering them useless in solving every problem of the domain. When the reward is only at goal states, the well known policy rollout approach cannot improve the performance of such faulty policies. Discrepancy Search has been developed in search to leverage the structural information of the heuristic functions which tends to be mostly-correct due to the human support. In this paper, we use reactive policies in discrepancy search for planning, in place of the heuristic functions. In our initial experiments, our proposed approach is effective in improving the performance of the given faulty reactive policies. The proposed approach outperformed the policy rollout as well as the reactive policies themselves. We will conclude with our research plan in the extension of the current proposal.

Introduction

Machine Learning (ML) has been successfully applied to many real life application domains, from image classification to natural language processing. AI Planning is no exception and ML techniques have been applied to AI planning and has shown some successes (Khardon 1999; Martin & Geffner 2000; Yoon, Fern, & Givan 2002). The role of ML here is producing a classifier that classifies an optimal or good action conditioned on the current state and the goal. Then, the sequential application of a good classifier to any problem in the target domain, will result in a goal state with high probability. The most successful approach of ML to planning was learning from sampled solution trajectories (Khardon 1999; Martin & Geffner 2000; Yoon, Fern, & Givan 2002) and the classifier produced by the ML techniques took the form of ordered list of rules (Khardon 1996). The resulting classifier is called (reactive) *policy* (Martin & Geffner 2000; Yoon, Fern, & Givan 2002) following the Markov Decision Process (MDP) framework or *control knowledge* (Estlin & Mooney 1996; Huang, Selman, & Kautz 2000; Aler, Borrajo, & Isasi 2002; Fern, Yoon, & Givan 2004) following search control framework. The classifier is called *reactive* control or policy when the only input to the classifier is the current state and goal information, without any information on the history of the action choices or state sequences that led to the current state.

To be useful in planning, a classifier for planning must be highly accurate. One wrong-selection among any of the state sequence of long plan trajectory could result in the failure on the whole planning problem, even if the classifier made correct choices all along the trajectory except in one state. As reported by (Khardon 1999; Martin & Geffner 2000; Yoon, Fern, & Givan 2002), ML technique found good reactive policies that achieve high success ratio on some planning domains like Blocksworld. Due to the nature of induction in ML techniques, often automatically found reactive policies, are prone to have some faults, typically caused by overfitting. *Ensemble* (Breiman 1996) can overcome overfitting but Ensemble itself can be faulty and it takes more time to train and get a set of reactive policies (Yoon, Fern, & Givan 2002). From decision theoretic planning, iterative updating of policy has been developed and one can use approximate policy iteration (API) (Bertsekas & Tsitsiklis 1996) in correcting the faulty policies. API (Fern, Yoon, & Givan 2003) has been shown effective in correcting faulty policies for planning, but it needs time in updating the policies and there is no guarantee that API always will produce improved policy over the current policy.

Ensemble and API techniques both need the assumption that the hypothesis space for the classifier is complete in every state. When the hypothesis space for the classifier cannot choose the correct action in every state of the target domain, either of the technique has no chance in selecting the correct actions on every state in the target domain.

In this proposal, we would like to answer the following question, "how to use reactive policies found by an ML technique when there is some possibility that the policies can be faulty". Or what if the hypothesis space for the ML is not complete, means that the knowledge representation (KR) for the hypothesis is good for most of the states in the target domain but for small portion of the states. We seek a deductive technique in using possibly faulty policies. Policy rollout (Bertsekas & Tsitsiklis 1996) is one such technique that can improve the performance of the given policy by sequentially choosing the actions considered best in one-step

look ahead policy evaluation. Although policy rollout is a powerful technique that can improve upon the given policy, when the reward is limited to a specific state like goal state and the faulty choice of the given policy is sporadic across the trajectory of the policy, then policy rollout may not be able to improve the performance of the given policy. Even multi-level policy rollout (Xiang Yan & Van Roy 2004) cannot fix the problem unless the faulty choices are limited to the initial part of the trajectories. In this proposal, we seek an answer to this sporadic distribution of faulty choices of a policy, in search techniques.

In search community, due to the involvement of human, many search application domains have a good set of well-designed features and have good heuristic functions. For some search application domains, heuristic functions are mostly-correct in guiding the search path but though infrequent, they can be wrong. *Discrepancy Search* (William D. Harvey 1995) is developed to leverage such structural information about heuristic function, or when the heuristic function is mostly-correct. In this proposal, we will employ discrepancy search in using mostly correct reactive policies.

The remainder of this proposal is structured as follows. We will describe the proposed technique. Then, we will show initial experimental results. Finally, we will give research direction following this proposal.

Proposed Algorithm

Figure 1 shows our proposed algorithm. The big picture of our algorithm is the same as the Discrepancy Search (DS). One difference is the consideration of the nature of the planning. Unlike the typical applications of the traditional DS, the depth of the search for planning problems can be arbitrarily long. And one cannot follow the given policy indefinitely. To address this issue, in the **Neighbors** function of figure 1, in every expansion of a node, we assign weight. Weight can be understood as the discrepancy from a state to another state with regard to the input policy. The paths favored by the policy will be weighted lower, enabling deeper search following those paths. The depth of the search is increased by the amount of the weight calculated by the assign-weight function. In this proposal, we suggested to use 1 - (the probability that the policy selects the path) + ϵ, where $\epsilon > 0$. Investigating alternative techniques like logarithmic use of the probability of the choice for the assign-weight function is in our agenda for the research. The parameter ϵ gives a natural horizon to the paths that follows the input policy, preventing paths with arbitrary depth.

Figure 1 also suggests a natural way of using stochastic policy. The assign-weight function is designed to consider a stochastic policy. During machine learning, based on the purity of the coverage in training data, one can easily assign the weight of the policy or the weight of each rule of the policy. And this information can be effectively used in stochastic search and in our algorithm. Weighting the selection of the heuristic function in DS has been studied (Walsh 1997; Bedrax-Weiss 1999). The weight setting in these works were on the level or depth of the tree, and not on the intrinsic stochastic nature of the heuristic function or the policy. So, our suggestion of using stochastic policy in DS will be an interesting extension of the weighted version of the DS technique.

Discrepancy-Search $(\mathbb{S}, \pi, D)$
// D: depth limit, $\mathbb{S}$: problem or state, π: input policy
$Q \leftarrow \{(\mathbb{S}, 0)\}$
// search queue: stores pairs of states and depths
$s \leftarrow$ **first**(Q)
repeat until **goal-reached**$(\text{first}(s))$
  Neighbors$(s, \pi, 0)$
  $s \leftarrow$ **first**(Q)
return Plan(s)

Neighbors (s, π, d)
// d: neighbor depth, s: state, π input policy
if **second**$(s) > D$; **return** // over the discrepancy limit
if $d > 1$; **return** // over the neighbor discrepancy limit
$N \leftarrow$ **Next-States**(**first**(s))
// enumerate direct neighbors
for-each n *in* N
  $w \leftarrow$ **assign-weight**(s, n, π)
  $Q \leftarrow$ **add**$(Q, (n, w +$ **second**$(s))$
  // add to search Q a new pair of state and depth
  Neighbors$((n, w +$ **second**$(n)), \pi, d + w)$

assign-weight (s', n, π)
// *example assign weight function*
//s': state, n: next state, π: input policy
return $(1 - p(\pi, s', n) + \epsilon)$
// $p(\pi, s', n)$: probability that the given policy moves to state n from the current state s'

Figure 1: Discrepancy Search with a Reactive Policy: The search algorithm is the same as that of limited discrepancy search. The discrepancy limit is D. Note that the discrepancy depth is not increased by one. The discrepancy depth is increased with proportional to the probability that the path is chosen by the input policy.

Preliminary Experiments

To test the performance of discrepancy search with reactive policies, we conducted experiments on Blocksworld and Driverlog. We randomly selected a policy learned from our previous research (Yoon, Fern, & Givan 2005) then we compared the performance of the policy in 3 techniques, policy as it is, rollout policy and discrepancy search with the policy. The figure 2 shows the performance of the each technique on the corresponding planning domains. Column labeled P shows the success ratio (SR) of the policy as it is. The success ratio here is measured as the number of solved problems in 100 randomly generated problems. Column labeled PR shows the SR of rollout policy of the given policy. Column labeled DS(n) shows the SR of discrepancy search with the policy, where we limit the number of the discrepancies to n.

For Blocksworld, we used 20 blocks problems and for Driverlog, we used 3 links, 4 drivers, 4 trucks and 8 package problems. As indicated in the figure 2, the rollout policy does not improve the performance of the faulty reactive policy. The faulty selections of actions happen sporadically across the trajectory, and the rollout policy does not address these faults. Rather the discrepancy search cures the faulty choices of the policy efficiently and improves the performance of the given policy. Even with discrepancy limit 1, the DS performs better than PR, and DS effectively correct the faulty choices while PR could not.

Domains	P	PR	DS(1)	DS(2)	DS(4)
Blocksworld	0.7	0.7	0.8	0.9	1
Driverlog	0.4	0.4	0.7	0.8	1

Figure 2: Using Reactive Policy

Research Direction

As an extension of the current proposal, we would develop discrepancy search techniques for stochastic planning domains and multi-agent planning domains, like Hearts or WarCraft domains. For the stochastic planning domains, the problem is the outcome distribution of the actions. We will assume that the reactive policy can designate not only the desired action choices but also the desired outcomes of the actions among possible outcomes of the actions. Similar idea of designating the outcome of an action is used in different purpose by (Boutilier, Dearden, & Goldszmidt 2000). Here the idea was used in symbolic value iteration. In our proposed work, the idea will reduce the branching factor to be considered in the AND-OR or ExpectiMax search. For the multi-agent domains, we would assume a similar situation, where the reactive policy can designate the expected action choices of the other agents as well as the current agent. This will again reduce the branching factor of the MIN-MAX search tree and will result in an effective and efficient algorithm.

Also we are planning to extend the proposed algorithm in heuristic based forward search setting in deterministic planning domains. In the heuristic search, the discrepancy search will be used in enumerating neighbors of a node. In this extension, we consider the nodes that are within one discrepancy from the current node with respect to the given policy, as neighbors of the current node. First, this will make the planner faster when the policy favors the same actions as the heuristics, since usually reactive policy's execution is faster than heuristic based action choices calculations. Second, when the heuristic is in a local minimum, and the policy favors orthogonal actions to the heuristic's favorites, this approach might help in escaping the local minimum, resulting in improvement on success ratio.

Note

Many of the ideas of this paper derive from joint work with Robert Givan

Acknowledgement

I thank Alan Fern for the helpful comments on this work.

References

Aler, R.; Borrajo, D.; and Isasi, P. 2002. Using genetic programming to learn and improve control knowledge. *AIJ* 141(1-2):29–56.

Bedrax-Weiss, T. 1999. *Optimal Search Protocols*. Ph.D. Dissertation, University of Oregon.

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 2000. Stochastic dynamic programming with factored representations. *AIJ* 121(1-2):49–107.

Breiman, L. 1996. Bagging predictors. *Machine Learning* 24(2):123–140.

Estlin, T. A., and Mooney, R. J. 1996. Multi-strategy learning of search control for partial-order planning. In *AAAI*.

Fern, A.; Yoon, S.; and Givan, R. 2003. Approximate policy iteration with a policy language bias. In *Proceedings of the 16th Conference on Advances in Neural Information Processing*.

Fern, A.; Yoon, S.; and Givan, R. 2004. Learning domain-specific control knowledge from random walks. In *ICAPS*.

Huang, Y.-C.; Selman, B.; and Kautz, H. 2000. Learning declarative control rules for constraint-based planning. In *ICML*, 415–422.

Khardon, R. 1996. Learning to take actions. In *AAAI/IAAI, Vol. 1*, 787–792.

Khardon, R. 1999. Learning action strategies for planning domains. *AIJ* 113(1-2):125–148.

Martin, M., and Geffner, H. 2000. Learning generalized policies in planning domains using concept languages. In *KRR*.

Walsh, T. 1997. Depth-bounded discrepancy search. In *IJCAI*, 1388–1395.

William D. Harvey, M. L. G. 1995. Limited discrepancy search. In Mellish, C. S., ed., *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95); Vol. 1*, 607–615. Montréal, Québec, Canada: Morgan Kaufmann, 1995.

Xiang Yan, Persi Diaconis, P. R., and Van Roy, B. 2004. Solitaire: Man versus machine. In *NIPS*.

Yoon, S.; Fern, A.; and Givan, R. 2002. Inductive policy selection for first-order MDPs. In *UAI*.

Yoon, S.; Fern, A.; and Givan, R. 2005. Learning measures of progress for planning domains. In *AAAI*.

Planning with Soft Regular Constraints

Alessandro Zanarini
Département de génie informatique
École Polytechnique de Montréal
C.P. 6079, succ. Centre-ville, Montreal, Canada H3C 3A7
azanarini@crt.umontreal.ca

Abstract

We introduce a new approach for encoding STRIPS planning problems as Constraint Satisfaction Problems: the encoding makes use of automata for modelling the dynamics of the objects involved in the domain. We describe a total order planner based on Constraint Programming that takes advantage of this encoding, using global constraints to model the automata and to reduce the search space significantly. The planner can be easily extended, using soft global constraints, in order to deal with preferences among the goals in infeasible problems. The soft planning infrastructure can also be exploited to build an effective search heuristic and to approximate the plan length.

Introduction

Constraint Programming (CP) is considered an efficient and effective paradigm for solving classical planning problems (see (Nareyek *et al.* 2005)). Several optimal parallel planners based on CP have been proposed by the research community over the years such as CPlan (Van Beek & Chen 1999), GP-CSP (Do & Kambhampati 2001), CSP-Plan (Lopez & Bacchus 2003) and they have shown their efficiency compared to other optimal parallel planners based on SAT or planning graph encodings. The common approach for CP-based planners is to encode the planning problem as a Constraint Satisfaction Problem (CSP) and to use well-known CP techniques to solve the CSP like generalized arc consistency (GAC) or conflict-based backjumping (CBJ) (see (Dechter 2003)). Most of the CP based planners encode the CSP starting from a planning graph representation and take advantage of this in order to add mutex constraints to the CSP model.

We propose a new kind of encoding that exploits automata for modelling the dynamics of the objects that are involved in the planning problem. We show how easily this model can be extended in order to take into account preferences on the goals. We show some experimental results on a preliminary version of the planner.

The remainder of the paper is organized as follow: in Section 2 we introduce the new encoding and the planner. In Section 3 we show the soft version of the planner that can handle preferences on the goals, Section 4 shows the benefits that the soft planner can bring also to speed up the search. In Section 5 we give some experimental results. Finally, in Section 6 conclusions are given.

Modelling planning problems with automata

The basic idea is to model the planning problem as a set of automata. Each automaton describes the dynamics of a single object (or entity) involved in the planning problem; the states of the automata denote the states of the objects and transitions between the automata states represent actions. Intuitively, if you look at the set of current states of the automata as a whole, this corresponds to the current world state. The initial state of the world and the goals are represented by the initial state and the final state of the automata. A valid (total order) plan is a sequence of actions that is recognized by each automaton; in other words it is a sequence of actions that brings every automaton (object) from the initial state to a final state.

Consider, for example, a simple instance of the blocks world domain in which you have as the initial state $\texttt{onTable}(\texttt{a}), \texttt{onTable}(\texttt{b}), \texttt{on}(\texttt{c},\texttt{b})$ and as the goal $\texttt{on}(\texttt{a},\texttt{b}), \texttt{on}(\texttt{b},\texttt{c}), \texttt{onTable}(\texttt{c})$. The problem is modelled as three automata that describe the state of the blocks. Figure 1a shows the automaton for block a; automata for block b and c are analogous.

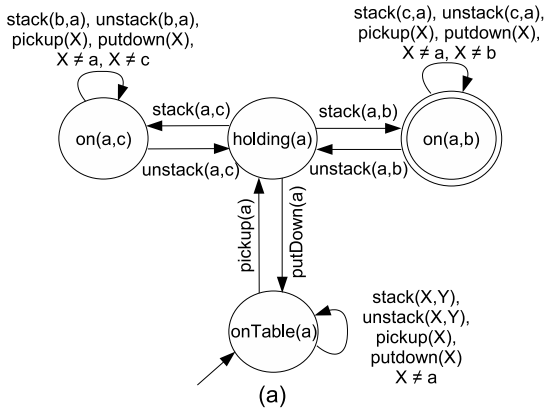


Figure 1: Automata of blocks world example.

Formally, given a plan of length L, the definition of the

$CSP = (X, D, C)$ is:

- $X = (X_1, \ldots, X_L)$: a sequence of variables that represents the total order plan;
- $D = (D_1, \ldots, D_L)$: the variable domains, each domain initially contains all the possible actions.
- $C = (Regular_1, \ldots, Regular_k)$: the set of constraints defined on the set of variables; each regular constraint represents an automaton (i.e. object dynamics in the planning problem).

Given a variable X_i, the instantiation $X_i = a_j$ denotes that the action a_j should be performed in the time step i.

Given an action a_j with preconditions $Pre(a_j)$ and effects $Eff(a_j)$, we denote by $Pre_o(a_j)$ the subset of preconditions that contain the literal (object) o and analogously with $Eff_o(a_j)$ the subset of effects that contain o. We use O to denote the set of objects involved in the planning problem. Given an object $o \in O$, we write $P(o)$ for the set of all the possible propositions that involve o (i.e. the propositions that contain the literal o).

Let o be an object of the planning problem: the related automaton A_o contains one state for each possible combination of the propositions in $P(o)$; in order to simplify the notation, given a state s_k of A_o, we use s_k to denote also the conjunction of propositions represented by the state itself. A transition (action a_{s_k,s_q}) is present between two states s_k and s_q iff $Pre_o(a_{s_k,s_q}) \subseteq s_k$ and $s_q = s_k \oplus Eff_o(a_{s_k,s_q})$ (where $A \oplus B$ is defined as the operation that adds to A all the positive effects of B and deletes from A all the negative effects of B).
The global constraint $Regular_o$ is used to model the automaton A_o. Note that the variable set is constrained by several Regular constraints; this implies that an action a_j for a given time step can be performed iff it is consistent for each regular constraint i.e. the preconditions of a_j are met in each automaton; formally, $\bigcup_{o \in O} Pre_o(a_j) = Pre(a)$.
Intuitively, the regular constraints filter the domains in such a way that only the actions for which the preconditions are met, are kept in the domains. Moreover the global nature of the regular constraints allows to filter also the actions that can be hypothetically instantiated in a given time step (i.e. the action precondition are met in that time step) but that do not lead to the final states within the given plan length horizon. This kind of reasoning restricts the search space and effectively guides the search towards the goals.

Generally, the automata built in such a way, have a high number of states; since the regular constraint propagation algorithm has a complexity that is proportional to the number of the state of the underlying automaton hence it is worth minimizing the number of states of the automata.

Description of the algorithm Given a lower bound (eventually equal to 1) and an upper bound on the plan length, the search for a valid plan is performed, following these basic steps:

- set the plan length L to the associated lower bound;
- solve the related CSP problem with a plan length equal to L;
  - if a solution is found then stop and return the optimal total order plan
  - if no solution is found then increase the plan length and solve the new CSP problem; the iteration is stopped when no valid plan is found with a length equal to the upper bound.

Softening the planner

Soft constraints (see (Petit, Régin, & Bessière 2001) for further explanation) are a convenient modeling feature to find plans that can lead us "close" to the goal, to express preferences among the goals, or in general to deal with unsatisfiable planning problems. The presented model and planner can be easily extended in order to introduce soft constraints.

In each automaton we introduce a set of transitions $t = (s_i, s_f) \in T_{fake}$ that go from each state to the final state; these transitions represent fake actions and are exploited to compute the violations. To do that, we introduce a cost function $f : T_{fake} \rightarrow \mathbb{R}^+$. In order to deal with the quantitative approach proposed with the planning description language PDDL 3.0, in which a goal is either satisfied or unsatisfied, we can use the following function:

$$\forall\, t \in T_{fake} : f(t) = \begin{cases} 0 & \text{if the transition starts from a final state} \\ 1 & \text{if the transition starts from a non final state} \end{cases}$$

Another interesting violation function can be the distance to a goal expressed as the number of remaining actions we should perform to reach it without considering the interaction with the other goals; this is equivalent to the number of states that are present in the automaton between a state and the final state. Given the function $d : S \rightarrow \mathbb{N}$ that represents the shortest sequence of action to achieve the goal then

$$\forall\, t = (s_i, s_f) \in T_{fake} : f_d(t) = d(s_i)$$

The CSP model and the planner are adapted in the following way:

- *Violation variables*: for each automaton (regular constraint) we associate a cost variable that represents the violation. A total violation variable $TotalViolation$ is added to the model as a function of the previously defined variables.
- *Cost Regular Constraints*: we use cost regular constraints to deal with automata in which there is the notion of cost associated to the transitions.
- *Objective*: a minimization objective is added to the model for the total violation variable.
- *Slack Variable*: given a plan of length L we add a variable X_{L+1} instantiated to the fake action. Clearly if all the goals are achieved within L time steps, then all the final transitions will be from final states to final states so the violation will be null. In the case in which there is at least one unachieved goal, the fake action will lead to the final states but with a corresponding violation cost.

Note that the function that relates $TotalViolation$ to the violations of single automata can be seen as a way to express preferences among the goals. Obviously if we want to give more importance to a given goal we should give it more weight in the function. The underlying CP framework also allows us to define more complex relationships between violation variables: for example, assuming we are using the distance violation function f_d, we can use a constraint that states that the absolute value of the difference between each pair of variables must be less than a given threshold: this can be seen as a way to express fairness (we do not allow a goal to be reached while another is very far from being achieved).

Further Advantages of this (soft) planner

Building a search heuristic from the soft planner

In many traditional planners every time a valid plan is not found, the plan length is increased and the search is restarted. Clearly, with this approach, we revisit a large part of the search space at each iteration. However, we can exploit soft planning infrastructure to build a heuristic to speed up the search in soft and also traditional planning problems.

We use the distance based violation function and we search for a plan that is as close as possible to the goals, that is it minimizes the total violation variable. For this variable we propose two functions: $TotalViolation_{sum} = \sum_i Violation_i$, the sum of the violations from individual automata, and $TotalViolation_{max} = max_i(Violation_i)$, the greatest individual violation. Once we prove that there is no valid plan of length L (i.e. $TotalViolation > 0$) we store the best solution found and exploit it for the next iteration, in two ways:

- *heuristic*: we branch first on the same values as in the stored solution, in order to quickly arrive to a promising region of the search space. Note that the proposed heuristic slightly differs from the one proposed in (Bonet & Geffner 2001); in that approach the chosen action is the one that leads to a state that is as close as possible to the goal but with the strong assumption of considering the goals independent and without considering the interference among the actions. In our approach the stored solution brings us as close as possible to the goal considering the goal interations and the interference among the actions. Even though we have not compared experimentally the two heuristics with this planner, we believe that our solution should be more effective.
- *violation bound*: the violation cost of the best solution found in the previous iteration is a valid upper bound on the violation variable. Clearly this will help to prune the search space better during the current iteration.

Plan length increase approximation

In the basic algorithm, when no solution is found for a given plan length, the plan length is increased by 1. Again, the soft planner gives us some information that we can use to get a closer approximation of the plan length.

Consider the soft planner with the distance based violation function and total violation equal to the max of the automata violations. Since we are minimizing the total violation, the cost of the best solution found indicates the minimum number of additional actions we should perform to achieve the goals. Hence, given a plan length L_i in iteration i and the best total violation found $TotalViolation_i^*$ at iteration i, we can set the plan length of the following iteration to $L_{i+1} = L_i + TotalViolation_i^*$.

Proposition 1. *The planner with the plan length increase approximation is optimal.*

Proof. Let L_i be the length of the plan at iteration i, P_i^* the best plan at iteration i with $TotalViolation_i^* > 0$ where the total violation is computed using the max function over the distance based violations. Suppose that there exists an optimal valid plan P^* of length $L^* < L_i + TotalViolation_i^*$. Consider then the partial plan $P^*_{partial}$ in which the first L_i actions are equal to the plan P^*. With $P^*_{partial}$ all the goals can be achieved with a number of actions at most equal to $L^* - L_i < TotalViolation_i^*$. So $P^*_{partial}$ has a violation strictly less than P_i^*, hence P_i^* is not the best plan with length L_i. □

Experimental results

The planner was implemented in ILOG Solver 6.1. To illustrate its behavior, we report preliminary experiments on reduced instances of the Zeno Travel problem (see (ICAPS06 2006)) in which two airplanes (A1 and A2), two persons (P1 and P2) and four cities (C1, C2, C3 and C4) are present. The instance has been modelled with four automata representing the two airplanes and the two persons. The violation function used is f_d that considers the distance to the final state of the automata. We consider different goals in order to test the soft and hard planners; the following table shows the different instances in terms of initial state and final state:

	A1	A2	P1	P2
Instance 1	C3→C2	C3→C3	C3→C1	C3→C2
Instance 2	C3→C4	C3→C3	C3→C1	C3→C2
Instance 3	C3→C2	C1→C3	C2→C1	C3→C2

The tests were performed on a Pentium-M 1.6GHz with 1GB RAM; the following table shows the results (plan length and the time expressed in seconds for finding a feasible plan) for solving the instances with the soft planner and the hard (traditional) planner; some basic techniques for breaking the symmetries have been introduced in the planners.

	Plan Length	Hard Planner	Soft Planner
Instance 1	6	1.6	1.5
Instance 2	7	23.4	3.9
Instance 3	8	23.2	5.2

We tested the impact of the different features of the soft planner on an instance with a feasible plan of length 9. The following table show the results (B: violation bound, P: plan

length increase approximation, S: basic symmetry breaking techniques, H: search heuristic, Back: backward search):

Planner features	Time	Planner Features	Time
Basic	153.3	+B+P+S+H	34.7
+B	160.0	+B+P+H+Back	28.9
+B+P	141.3	+P+S+H+Back	25.3
+B+P+S	123.8	+B+P+S+H+Back	24.3

In these preliminary tests, we can see that the search heuristic allows an interesting performance boost and in general each proposed feature brings some performance increase.

In order to experiment with preferences, we defined some preferences on the goals of the first 3 instances: particularly, we expressed the total violation as $TV = V_{A1} + V_{A2} + 2 * V_{P1} + 2 * V_{P2}$ where V_{A1}, V_{A2}, V_{P1} and V_{P2} are the violations of the airplanes and of the persons; with this objective function the preference is clearly given to the persons. We used the distance based violation function for the single violations and we searched for the best plan with a tighter plan length upper bound:

	Plan Length	Time	Violations
Instance 1	4	2.2	4
Instance 1	5	4.4	2
Instance 2	5	8.8	3
Instance 2	6	6.1	1
Instance 3	6	2.0	2
Instance 3	7	3.9	1

With a tighter upper bound on the plan length, it was not possible to satisfy all the goals hence the plan with the minimum violation has been found. The time for solving the instances with preferences is comparable to the time for solving instances without preferences (with the same plan length). Note that the flexibility of the framework allowed us to introduce preferences among the goals simply by adding an ad hoc objective function.

Discussion and open issues

The presented encoding raises one main challenge: it is not always obvious how to choose the set of objects to fully and correctly model the problem (actually we could choose each entity of the planning problem but then the number of automata would become intractable). Furthermore, the termination condition for the soft planner is actually given by achieving all the goals or reaching the upper bound on the plan length. Both issues are currently under investigation.

An interesting aspect that we are currently studying, is the introduction of no-goods recording in the planner. Most of the current state-of-the-art planners showed that it is a very powerful method to improve the performance of the planner.

Another aspect to investigate in future studies is the introduction of stronger symmetry breaking techniques. It is well known that total order plans present a lot of symmetries (two or more actions can be executed in whatever order) and this can degrade the performance in cases where there is no valid plan for a given plan length. In order to prove the infeasibility of a problem for a given plan length, the actual planner explores all the search space while symmetry breaking methods can help to reduce it significantly.

To the best of our knowledge, no proposed CP-based planner exploits global constraints that are commonly known as a powerful tool to speed up the search. The contributions of this paper are:

- a new encoding for the planning problem;
- use of global constraints for solving planning problems;
- a new violation measure for the soft regular constraint;
- a CP-based planner that provides tools to express preferences on goals;
- exploitation of the soft planning infrastructure for building an effective heuristic.

The implementation of the planner in ILOG Solver (probably the best CP framework commonly used by the research community and by industry) will allow us to introduce and exploit several of the sophisticated techniques that the CP community has proposed. Even if the proposed planner has some limitations, the actual implementation and the possible improvements that can be introduced to speed up the search seem promising.

References

Bonet, B., and Geffner, H. 2001. Planning as Heuristic search. *Artificial Intelligence* 129:5–33.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann Publishers.

Do, M. B., and Kambhampati, S. 2001. Planning as constraint satisfaction: Solving the planning garph by compiling it into CSP. *Artificial Intelligence* 132:151–182.

ICAPS06. 2006. Workshop on Preferences and Soft Constraint in Planning. *http://www.cis.strath.ac.uk/derek/PSCinP.html*.

Lopez, A., and Bacchus, F. 2003. Generalizing GraphPlan by Formulating Planning as a CSP. *International Joint Conference on Artificial Intelligence IJCAI-2003* 954–960.

Nareyek, A.; Freuder, E. C.; Fourer, R.; Giunchiglia, E.; Goldman, R. P.; Kautz, H.; Rintanen, J.; and Tate, A. 2005. Constraints and AI Planning. *IEEE Intelligent Systems* 20:62–72.

Petit, T.; Régin, J.-C.; and Bessière, C. 2001. Specific Filtering Algorithms for Over Constrained Problems. In *Principles and Practice of Constraint Programming – CP-2001: Proceedings of the Seventh International Conference*. Springer-Verlag LNCS 2239.

Van Beek, P., and Chen, X. 1999. CPlan: A Constraint Programming Approach to Planning. *Proceedings of the 16th National Conference on Artificial Intelligence* 585–590.