ICAPS 2006

The English Lake District, Cumbria, UK

VS1

Workshop on Preferences and Soft Constraints in Planning

Alfonso Gerevini
University of Brescia, Italy

Derek Long
University of Strathclyde, UK














ICAPS 2006

The English Lake District, Cumbria, UK

Workshop on Preferences and Soft Constraints in Planning

WS1

Alfonso Gerevini
University of Brescia, Italy

Derek Long
University of Strathclyde, UK

ICAPS 2006
Workshop on Preferences and Soft Constraints in Planning

Table of contents

ICAPS 2006
Workshop on Preferences and Soft Constraints in Planning

Preface

The planning community has traditionally been primarily concerned with the problem of generating a plan that satisfies a collection of goals. In many real-world planning problems, it is necessary to trade the benefits of achieving goals against the cost of achieving them. In practice, many goals are not absolute requirements, but an expression of preferences of varying degrees of importance. Some of these goals might even be unsatisfiable, perhaps because achieving them all would demand more resources than are available. In these problems, it is important to generate plans of good or optimal quality by achieving a subset of the goals, while attempting to maximise the benefits and miminise the costs in doing so. Relative value of plans can be expressed as preferences over the goals to achieve, constraints on the trajectory of the plan, or over other properties such as the resources used by the plan.

Soft constraints on plan trajectories are constraints over possible actions in the plan and intermediate states reached by the plan. Soft problem goals are goals that are desirable, but that do not have to be necessarily achieved. When a planning problem involves soft constraints or goals, some of them may be more important than others. In planning with soft constraints and goals, the best quality plan should satisfy as many of the soft constraints and goals as possible, according to one or more specified preference relations distinguishing alternative feasible plans.

Soft constraints have been extensively studied in the CP community, but are still a relatively new topic in the planning community. In this workshop we seek to bring together the experiences and expertise of the CP community and of the planning community in order to identify the common issues of interest and the approaches to these problems that can be shared. We also intend to use the workshop as an opportunity to present approaches to the management of soft constraints and goals in planning developed by competitors in the ICAPS'06 International Planning Competition, in which this problem is a focus.

This volume contains 11 papers. The papers concern various approaches to planning extended with different methods for representing and handling preferences and soft constraints. In particular, five papers describe implemented planning systems that are among the participants in the 2006 International Planning Competition. At the time of writing, these planners are running the competition benchmarks, which include many test problems requiring to handle preferences and soft constraints.

We hope that the workshop will be a stimulating and fruitful opportunity to discuss the state-of-the-art in planning with preferences and soft constraints, as well as an to identify promising directions for current and future research in this important emerging field.

Alfonso Gerevini (co-chair) Derek Long (co-chair)

Organizers

- *Alfonso Gerevini (co-chair), University of Brescia, Italy*
- *Derek Long (co-chair), University of Strathclyde, UK*

Programme Committee

- *Alfonso Gerevini (co-Chair), University of Brescia, Italy*
- *Derek Long (co-Chair), Strathclyde University, UK*
- *Peter van Beek, University of Waterloo, Canada*
- *Ronen Brafman, Ben Gurion University, Israel*
- *Yannis Dimopoulos, University of Cyprus, Cyprus*
- *Carmel Domshlak, Technion, Israeli Institute of Technology, Israel*
- *George Ferguson, Rochester University, USA*
- *Maria Fox, Strathclyde University, UK*
- *Patrik Haslum, Australia National University, Australia*
- *Lina Khatib, NASA Ames, USA*
- *Rao Kambhampati, Arizona State University, USA*
- *Robert Morris, NASA Ames, USA*
- *Bart Peintner, SRI, USA*
- *Francesca Rossi, University of Padova, Italy*
- *Alessandro Saetti, University of Brescia, Italy*
- *Toby Walsh, University of New South Wales, Australia*

Planning with Temporally Extended Preferences by Heuristic Search

Jorge Baier and **Jeremy Hussell** and **Fahiem Bacchus** and **Sheila A. McIlraith**
Department of Computer Science
University of Toronto
Toronto, Canada
[jabaier|hussell|fbacchus|sheila]@cs.toronto.edu

Abstract

In this paper we describe a planner that extends the TLPLAN system to enable planning with temporally extended preferences specified in PDDL3, a variant of PDDL that includes descriptions of temporal plan preferences. We do so by compiling preferences into nondeterministic finite state automata whose accepting conditions denote achievement of the preference described by the automata. Automata are represented in the planning problem through additional predicates and actions. With this compilation in hand, we are able to use domain-independent heuristics to guide TLPLAN towards plans that realize the preferences. We are entering our planner in the *qualitative preferences* track of IPC5, the 2006 International Planning Competition. As such, the planner description provided in this paper is preliminary pending final adjustments prior to the close of the competition.

Introduction

Standard goals in planning allow us to distinguish between plans that satisfy the goal and those that do not, however, they fail to discriminate between the quality of different successful plans. Preferences, on the other hand, express information about how "good" a plan is thus allowing us to distinguish between desirable successful plans and less desirable successful plans.

PDDL3 (Gerevini & Long 2005) is an extension of previous planning languages that includes facilities for expressing preferences. It was designed in conjunction with the 2006 International Planning Competition. One of the key features of PDDL3 is that it supports temporally extended preference statements, i.e., statements that express preferences over sequences of actions. In particular, in the *qualitative preferences* category of the planning competition preferences can be expressed with temporal formulae that are a subset of LTL (linear temporal logic). A plan satisfies a preference whenever the sequence of states generated by the plan's execution satisfies the LTL formula representing the preference.

PDDL3 allows each planning instance to specify a problem-specific metric used to compute the value of a plan. For any given plan, over the course of its execution various preferences will be violated or satisfied with some preference perhaps being violated multiple times. The plan value metric can depend on the preferences that are violated and the number of times that they are violated. The aim in solving the planning instance is to generate a plan that has the best metric value, and to do this the planner must be able to "monitor" the preferences to determine when and how many times different preferences are being violated. Furthermore, the planner must be able to use this information to guide its search so that it can find best-value plans.

We have crafted a preference planner that uses various techniques to find best-value plans. Our planner is based on the TLPLAN system (Bacchus & Kabanza 1998), extending TLPLAN so that fully automated heuristic-guided search for a best-value plan can be performed. We use two techniques to obtain heuristic guidance. First, we translate temporally extended preference formulae into nondeterministic finite state automata that are then encoded as a new set of predicates and action effects. When added to the existing predicates and actions, we thus obtain a new planning domain containing only standard ADL-operators. Second, once we have recovered a standard planning domain we can use a modified relaxed plan heuristic to guide search. In what follows, we describe our translation process and the heuristic search techniques we use to guide planning. We conclude with a brief discussion of related work.

Translation of LTL to Finite State Automata

TLPLAN already has the ability to evaluate LTL formulae during planning. It was originally designed to use such formulae to express search control knowledge. Thus one could simply express the temporally extended preference formulae in TLPLAN directly and have TLPLAN evaluate these formulae as it generates plans. The difficulty, however, is that this approach is by itself not able to provide any heuristic guidance. That is, there is no obvious way to use the partially evaluated LTL formulae maintained by TLPLAN to guide the planner towards satisfying these formulae (i.e., to satisfy the preferences expressed in LTL).

Instead our approach is to use the techniques presented in (Baier & McIlraith 2006) to convert the temporal formulae into nondeterministic finite state automata. The algorithm we use is a modification of one originally proposed in (Gerth *et al.* 1995). Intuitively the states of the automata "monitor" progress towards satisfying the original temporal formula. In particular, as the world is updated by actions added to the plan, the state of the automata is also updated dependent on changes made to the world. If the automata enters an accept-

ing state then the sequence of worlds traversed by the partial plan has satisfied the original temporal preference formula.

There are various issues involved in building efficient automata from an arbitrary temporal formula, and more details are provided in (Baier & McIlraith 2006). However, once the automaton is built, we can integrate it with the planning domain by creating an augmented planning domain. In the augmented domain there is a predicate specifying the current set of states that the automata could be in (it is a nondeterministic automata so there are a set of current states). Moreover, for each automata, we have a single predicate (the *accepting predicate*) that is true iff the automata has reached an accepting condition, denoting satisfaction of the preference. In addition, we define a post-action update sequence of ADL operators, which take into account the changes just made to the world and the current state of the automata in order to compute the new set of possible automata states. This post-action update is performed immediately after any action of the domain is performed. TLPLAN is then asked to generate a plan using the new augmented domain.

To deal with multiple preference statements, we apply this method to each of the preferences in turn. This generates multiple automata, and we combine all of their updates into a single ADL action (actually to simplify the translation we use a pair of ADL actions that are always executed in sequence).

A number of refinements must be made however to deal with some of the special features of PDDL3. First, in PDDL3 a preference can be scoped by a universal quantifier. Such preferences act as parameterized preference statements, representing a set of individual preference statement one for each object that is a legal binding of the universal variable. To avoid the explosion of automata that would occur if we were to generate an distinct automata for each binding, we translate such preferences into "parameterized" automata. In particular, instead of having a predicate describing the current set of states the automata could be in, we have a predicate with extra arguments which specifies what state the automata could be in for different objects. Similarly, the automata update actions generated by our translator are modified so that they can handle the update for all of the objects through universally quantified conditional effects.

Second, PDDL3 allows preference statements in action preconditions. These preferences refer to conditions that must ideally hold true immediately before performing an action. These conditions are not temporal, i.e., they refer only to the state in which the action is performed. Therefore, we do not model these preferences using automata but rather as conditional effects of the action. If the preference formula does not hold and the action is performed, then, as an effect of the action, a counter is incremented. This counter, representing the number of times the precondition preference is violated, is used to compute the metric function, described below.

Third, PDDL3 specifies its metric using an "is-violated" function. The is-violated function takes as an argument the name of a preference type, and returns the number of times preferences of this type were violated. Individual preferences are either satisfied or violated by the current plan. However, many different individual preferences can be grouped into a single type. For example, when a preference is scoped by a universal quantifier, all of the individual preference statements generated by different bindings of the quantifier yield a preference of the same type. Thus the is-violated function must be able to count the number of these preferences that are violated. Similarly, action precondition preferences can be violated multiple times, once each time the action is executed under conditions that violated the precondition preference. The automata we construct utilizes TLPLAN's ability to manipulate functions to keep track of these numbers.

Finally, PDDL3 allows specification of hard temporal constraints, which can also be viewed as being hard temporally extended goals. We also translate these constraints into automata. The accepting predicate of these automata are then treated as additional final-state goals. Moreover, we use TLPLAN's ability to incrementally check temporal constraints to prune from the search space those plans that already have violated the constraint.

Heuristic Search

The new augmented planning domain no longer has temporally extended preferences. Instead, the domain is much like a standard planning domain. Thus, we can compute relaxed plans and use those relaxed plans to compute heuristics.

In particular, we have augmented TLPLAN to allow it to compute relaxed state sequences: sequences of states that can be generated from the current state when ignoring the delete effects of actions. Notice that since the automata predicates are part of the new domain, the relaxed state sequences include predicates describing the "relaxed state" of the automata. Thus in the relaxed sequence of states not only can we compute various goal distance functions, but we can also compute various functions that depend on automata states. That is, we can compute information about the distance to satisfying various preferences. Since each preference is given a different weight in valuing a plan we can even weight the "distance to satisfying a preference" differently depending on the value of the preference.

Specifically, our heuristic function is a combination of the following functions, which are evaluated over partial plans. At the time of writing, these functions were not finalized.

Goal distance A function that is a measure of how hard it is to reach the goal. It is computed using the relaxed plan graph (similar to the one used by the FF planner (Hoffmann & Nebel 2001)). It computes a heuristic distance to the goal facts using a variant of the heuristic proposed by (Zhu & Givan 2005). The exact value of the k exponent in this heuristic is still being finalized.

Preference distance A measure of how hard it is to reach the preference goals, i.e., how hard it is to reach the accepting states of the various preference automata. Again, we use Zhu & Givan's heuristic to compute this distance.

Optimistic metric A lower bound[1] for the metric function

[1]Without loss of generality, we assume that we are minimizing the metric function.

of any plan that completes the partial plan, i.e., the best metric value that the partial plan could possibly achieve if completed to satisfy the goal. We compute this number assuming that no precondition preferences will be violated in the future, and assuming that all temporal formulae that are not currently violated by the partial plan will be true in the completed plan. To determine whether a temporal formula is not violated by the partial plan, we simply verify that its automaton is currently in a state from which there is a path to an accepting state. Finally, we assume that the goal will be satisfied at the end of the plan.

Relaxed Optimistic Metric In the same spirit as the optimistic metric, the relaxed optimistic metric is an *estimation* of the best metric value that a partial plan can have when completed to satisfy the goal. We compute this value as the best metric value of the relaxed states. As with the optimistic metric, we assume that no precondition preferences will be violated in the future.

The relaxed optimistic metric is more realistic than the optimistic metric since it takes into account that some automata satisfaction facts are possibly not achievable from certain states. The preferences corresponding to those unreachable facts are regarded by this metric as violated.

Discounted metric A weighting of the metric function evaluated in the relaxed states. Let $M(s_0)$ be the metric value of a state s_0, and $s_1, \ldots, s_n$ be the relaxed states reachable from state s until a fixed point is found. The discounted metric for s and discount factor r, $D(s, r)$, is computed as:

$$D(s,r) = M(s_0) + \sum_{i=0}^{n-1} (M(s_{i+1}) - M(s_i)) r^i.$$

The factor of r we are finally going to use is not yet decided.

The final heuristic function is obtained by a combination of the functions defined above.

Our planner is able to return plans with incrementally improving metric values. It does best-first search using the heuristic described above. At all times, it keeps the metric value of the best plan found so far. Additionally, the planner prunes from the search space all those plans whose relaxed optimistic metric is worse than the best metric found so far. This is done by dynamically adding a new TLPLAN hard constraint into the planning domain.

Discussion

The technique we use to plan with temporally extended preferences presents a novel combination of techniques for planning with temporally extended goals, and for planning with preferences.

A key enabler of our planner is the translation of LTL preference formulae into automata, exploiting work described in (Baier & McIlraith 2006). There are several papers that address related issues. First is work that compiles temporally extended goals into classical planning problems such as that of Rintanen (Rintanen 2000), and Cresswell and Coddington (Cresswell & Coddington 2004). Second is work that exploits automata representations of temporally extended goals (TEGs) in order to plan with TEGs, such as Kabanza and Thiébaux's work on TLPLAN (Kabanza & Thiébaux 2005) and work by Pistore and colleagues (Lago, Pistore, & Traverso 2002). A more thorough discussion of this work can be found in (Baier & McIlraith 2006).

There is also a variety of previous work on planning with preferences. In (Bienvenu, Fritz, & McIlraith 2006) the authors develop a planner for planning with temporally extended preferences. Their planner performs best first-search based on the optimistic and pessimistic evaluation of partial plans relative to preference formulae. Preference formulae are evaluated relative to partial plans and the formulae progressed, in the spirit of TLPLAN, to determine aspects of the formulae that remain to be satisfied. Also noteworthy is the work of Son and Pontelli (Son & Pontelli 2004) who have constructed a planner for planning with temporally extended goals using answer-set programming (ASP). Their work holds promise however ASP's inability to deal efficiently with numbers has hampered their progress. Brafman and Chernyavsky (Brafman & Chernyavsky 2005) recently addressed the problem of planning with preferences by specifying qualitative preferences over possible goal states using TCP-nets. Their approach to planning is to compile the problem into an equivalent CSP problem, imposing variable instantiation constraints on the CSP solver, according to the TCP-net. This is a promising method for planning, though at the time of publication of their paper, their planner did not deal with temporal preferences.

References

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

Baier, J. A., and McIlraith, S. 2006. Planning with first-order temporally extended goals. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*. To appear.

Bienvenu, M.; Fritz, C.; and McIlraith, S. 2006. Planning with qualitative temporal preferences. In *Proceedings of the Tenth International Conference on Knowledge Representation and Reasoning (to appear)*.

Brafman, R., and Chernyavsky, Y. 2005. Planning with goal preferences and constraints. In *Proceedings of The International Conference on Automated Plann ing and Scheduling*.

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

Gerevini, A., and Long, D. 2005. Plan constraints and preferences for pddl3. Technical Report 2005-08-07, Department of Electronics for Automation, University of Brescia, Brescia, Italy.

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

Hoffmann, J., and Nebel, B. 2001. The FF planning sys-

tem: Fast plan generation through heuristic search. *Journal of Art. Int. Research* 14:253–302.

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

Lago, U. D.; Pistore, M.; and Traverso, P. 2002. Planning with a language for extended goals. In *Proc. AAAI/IAAI*, 447–454.

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

Son, T., and Pontelli, E. 2004. Planning with preferences using logic programming. In Lifschitz, V., and Niemela, I., eds., *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-2004)*, number 2923 in Lecture Notes in Computer Science. Springer. 247–260.

Zhu, L., and Givan, R. 2005. Simultaneous heuristic search for conjunctive subgoals. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-2005)*, 1235–1241.

A Multivalued planning model for soft preconditions and preferences

M. Baioletti and **A. Milani** and **V. Poggioni** and **S. Suriani**
Dipartimento di Matematica e Informatica
Università di Perugia, Italy
{baioletti,milani,poggioni,suriani}@dipmat.unipg.it

Abstract

In this work a model for planning with multivalued fluents and graded actions, that allows preferences specifications and plan metric, is introduced. This model is based on the infinity-valued Łukasiewicz logic. In multivalued planning, fluents can assume truth values in the interval $[0, 1]$ and actions can be executed at different application degrees also varying in $[0, 1]$. The notions of planning problem and solution plan also reflect a multivalued approach. Multivalued fluents and graded actions allow to model many real situations where some features of the world are fuzzy and where actions can be executed with varying strength. Moreover, the specification of soft preconditions allows a more flexible model for the solution plan while the use of soft preferences increases the expressive power of the plan quality specification. A correct/complete algorithm which solves bounded multivalued planning problems based on MIP compilation is also described and a prototype implementation is presented.

Introduction

In many planning domains disagreement and fuzziness of real world properties need to be modeled. Since classical actions and constraints with crisp boolean values are not adequate to represent these domains, it is necessary to introduce into planning no–boolean fluents and soft constraints that can partially violated.

Consider for instance a simple scenario where the action *open_door* is available in the domain and modifies the truth value of the fluent *door_is_open*.

A typical probabilistic planning model would regard to the degree of uncertainty about the fluent *door_is_open* and the action *open_door* as the uncertainty of the system, or belief degree.

A planner able to handle numerical resources would treat the fluent *door_is_open* as a real function whose initial value has to be measured, for instance by the openness angle. Since it is not possible to model the strength of *open_door*, the value of *door_is_open* can only be changed by a fixed value.

From a multivalued point of view, instead, the truth value of the fluent *door_is_open* denotes a real world property which can vary from 0 (the door is completely closed) to 1 (the door is completely opened). In other words the door can "really" be opened at an intermediate degree, between "completely opened" and "completely closed". Moreover the graded action *open_door* can be applied with different "strength", ranging from 1 (completely applied) to 0 (not applied at all). The resulting effect on the fluent *door_is_open* will depend on the application degree of the action *open_door* and on the previous truth value of the fluent itself.

The idea underlying multivalued planning is to model in a logical framework fuzzy concepts, such as "the door is slightly open" or "the door is wide open", and graded application of actions with additive effects, such as "open the door a bit more" or "open the door as much as possible".

It is worth noticing that many fuzzy concepts cannot be represented by numerical resources, because the fuzzy values could not be obtainable by a measurement process or they could be affected by subjective factors.

In this sense the approach is orthogonal with respect to planning models for uncertainty and incompleteness. Since in this approach an underlying multivalued logic is used, the management of soft constraints in action preconditions, goals, and preferences is straightforward.

In the multivalued/fuzzy logics scenario we preferred to use a logic based on T-norms because of its well-founded mathematical aspects. Among this class of logics we have chosen the Łukasiewicz logic because the semantics of its operators is more suitable for our framework. For these reasons we have excluded other well-known logics as Product Logic, in which for instance the negation operator is not involutive.

In this paper a new framework for a general theory of multivalued planning based on the infinite-valued Łukasiewicz logic (Cignoli, D'Ottaviano, & Mundici 2000; Hajek 1998) is introduced and an algorithm for a multivalued planner is described. The proposed model can be seen as a generalization of the classical planning, since boolean and multivalued actions and fluents can be used in the same domain.

The paper is organized as follows. First of all some basic concepts on Łukasiewicz logic are recalled, then the model of multivalued planning is presented and an algorithm solving problems in such a model is proposed. An example is introduced and finally related works and conclusions are pointed out.

A brief introduction to Łukasiewicz logic

Łukasiewicz logic is a multivalued extension of classical logic, well known from the theoretical point of view (Hajek 1998; Cignoli, D'Ottaviano, & Mundici 2000). In this section we will report only some useful definitions. The conjunction operator, $x \odot y = \max(0, x + y - 1)$, has the properties of a generic $T - norm$: it is commutative, associative, monotonic with respect to all the variables; it has 0 as absorbing element and 1 as unit element.

The implication is defined as the residual of the conjunction operator and is expressed by $x \to y = \min(1, 1-x+y)$.

The negation, which can be defined as $\neg x = x \to 0$, is expressed as $\neg x = 1 - x$ and is involutive. Using the last property and De Morgan's law it is possible to define a disjunction operator, expressed as $x \oplus y = min(1, x + y)$. This operation is a *co*–norm: it is commutative, associative and monotonic with respect to all the variables; it has 1 as absorbing element and 0 as unit element.

Implication and disjunction are related by the condition $x \to y = \neg x \oplus y$.

It is possible to define an operation similar to subtraction as $x \ominus y = x \odot \neg y = \max(0, x - y)$, which under some condition on x and y can behave as the inverse of additive disjunction. Moreover $(x \ominus y) \ominus z = x \ominus (y \oplus z)$ holds.

Note that in Łukasiewicz logic the lattices operators called *min*–conjunction $x \wedge y = x \odot (x \to y)$ and *max*–disjunction $x \vee y = (x \to y) \to y$ coincide, respectively, with the minimum and the maximum between x and y.

The Model

In this section the model of multivalued planning is presented. This model allows to use boolean fluents and classical actions jointly with two new kinds of fluents and actions, called multivalued fluents and graded actions. Here the model is shortly presented, for details see (Baioletti *et al.* 2006).

Multivalued States

Let $\mathcal{F}$ be the finite set of fluents. A state is described by a function S assigning to each fluent f at a given time–step t a truth value belonging to $[0, 1]$, denoted by $S_t(f)$, i.e.

$$S : \mathcal{F} \times \mathcal{T} \to [0, 1].$$

It can be seen as a natural extension of the classical state definition in which the range of the state function is $[0, 1]$ instead of $\{0, 1\}$.

The value at a time–step t of a negated fluent is denoted by $S_t(\neg f)$ and it is computed as the negation of the truth value of f in S_t, i.e. $S_t(\neg f) = \neg S_t(f)$.

Multivalued Constraints

Definition 1 *Multivalued Constraints.*

A multivalued constraint *is represented by a tuple* $(\mathtt{k}, \chi, \mathtt{f})$*, where* $\mathtt{k} \in \{\mathtt{b}, \mathtt{s}\}$ *is a flag that denotes the kind of the constraint (boolean or soft),* χ *is a value in* $[0, 1]$ *representing a threshold and* $\mathtt{f}$ *is a literal (i.e. a fluent or its negation). A boolean constraint in the state* S_t *is defined by the classical constraint* $S_t(f) \geq \chi$*, while the soft constraint is defined by the Łukasiewicz implication* $\chi \to S_t(f)$*.*

The expression $\chi \to S_t(f)$ is chosen to encode the soft constraints because it is a fuzzy extension of the crisp relation $S_t(f) \geq \chi$. In fact, if $S_t(f) \geq \chi$, then the implication assumes truth value 1, while if $S_t(f) < \chi$, the implication assumes a value lesser than 1. The higher the difference between $S_t(f)$ and χ, the lesser the value of the implication.

Definition 2 *Satisfaction Degree of a Multivalued Constraint.*

Let $\mathcal{S}$ *a set of states,* X *a set of constraints* $mc = (\mathtt{k}, \chi, \mathtt{f})$*, where* χ *is a threshold in* $[0, 1]$*,* $\mathtt{f}$ *is a literal and* $\mathtt{k}$ *a flag in* $\{\mathtt{b}, \mathtt{s}\}$*. Let* $S \in \mathcal{S}$*, then the* satisfaction degree *of the constraint in the state* S *is defined by the function* $\mathcal{D} : \mathcal{S} \times X \to [0, 1]$*,*

$$\mathcal{D}(S, mc) = \begin{cases} 1 & \text{if } \mathtt{k}=\mathtt{b} \text{ and } S(\mathtt{f}) \geq \chi \\ 0 & \text{if } \mathtt{k}=\mathtt{b} \text{ and } S(\mathtt{f}) < \chi \\ \chi \to S(\mathtt{f}) & \text{if } \mathtt{k}=\mathtt{s} \end{cases} \quad (1)$$

Although we restricted our attention to constraints over literals, the extension to constraints over arbitrary propositional formulae of Łukasiewicz logic is straightforward.

Graded Actions

Extending the classical action definition introduced in STRIPS, an action A is described by a list of preconditions *prec(A)*, a list of effects *eff(A)* and list of preferences *pref(A)*.

Each precondition and each preference are defined by a multivalued constraint of the form presented in Definition 1. The preferences do not affect the executability of the actions but they are involved only in the plan quality evaluation. Differently from the classical model, an application degree belonging to $[0, 1]$ is assigned to each action. The degrees 0 and 1 have the same meaning as in the classical model.

The execution of an action can modify the truth value of fluents. In this framework we propose to model the action effects by means of additive changing over the fluent values.

Moreover, this model allows to use classical boolean actions simply bounding/restricting the domain for its application degree. In other words, when a multivalued planning domain needs boolean actions, the user have to restrict the action application domain to $\{0, 1\}$.

Action Preconditions and Action Executability

Definition 3 *Action Preconditions.*

The preconditions of an action A *are defined by a list of multivalued constraints* $\mathtt{mc}=(\mathtt{k}, \chi, \mathtt{f})$*.*

The value of χ can be read as the threshold for the fluent truth value. The maximum satisfaction degree of the precondition is reached when the fluent truth value is above the threshold.

The *satisfaction degree of a precondition* corresponds to the satisfaction degree of the multivalued constraint computed by $\mathcal{D}(S, mc)$.

Definition 4 *Action Executability.*

An action $A \in \mathcal{A}$ *is executable in a state* S_t *with application degree* $\alpha_t \in [0, 1]$ *if* $\alpha_t \leq \alpha_{max}(A, S_t)$*, where*

$$\alpha_{max}(A, S_t) = \bigwedge_{mc \in pre(A)} \mathcal{D}(S_t, mc). \quad (2)$$

α_{max} provides the maximum application degree of the action A in the state S_t.

It is reasonable to compute a maximum application degree because in this multivalued extension of planning the planner can choose how much to apply an action. While in the classical model the planner can only decide if to apply an executable action A, in this model the planner can choose to apply A with any actual application degree smaller or equal to $\alpha_{max}(A, S_t)$.

The value of $\alpha_{max}(A, S_t)$ is computed using the lattice conjunction (i.e. the minimum) among the precondition contributions $\mathcal{D}(S_t, mc)$.

The use of the *min*–conjunction $\wedge$ instead of the usual Łukasiewicz conjunction $\odot$ is justified by the fact the latter, differently from the former, is not idempotent and if x and y are smaller than 1, $x \odot y$ is smaller than x and y.

Finally, it is important to note that a negative precondition has a particular semantics. In the classical model, a negative precondition is equivalent to require the fluent to be false. In this model it requires that the truth value of the fluent is small, i.e. that its negation is enough true. In fact $\chi \rightarrow S(\neg f)$ holds 1 if and only if $S(\neg f) \geq \chi$, i.e. $S(f) \leq 1 - \chi$.

From the point of view of the action executability we have to point out that the violation of boolean constraints yields to a maximum application degree equal to 0, while the use of soft constraints allows to execute actions even if $S(f) \leq \chi$, i.e it is possible to have a solution plan that violates soft preconditions.

However the violations of soft preconditions have to be taken into account, i.e. a plan with some unreached preconditions cannot have the same quality of a plan with all satisfied preconditions (see *Plan Quality* section).

Action Preferences

Recently, the AI planning community has driven its attention to the specification of preferences in goal and action precondition conditions (Gerevini & Long 2005). The aim is to specify some conditions that are not necessary for the action executability or the goal satisfiability but that are considered important for the plan quality estimation. In this multivalued model we consider both boolean and soft preferences specifying either a crisp or a fuzzy threshold.

Definition 5 *Preferences.*
The preferences of an action A are defined by a list of multivalued constraint `mc=(k,`χ`,f)`*, where χ is called* preference threshold *of* `f`.

How the penalties for violations of preferences are used in the metric is shown in *Plan Quality* section.

Action Effects and Action Execution

In this model, effects are intended as modifications of the truth value of fluents. A positive effect provides an increment of the truth value, while a negative effect provides a decrement. The amount of increment is proportional to the application degree of the action and can be tuned by a parameter which models the strength of the effect. It is not allowed to have an action having both a positive and a negative effect over the same fluent.

Definition 6 *Action Effects.*
The effects of an action A are defined by a list of pair (γ, e), where e is a literal and γ is a real number in $[0, +\infty)$, called weight *of A over e.*

Note that positive (negative) literals denote positive (negative) effects.

Definition 7 *Action Execution.*
Let A an action executed with application degree α_t in a state S_t and $(\gamma, e) \in eff(A)$, then

$$S_{t+1}(e) = S_t(e) \oplus \Delta_{Ae}, \tag{3}$$

where $\Delta_{Ae} = \min(\gamma \cdot \alpha_t, 1)$.

The quantity Δ_{Ae} can be seen as a unary operator depending on γ; it allows to reduce or to amplify the effect over the fluent e with respect to the application degree. In particular, it reflects the idea of effects which are proportional to the application degree of the action, in fact the amount Δ_{Ae} is directly proportional to both the application degree α_t and the weight of A over e.

It is easy to prove that applying the definition (3) to a negated fluent $\neg e$ we obtain

$$S_{t+1}(e) = S_t(e) \ominus \Delta_{Ae}.$$

Therefore, as expected, the action execution decrements the truth value of fluents in its negative effects by the quantity Δ_{Ae}, while increments the the truth value of fluents in its positive effects.

It is also possible to encode boolean effects that assign 0 or 1 to the truth value of fluents, independently from the execution degree. We chose the symbolic value ∞ denoting an infinite weight γ. In this case the effect is equal to a classical action effect that makes true or false the value of the fluent. In fact, the intended semantics for these effects is: `(`∞`,e)` over the fluent `e` produces $S_{t+1}(e) = 1$, while `(`$\infty, \neg$ `e)` produces $S_{t+1}(e) = 0$.

Multivalued Planning Problems

Definition 8 *Multivalued Planning Problems.*
A multivalued planning problem *is defined by a quadruple $(I, \mathcal{O}, G, \sigma)$ where $I : \mathcal{F} \rightarrow [0, 1]$ is a truth assignment over the fluents and denotes the initial state; $\mathcal{O}$ is a set of boolean and multivalued actions; G is the goal list and is defined by a list of multivalued constraints* `mc=(k,`χ`,f)`*; σ is a real number in $[0, 1]$ called* global threshold.

A valid plan is an ordered sequence of pairs $\{(a_0, \alpha_0), (a_1, \alpha_1), \ldots, (a_{n-1}, \alpha_{n-1})\}$, where each α_i represents the application degree of action $a_i \in \mathcal{O}$; moreover, let $S_0 = I$ and S_{i+1} the state resulting from the execution of the action a_i in S_i, each action a_i must be executable in S_i with application degree α_i.

Definition 9 *Solution Plan.*
A solution plan is a valid plan where in the final state S_n

$$\bigwedge_{mc \in G} \mathcal{D}(S_n, mc) \geq \sigma \tag{4}$$

Note that a solution plan is a plan which achieves all the goals in a multivalued fashion. Similarly to the preconditions, the problem goals are defined by classical/soft constraints and preferences and their satisfaction degrees are computed by the $\mathcal{D}$ function. Then, σ specifies a global parameter which represents a minimal satisfaction degree desired for the solution plan.

Plan Quality

The plan quality is evaluated by a metric specified by any arithmetic expression. The presented model allows involving in the metric violations of preferences or soft preconditions, as well as action application degrees. An optimal plan will minimize (maximize) the metric.

In order to compute the penalties of violations of soft preconditions and preferences we suppose that every multivalued constraint `mc` is labeled by an univocally constant `name`. The total of penalties for the violations of constraints labeled by `name` is computed by

$$\Lambda(\texttt{name}) = \sum_i \Lambda_i(\texttt{name}). \tag{5}$$

Let $\texttt{mc} = (\texttt{k}, \chi, \texttt{f})$ the multivalued constraint labeled by `name` and S_i a state, then

$$\Lambda_i(\texttt{name}) = \neg\, \mathcal{D}(S_i, \texttt{mc}).$$

The sum $\Lambda(\texttt{name})$ is computed considering only the actions that are in the solution plan, having the constraint `mc` as precondition or preference. Note that, the function $\mathcal{D}(S_i, \texttt{mc})$ is defined both for boolean and soft preconditions. But since violations of preconditions can exist only for soft preconditions, then $\mathcal{D}(S_i, \texttt{mc}) = 1$ for every boolean constraint.

A violation of a boolean preference increases by 1 the value of Λ, while a violation of a soft preference increase Λ by a value in [0,1] depending on "how much" the soft constraint is violated. This yields that a violation of a boolean preference can have much more weight than the same violation of a soft preference. In fact, let two preferences `mc1`, `mc2` respectively labeled by `p1` and `p2` with $\texttt{mc1} = (\texttt{b}, \chi, \texttt{f})$, $\texttt{mc2} = (\texttt{s}, \chi, \texttt{f})$ and $S_t(f) = v < \chi$, then $\Lambda_t(\texttt{p1}) = 1$ while $\Lambda_t(\texttt{p2}) = \chi - v$. It is simple to prove that $\Lambda_t(\texttt{p1}) \geq \Lambda_t(\texttt{p2}),\ \forall v, \chi \in [0, 1]$.

This difference is very useful for the user because it allows to take into account not only the satisfaction degrees of the soft constraints but also the differences between classical and soft constraints. The user can specify a boolean preference when he/she considers very important its violation, while it can use a soft preference when he/she considers that is important only the difference between the desired value and the actual value.

The presence of precondition violations can appear in contradiction with the classical planning model. In fact a sequence of actions having not achieved preconditions is not a valid plan in the classical model, while in this multivalued model, preconditions can be also partially achieved.

Further, it can be interesting to specify action application degrees in the metric for the plan quality, especially in such domains where action application degrees are directly proportional to action costs. In this case the function

$$\Omega(A) = \sum_{t=0}^{T-1} \alpha_t \tag{6}$$

is used to compute the total of application degrees α_t of each occurrence of the action A in the solution plan.

An Algorithm for Multivalued Planning

The idea of the algorithm is to solve bounded multivalued planning problems in three steps. At first, a planning graph (Blum & Furst 1997) is constructed until either the upper bound U for time–steps is exceeded or a state verifying the necessary condition for the solution existence is reached. Then the graph representation is translated into a MIP problem (Mixed Integer Programming), and finally a MIP problem solver is called. If the MIP problem has a solution, then also the planning problem has a solution which is directly derived from the MIP problem solution. If the MIP problem has no solution, the main loop continues by adding a new level until a solution is found or the bound U is exceeded. It is easy to prove that this algorithm is correct and complete in the class of the U–bounded multivalued planning problems. The pseudo–code of the algorithm is presented in Figure 1, where `initialize` is the procedure which creates the planning graph starting from the initial state; `goal_sat` is the function which checks the goal condition, `add_new_level` is the procedure which adds to the graph a new level of actions and the corresponding new level of fluents; `translate_to_MIP` and `solve_MIP` create and solve the MIP problem (if the MIP problem has no solution, $\perp$ is returned by `solve_MIP`); finally, the function `extract_solution` translates the solution of the MIP problem into the corresponding solution of the planning problem.

Planning Graph Construction

The planning graph built in our system is similar to the planning graph used in Graphplan and other planning systems. The planning graph is a $(2T+1)$–leveled graph $G = (V, E)$ whose vertex set is $V = F_0 \cup A_0 \cup F_1 \cup A_1 \cup \ldots \cup F_T$, where F_t is the set of *multivalued fluents* at time $t = 0, \ldots, T$ and A_t is the set of *graded actions* at time $t = 0, \ldots, T-1$. Each edge in E links either a fluent $f \in F_t$ to an action $a \in A_t$, such that $(k, \chi, f) \in pre(a)$, or an action $a \in A_t$ to a fluent $f \in F_{t+1}$, such that $(\gamma, f) \in eff(a)$.

In the following, $\alpha_t^*(a)$ denotes the upper bound for the action application degree α_t. It is computed by a function depending on the precondition set of the action and on upper and lower bounds of the truth values of the fluents involved in such preconditions.

```
function planner
begin
    t := 0;
    G := initialize(I);
    while t<U and not goal_sat(G) do
          t := t+1;
          add_new_level(G,t);
    end;
    sol := ⊥;
    while t<U and sol=⊥ do
          M := translate_to_MIP(G);
          sol := solve_MIP(M);
          if sol=⊥ then
            t := t+1;
            add_new_level(G,t);
          endif;
    end;
    if sol≠ ⊥ then
      return extract_solution(sol);
    else
      return ⊥ ;
    endif
end
```

Figure 1: The algorithm

The values of $S_0(f)$ and $\alpha_0^*(a)$ can be computed in an exact way, because the initial state is completely known. $S_0(f)$ is the initial truth value of the fluent f and $\alpha_0^*(a)$ is computed directly using the formula (2).

If $t > 0$, since it is not known which action will be executed in each time–step, the truth value $S_t(f)$ for each fluent $f \in F_t$ is unknown and, for the same reason, it is not possible to compute a realistic value of $\alpha_t^*(a)$ for each action $a \in A_t$.

Anyway, it is possible to compute a lower bound $\underline{S}_t(f)$ and an upper bound $\overline{S}_t(f)$ for the value $S_t(f)$ by taking into account the upper and the lower bounds for $S_{t-1}(f)$ and the maximum change (in the positive and in the negative cases) caused to f by any possible action executed at time $t-1$.

In particular $\underline{S}_0(f) = \overline{S}_0(f) = S_0(f)$ holds.

Therefore for each fluent $f \in F_t$ and each action $a \in A_{t-1}$, we define the interval $[\underline{S}_t^a(f), \overline{S}_t^a(f)]$ which provides bounds for $S_t(f)$, knowing that the action a will be executed at time $t-1$. Clearly, since the maximum change will be achieved when a is executed at the largest possible application degree $\alpha_{t-1}^*(a)$, we have that $\underline{S}_t^a(f) =$

$$\begin{cases} \underline{S}_{t-1}(f) & \text{if } (\gamma, \neg f) \notin eff(a) \\ \underline{S}_{t-1}(f) \ominus \Pi_\gamma(\alpha_{t-1}^*(a)) & \text{if } (\gamma, \neg f) \in eff(a) \text{ and } \gamma \neq \infty \\ 0 & \text{if } (\infty, \neg f) \in eff(a) \end{cases}$$

and $\overline{S}_t^a(f) =$

$$\begin{cases} \overline{S}_{t-1}(f) & \text{if } (\gamma, f) \notin eff(a) \\ \overline{S}_{t-1}(f) \oplus \Pi_\gamma(\alpha_{t-1}^*(a)) & \text{if } (\gamma, f) \in eff(a) \text{ and } \gamma \neq \infty \\ 1 & \text{if } (\infty, f) \in eff(a) \end{cases}$$

Finally, we obtain the interval

$$[\underline{S}_t(f), \overline{S}_t(f)] = [\min_{a \in A_{t-1}} \underline{S}_t^a(f), \max_{a \in A_{t-1}} \overline{S}_t^a(f)] \quad (7)$$

We can now show how to compute the values of $\alpha_t^*(a)$, for an action $a \in A_t$, by using the intervals $[\underline{S}_t^b(f), \overline{S}_t^b(f)]$ computed for each fluent $f \in F_t$ and each action $b \in A_{t-1}$.

For this purpose we define $\alpha_{bt}^*(a)$ as the maximum execution degree of the action a, computed considering $[\underline{S}_t^b(p), \overline{S}_t^b(p)]$ as the interval bounding the truth value at time t for each fluent p involved in the preconditions of the action a. In other words, $\alpha_{bt}^*(a)$ is the maximum execution degree of the action a at the time t, knowing that the action b has been executed at time $t-1$. More formally, $\alpha_{bt}^*(a) =$

$$\bigwedge_{(k,\chi,f) \in pre(a)} \mathcal{D}(\overline{S}_t^b, (k, \chi, f)) \quad \wedge \quad \bigwedge_{(k,\chi,\neg f) \in pre(a)} \mathcal{D}(\underline{S}_t^b, (k, \chi, \neg f)) \quad (8)$$

where f is a fluent.

Finally, we compute

$$\alpha_t^*(a) = \max_{b \in A_{t-1}} \alpha_{bt}^*(a). \quad (9)$$

It is easy to prove that the computation of $\alpha_t^*(a)$ directly from $\underline{S}_t(f)$ and $\overline{S}_t(f)$, would produce a higher estimate of the maximum application degree.

Anyway a complete implementation of this method is not feasible. In fact the computation of $\alpha_t^*(a)$ at a time t would require the computation of $\underline{S}_t^{a_0,\ldots,a_{t-1}}(f)$ and $\overline{S}_t^{a_0,\ldots,a_{t-1}}(f)$ for every possible sequence of actions $a_0, \ldots, a_{t-1}$. Therefore we restrict the method of computing $\alpha_t^*(a)$ to take into account only the possible actions executed at the previous level $t-1$.

The first time–step where a solution can exist is the first t in which the satisfaction degree for the goals, computed by using $\underline{S}_t(f)$ and $\overline{S}_t(f)$, can be greater than the goal threshold σ. Therefore, the graph construction phase ends at a time $t \leq U$, if any, when the goal condition $G_t =$

$$\bigwedge_{(k,\chi,f) \in G} \mathcal{D}(\overline{S}_t, (k, \chi, f)) \wedge \bigwedge_{(k,\chi,\neg f) \in G} \mathcal{D}(\underline{S}_t, (k, \chi, \neg f)) \geq \sigma \quad (10)$$

is verified.

Constraints Extraction

A planning problem can be compiled into a Mixed Integer Programming (MIP) problem $\mathcal{M}$ by translating the constraints expressed by operators of Łukasiewicz logic into linear constraints on integer/real variables (Hähnle 1997).

The truth value of each fluent f and the application degree of each action a at time t are respectively denoted by the real variables x_{ft} and x_{at} in $[0, 1]$. For the boolean cases x_{ft} and x_{at} will be integer variables in $\{0,1\}$.

Different kinds of constraint must be satisfied in order to have a solution plan according to the model presented. They can be grouped in *action executability rules*, *action execution rules* and *goal satisfiability rules*; moreover, according to the definition of valid plan, we have to express a linearity condition for each plan called *plan linearity constraint*.

Finally if a metric is present in the problem, a suitable *objective function* has to be added to the MIP system.

It is easy to see that the number of constraints and the number of variables in the MIP problem $\mathcal{M}$ will be a polynomial of the number of fluents and actions in the planning graph.

Action Executability Rules At each time–step t, the actual application degree of an action a must be less (or equal) than the upper bound represented by $\alpha_t^*(a)$. Since the formula (2) requires to compute a minimum, it is easy to separate the contribution of soft preconditions from the contribution of boolean preconditions.

First, we have that for the soft preconditions

$$x_{at} \leq \min(\begin{array}{l} \min_{(s,\chi,f)\in pre(a)}\{\chi \to x_{ft}\}, \\ \min_{(s,\chi,\neg f)\in pre(a)}\{\chi \to \neg x_{ft}\} \end{array}).$$

Therefore for each $a \in A_t$ and for every $t = 0, \ldots, T-1$ we add to $\mathcal{M}$ the following constraints

$$\begin{cases} x_{at} \leq (1-\chi) + x_{ft} & \text{for all } (s,\chi,f) \in pre(a) \\ x_{at} \leq (1-\chi) + (1-x_{ft}) & \text{for all } (s,\chi,\neg f) \in pre(a) \end{cases}$$

considering that $x_{at} \leq 1$ is always verified.

The treatment of boolean preconditions is slightly more difficult. In fact for each $(b,\chi,p) \in pre(a)$, where $p = f$ or $p = \neg f$, it is required the creation of a new $0/1$ integer variable μ_{ft}^a and the addition to $\mathcal{M}$ of the following constraint

$$x_{ft} - \chi \leq \mu_{ft}^a \leq x_{ft} - \chi + 1, \text{ if } p = f$$

or

$$(1-\chi) - x_{ft} \leq \mu_{ft}^a \leq (1-\chi) - x_{ft} + 1, \text{ if } p = \neg f$$

and, in any case, the addition of the constraint

$$x_{at} \leq \mu_{ft}^a.$$

Action Execution Rules The truth value of a fluent f at a time–step $t = 1, \ldots, T$ is equal to:

- the truth value at the time–step $t-1$, if the action chosen at the time–step $t-1$ does not modify f
- the truth value at the time–step $t-1$ increased/decreased (with Łukasiewicz operators $\oplus$ and $\ominus$) by $\Delta_{f,t-1}$, if the action chosen has f as positive/negative effect and finite γ
- the truth value 1/0 if the action chosen has f as positive/negative effect and infinite γ

Therefore $x_{f,t+1}$ can be computed by the following equation $x_{f,t+1} =$

$$x_{ft} \oplus \left(\sum_{(\infty,f)\in eff(a)} \lambda_{at} + \bigoplus_{(\gamma,f)\in eff(a)} \Pi_\gamma(x_{at}) \right) \ominus \left(\sum_{(\infty,\neg f)\in eff(a)} \lambda_{at} + \bigoplus_{(\gamma,\neg f)\in eff(a)} \Pi_\gamma(x_{at}) \right) \quad (11)$$

where λ_{at} is the $0/1$ variable, which will be introduced in the following section *Linearity Plan Rule*.

Note that since only one action is executed at each time–step, in (11) at most one term among $\Pi_\gamma(x_{a,t-1})$ or λ_{at} is greater than zero and affects the value of x_{ft}.

The translation of (11) in MIP constraints is performed in several steps. First, it is possible to encode constraints of the kinds $z = x \oplus y$ and $z = x \ominus y$ in small linear systems with an additional binary variable. Then, to encode a constraint like $z = \oplus_{j=1}^n x_j$, instead of iterating the encoding of the binary operator $\oplus$, it is possible to use a linear system with a fixed number of inequalities obtained by a method of disjunctive programming (Balas 1979). Finally, the constraints representing (11) so obtained are added to the MIP subsystem $\mathcal{M}$.

Goal Satisfiability Rule This rule derives directly from the goal condition defined in (10) and requires that each subgoal (k,χ,f) must be fulfilled with a truth value greater or equal to the global threshold. Similarly to what happens for the preconditions, it is possible to separate the contribution of soft goals and boolean goals. Soft goals are encoded as

$$\begin{cases} (1-\chi) + x_{fT} \geq \sigma & \text{for all } (s,\chi,f) \in G \\ (1-\chi) + (1-x_{fT}) \geq \sigma & \text{for all } (s,\chi,\neg f) \in G \end{cases},$$

while, for the boolean goals $(b,\chi,p) \in G$, where $p = f$ or $p = \neg f$, the encoding uses the constraint

$$x_{fT} - \chi \leq \nu_f \leq x_{fT} - \chi + 1, \text{ if } p = f$$

or the constraint

$$(1-\chi) - x_{fT} \leq \nu_f \leq (1-\chi) - x_{fT} + 1, \text{ if } p = \neg f$$

and, in any case, the constraint

$$\nu_f \geq \sigma$$

(ν_f is a new $0/1$ variable).

Linearity Plan Rule With this constraint we express the condition that only one action can be executed in a given time–step, i.e. at each time t, $\exists! a \in A_t : x_{at} \neq 0$.

This constraint can be represented directly by the MIP system

$$\begin{cases} x_{at} \leq \lambda_{at} & \text{for all } a \in A_t \\ \sum_{a\in A_t} \lambda_{at} = 1 \end{cases} \quad (12)$$

where λ_{at} is a new $0/1$ variable, for each $a \in A_t$.

It is easy to check that this condition is true if and only if at each time–step t either $x_{at} = 0$ for all $a \in A_t$, or there exists a unique a such that $x_{at} \neq 0$.

The first case cannot really happen. In fact if there exists a solution where in at least one time–step no action was executed, such a solution would be equivalent to a shorter solution. Since our algorithm operates in an incremental way, this shorter solution would have been found in an earlier stage.

Objective function If a metric is present in the planning problem specification, then a suitable objective function ϕ is added to the MIP system $\mathcal{M}$. It is obvious that a MIP solver can only optimize linear functions, therefore we restrict the functional form of the metric to a linear combination of terms involving violations of preferences and soft preconditions and terms related to action execution degrees.

Suppose that in the metric a term $(\texttt{is}-\texttt{violated p})$ appears, where p is the name of a soft preference or a soft precondition (s, χ, q) in the action a. Then a new real variable, called y_{pt}, for $t = 0, \ldots, T-1$, is added to $\mathcal{M}$, whose value is exactly $\Lambda_t(p)$. Since the violation has not be taken into account if the action a is not executed, then it is easy to see that if $q = f$, for a fluent f,

$$y_{pt} = \lambda_{at} \ominus (\chi \rightarrow x_{ft})$$

while, if $q = \neg f$, the same formula holds, where x_{ft} is obviously replaced with $1 - x_{ft}$. Then the translation of the former equation is added to $\mathcal{M}$. In ϕ the term $(\texttt{is}-\texttt{violated p})$ is represented with the term $\sum_{t=0}^{T} y_{pt}$.

If p is the name of a boolean preference (b, χ, q), the variable y_{pt} associated is now a $0/1$ variable. We will also use a new $0/1$ variable, called μ_{pt}, subject to the constraint

$$x_{ft} - \chi \leq \mu_{pt} \leq x_{ft} - \chi + 1, \text{ if } q = f$$

or the constraint

$$(1 - \chi) - x_{ft} \leq \mu_{pt} \leq (1 - \chi) - x_{ft} + 1, \text{ if } q = \neg f.$$

We add to $\mathcal{M}$ the previous constraint and the translation of

$$y_{pt} = \lambda_{at} \ominus \mu_{pt}.$$

Finally, in ϕ the term $(\texttt{is}-\texttt{violated p})$ is again represented with the term $\sum_{t=0}^{T} y_{pt}$.

The treatment of terms involving the execution degrees is straightforward. A term $(\texttt{is}-\texttt{executed a})$ is represented in ϕ with the linear term $\sum_{t=0}^{T-1} x_{at}$.

Implementation

A prototype implementation of the previously described algorithm has been made in C++, using as a MIP solver the open–source library *lpsolve*, which is not very performant, although the preliminary results obtained on small examples are encouraging.

We expect that some further code optimizations and mainly the use of some more efficient linear programming libraries, like CPLEX, will produce good results from the computational point of view.

Another different approach to solve multivalued planning problems could be a compilation into a mixed boolean– real linear system, handled by solvers like MathSAT (Bozzano 2005), if there is no metric.

An Example

In order to have domains and problems with multivalued fluents and graded actions we have to add some little features to PDDL3. We started from this version of the language because it allows to specify preferences both in the action precondition and goal fields. In particular, we had to add:

- the field `(:type boolean|graded)` in action specification, in order to specify if the action is boolean or graded,
- two parameters in the structure of each precondition, preference or goal, in order to specify the kind of constraint (boolean or soft) and the value for the threshold,
- a parameter in the structure of each effect, in order to specify the weight of the action over the effect,
- the field `(:threshold v)` for the global threshold,
- the initial truth value for each fluent in the initial state,
- the possibility to label soft preconditions in order to use them in the function `(is-violated p)`, as the PDDL3 allows for preferences,
- the use of `(is-executed a)` in the metric specification, in order to use the action application degrees in the metric for the plan quality.

The SLIPPERY-GRIPPER Domain

In this small example we present a domain having generic objects that can be wet and must to be painted; there are also some grippers that can be wet and hold objects. These domain properties are represented by the following fluents: `(obj_wet o)`, `(painted o)`, `(gr_wet g)`, `(holding o g)` where `o` is an object and `g` is a gripper.

This domain has four actions. `dry` is a graded action that can dry a free gripper (represented by `(free g)`). `paint` is a graded action that can paint an object held by a gripper and prefers to paint not too much wet objects. `pickup` and `putdown` are boolean actions that respectively pick an object up from the table and put an object down to the table. The property to be ontable is represented by `(ontable o)`. A gripper can hold only one object; it can pick an object up only if it is free and the object is on the table; it can put an object down only if it holds it. A gripper can be wet, and we can safely use it only if it is not too much wet.

This example cannot realistically be modeled by classical planning domains, because qualitative properties like ”be wet” or ”be painted” cannot suitable represented by boolean propositions. A gripper is not only wet or not wet, it can be ”a little wet” or ”just damp”. To have soft preconditions allows, for instance, to execute the action *pickup(gr,obj)* even if the gripper is not totally dry. Moreover, the classical planning model is too rough for an action like *dry(gr)* because any planner applies it with the same strength, both when the gripper is *very wet* and when is *a little wet*. These are the action descriptions:

```
(:action paint
   :type graded
   :parameters (?obj - object ?gr - gripper)
   :precondition (and (b 1 (holding ?obj ?gr))
           (preference p1 s 0.5 (not(obj_wet ?obj))) )
   :effect (0.8 (painted ?obj)) )
(:action dry
   :type graded
   :parameters (?gr - gripper)
   :precondition (s 0.8 (gr_wet ?gr))
   :effect (0.9 not(gr_wet ?gr)) )
(:action pickup
   :type boolean
   :parameters (?obj - object ?gr - gripper)
   :precondition (and (b 1 (free ?gr))
           (b 1 (ontable ?obj))
           (p2 s 0.7 (not(gr_wet ?gr))) )
   :effect (and(∞ not(free ?gr)) (∞ (not(ontable ?obj)))
               (∞ (holding ?obj ?gr)) ))
```

```
(:action putdown
   :type boolean
   :parameters (?obj - object ?gr - gripper)
   :precondition (b 1 (holding ?obj ?gr))
   :effect (and(∞ (free ?gr)) (∞ (ontable ?obj))
               (∞ (not(holding ?obj ?gr))) ))
```

An example of a problem is presented. We suppose to have the goal to paint the objects minimizing the `dry` application and the violations of the preference `p1` and of soft precondition `p2`.

```
(define (problem example)
  (:domain slippery_gripper)
  (:objects gr1-gripper gr2-gripper obj1-object obj2-object)
  (:init (= 1 (free gr1)) (= 1 (free gr2))
         (= 1 (ontable obj1)) (= 1 (ontable obj2))
         (= 0.2 (gr_wet gr1)) (= 0.5 (gr_wet gr2))
         (= 0.2 (obj_wet obj1)) (= 0.7 (obj_wet obj2))
         (= 0.3 (painted obj1)) (= 0.6 (painted obj2))  )
  (:goal (and (s 0.8 (painted obj1))(s 0.7 (painted obj2))) )
  (:threshold 0.9)
  (:metric minimize (+ (* 5 (is-violated p1))
         (* 3 (is-violated p2)) (* 10 (is-executed dry)) ))
)
```

Related Works

One of the main related planning models is the non–Boolean approach introduced in (Miguel, Jarvis, & Shen 2001) based on *flexible operators* and *propositions*. The main difference is that the multivalued planning approach proposed here can manage a continuous range of values in [0,1]; another difference is that, in the flexible planning model, the mapping between preconditions and effects is obtained by a set of disjoint conditional clauses, where each clause determines the satisfaction degree of the effects associated with it. Despite of the fuzziness of the approach, the activation of a conditional clause is crisp. Finally the concept of graded action is not present: the operators are applied only at a fixed application degree and the effects of the action depend only by the satisfaction degree of preconditions. In our approach, instead, the planner decides which application degree to assign to an action among the available values.

Models for planning with resources (for example as in (Koehler 1998)) can also be related to our work. In these approaches numerical constraints can be posed on resources which are incremented/decremented by actions. It is easy to see that most types of resources can be encoded in our model by appropriate multivalued properties values and by additive effects.

Recently PDDL (Gerevini & Long 2005) has been extended in order to allow boolean preferences; the number of preference violations are used to estimate the plan quality. It is worth noticing that, in our approach, preferences are encoded by multivalued constraints with satisfaction/violation degree in $[0, 1]$. This allows to define more expressive and accurate metric functions because we can take into account not only the number of violations of boolean preferences but also the violations degree of the soft preferences.

Conclusions

In this work a model for planning in a multivalued framework has been introduced. The model allows to manage multivalued fluents and graded execution of actions. Moreover the model allows to specify soft constraints in preferences, preconditions and goals. This model is yet preliminary and requires further investigations: several extensions can be proposed in order to integrate the management of plan trajectory constraints and resources. Another important extension would be to allow the use of parallel actions in the plan, with the definition of a suitable concept of mutually exclusive actions.

An interesting direction for future investigations is how to redefine our multivalued planning model in a finite-valued logic. While the treatment of the constraints used in the preconditions and goals would not need major modifications, it is not clear how to model the multiplicative weights of the effects with a finite number of truth values. Anyway in a finite-valued logic the elicitation of fuzzy values would be easier than in an infinite valued one.

Different and more efficient approaches to solve multivalued planning problems need to be considered, such as graph based algorithms or using mixed boolean–real solvers. On the other hand the current algorithm, which uses a translation into a MIP problem, allows to introduce into the model linear objective functions and plan metrics and to solve the corresponding optimization problems.

Finally, from an experimental point of view, it is necessary to develop a set of benchmark problems for significant domains involving graded actions in order to test the performance of the implementation.

References

Baioletti, M.; Milani, A.; Poggioni, V.; and Suriani, S. 2006. A multivalued logic model of planning. In *Proc. of ECAI-06*.

Balas, E. 1979. Disjunctive programming. *Annals of Discrete Mathematics* 5:3–51.

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Art. Int.* 90(1–2):279–298.

Bozzano, M. e. a. 2005. The mathsat 3 system. In *Proc. CADE-20*.

Cignoli, R.; D'Ottaviano, I.; and Mundici, D. 2000. *Algebric Foundations of many–valued Reasoning.* Kluwer Academic Publisher, Dordrecht, Boston, London.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. *Tech. Rep. of Dept. of Electronics for Automation, University of Brescia, Italy.*

Hähnle, R. 1997. Proof theory on many–valued logic – linear optimization – logic design: Connections and interactions. *Soft Computing* 1(3):107–119.

Hajek, P. 1998. *Mathematics of Fuzzy Logic.* Kluwer Academic Publisher, Dordrecht, Boston, London.

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

Miguel, I.; Jarvis, P.; and Shen, Q. 2001. Efficient flexible planning via dynamic flexible constraint satisfaction. *Engineering Applications of AI* 14(3):301–327.

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

Subbarao Kambhampati
Department of Computer Science
Arizona State University
Tempe AZ, 85287-8809
rao@asu.edu

Thomas Vossen
Leeds School of Business
University of Colorado at Boulder
Boulder CO, 80309-0419
vossen@colorado.edu

Abstract

We present an integer programming approach for handling preferences and trajectory constraints in planning. Our main aim is to illustrate through examples how simple it is to express preferences and trajectory constraints by linear constraints over 0-1 variables. We are currently in the process of incorporating these constraints in the context of efficient integer programming encodings that we developed recently.

Introduction

Given the recent success of integer programming approaches to automated planning (van den Briel, Vossen, & Kambhampati 2005), we 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 explored 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. 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 paper we show numerous examples of how to express preferences and trajectory constraints by linear constraints over 0-1 variables. These constraints would need to be added to the integer programming formulation of the planning problem. Currently, we are still in the process of incorporating these constraints in the formulations described by van den Briel, Vossen and Kambhampati (2005).

The organization of this this paper is as follows. The next section is concerned with the formulation of integer optimization problems, that is, how to translate a verbal description of a problem into mathematical state-

ment. In particular, we will show how we can use 0-1 variables to model different relations between events. We then show various planning examples for both simple preferences and qualitative preferences, and some concluding remarks will be given at the end.

Modeling with 0-1 Variables

Integer programming is a powerful and natural modeling framework for solving optimization problems involving integer decision variables. The case where the integer variables are restricted to be 0 or 1 are referred to as 0-1 programming problems or binary integer programming problems. In mathematical programming, 0-1 variables are commonly used to represent binary choice, where binary choice is simply a choice between two things. For example, consider the problem of deciding whether an event should or should not occur. This decision can be modeled by a binary variable x, where $x = 1$, if the event occurs and, $x = 0$, if the event does not occur. Depending on the problem being considered, the event itself could be almost anything. For example, in scheduling, an event x could represent whether some job should be scheduled before another job.

Here we show some standard relationships between events and how they can be modeled by 0-1 variables.

- The relation that *at most one* of a set J of events is allowed to occur is represented by a packing constraint, $\sum_{j \in J} x_j \leq 1$.
- The relation that *at least one* of a set J of events is allowed to occur is represented by a cover constraint, $\sum_{j \in J} x_j \geq 1$.
- The relation that *exactly one* of a set J of events is allowed to occur is represented by a partitioning constraint, $\sum_{j \in J} x_j = 1$.
- The relation that neither or both events 1 and 2 must occur, that is, event 1 *equals* event 2, is represented by the linear equality $x_2 - x_1 = 0$.
- The relation that event 2 can occur only if event 1 occurs, that is, event 2 *implies* event 1, is represented by the linear inequality $x_2 - x_1 \leq 0$.

Sometimes, the occurrence of an event is limited to a set of pre-specified time periods $1 \leq t \leq T$. The decision whether an event should or should not occur at time period t can can be modeled by a time-indexed binary variable x_t, where $x_t = 1$, if the event occurs at time period t and, $x_t = 0$, if the event does not occur at time period t. For example, in planning, an event x_t could represent the execution of an action or the truth value of a proposition at a specific plan step.

Here we show some standard relationships between time dependent events and how they can be modeled by time-indexed 0-1 variables.

- The relation that event 1 may occur *at most once* during the time horizon, is represented by the linear inequality $\sum_t x_{1,t} \leq 1$.
- The relation that event 1 must occur *sometime* during the time horizon, is represented by the linear inequality $\sum_t x_{1,t} \geq 1$.
- The relation that if event 1 occurs, event 2 must occur *sometime-before* event 1, is represented by the linear inequalities $x_{1,t} \leq \sum_{1 \leq s < t} x_{2,s}$ for all $1 \leq t \leq T$.
- The relation that if event 1 occurs, event 2 must occur *sometime-after* event 1, is represented by the linear inequalities $x_{1,t} \leq \sum_{t \leq s \leq T} x_{2,s}$ for all $1 < t \leq T$.
- The relation that event 1 must *always* occur is represented by the linear equalities $x_{1,t} = 1$ for all $1 \leq t \leq T$.
- The relation that event 1 must occur *at the end* of the time horizon is represented by the linear equality $x_{1,T} = 1$.

Note that most of these standard relationships coincide with the new modal operators: `at-most-once`, `sometime`, `sometime-before`, `sometime-after`, `always`, `at end`, in PDDL3.0. Even though there some differences in semantics, this suggests that modeling these modal operators and the preferences and trajectory constraints that are expressed by them through integer programming should be rather straightforward.

In the next two sections we give various examples of how to model preferences and trajectory constraints by linear constraints over 0-1 variables. The examples are all borrowed from the International Planning Competition resources [1].

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 $y_{f,t}$

[1] http://zeus.ing.unibs.it/ipc-5/

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 it provides a concise representation of the resulting constraints.

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.

The goal preference p_4 "We would like that person3 and person4 are at city3" is expressed as follows.

```
(:goal (and (preference p4 (and
  (at person3 city3) (at person4 city3)))))
```

Note that preference p_4 is very similar to preference p_1 with the exception that p_4 is defined over a conjunction of fluents. For each fluent in the conjunction we will state a separate constraint, thus the inequalities corresponding to preference p_4 are given by:

$$p_4 \geq 1 - y_{\text{at person3 city3},T} \quad (4)$$

$$p_4 \geq 1 - y_{\text{at person4 city3},T} \quad (5)$$

Now, preference p_4 is violated if either or both person3 and person4 are not at city3 at the end of the plan.

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 $p_{5,\text{fly}?a?c1?c2,t}$ "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 p5
     (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:

$$p_{5,\text{fly}?a?c1?c2,t} \geq x_{\text{fly ?a ?c1 ?c2,t}} - \sum_{?p} y_{\text{in ?p ?a},t} \quad \forall 1 \leq t \leq T \quad (6)$$

Thus, preference $p_{5,\text{fly}?a?c1?c2,t}$ 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:

$$y_{\text{fragile ?b},t} - y_{\text{clear ?b},t} \leq 0 \quad \forall ?b, 1 \leq t \leq T \qquad (7)$$

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:

$$y_{\text{holding ?b},0} + \sum_{a \in A, 1 \leq t \leq T : \text{holding ?b} \in ADD(a)} x_t^a \leq 1 \quad \forall ?b \quad (8)$$

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:

$$\sum_t y_{\text{holding ?b},t} \geq 1 \quad \forall ?b \qquad (9)$$

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:

$$\sum_{1 \leq s < t} y_{\text{at ?t city2},s} \geq y_{\text{at ?t city1},t} \quad \forall ?t, 1 \leq t \leq T \quad (10)$$

Similarly the trajectory constraint "If a taxi has been used and it is at the depot, then it has to be cleaned."

```
(:constraints (and (forall (?t - taxi)
  (sometime-after (and (at ?t depot) (used ?t))
    (clean ?t)))))
```

This translates to a *sometime-after* relationship for all taxis $?t$. Note, however, that this trajectory constraint has two conditions, which if satisfied, require that taxi $?t$ is to be cleaned. The inequality corresponding to this trajectory constraint is given by:

$$y_{\text{at ?t depot},t} + y_{\text{used ?t},t} - 1 \leq \sum_{t \leq s \leq T} y_{\text{clean ?t},s} \quad \forall ?t, 1 \leq t \leq T \qquad (11)$$

Now, if taxi $?t$ is at the depot at step t ($y_{\text{at ?t depot},t} = 1$) and if it has been used ($y_{\text{used ?t},t} = 1$), then it must be cleaned sometime after step t ($\sum_{t \leq s \leq T} y_{\text{clean ?t},s} \geq 1$).

More examples can be presented, but we hope it is enough to bring the point across that integer programming provides a natural framework for modeling propositional planning with preferences and trajectory constraints.

Conclusions

We have shown numerous examples of how to model preferences and trajectory constraints by integer programming. 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. Even though we haven't had the time to implement this yet, we believe this can be done.

An interesting analysis for future work would be to see the impact on performance when preferences and trajectory constraints are added to integer programming formulation of the planning problem. Also we would like to compare the performance of the integer programming formulations that use preferences and trajectory constraints as side constraints (as shown in the examples in this paper) with integer programming formulations that handle preferences and trajectory constraints which are compiled down into PDDL2.2.

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.

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 using 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.

Planning with Goal Utility Dependencies

Minh B. Do
Embedded Reasoning Area
Palo Alto Research Center
Palo Alto, CA 94304
minh.do@parc.com

J. Benton & Subbarao Kambhampati
Computer Science & Engineering Department
Arizona State University
Tempe, AZ 85287
{j.benton,rao}@asu.edu

Abstract

Work in partial satisfaction planning (PSP) has hither to assumed that goals are independent. This implies that that individual goals have additive utility values. In many real-world problems we cannot make this assumption and thus goal utility is not additive. In this paper, we motivate the need for representing and handling goal utility dependencies in PSP and we provide a framework for representing them using the General Additive Independence (GAI) model (Bacchus & Grove 1995). We then present an algorithm based on forward heuristic planning to solve this problem using heuristics derived from the planning graph. To show the effectiveness of our framework, we provide empirical results on benchmark planning domains.

Introduction

Classical planning aims at finding a plan that achieves a set of conjunctive goals. Partial satisfaction (or oversubscription) planning relaxes this all-or-nothing constraint, focusing on finding a plan to achieve the "best" subset of goals (i.e. the plan that gives the maximum tradeoff between total achieved goal utilities and total incurred action cost). The process of selecting the set of goals on which to focus 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 actions to achieve them (making the cost of achieving them together significantly more or less than the sum of costs of achieving them in isolation) (ii) A set of goals may have *utility dependencies* in that achieving the goals together may lead to utility that is significantly different from the sum of utilities of achieving each goal in isolation. Both cost and utility dependencies are common in many real world applications such as NASA's data collection domains.

Although some recent work in partial satisfaction planning (van den Briel *et al.* 2004) has begun to handle cost dependencies between goals, there has not yet been any work on handling utility dependencies. The primary contribution of our paper is a systematic approach for handling cost and utility dependencies together in PSP. The main technical challenges here include developing a model where goal utility dependencies can be compactly represented and using utility interactions with cost interactions to find a high net benefit plan. Our approach builds upon the methods for handling utility dependencies from decision theory (c.f. (Bacchus & Grove 1995; Boutilier et. al. 2001)) and combinatorial auctions (c.f. (Nisan 2005)).

We start with a brief overview of two types of utility dependencies that need to be handled:

1. *Mutual dependency:* Utility of the set of goals is different from the summation of the utility of each individual goal. Thus, for $S \subseteq G, u(S) \neq \Sigma_{g \in S} u_g$. Examples: (1) while the utility of having either a left or right shoe alone is *zero*, utility of having both of them is much higher (i.e. the goals "complement" each other); (2) utility of having two cars is smaller than the summation of the individual utilities of having each of them (i.e. the goals "substitute" each other).
2. *Conditional dependency:* The utility of a goal or set of goals depend on whether or not another goal or set of goals are already achieved. Examples: (1) the utility of having a hotel reserved in Hawaii depends on whether or not we have a ticket to Hawaii; (2) in the logistics domain, packages containing parts of a machine that need to be delivered to a given location are only useful in the presence of other parts in the same group.

In this paper, we develop an approach for representing these utility dependencies between planning goals using the *Generalized Additive Independence (GAI)* model (Bacchus & Grove 1995) and describe a planning algorithm based on forward search that solves this extended PSP problem. The algorithm is based on the forward heuristic search used in the *Sapa*$^{\mathcal{PS}}$ planner (van den Briel *et al.* 2004). To guide the heuristic search algorithm, we introduce two different heuristics based on the planning graph structure. The first one is admissible and can be used to find optimal solutions. The second one is inadmissible but is more effective in finding good quality solutions in less time. The main innovation of our heuristic approach is its ability to take into account both goal utility and goal achievement cost dependencies. Our heuristic framework combines both (1) a greedy search for low-cost relaxed plans that handle cost interactions between goals, and (2) a declarative Integer Linear Programming (ILP) encoding that captures both mutual goal achieve-

ment cost and goal utility dependencies to select the best subset of goals. The solution of this latter ILP encoding is used to guide an anytime best-first search algorithm that returns higher net benefit solutions.

For the rest of this paper, we will first formalize the problem of goal utility dependency. After that we will introduce the GAI model (Bacchus & Grove 1995) and then describe the important steps of the main heuristic search framework. We finish the paper with empirical results showing the effectiveness of our approach and the related and future work.

Problem Formulation

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 $a \in A$ defined by pre- and post-conditions $Precond(a), Effect(a) \subseteq F$. A plan P is a sequence of actions from A such that, when executed from I, P will result in a state that achieves all $g \in G$.

In partial satisfaction planning (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, which represents the resources spent executing each action. 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 all actions $a \in P$:

$$maximize_{G' \subseteq G} \; u(G') - \sum_{a \in P} c_a \qquad (1)$$

Work on PSP has until now assumed that goals have no utility dependencies and thus their utilities are additive: $u(G') = \Sigma_{g \in G'} u_g$. As we showed in the previous section, there are many scenarios in which this assumption is not true. To represent the goal utility dependencies as discussed, we adopt the *Generalized Additive Independence* (GAI) model (Bacchus & Grove 1995). Specifically, we assume that the utility of the goal set G can be represented by k local utility functions $f^u(g[k]) \in R$ over sets $g[k] \subseteq G$ of goals where $g[k]$ may contain a single goal thereby capturing the utility of that goal. For any subset $G' \subseteq G$ the utility of G' is:

$$u(G') = \sum_{g[k] \subseteq G'} f^u(g[k]) \qquad (2)$$

For the rest of this paper, we name the new PSP problem with utility dependencies represented by GAI model ***PSP***$^{\mathcal{UD}}$. If there are $|G|$ local functions $f^k(g[k])$ and each $g[k]$ contains a single goal then there are no utility dependencies and thus ***PSP***$^{\mathcal{UD}}$ reduces to the original PSP problem. We chose the GAI model because it is simple, intuitive, expressive and it is more general than other commonly used models such as UCP-Net (Boutilier et. al. 2001). To facilitate the discussion on the GAI model for ***PSP***$^{\mathcal{UD}}$, we will use the following example in the Mars Rover domain (Smith 2004).

Example: In the Mars Rover domain, a Rover needs to travel between different locations. It then collects the samples and takes either high or low resolution pictures at different locations. Achievement of each goal gives a utility value. We have $g_1^l = sample(l)$, $g_2^l = high\_res(l)$, $g_3^l = low\_res(l)$ with the utility values $u(g_1^l) = 200, u(g_2^l) = 150$ and $u(g_3^l) = 100$ for all locations l of interest. There are also utility dependencies between combinations of goals such as:

- *complement* relation: Utility of having samples at related locations l_1 and l_2 will give additional utility (e.g. $u(\{sample(l_1), sample(l_2)\}) = u(g_1^{l_1}) + u(g_1^{l_2}) + 50$).
- *substitute* relation: Taking both low and high resolution images of the same location will reduce the overall utility (e.g. $u(\{g_2^l, g_3^l\}) = u(g_2^l) + u(g_3^l) - 80$).
- *conditional* relation: Finally, if we already have a picture of a given location l, then the utility of taking a sample at l increases, due to the available information to aid future analysis (e.g. if g_2^l then $u(g_1^l)$ += 100, if g_3^l then $u(g_1^l)$ += 50 and if $g_2^l \wedge g_3^l$ then $u(g_1^l)$ += 110).

For our GAI model, the local functions for these relations in this example would be: $f(g_1^l) = 200$, $f(g_2^l) = 150$, $f(g_3^l) = 100$, $f(\{g_1^{l_1}, g_1^{l_2}\}) = 50$, $f(\{g_2^l, g_3^l\}) = -80$, $f(\{g_1^l, g_2^l\}) = 100$, $f(\{g_1^l, g_3^l\}) = 50$ and $f(\{g_1^l, g_2^l, g_3^l\}) = 110 - (100+50) = -40$. At first glance, it may seem strange to have the local function $f(\{g_1^l, g_2^l, g_3^l\})$ having negative value even though those three goals have a complement relation. That's because the formulation to calculate the utility of $\{g_1^l, g_2^l, g_3^l\}$ includes two other complement functions for $\{g_1^l, g_2^l\}$ and $\{g_1^l, g_3^l\}$. Using GAI functions, the utility of a set of goals can be calculated as: $U(\{g_1^l, g_2^l, g_3^l\}) = f(g_1^l) + f(g_2^l) + f(g_3^l) + f(\{g_1^l, g_3^l\}) + f(\{g_1^l, g_2^l\}) + f(\{g_1^l, g_2^l, g_3^l\}) = 200 + 150 + 100 + 50 + 100 - 40 = 560$

Search Algorithm

There are several approaches to solve the PSP problems such as selecting the subset of goals up-front, compilation to ILP, or adapting the A^* search algorithm to PSP (Smith 2004; van den Briel *et al.* 2004). We choose to extend the best-first search framework in ***Sapa***$^{\mathcal{PS}}$ to handle ***PSP***$^{\mathcal{UD}}$, which uses an $A*$ based search algorithm. With this framework we can analyze the action cost and utility dependencies at each search node.

The search algorithm, which we call A^*_{PSP} is sketched in Figure 1. It starts with the initial state S_{init} and continues to dequeue the most promising node S (i.e. highest $f(S) = g(S) + h(S)$ value). For each search node S, let P_C be the partial plan leading from S_{init} to S, let G_S be the set of goals satisfied in S, $U(S) = \Sigma_{g \in G_S} u_g$ be the utility of S, and let $c(S) = c(P_C) = \Sigma_{a \in P_C} c_a$ be the total cost to visit S. We have $g(S) = u(S) - c(S)$. Let P_R be the plan that, when we apply in S, will lead to S' such that P_R maximizes $h(S) = (U(S') - U(S)) - c(P_R)$. While calculating $g(S)$

```
Open State Queue: SQ={S_init}
Best achieved benefit: B_B = U(S_init)
while SQ≠{}
    S:= Dequeue(SQ)
    if (g(S) > 0) ∧ (h(S) = 0) then
        Terminate Search;
    forall a applicable in S
        S' := Apply(a,S)
        if g(S') > B_B then
            Print BestBeneficialNode(S')
            B_B ← g(S')
        if f(S') = g(S') + h(S') ≤ B_B then
            Discard(S)
        else Enqueue(S',SQ)
end while;
```

Figure 1: A* search with negative edges for PSP problems.

is trivial, having a good estimate of $h(S)$ is hard and is the key to the success of best-first search algorithms.

Definition S_T is a termination node if: $h(S_T) = 0$, $g(S_T) > 0$, and $\forall S : g(S_T) > f(S)$.

If a state S is a termination node, we stop the search. If not, we generate children of S by applying applicable actions a to S. If the newly generated node $S' = Apply(a, S)$ is a beneficial node (i.e. $g(S') > 0$) and has a better $g(S')$ value than the best beneficial node visited so far, then we print the plan leading from S_{init} to S'. Finally, if S' is a promising node (i.e. $f(S') > B_B$ where $f(S')$ is the f value of state S' and B_B is the g value of the best beneficial node found so far), then we will put it in the search queue SQ sorted in the decreasing order of f values. Notice that because we keep outputting the best beneficial plans while conducting search (until a terminal node is found), the algorithm has an "anytime" property. Thus, it can quickly return some beneficial plan. It also can continue to return plans with better net benefit[1].

Proposition *If $h(S)$ is admissible (over-estimates the achievable benefit), then A^*_{PSP} returns an optimal solution.*

Proof sketch: If $f(S)$ over-estimates the real achievable benefit, then the discarded nodes (not enqueued) cannot lead to nodes with higher benefit value than the current best node (B_B). If A^*_{PSP} finishes with an empty queue then the optimal solution should be found because all nodes enqueued are visited. If A^*_{PSP} found a termination node S_T, all nodes remaining in the queue can lead to solutions with lower total benefit than S_T. Thus S_T is an optimal solution.

[1]If we consider heuristic planning search as a graph search problem, then PSP has some interesting properties (1) the edge cost $v = (u(S') - u(S)) - c_a$ of moving from a state S to state $S' = apply(S, a)$ can be negative; (2) any reachable state can be a valid goal state. We have not found a cyclic graph search algorithm dealing with problems having the same properties.

Heuristics for Maximizing Plan Benefit

The main objective of PSP planners, as described in Equation 1, is to find the best plan in terms of total benefit, which is calculated as *benefit = total achieved utility - total action cost*. The key to the success of A^*_{PSP} is an effective heuristic function capable of estimating the remaining achievable benefit $h(S)$ for each generated state S during forward search. We base our heuristic routine on the planning graph cost-propagation framework first used in the $Sapa$ planner (Do & Kambhampati 2001). We how describe how our heuristic estimates cost and utility.

Cost-Propagation on the Relaxed Planning-Graph

For PSP problems, the cost-propagation process on the planning graph is used to estimate the achievement cost for each individual goal. Starting with the achievement cost of $c(f) = 0$ for facts f in the initial state I and $c(f) = c(a) = \infty$ for all other facts and all actions, the propagation rules to estimate costs to achieve different facts p and to execute actions a are[2]:

1. Facts: $\forall f : c(f) = min\ (c(a) + c_a) : f \in Effect(a)$
2. Max-prop: $\forall a \in A, f \in Precond(a) : c(a) = max\ c(f)$
3. Sum-prop: $\forall a \in A, f \in Precond(a) : c(a) = \Sigma c(f)$

The update rules are used while extending the (relaxed) planning graph structure (Blum & Furst 2001) from the initial state, with each $c(f)$ or $c(a)$ value updated exactly once. After the propagation is done, for each individual goal $g \in G$, the value $c(g)$ is an estimate on the cost to achieve 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. Using $c(g)$ calculated by *max* propagation, we can estimate the achievable benefit value as below:

$$h_{max} = max_{G' \subseteq G}\ [u(G') - (max_{g \in G'} c(g))] \quad (3)$$

It's easy to see that h_{max} overestimates the real achievable benefit and thus A^*_{PSP} using h_{max} will output an optimal solution. One brute force way to estimate h_{max} is by enumerating over all ($2^{|G|}$) possible subsets of G, which can be prohibitive for a large goal set. In the following, we will introduce an approach for estimating h_{max} using an Integer Linear Programming (ILP) encoding.

Relaxed-Plan based Heuristic

Because the cost estimate for each goal using the cost-propagation on the planning graph can highly underestimate the real cost of a set of goals, the max family of heuristics as in h_{max} tends to perform badly in practice. Therefore, we use an alternative approach of utilizing the relaxed plan employed by *Sapa*^{PS} for partial satisfaction

[2]c_a, which is the execution cost of a, is different from $c(a)$, which is the cost to enable the execution of a (i.e. cost to achieve preconditions of a)

planning[3]. For each state S explored in a progression planner we build the relaxed planning graph and perform a forward cost propagation on the graph. After this we use the following general procedure to extract a relaxed plan to support a subset of goals $G' \subseteq G$:

1. Let subgoal set $SG = G$ and the relaxed-plan $RP = \emptyset$.
2. For each $g \in SG \setminus I$ select action $a : g \in Effect(a)$. Add a to RP and $p \in Precond(a) \setminus I$ to SG.
3. Repeat until $SG = \emptyset$.

The plan RP is called a "relaxed plan" because we ignore negative effects of all actions in the planning graph while extracting it. This relaxation allows a non-backtracking plan extraction process that finds RP quickly. Ideally, we would like to extract the relaxed-plan with the highest net benefit. 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:

$$h_{relax} = max_{G' \subseteq G} \left(u(G') - \sum_{a \in RP(G')} c_a\right) \qquad (4)$$

Extracting the relaxed plan that has the highest net benefit (i.e. utility minus cost), tends to get complex as the set of goals G' we need to focus on depends both on the cost dependencies as well as their utility dependencies. To approximate h_{relax} in previous work, the cost dependencies were partially handled by biasing the relaxed plan extraction to: (i) greedily select actions with lowest achievement cost ($c(a) + c_a$); (ii) reuse actions selected for other goals. We further extend this approach to approximate the optimal relaxed plan in the presence of utility dependencies as follows:

1. Greedily extract the lowest-cost relaxed plan RP that achieves the *largest* set of achievable goals. The relaxed plan extraction is not sensitive to utility dependencies. We now can search in its neighborhood for a plan that attempts higher net benefit, while taking utility dependencies into account.
2. Capture the achievement cost dependencies between achievable goals using the causal structure of RP.
3. Pose the problem of extracting the optimal relaxed plan within RP that takes both cost and benefit dependencies into account as an ILP encoding. This encoding is then solved by an ILP solver, and the solution is used as the heuristic $h(S)$ to guide A^*_{PSP}.

Alert readers would have noted that if we compile the entire relaxed planning graph (rather than just the greedily extracted relaxed plan) we can post the entire relaxed plan extraction process (including step 1) as an ILP problem. Despite its conceptual elegance, we chose not to follow this route as the cost of heuristic computation increases quite significantly (especially for our progression planner which extracts relaxed plans at each node). Related to this is whether we can encode the full general problem as an ILP. While this is possible, previous work has shown that this approach does not scale well in even simpler problems (van den Briel *et al.* 2004).

[3]Variants of this approach are also used in several other PSP planners such as *AltAlt*^{ps} (van den Briel *et al.* 2004) and the orienteering planner (Smith 2004).

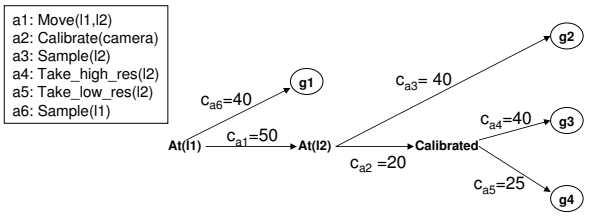

Figure 2: Relaxed plan for Mars Rover domain.

As another observation, note the possibility of performing steps 2 and 3 in a greedy procedural form. We found that such a procedure becomes quite hairy because of the complex utility dependencies. We also note that because the relaxed planning graph may not contain the optimal relaxed plan, h_{relax} is not guaranteed to be admissible (and can thus underestimate the achievable net benefit).

Cost Dependencies In our ongoing example, plans consist of actions for traveling between different locations, calibrating instruments and carrying out experiments; all have costs (e.g. in proportion to the amount of energy and time consumed by different actions). Because certain actions contribute to the achievement of multiple goals, there are also mutual dependencies between the costs of achieving sets of goals. Those relations can be discovered by using the causal structure of the extracted relaxed plan.

Figure 2 shows the relaxed plan for the planning instance in which the Rover is initially located at l_1 and the desired goals are $\{g_2, g_3, g_4\}$ with $g_2 = sample(l_2)$, $g_3 = high\_res(l_2)$, and $g_4 = low\_res(l_2)$. The relaxed plan contains 5 actions: $a_1 = move(l_1, l_2)$ with cost $c_{a_1} = 50$, $a_2 = calibrate(camera)$ with $c_{a_2} = 20$, $a_3 = take\_sample(l_2)$ that achieve goal g_2 with cost $c_{a_3} = 40$, and two actions a_4, a_5 to take the pictures at l_2 of different quality with cost $c_{a_4} = 40$ and $c_{a_5} = 25$ respectively. As discussed above, partial plans achieving different individual goals can overlap. For example, the partial plans to achieve individual goals g_2, g_3 and g_4 all share action a_1 and plans for g_3 and g_4 share the action a_2. Thus, the cost to achieve the set S_g of goals follows the *substitute* dependencies in which the cost to achieve S_g can be smaller than the summation of the individual costs to achieve each $g \in S_g$. For example, the cost to achieve goal g_3 is $c(g_3) = c_{a_1} + c_{a_2} + c_{a_4} = 110$ and $c(g_4) = c_{a_1} + c_{a_2} + c_{a_5} = 95$ while $c(\{g_3, g_4\}) = c_{a_1} + c_{a_2} + c_{a_4} + c_{a_5} = 135 < c(g_3) + c(g_4) = 205$.

To capture the mutual dependencies between the goal achievement costs, we find the set of actions shared between different partial plans achieving different goals. This procedure utilizes the causal links, each of which specifies the achievement action for a goal or action precondition, gathered while extracting the relaxed plan.

1. Initialize: $\forall a \in P : GS(a) = \emptyset$; $\forall p \in Effect(a) \bigcup Prec(a) : GS(p) = \emptyset$; $\forall g \in G : GS(g) = \{g\}$.
2. Backward sweep from goals and update: $GS(a) = \bigcup GS(p) : p \in Effect(a)$ and $GS(p) = \bigcup GS(a) : p \in Precond(a)$

Using the update procedure above, for each action a, $GS(a)$ contains the set of goals g to which a contributes. For example, $GS(a_1) = \{g_2, g_3, g_4\}$, $GS(a_2) = \{g_3, g_4\}$ and $GS(a_3) = \{g_2\}$.

Estimating Achievable Benefit A realistic planning problem with goal utility dependencies will likely include several goals involved in multiple dependencies, considerably increasing the complexity of the problem. The challenge is to use the relaxed plan as means of capturing possible goal combinations while also being informed by the cost of the relaxed actions involved in achieving these goals.

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}:

- Binary Variables:
  - $\forall a \in P$: create one binary integer variable X_a.
  - $\forall g \in G$: create one binary integer variable X_g.
  - $\forall G' \subseteq G,\ f^u(G') \neq 0$: create one binary integer variable $X_{G'}$.
- Constraints:
  - $\forall a \in P, \forall g \in GS(a) : (1 - X_g) + X_a \geq 1$
  - $\sum_{g \in G'} (1 - X_g) + X_{G'} \geq 1$
  - $\forall g \in G' :\ (1 - X_{G'}) + X_g \geq 1$
- Objective function: $max\ (\Sigma f^u(G') * X_{G'} - \Sigma X_a * c_a)$

The purpose of this encoding is to capture the set of goals $G' \subseteq G$ that gives the maximum tradeoff between the utility of G' and the cost of actions in the relaxed plan supporting G'.

The first constraint enforces that if a given goal is selected for achievements, then any action that contributes to the achievement of that goal should be selected too. The second and third types of constraints ensure that if there is a GAI local function for a set of goals $G' \subseteq G$, then this local function (represented by a binary variable $X_{G'}$) will be activated ($X_{G'} = 1$) if and only if all goals $g \in G'$ are selected ($X_g = 1$). The value we get from solving this ILP encoding can then be used as an estimate of the achievable benefit for a given state ($h(S)$ value) for the A^*_{PSP} search algorithm outlined in Figure 1.

For h_{max} we can also setup an ILP encoding which is simpler than the encoding for h_{relax} because there is no need for variables and constraints related to actions and goal supporting $GS(a)$ sets.

- Variables: besides X_g and $X_{G'}$, create one variable X_{C_G} representing the cost to achieve G.
- Constraints: besides the second and third types of constraints as in the encoding for h_{relax} above, introduce one constraint: $\forall g \in G :\ X_{C_G} - X_g * c(g) \geq 0$ where $c(g)$ is calculated during cost propagation on the graph (as in Equation 3).
- Objective Function: $max\ (\Sigma f^u(G') * X_{G'} - X_{C_G})$

The variable X_{C_G} and the constraint with X_g guarantee that the cost to achieve a set of goals G' is the maximum of the cost to achieve any goal $g \in G'$.

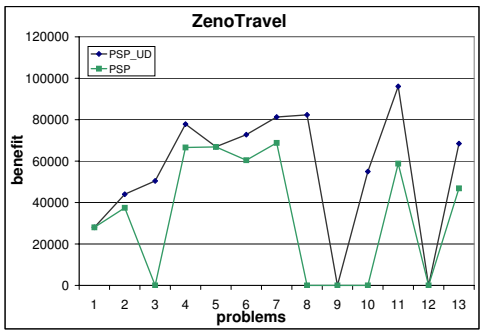

Figure 3: *SPUDS* and *Sapa*$^{\mathcal{PS}}$ in ZenoTravel domain.

Empirical Results

We have implemented the heuristic search algorithm for the *PSP*$^{\mathcal{UD}}$ problems discussed in this paper on top of the *Sapa*$^{\mathcal{PS}}$ planner. We call the new planner *SPUDS*. We tested our planner on two sets of random *ZenoTravel* and *Satellite* problems, which were generated on top of the problem sets used in the Third International Planning Competition (Long & Fox 2003). The ZenoTravel domain involves moving people by airplanes between different cities and the Satellite domain involves turning satellites to point at different objects and taking pictures of them. A more detailed description of these domains can be found at the IPC3 website[4].

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 a termination node is found), the results reported in this section represent the plan with highest benefit value found within the time limit. For solving the ILP encoding, we use the C version of `lp_solve version 5.5`, a free solver with a Java wrapper.

Generating Test Problems: Given that in general, action cost is decided by the amount of resources consumed and/or the time spent by that action, we decided to automatically generate the *PSP*$^{\mathcal{UD}}$ problems from a set of metric temporal planning problems from the ZenoTravel and Satellite domains (used in IPC3) as follows:

- *Domain File:* We modified the domain files by adding a cost field to each action description. Action cost is represented as a mathematical formula involving numeric variables that exist in the original problem description and

[4] http://planning.cis.strath.ac.uk/competition/

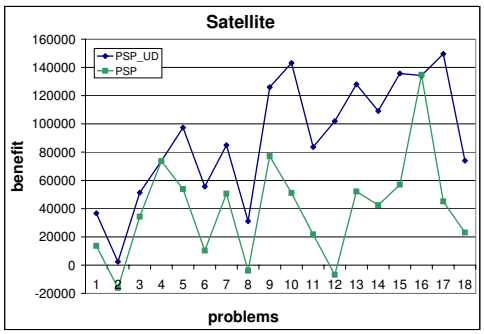

Figure 4: *SPUDS* and *Sapa*$^{\mathcal{PS}}$ in Satellite domain.

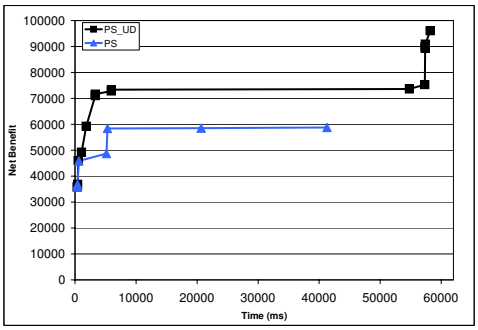

Figure 5: Example run of planners. (Problem 11 in Zeno-Travel)

also new numeric variables that we have added to the domain description file. The cost field utilizes both the functions representing numeric quantities in the original problem descriptions, and the newly introduced cost-related functions used to convert the temporal and resource consumption aspects of each action into a uniform plan benefit represented by the amount of money spent (as in the ZenoTravel domain) or energy consumed (as in Satellite domain).

- *Problem File:* For each domain, we implemented a Java program that parses the problem files used in IPC3 and generates the *PSP*$^{\mathcal{UD}}$ version with cost-related function values randomly generated within appropriate upper and lower bounds. The goal utilities are also randomly generated within different upper and lower bounds. The goals are randomly selected to be "hard" or "soft". The set of goal dependencies along with their utility values are also randomly generated. Thus, the number of dependencies, size of the dependencies, set of goals involved and the utility values are all randomly selected within certain lower and upper bounds (e.g. upper bound on the number of dependencies is $3 * |G|$).

Analysis: We ran both *Sapa*$^{\mathcal{PS}}$ and *SPUDS* on problems from the two domains. While the latter is sensitive to both cost and utility dependencies, the former (*Sapa*$^{\mathcal{PS}}$) only accounts for cost dependencies. Due to poor performance of h_{max} in comparison to h_{relax} in tests[5], we focus only on h_{relax} in evaluating *SPUDS*. The empirical evaluation is designed to test whether *SPUDS* is able to solve the *PSP*$^{\mathcal{UD}}$ problems more effectively (i.e. with higher net benefit). Figures 3 and 4 show the comparison between those two planners.

In the ZenoTravel domain, *SPUDS* is better than *Sapa*$^{\mathcal{PS}}$ in 10/13 problems. Both planners find the same solutions in the remaining three problems.

In the Satellite domain, *SPUDS* is better in 16/18 problems, and most of the time is significantly better (up to 16 times better in overall best plan quality). They are equal in 1 problem and *Sapa*$^{\mathcal{PS}}$ is slightly (0.3%) better in 1 problem.

We found that the ILP encoding increases the time spent per search node by 3 to 200 times with the highest increase occurring on the larger problems. The advantage gained by *SPUDS* in heuristic guidance offsets this additional computation cost. This behavior is shown in Figure 5 through a plot of a run on problem 11 of ZenoTravel. For this case, *SPUDS* finds a better plan than *Sapa*$^{\mathcal{PS}}$ at 533 milliseconds and this trend continues; *SPUDS* retains its net benefit superiority.

Of the problems solved by *SPUDS*, the average number of actions in the plans was 14.5 for ZenoTravel and 126.2 in Satellite. For Satellite, this number excludes a single outlier problem that included 1850 actions. For *Sapa*$^{\mathcal{PS}}$, the average number of actions included in ZenoTravel plans was 13.75 and in Satellite plans was 121.1. These averages exclude the same outlier problem in Satellite. Our results underscore the fact that *SPUDS* finds higher net benefit plans even though shorter plans are found using *Sapa*$^{\mathcal{PS}}$. In fact, most of the plans found by *Sapa*$^{\mathcal{PS}}$ are shorter because they achieve fewer goals and therefore lose out on the utility that those goals give.

In conclusion, *SPUDS* is significantly better than *Sapa*$^{\mathcal{PS}}$ in both the ZenoTravel and Satellite domains. This shows that our declarative heuristic technique of using an ILP encoding to extract the best part of the relaxed plan regarding the utility dependencies pays off despite higher overhead. We are in the process of more creating test problems based on other benchmark problem sets. We also hope to enable other PSP planners such as *AltAlt*^{ps} and *OptiPlan* (van den Briel *et al.* 2004) to handle *PSP*$^{\mathcal{UD}}$ problems and compare these with *SPUDS*.

Related Work

There has been work on PSP problems using *orienteering* to select goal subsets by David Smith (2004). Also, van den Briel et. al. (2004) introduced several planners such as *AltAlt*^{ps}, *Sapa*$^{\mathcal{PS}}$, and $OptiPlan$ that tackle PSP by either greedily selecting goals up-front, heuristically searching for solutions, or compiling the problem into ILP. None of those planners deal with utility dependencies as described in this paper.

[5]In our empirical tests using h_{relax} allowed *SPUDS* to find plans that were on average 27% better than those found when using h_{max} (when using h_{max} found a plan at all).

PrefPlan (Brafman & Chernyavsky 2005) can find optimal plans with preferences between goals specified in CP-Net. Both PSP planners and PrefPlan can handle soft-goals; however, PSP planners explicitly reason about quantitative goal utility and action costs, while PrefPlan handles qualitative goal preferences.

Besides the GAI model that we used to represent the utility dependencies, there are several other attractive models such as UCP-Net (Boutilier et. al. 2001) and the graphical model (Bacchus & Grove 1995). While both provide a graphical representation that can make it easier for users to understand the dependencies, the GAI has advantages in that it is more general than UCP-Net while still being simple and intuitive.

In combinatorial auctions, the utility for a set of items up for bid are normally non-additive and share many similarities with reasoning about sets of goals that are dependent in PSP. While a bidding process is different from planning, the bidding language (Nisan 2005) can be used to represent utility dependencies in $PSP^{\mathcal{UD}}$.

Conclusion & Future Work

In this paper, we discussed a framework of solving partial satisfaction planning (PSP) problems with utility dependencies. We show how to represent various types of dependencies using the GAI framework. We also introduced an admissible heuristic, h_{max}, and an inadmissible heuristic, h_{relax}, that when used with the A^*_{PSP} search algorithm, will find optimal or inoptimal solutions respectively. We empirically demonstrated the effectiveness of the new heuristic framework on two benchmark planning domains.

We plan to extend this work to combine both quantitative preferences as in PSP with qualitative preference model as handled in PrefPlan. To improve the performance, we plan on investigating more effective admissible heuristics and more aggressively take into account negative information, such as residual cost as described in AltWlt (Sanchez & Kambhampati 2005) to improve the heuristic quality. We are also looking at the possibility of converting the utility dependencies into dummy goals to simplify the problems.

References

Bacchus, F. and Grove, A. 1995. Graphical Models for Preference and Utility In *Proc. of UAI-95*.

Blum, A., and Furst, 1997. Fast Planning through Planning Graph Analysis. In *Artificial Intelligence*, 90:281–300.

Boutilier, C., Bacchus, F., and Brafman, R. 2001. UCP-Networks: A Directed Graphical Representation of Conditional Utilities. In *Proc. of UAI-2001*.

Boutilier, C., Brafman, R., Hoos, H., and Poole, D. 1999. Reasoning with conditional ceteris paribus preference statements. In *Proc. of UAI-2001*.

Brafman, R., and Chernyavsky, Y. 2005. Planning with Goal Preferences and Constraints. In *Proc. of ICAPS-2005*.

Brafman, R. 2004. Eliciting, Modeling, and Reasoning about Preference using CP-nets *Tutorial at UAI-2004*

Dasgupta, P., Sen, A., Nandy, S., and Bhattacharya, B. 2001. Searching Networks with Unrestricted Edge Costs. *IEEE Transactions on Systems, Man, and Cybernetics*

Dechter, R. 2003. Constraint Processing. *Morgan Kaufmann Publishers.*

Do, M. and Kambhampati, S. 2001. Sapa: A Domain-Independent Heuristic Metric Temporal Planner. In *Proc. of ECP-01*.

Long, D. and Fox, M. 2003. The 3rd International Planning Competition: Results and Analysis. *Journal of AI Research.* 20:1-59.

Nisan, N. 2000. Bidding and Allocation in Combinatorial Auctions In *Proc. of ACM conf. on Electronic Commerce*.

Sanchez, R. and Kambhampati, S. 2005. Planning Graph Heuristics for Selecting Objectives in Over-subscription Planning Problems. In *Proc. of ICAPS-05*..

Smith, D. 2004. Choosing Objectives in Over-Subscription Planning. In *Proc. of ICAPS-04*.

van den Briel, M., Sanchez, R., Do, M, and Kambhampati, S. 2004. Effective Approaches for Partial Satisfaction (Over-Subscription) Planning. In *Proc. of AAAI-04*

Williamson, M., and Hanks, S. 1994. Optimal Planning with a Goal-Directed Utility Model. In *Proc. of AIPS-94*.

Cost-Optimal Symbolic Pattern Database Planning with State Trajectory and Preference Constraints

Stefan Edelkamp*
Otto-Hahn Straße 14
University of Dortmund
eMail: stefan.edelkamp@cs.uni-dortmund.de

Abstract

This paper considers strategies for planning problems with state trajectory and plan preference constraints – two recently added PDDL features that have been introduced in the context of the 5^{th} international planning competition.

A symbolic breadth-first search exploration algorithm is devised that is guaranteed to find the cost-optimal plan wrt. the preference conditions imposed. For untimed trajectory constraints we propose to run a synchronized property automata concurrent to the unconstrained state space, while timed trajectory constraints are checked during plan construction. Plan preferences are evaluated in the cost function after a successful plan has been generated. For deriving good search heuristics, we turn to the construction of symbolic pattern databases and the pattern selection problem.

Introduction

State trajectory and plan preference constraints are the two language features introduced in PDDL3 (Gerevini & Long 2005) for describing benchmarks of the 5^{th} international planning competition.

State trajectory constraints provide an important step of the agreed fragment of PDDL towards the description of *temporal control knowledge* (Bacchus & Kabanza 2000; Kabanza & Thiebaux 2005) and *temporally extended goals* (DeGiacomo & Vardi 1999; Lago, Pistore, & Traverso 2002). They assert conditions that must be met during the execution of a plan and are often expressed using using quantification over domain objects. Through the decomposition of metric and temporal plans into *happenings*, state trajectory constraints feature also higher levels of the PDDL hierarchy (Fox & Long 2003). For sequential plans $\pi = (S_0, \ldots, S_n)$ with unit time actions, trajectory constraints are interpreted as shown in Fig. 1. Note that the definitions of some constraints have been adapted, identifying time with the position in the state sequence.

$\pi \models \phi \equiv \pi \models (at\ end\ \phi)$	$\Leftrightarrow$	$S_n \models \phi$
$\pi \models (always\ \phi)$	$\Leftrightarrow$	$\forall 0 \leq i \leq n : S_i \models \phi$
$\pi \models (sometimes\ \phi)$	$\Leftrightarrow$	$\exists 0 \leq i \leq n : S_i \models \phi$
$\pi \models (within\ t\ \phi)$	$\Leftrightarrow$	$\exists 0 \leq i \leq t : S_i \models \phi$
$\pi \models (at\text{-}most\text{-}once\ \phi)$	$\Leftrightarrow$	$\forall 0 \leq i \leq n : S_i \models \phi$
$\Rightarrow \exists i \leq j\ \forall i \leq l \leq j : S_l \models \phi$	$\wedge$	$\forall j < l \leq n : S_l \not\models \phi$
$\pi \models (sometimes\text{-}after\ \phi\ \psi)$	$\Leftrightarrow$	$\forall 0 \leq i \leq n : S_i \models \phi$
$\Rightarrow \exists i < j \leq n : S_j \models \psi$		
$\pi \models (sometimes\text{-}before\ \phi\psi)$	$\Leftrightarrow$	$\forall 0 \leq i \leq n : S_i \models \phi$
$\Rightarrow \exists 0 \leq j < i : S_j \models \psi$		
$\pi \models (always\text{-}within\ t\ \phi)$	$\Leftrightarrow$	$\forall 0 \leq i \leq n : S_i \models \phi$
$\pi \models (hold\text{-}during\ \ t\ t'\ \phi)$	$\Leftrightarrow$	$\forall t \leq i < t' : S_i \models \phi$
$\pi \models (hold\text{-}after\ t\ \phi)$	$\Leftrightarrow$	$\forall t < i \leq n : S_i \models \phi$

Table 1: Semantics of State Trajectory Constraints.

Annotating goal conditions or state trajectory constraints with preferences models soft constraints. For example, if we prefer block a to reside on the table after the plan's execution, we add *(preference p (on-table a))* to the planning goal and maintain a proposition *(is-violated-p)* that denotes the inverse of preference satisfaction. Such propositions are interpreted as natural numbers that can be included into the plan's *cost function*. This function allows planners to search for cost-optimal plans. For planning with preferences the cost function first scales and then accumulates the numerical interpretation of the propositions referring to the violation of the preference constraints.

Symbolic exploration based on BDDs (Bryant 1985) acts on sets of states rather than on singular ones and exploit redundancies in the joint state representation. In this work we make optimal BDD solver technology applicable to planning with PDDL3 domains. We compile state trajectory expressions back to PDDL2 (Fox & Long 2003). The grounded representation is annotated with propositions that maintain the truth of preferences and operators that model that the synchronized execution or an associated property automaton (Edelkamp 2006b). We introduce to *Cost-Optimal Symbolic BFS* (Edelkamp 2006a) and adapt it to the search with preference constraints. We compare the

*The author is supported by the German Research Foundation (DFG) project *Heuristic Search* Ed 74/3.

approach with *Cost-Optimal Symbolic Breadth-First Branch-And-Bound* and show the latter's limitation with respect to currently considered cost functions. Last but not least, we consider the acceleration of the search process with symbolic pattern databases.

Symbolic Breadth-First Search

Symbolic search is based on satisfiability checking. The idea is to make use of Boolean functions to avoid (or at least lessen) the costs associated with the exponential memory blow-up for the state set involved as problem sizes get bigger. For propositional action planning problems we can encode the atoms that are valid in a given planning state individually by using the binary representation of their ordinal numbers, or via the bit vector of atoms being true and false.

There are many different possibilities to come up with an encoding of states for a problem. The more obvious ones seem to waste a lot of space, which often leads to bad performance of BDD algorithms. We implemented the approach of (Helmert 2004) to infer a minimized encoding of a propositional planning domain.

Given a fixed-length binary encoding for the state vector of a search problem, characteristic functions represent state sets. The function evaluates to true for the binary representation of a given state vector, if and only if, the state is a member of that set. As the mapping is 1-to-1, the characteristic function can be identified with the state set itself. Transitions are formalized as relations, i.e., as sets of tuples of predecessor and successor states, or, alternatively, as the characteristic function of such sets. The *transition relation* has twice as many variables as the encoding of the state. If x is the binary encoding of a state and x' is the binary encoding of a successor state, then $T(x, x')$ evaluates to true. We observe that T is the disjunct of all individual state transitions T_O, with O being an operator in $\mathcal{O}$. What we are really interested in, is to compute the (partitioned) *image* $\bigvee_{O\in\mathcal{O}} \exists x\ (T_O(x, x') \wedge S(x))$ of a state set represented by S wrt. a transition relation T.

For symbolic breadth-first search, let S_i be the boolean representation of a set of states reachable from the initial state $\mathcal{I}$ in i steps, initialized with $S_0 = \mathcal{I}$, and $S_{i+1}(x') = \bigvee_{O\in\mathcal{O}} \exists x\ (T_O(x, x') \wedge S_i(x))$. Note that S on the right hand side of the equation depends on x compared to x' on the left hand side. Thus, it is necessary to substitute x' with x in S_i, written as $S_i[x \leftrightarrow x']$. To terminate the exploration, we check $S_i \Rightarrow \mathcal{G}$, or equivalently, $S_i \wedge \mathcal{G} \neq \bot$ (*false* function).

In order to retrieve the plan path we assume that all sets $S_0, \ldots, S_i$ are available. We start with a state S that is in the intersection of S_i and the goal $\mathcal{G}$. This state is the last one on the sequential optimal plan path. We take its characteristic function S into the relational product with T to compute its potential predecessors. Next we compute the second last state on the optimal plan path in the intersection of *Pred* and S_{i-1}, and iterate until the entire plan has been constructed.

We employ BDDs for symbolic exploration. A BDD is a data structure for a concise representation of Boolean functions in form of a DAG with a single root node and two sinks, labeled "1" and "0", respectively. For evaluating the represented function for a given input, a path is traced from the root node to one of the sinks. By detecting unnecessary variable tests and isomorphisms in subgraphs, one obtains a unique representation that is polynomial in the length of the bit strings for many interesting functions. Given the uniqueness of BDD representations we have that the satisfiability test is available in $O(1)$ time. This is a clear benefit to a general satisfiability testing, which – by the virtue of Cook's Theorem – is NP hard.

The variable ordering has a huge influence on the size of a reduced and ordered BDD. In an interleaved representation, that we employ for the transition relation, we alternate between x and x' variables. In usual implementations of BDD libraries, different BDDs share their structures. Such libraries have efficient operations of combining BDDs and feature the computation of an image. Moreover, BDD packages often support arithmetics (on finite domains) with BDDs.

Cost-Optimal Symbolic BFS

Action planning refers to a world description in logic. A number of propositions describe what can be true or false in each state of the world. By applying operations in a world, we arrive at another world where different atoms might be true or false. Usually, only some few atoms are affected by an operator, and most of them remain the same.

We build the binary representation for the objective function as follows. For ordinary preferences of type (*preference* p ϕ_p) we associate a Boolean variable v_p (denoting the violation of p) and construct the following indicator function: $X_p(v) = (v_p \Leftrightarrow \phi_p)$.

For the computation of a BDD for a linear objective function $f(x) = \sum_{i=1}^{n} a_i v_i$ we first compute the minimal and maximal value that f can take. This defines the range that has to be encoded in binary. For the ease of presentation we assume $v_i \in \{0, 1\}$.

The work of (Bartzis & Bultan 2006) shows that the BDD for representing f has at most $O(n \sum_{i=1}^{n} a_i)$ nodes and can be constructed with matching time performance. Even while taking the most basic diagram representation, this result improves on alternative, more expressive structures like ADDs. Moreover, the result generalizes to variables $v_i \in \{0, \ldots, 2^b\}$ and the conjunction/disjunction of several linear arithmetic formulas. This implies that metric planning for bounded linear arithmetic expressions in the preconditions and effects is actually efficient for BDDs. The BDD construction algorithm in our planner differs from the specialized construction above, but computes the same result. As most BDD packages support variables of finite domain, FDD variables for short, we can abstract from their binary encoding. We first combine $mult(a_i, v_i, t_i)$, encoding the ternary relation $a_i \cdot v_i = t_i$ and compute

Procedure Cost-Optimal-Symbolic-BFS
Input: PDDL3 Instance with Transition Relation T,
Linear Objective F (to be minimized on $[0, \max_f]$),
List of Symbolic Goal Preferences X_p,
Goal BDD $\mathcal{G}$, and Initial BDD $\mathcal{I}$
Output: Cost-Optimal Plan (stored)

$U \leftarrow \max_f$
loop
$Reach(x') \leftarrow \mathcal{I}(x')$; $Open(x') \leftarrow \mathcal{I}(x)$
$Intersection(x) \leftarrow \mathcal{I}(x) \wedge \mathcal{G}(x)$
$Bound(v) \leftarrow F(v) \wedge \bigvee_{i=0}^{U}[v = i]$
$Eval(v, x) \leftarrow Intersection(x) \wedge \bigwedge_p X_p(v, x)$
$Metric(x) \leftarrow \exists v : Eval(v, x) \wedge Bound(v)$
while $(Metric(x) \neq \bot)$
if $(Open = \bot)$ **return** "Exploration completed"
$Succ(x') = \bigvee_{O \in \mathcal{O}} \exists x\ T_O(x, x') \wedge Open(x)$
$Open(x) \leftarrow (Succ(x') \wedge \neg Reach(x'))[x' \leftrightarrow x]$
$Reach(x') \leftarrow Reach(x') \vee Succ(x')$
$Intersection(x) \leftarrow Open(x) \wedge \mathcal{G}(x)$
$Eval(v, x) \leftarrow Intersection(x) \wedge \bigwedge_p X_p(v, x)$
$Metric(x) \leftarrow \exists v : Eval(v, x) \wedge Bound(v)$
$U \leftarrow ConstructAndStorePlan(Metric(x))$

Figure 1: Cost-Optimal BFS Planning Algorithm.

$X(v) = s_n$ with $s_0 = 0$ and $\exists s_0, \ldots, s_{n-1}, \exists t_1, \ldots, t_n$: $add(s_i, t_{i+1}, s_{i+1})$, with $add(s_i, t_{i+1}, s_{i+1})$ encoding the ternary relation $s_i + t_{i+1} = s_i + 1$.

Figure 1 displays the pseudo-code for a symbolic BFS-exploration incrementally improving an upper bound U on the plan length. For the sake of brevity in the pseudo-code we assume that we have a minimization problem in the range $[0, \max_f]$. The implemented system, however, is capable of maximization and general intervals. It computes a lower and upper bound of f and improves them, until the search space is exhausted or the two values match.

The state sets that are used are represented in form of BDDs. With each BDD, the underlying variable is provided. The search frontier denoting the current BFS layer is tested for an intersection with the goal, and this intersection is further reduced according to the already established bound.

There might be many different goal states contained in the *Intersection*. The plan construction process in *ConstructAndStorePlan* first determines the state in the intersection that has the minimal cost. This is achieved by iterative intersection of $[v = i]$, for $i \in \{0, \ldots, U\}$. The obtained value minus 1 is the return value for the next threshold.

Theorem 1 *The latest plan stored by the symbolic search planner* Cost-Optimal-Symbolic-BFS *has minimal cost.*

PROOF: The algorithm applies full duplicate detection and traverses the entire planning state space. It generates each possible planning state exactly once. Only clearly inferior states are pruned in the intersection when evaluated in *Eval* and taken into a conjunct with *Bound*. Therefore, *Metric* is empty only if there is no state in the intersection of the search frontier *Open* with the goal $\mathcal{G}$ that has an improved bound. ■

BDDs already save space for large state sets. But still, memory is the main concern in practice. One implemented idea for further memory limitation is *Frontier-Search* (Korf *et al.* 2005), which has been proposed for undirected or directed acyclic graph structures. In more general planning problems we have established that a small duplicate detection scope (a.k.a. *locality*) is sufficient to guarantee termination for *Cost-Optimal-Symbolic-BFS*.

For purely propositional domains we can apply bidirectional symbolic BFS to obtain step-optimal plans (Edelkamp & Helmert 2001), which is often much faster as unidirectional search. For cost-optimal BFS search in PDDL3, however, this is no longer available as computing the goal preferences and selecting the states on the optimal path is involved. To overcome this limitation. instead of evaluating the intersection, it is also possible to encode goal preferences directly into the operators. While we have done this for single-state exploration approach. for symbolic exploration, however, we stick to evaluate goal constraints for the intersection. As an advantage of this, we do not need to reason about goal preferences for non-goals. As a disadvantage we loose the option for invoking bidirectional and heuristic search.

State Trajectory Constraints

State trajectory constraints (Gerevini & Long 2005) can be interpreted in Linear Temporal Logic (LTL) (Clarke, Grumberg, & Peled 1999) and translated into automata that run concurrent to the search and accept when the constraint is satisfied (Gastin & Oddoux 2001). LTL includes temporal modalities like **A** for *always*, **F** for *eventually*, and **U** for *until*. We propose to compile the automata back to PDDL with each transition introducing a new operator (Edelkamp 2006b). Predicate *at* denotes an automaton state for each automaton that has been created. For detecting accepting states we include a flag *accepting*. The initial state of the planning problem includes the start state of the automaton and an additional proposition if it is accepting. For all automata, the goal includes their acceptance.

Allowed transitions in form of planning actions take the current automaton state and the transition label as a precondition and modify them accordingly. For running the constraint automaton and the original planning space in parallel, we toggle *synchronization flags* to ensure perfect alternation. More formally, for

(***hold-after*** $t\ \phi$) we impose that ϕ is satisfied for the search frontier in all steps $i > t$ by applying *frontier shaping* $Open_i \leftarrow Open_i \wedge \phi$

(***hold-during*** $t_1\ t_2\ \phi$) and for all $t_1 \leq i < t_2$ we additionally impose $Open_i \leftarrow Open_i \wedge \phi$.

(***sometimes*** ϕ) we apply automata-based model checking. Let $\mathcal{P}$ be the original planning space and $A_{\mathbf{F}\phi}$ be the constructed automaton for formula $A_{\mathbf{F}\phi}$ and $\otimes$ the cross product between two automata, then $\mathcal{P} \leftarrow \mathcal{P} \otimes A_{\mathbf{F}\phi}$ and $\mathcal{G} \leftarrow \mathcal{G} \cup \{accepting(A_\phi)\}$. The initial state is extended by the initial state of the automaton, which, in this case, is not accepting.

(***sometimes-before*** ϕ ψ) we compile

$$\mathcal{P} \leftarrow \mathcal{P} \otimes A_{(\neg\phi\wedge\neg\psi)\mathbf{U}((\neg\phi\wedge\psi)\vee(\mathbf{A}(\neg\phi\wedge\neg\psi)))}$$

and adapt the goal and initial states accordingly.

(***always*** ϕ) we construct $\mathcal{P} \leftarrow \mathcal{P} \otimes A_{\mathbf{G}\phi}$. Alternatively, for all i we could impose $Open_i \leftarrow Open_i \wedge \phi$ in analogy to *hold-during* and *hold-after*.

(***at-most-once*** ϕ) we assign the planning problem $\mathcal{P}$ to $\mathcal{P} \otimes A_{\mathbf{G}(\phi\rightarrow(\phi\mathbf{U}(\mathbf{G}\neg\phi)))}$.

(***within*** t ϕ) we build $\mathcal{P} \leftarrow \mathcal{P} \otimes A_{\mathbf{F}\phi}$. Moreover, we set $Open_t \leftarrow Open_t \wedge \{accepting(A_{\mathbf{F}\phi})\}$.

We do not support *preference preconditions* (Gerevini & Long 2005). Actually, we can parse and process the conditions, but as the domains of the *violation*-variables are increased and potentially unbounded this affects a possible encoding as a BDD.

For space efficiency, we do not store intermediate BDD layers corresponding to state trajectory automata transitions. Only the layers that correspond to the original unconstrained state space are stored and only for these layers duplicates are eliminated.

For preferences on state trajectory constraints that are constructed via automata theory, we apply the following construction. Instead of adding the automaton acceptance to the goal state we combine the acceptance with the violation predicate. If the automaton accepts then the preference is not violated; if it is located in a non-accepting state, then it is violated. A specialized operator *skip* allows to fail the automata completely. If an automaton is failed once, it remains invalid for the rest of the computation.

Cost-Optimal Symbolic Breadth-First Branch-and-Bound

Having a monotonically increasing cost-function usually allows for better pruning in a branch-and-bound algorithm. Instead of the intersection with the goal we can now evaluate the search frontier, and prune away the states that have a worse evaluation. As one representative for the symbolic search according to this principle we have depicted the pseudo-code for *Cost-Optimal Symbolic Breadth-First Branch-and-Bound* in Figure 2. Different to *depth-first branch-and-bound* (Korf 1993), the breadth-first search order allows an easier implementation with BDDs. The main difference wrt. Cost-Optimal-Symbolic-BFS is that the search frontier itself is evaluated and used for pruning.

The core search routine relies on an additional bound U on the optimal plan length, which can be provided by the user or non-optimal search algorithms like beam search. Using the bound, states are neglected from search if their f-values are larger than U.

Procedure Cost-Optimal-Symbolic-BFBnB
Input: PDDL3 Instance with Transition Relation T,
Linear Objective F (to be minimized on $[0, \max_f]$),
List of Symbolic Goal Preferences X_p,
Goal BDD $\mathcal{G}$, and Initial BDD $\mathcal{I}$
Output: Cost-Optimal Plan (stored)

$U \leftarrow \max_f$
loop
  $Reach(x') \leftarrow \mathcal{I}(x')$
  $Intersection(x) \leftarrow \mathcal{I}(x) \wedge \mathcal{G}(x)$
  $Bound(v) \leftarrow F(v) \wedge \bigvee_{i=0}^{U}[v = i]$
  $Eval(v, x) \leftarrow \mathcal{I}(x) \wedge \bigwedge_p X_p(v, x)$
  $Open(x) \leftarrow \exists v : Eval(v, x) \wedge Bound(v)$
  while $(Intersection(x) \neq \bot)$
    if $(Open = \bot)$ **return** "Exploration completed"
    $Succ(x') = \bigvee_{O\in\mathcal{O}} \exists x\ T_O(x, x') \wedge Open(x)$
    $Open(x) \leftarrow (Succ(x') \wedge \neg Reach(x'))[x' \leftrightarrow x]$
    $Eval(v, x) \leftarrow Open(x) \wedge \bigwedge_p X_p(v, x)$
    $Open(x) \leftarrow \exists v : Eval(v, x) \wedge Bound(v)$
    $Reach(x') \leftarrow Reach(x') \vee Succ(x')$
    $Intersection(x) \leftarrow Open(x) \wedge \mathcal{G}(x)$
  $U \leftarrow ConstructAndStorePlan(Intersection(x))$

Figure 2: Cost-Optimal Anytime Breadth-First Branch-and-Bound Planning Algorithm.

Theorem 2 *Let $\mathcal{P}$ be a propositional PDDL3 planning problem with cost function $f(v) = \sum_{i=1}^{k} a_i v_i$, where v_i defines the violation of ith preference constraint and $a_i > 0$, for all $i \in \{1, \ldots, k\}$. If all preference constraints are monotone increasing wrt. f (i.e. on every path they are violated but not satisfied again), then the latest plan stored by the symbolic search planner* Cost-Optimal-Symbolic-BnB *has minimal cost.*

PROOF: Symbolic breadth-first branch-and-bound prunes away frontier states, which have a cost evaluation that is larger than an already established one. As the cost is monotonically increasing no plan of lower cost gets pruned. ■

If we do not have an cost function that is monotonically increasing, we cannot easily prune states with an evaluation larger than the current one. Unfortunately, such non-monotonicities are frequent in PDDL3 planning. For a goal preference (*preference* (*at end* ϕ)) it may happen that ϕ is set in one operator and deleted in another and vice versa. For example, take $\phi = a$ and $a \in add(O') \cup del(O') \setminus add(O)$ and O is executed directly before O'. Let u be the state after the application of O, and v be the state after the application of O'. Then the value of $a_i v_i$ at u is a_i, which decreased to $a_i v_i = 0$ at v.

This unfortunate observation is also true for

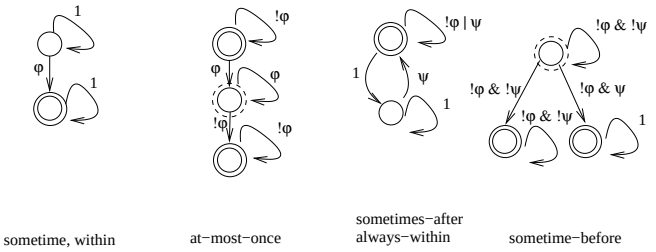

Figure 3: Automata for State Trajectory Constraints.

preferences on state trajectory constraints. For preference like (*sometimes* ϕ), (*sometimes-after* ϕ), (*at-most-once* ϕ), (*within* t ϕ), and (*always-within* t ϕ) we consider a prototypical automaton representation of the condition as illustrated in Fig. 3[1][2]. There is only one automata that allows a transition from a accepting state to an non-accepting state, namely the one for *always-within* and *sometimes-after*. On all other automata there are only transitions from accepting to non-accepting state such that the corresponding *violation*-variable in the objective function would decrease on every path.

If we allow sub-optimal plans there are many good strategies to apply *symbolic beam-search* by omitting non-promising states from consideration. In the current search frontier, we can simply drop states that have a large f-value (compared to the others).

Beam search ranks successors by their f-value, a common strategy in heuristic search planners. Some symbolic algorithms have been proposed for heuristic search planning like BDDA* (Edelkamp & Reffel 1998), SetA* (Jensen, Bryant, & Veloso 2002) and Symbolic BFBnB (Jensen *et al.* 2006). In the following, we consider one option to compute search heuristics for directed symbolic search fully automatically. We will only consider propositional domains, where cost-optimal plans are the ones with minimal number of actions. Nonetheless it is not difficult to extend the results to cost-optimal PDDL3 planning by providing estimates not on the plan length but on its cost.

Symbolic Planning Pattern Databases

Admissible heuristics for planning under-approximate the shortest path distance of the current state to the goal. They are important to guarantee optimality in heuristic search algorithms like A* and IDA*. The *max-atom* heuristic (Haslum & Geffner 2000) is an approximation of the optimal cost for solving a relaxed problem in which the delete lists are ignored. Its extension *max-pair* improves the information without loosing admissibility, approximating the cost of atom pairs. The heuristic h^+ (Hoffmann & Nebel 2001) is another extension to *max-atom* defined as the length of the shortest plan that solves the relaxed problem with ignored delete lists. The heuristic is admissible but solving relaxed plans is computationally hard.

Planning pattern databases as proposed by (Edelkamp 2001; Haslum, Bonet, & Geffner 2005) refer to state space abstraction, where propositions are neglected. This forms a state space to be used for abstraction (Knoblock 1994). An *abstraction* $\mathcal{P}|_R = <\mathcal{S}|_R, \mathcal{O}|_R, \mathcal{I}|_R, \mathcal{G}|_R>$ of a STRIPS planning problem $\mathcal{P} = <\mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G}>$ wrt. a set of atoms R is formally defined by $\mathcal{S}|_R = \{S \cap R \mid S \in \mathcal{S}\}$, $\mathcal{G}|_R = \{G \cap R \mid G \in \mathcal{G}\}$, $\mathcal{O}|_R = \{O|_R \mid O \in \mathcal{O}\}$, where $O|_R$ for $O \in \mathcal{O}$ is given as $(pre(O) \cap R, add(O) \cap R, del(O) \cap R)$. Knoblock's idea has been extended to automatically inferred *fact groups*. The approach refers of mutual exclusive atoms and leads to a multi-variate variable encoding of a state (Helmert 2004). Fact groups are distributed into *patterns*, where each pattern corresponds to an abstraction of the state space. This is achieved by abstract states that represent only a subset of the original atoms - taken from the groups of the chosen pattern. Using this approach each concrete state is mapped to an abstract state. The projection can also be achieved in operators, intersecting the precondition, add and delete list with the pattern. The selection of and the partitioning into patterns is a computationally hard problem, critically influencing the quality of the estimate (Haslum, Bonet, & Geffner 2005).

For multiple pattern databases (Holte *et al.* 2004), in each abstraction i and for each state S we compute estimated costs $h_i(S)$. The estimate $h(S)$ is aimed to be the accumulated cost of the costs in the different abstractions, i.e., $h(S) = \sum_i h_i(S)$. In general we cannot expect that each operator contributes to only one pattern. This implies that the inferred heuristic is no longer admissible. There are two options. First, we may assume that each operator is applicable for only one pattern. Another option considered is to count the abstract problem graph edge weight induced by an operator in only one abstraction. In each BFS level, each zero-cost operator is fired until a fixpoint is reached. If an admissible heuristic is obtained, we call the databases *additive* or *disjoint* (Korf & Felner 2002).

The main limitation for applying pattern databases in search practice is the restricted amount of (main) memory. Many strategies for leveraging the problem have been suggested, e.g. (Zhou & Hansen 2004). Symbolic pattern databases as introduced by (Edelkamp 2002) are pattern databases that have been constructed symbolically for a latter use either in symbolic or explicit heuristic search. Each state set in a shortest path layer is efficiently represented by a corresponding characteristic function. Different to the posterior compres-

[1] As *within* is the constraint version of *sometimes* and *always-within* is the constraint version of *sometimes-after*, we only have shown one automata for both.

[2] The double circles denote accepting state in the corresponding automata, single ones non-accepting states. The dotted circles are non-accepting in the Büchi automata interpretation for infinite words and accepting for the finite state interpretation that we use.

sion of the state set, the construction itself works on compressed representation, allowing much larger databases to be constructed.

Automated Pattern Selection

Finding good patterns is not simple, as there are usually an exponential number of possible choices. Manual pattern selection implies that the planning process is problem-dependent. On the other hand, the problem of pattern selection is involved, especially if multiple pattern databases are requested. So far, automated pattern selection is an unresolved challenge. One reason is that the quality of approximation algorithm for pattern partitioning is hard to predict.

Since the number of vector elements can be considerably large, for explicit search one simplifies the problem of finding a suitable partition to *bin-packing*. The goal is to distribute the possible state variables into the bins in such a way that a minimal number of bins is used. A state variable is greedily added to an already existing bin, until the corresponding abstract state space exceeds main memory. Adding entities to the pattern corresponds to a multiplication of the domain sizes of the selected state variables to the expected abstract state size, so that the bin-packing variant that is needed for automated pattern selection is based on multiplying variable domain sizes. Bin-packing is NP complete such that efficient approximations are used.

Representation and Initialization

For the implementation of pattern optimization we adapt a genetic algorithm. Patterns are represented as binary chromosomes of size $p \times g$, where g is the number of variable domains (fact groups) and p is the number of (disjoint) patterns with $p \leq g$. In the first dimension (row), state variables are indexed, while in the second dimension (column) patterns are selected. A chromosome represents a complete distribution of state variables into multiple PDBs. In Figure 4 we illustrated such a binary chromosome. In Pattern 1, the groups 1,5,6,8 and n are included, whereas in Pattern 2, the groups 3,5 and 7 are present.

We found that the amount of work to find an good partition by performing a randomized assignment of the chromosomes is by far larger as with prior bin packing and has no significant advantage in the overall search. Therefore, we initialized the chromosomes with bin packing. To avoid all chromosomes of the initial population to be identical, we chose a random permutation to the fact groups prior to their automated partitioning into patterns. This leads to comparable good but different distributions of groups into patterns and a feasible initial population for the genetic algorithm.

Recombination and Mutation

The motivation of *recombination* of two parent chromosomes is the hope, that the good properties of the one combines well with the good properties of the other.

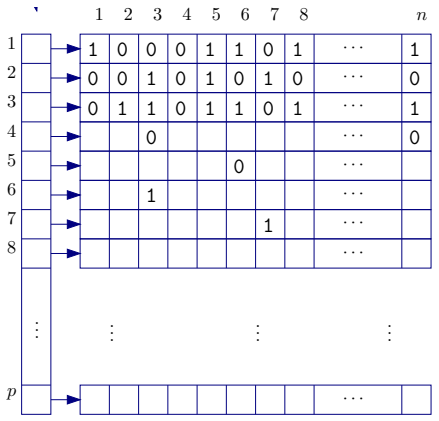

Figure 4: Bitvector Representation of a Chromosome.

The simplest technique is the *n-point-crossover*. For our representation a 2-point crossover in the dimension of the pattern is adequate. This means, that the parent chromosomes exchange parts of their patterns. If the two parents have a different number of patterns, so do the two children.

During *mutation*, chromosome bits are flipped with a small probability. For our case this operator is an important, as it allows to add or delete groups in patterns. We also included a mutation operator which adds and deletes entire patterns. In the bin packing analogy of multiple pattern partition, adding a pattern corresponds to opening a bin, and deleting a pattern corresponds to closing a bin.

Selection

During *selection* an enlarged population (as produced using recombination) is truncated back based on the fitness value(s) to its original size. The normalized fitness evaluation for the population is interpreted as a distribution function, which governs the selection process for the next population. Chromosomes with a better fitness are chosen with higher probability.

The fitness function plays a central role in a genetic algorithm. It defines the evolutionary strength of chromosomes. The construction of an fitness function is often difficult, like in our case, where the conditions for *good* patterns are hardly accessible.

A fundamental question concerns the relationship between the size of the pattern database for a heuristic, and the number of nodes expanded when the heuristic is used to guide the search. (Korf 1997) gives some more insights in the performance of pattern databases. As one result of his research, we can characterize the effectiveness of an admissible heuristic function h by its expected value $\overline{h}$ over the problem space. If the heuristic value of every state were equal to its expected value $\overline{h}$, then a search to depth d would be equivalent to searching to depth $d - \overline{h}$ without a heuristic, since the f-value for every state would be its depth plus $\overline{h}$.

This means that in most search spaces, a linear gain in $\overline{h}$ corresponds to an exponential gain in search efforts.

For the pattern selection problem we conclude that the higher the average heuristic values, the better the corresponding pattern database. As a consequence we compute the mean heuristic value for each database and More formally, the average estimate of a singular pattern database wrt. state space abstraction ϕ is

$$\overline{h} = \sum_{h=0}^{\max} \frac{h \cdot |\{\phi(u) \in PDB \mid entry(u) = h\}|}{|PDB|},$$

where *entry* is the distance value stored for u. For multiple pattern databases, we compute the mean heuristic value for each database and add these for a chromosome. More formally, if PDB_i is the i-th pattern database, $i \in \{1, \ldots, k\}$, then the *fitness* of a chromosome c is

$$\sum_{i=1}^{k} \sum_{h_i=0}^{\max_i} \frac{h_i \cdot |\{\phi_i(u) \in PDB_i \mid entry_i(u) = h_i\}|}{|PDB_i|}.$$

For experimenting with symbolic pattern databases, we choose various problems from IPC benchmarks. For domains from IPC-4 good exploration results of symbolic pattern databases are already known (Edelkamp 2005). For the construction of symbolic pattern databases we choose a population size of 5, and a number of 20 epocs (resulting in at most 100 pattern databases to be evaluated; some of them were eliminated due to size and state variable constraints). The random seed was fixed for all experiments.

The initial population the GA is based on randomizing the order of groups in the state vector. Therefore bin packing yields better results than GA search in some cases. We apply symbolic A* search with full duplicate elimination.

In Fig. 5 our results of comparing greedy bin-packing with GA pattern selection are shown. The abstract state space size limit (2^l) is an additional parameter in both cases. The searching time t_s is compared to the total running time t which includes the parsing of the grounded domain. The time for instantiation is not counted. The setup times $t - t_s$ for the genetic algorithm includes the time for computing *all* pattern databases during the optimization process. The additional time for pattern optimization contributes to the gain in the quality of the heuristic estimate, measured in $\overline{h}$, the mean heuristic estimate of the first (greedy bin-packing) or best surviving (genetic algorithm) pattern. To judge the quality of the heuristic, we provide the plan lengths. While greedy bin packing often runs out of memory, improved pattern selection with GAs scales better.

Conclusion

We have devised cost-optimal propositional PDDL3 planning algorithms and shown a promising approach so generate well-informed symbolic estimator functions fully automatically. To evaluate the fitness function it is necessary to compute the PDBs for every chromosome in every generation. This is very costly, but as symbolic pattern database construction is faster than an explicit one, we showed the potential to save total search efforts compared to the simpler bin-packing-only heuristic. The performance of a randomized GA search is still hard to predict, so even if it can result in much better explorations in most cases, under the hard time and space constraints at IPC-5, we actually switched this feature off.

As the approach for state trajectory constraints relies on a translation to LTL, it has the potential to deal with much larger language expressiveness than currently under consideration. Nonetheless, there is still a small discrepancy between LTL semantics that is devised for infinite words and state trajectory constraints that are proposed for finite plans. This is the price for applying the existing tool LTL2BA for translation. It is possible but still unfinished work to transfer the tool to an LTL2NFA equivalent.

As related work, a neat solution to compile preferences as PDDL2 derived predicates (Baier 2006) and apply the FF_X planning system for derived predicates. So far, it has shown advances to TLPlan. For the future, we plan to generalize the symbolic planning approach from finite domain variables to integer and real variables over linear arithmetic (Boigelot, Jodogne, & Wolper 2005).

References

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116:123–191.

Baier, J. 2006. Planning with first-order temporally extended goals using heuristic search. In *AAAI*, Accepted.

Bartzis, C., and Bultan, T. 2006. Efficient BDDs for bounded arithmetic constraints. *STTT* 8(1):26–36.

Boigelot, B.; Jodogne, S.; and Wolper, P. 2005. An effective decision procedure for linear arithmetic over the integers and reals. *ACM Trans. Comput. Log.* 6(3):614–633.

Bryant, R. E. 1985. Symbolic manipulation of boolean functions using a graphical representation. In *ACM/IEEE Design Automation Conference*, 688–694.

Clarke, E. M.; Grumberg, O.; and Peled, D. A. 1999. *Model Checking*. MIT Press.

DeGiacomo, G., and Vardi, M. Y. 1999. Automata-theoretic approach to planning for temporally extended goals. In *ECP*, 226–238.

Edelkamp, S., and Helmert, M. 2001. Mips: The model-checking integrated planning system. *AI Magazine* 22(3):67–72.

Edelkamp, S., and Reffel, F. 1998. OBDDs in heuristic search. In *KI*, 81–92.

Edelkamp, S. 2001. Planning with pattern databases. In *ECP*, 13–24.

		Greedy Bin Packing					GA-Optimization				
problem	2^l	length	iterations	$\overline{h}$	t_s	t	length	iterations	$\overline{h}$	t_s	t
logistics-4-1	10	19	63	9.28	0.39	0.77	19	63	8.57	0.37	0.79
logistics-6-1	20	14	42	21.9	0.39	0.77	14	39	20.34	0.30	1.01
logistics-8-1	20	44	166	26.32	11.98	19.92	44	44	29.51	5.7	1.42
logistics-10-1	30	-	-	-	-	-	42	351	38.82	33.78	85.69
logistics-12-1	30	-	-	-	-	-	69	826	51.49	138.02	498.76
blocks-9-0	30	30	79	8.86	0.47	52.51	30	358	20.03	8.89	19.4
blocks-10-0	40	-	-	-	-	-	34	692	25.15	8.53	34.94
blocks-11-0	40	-	-	-	-	-	32	1,219	24.20	49.30	58.44
blocks-12-0	40	-	-	-	-	-	34	942	25.36	101.95	104.55
zeno-2	10	6	6	6.37	0.17	0.55	6	14	3.83	0.19	0.57
zeno-4	10	8	19	5.74	0.27	0.73	8	14	3.83	0.19	0.57
zeno-6	20	11	24	6.6	0.64	1.21	11	14	8.58	0.58	1.51
zeno-10	25	-	-	-	-	-	22	23	15.7	43.12	190.56
zeno-11	25	-	-	-	-	-	14	37	15.06	15.11	833.16
driverlog-9	25	22	109	12.9	86.76	87.59	22	107	15.3	52.46	72.25
driverlog-11	25	-	-	-	-	-	19	110	10.67	34.48	44.60
driverlog-13	35	-	-	-	-	-	26	143	13.7	553.01	778.03
openstack-1	20	23	96	3.71	0.51	1.29	23	116	5.06	0.60	3.04
openstack-3	20	23	96	3.71	0.51	1.29	23	110	5.40	0.60	3.02
openstack-6	30	20	65	4.99	70.38	96.17	20	39	5.44	52.42	216.19
openstack-7	30	-	-	-	-	-	20	38	6.88	31.05	484.19

Figure 5: Symbolic A* Search with and without GA Optimization.

Edelkamp, S. 2002. Symbolic pattern databases in heuristic search planning. In *AIPS*, 274–293.

Edelkamp, S. 2005. External symbolic heuristic search with pattern databases. In *ICAPS*, 51–60.

Edelkamp, S. 2006a. Cost-optimal symbolic planning with state trajectory and preference constraints. In *ECAI*, To Appear.

Edelkamp, S. 2006b. On the compilation of plan constraints and preferences. In *ICAPS*, To Appear.

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.

Gastin, P., and Oddoux, D. 2001. Fast LTL to Büchi automata translation. In *CAV*, 53–65.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. Technical report, Department of Electronics for Automation, University of Brescia.

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

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

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *ICAPS*, 161–170.

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

Holte, R. C.; Newton, J.; Felner, A.; Meshulam, R.; and Furcy, D. 2004. Multiple pattern databases. In *ICAPS*, 122–131.

Jensen, R.; Hansen, E.; Richards, S.; and Zhou, R. 2006. Memory-efficient symbolic heuristic search. In *ICAPS*, To Appear.

Jensen, R. M.; Bryant, R. E.; and Veloso, M. M. 2002. SetA*: An efficient BDD-based heuristic search algorithm. In *AAAI*, 668–673.

Kabanza, F., and Thiebaux, S. 2005. Search control in planing for termporally extended goals. In *ICAPS*, 130–139.

Knoblock, C. A. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68(2):243–302.

Korf, R. E., and Felner, A. 2002. Disjoint pattern database heuristics. *Artificial Intelligence* 134(1-2):9–22.

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

Korf, R. E. 1993. Linear-space best-first search. *Artificial Intelligence* 1(62):41–78.

Korf, R. E. 1997. Finding optimal solutions to Rubik's Cube using pattern databases. In *AAAI*, 700–705.

Lago, U. D.; Pistore, M.; and Traverso, P. 2002. Planning with a language for extended goals. In *AAAI*, 447–454.

Zhou, R., and Hansen, E. 2004. Space-efficient memory-based heuristics. In *AAAI*, 677–682.

Cost-Optimal Planning with Constraints and Preferences in Large State Spaces

Stefan Edelkamp[1], Shahid Jabbar[2], and Mohammed Nazih[3] *
Computer Science Department
University of Dortmund, Dortmund, Germany
[1]stefan.edelkamp@cs.uni-dortmund.de
[2]shahid.jabbar@cs.uni-dortmund.de
[3]mohammed.nazih@uni-dortmund.de

Abstract

This paper deals with planning in the presence of constraints and preferences as proposed for the 5^{th} International Planning Competition. State trajectory constraints are translated into LTL formulae and are compiled into Büchi automata in PDDL format. Preference constraints are compiled into numerical fluents. Values of these fluents are changed by grounded operator effects upon violation.

We propose two exploration strategies for optimal planning in PDDL3 domains: (i) a best-first branch-and-bound weighted heuristic search; (ii) an external breadth-first search exploration algorithm that exploits secondary memory, such as harddisk, to save the *open* and *closed* lists. We prove an upper bound on the locality of the search in planning graphs that dictates the number of layers that have to be kept to avoid re-openings. For non-optimal planning, we present an external variant of enforced hill climbing.

Introduction

In recent years, AI Planning has seen significant growth in both theory and practice. PDDL (Planning Domain Description Language) provides a common framework to define planning domains and problems. Starting from a pure propositional framework, it has now grown into accommodating more complex planning problems. In Metric planning, we see a numerical extension to the STRIPS planning formalism, where actions can contribute an increase or decrease of numeric variables. The task is then to find a path from an initial state to a state where all goal criteria are fulfilled, *additionally*, the values of a set of numeric variables are minimized (or maximized).

State trajectory and preference constraints are the two language features introduced in PDDL3 (Gerevini & Long 2005) for describing benchmarks of the 5^{th} international planning competition. *State trajectory constraints* provide an important step of the agreed fragment of PDDL towards the description of *temporal control knowledge* (Bacchus & Kabanza 2000; Kabanza & Thiebaux 2005) and *temporally extended goals* (DeGiacomo & Vardi 1999; Lago, Pistore, & Traverso 2002; Pistore & Traverso 2001). They assert conditions that must be met during the execution of a plan and are often expressed using quantification over domain objects. Through the decomposition of metric and temporal plans into happenings, state trajectory constraints also feature higher levels of the PDDL hierarchy (Fox & Long 2003).

Unfortunately, as the planning problems get complicated, the size of the state and the number of states grow significantly too - easily reaching the limits of main memory capacity. Having a systematic mechanism to flush the already seen states to the disk can circumvent the problem. Algorithms that utilize secondary storage devices have seen significant success in single-agent search. In (Korf & Schultze 2005) we see a complete exploration of the state space of 15-puzzle made possible utilizing a 1.4 Terabytes of secondary storage. In (Jabbar & Edelkamp 2005) a successful application of external memory heuristic search for LTL model checking is presented. (Zhou & Hansen 2004b) proposed structured duplicate detection for external search, where the state space structure was exploited to define a partition on the state space. Among the reported results are applications on STRIPS planning problems.

The paper is structured as follows: We first discuss the planning problem with temporal and preference constraints as proposed for the 5^{th} International Planning Competition. The temporal constraints are compiled into Büchi automata that are synchronized with the exploration of the planning problem, while preference constraints are transformed into numerical fluents. Upon violation, a penalty cost is imposed to the corresponding fluent. We then discuss some implementation details for the realization of this compilation procedure. An overview of the external memory model is presented afterward. Then, we introduce external Breadth-First Search for implicit undirected graphs. Directed graphs are treated in the next section, where we discuss a formal basis to determine the locality of planning graphs which dictates the number

*All three authors are supported by the German Research Foundation (DFG) projects *Heuristic Search* Ed 74/3 and *Directed Model Checking* Ed 74/2.

of previous layers to look at during duplicates removal to avoid re-expansions (Edelkamp & Jabbar 2006). For non-optimal planning, we discuss an external memory variant of enforced hill climbing.

State Trajectory Constraints

We briefly recall *automata-based model checking*, a common approach for model checking of softwares.

Automata-based Model Checking

In automata-based model checking both the model to be analyzed and the specification to be checked are modeled as non-deterministic *Büchi automata*. Syntactically, Büchi automata are ordinary automata. For accepting *infinite words*, or *runs*, a different acceptance condition is applied. Let ρ be a run and $inf(\rho)$ be the set of states reached infinitely often in ρ, then a Büchi automaton accepts, if the intersection between $inf(\rho)$ and the set of final states F is not empty.

The desired property of the system is specified in some form of temporal logic. We briefly introduce *linear temporal logic (LTL)*. A path in model $\mathcal{M}$ is a sequence of states $\pi = S_0, S_1, \ldots$ and π^i denotes the suffix of π starting at S_i. LTL formulae have the form "Always f", where f is a *path formula*. If p is an atomic proposition then p is a path formula. If f and g are path formulae so are $\neg f, f \vee g, f \wedge g, \mathbf{X}\ f, \mathbf{F}\ f, \mathbf{G}\ f$, and $f\ \mathbf{U}\ g$.

For the *next time* operator $\mathbf{X}$ we have $M, \pi \models \mathbf{X}\ f \Leftrightarrow M, \pi^1 \models f$. For the *until* operator $g\ \mathbf{U}\ f$ we have $M, \pi \models g\ \mathbf{U}\ f \Leftrightarrow \exists 0 \leq k : M, \pi^k \models f \wedge \exists k \leq j : M, \pi^j \models g$, for the *eventually* operator we have $M, \pi \models \mathbf{F}\ f \Leftrightarrow \exists 0 \leq k : M, \pi^k \models f$, and for the *globally* operator we have $M, \pi \models \mathbf{G}\ f \Leftrightarrow \forall 0 \leq k : M, \pi^k \models f$.

Transforming the model and the specification into Büchi automata assumes that the system can be modeled by a deterministic finite state machine, and that the LTL formula can be transformed into an equivalent Büchi automaton. The contrary is not always possible, since Büchi automata are clearly more expressive than LTL expressions (Wolper 1983). Checking correctness is reduced to checking language emptiness. More formally, the model checking procedure validates that a model represented by an automaton $\mathcal{M}$ satisfies its specification represented by an automaton $\mathcal{S}$. The task is to verify if $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}(\mathcal{S})$. In words: the *language accepted by the model is included in that of the specification*. We have $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}(\mathcal{S})$ if and only if $\mathcal{L}(\mathcal{M}) \cap \overline{\mathcal{L}(\mathcal{S})} = \emptyset$.

Büchi automata are closed under intersection and complementation (Buchi 1962), so that there exists an automaton that accepts $\overline{\mathcal{L}(\mathcal{S})}$ and an automata that accepts $\mathcal{L}(\mathcal{M}) \cap \overline{\mathcal{L}(\mathcal{S})}$.

It is possible to complement a Büchi automaton, *but* the worst-case running time of such a construction is double-exponential in the size of the formula. Therefore, in practice, one constructs the automaton for negation of the LTL formula, avoiding complementation.

The product is *synchronous*, that is, each transition in one automata implies one in the other. The property automaton is non-deterministic, such that both the successor generation and the temporal formula representation may introduce branching to the overall exploration module. The construction assumes that all states in the model are accepting.

Application to Temporal Plan Constraints

In the proposed extension to planning we do not have to negate the property formula. Planning goals already correspond to the negations of properties in model checking. If ordinary goals without temporal modalities are used, we add their satisfaction to the acceptance condition of the model. For state trajectory constraints ϕ, we search for a witness in $L(\mathcal{M}) \cap L(\phi) \neq \emptyset$, where $\mathcal{M}$ is the original plan space.

For the exploration we, therefore, need a Büchi automaton for the model and one for the trajectory constraint, together with some algorithm that validates if the language intersection is not empty. By the semantics of (Gerevini & Long 2005) it is clear that all sequences are finite, so that we can interpret a Büchi automaton as a non-deterministic finite state automaton (NFA), which accepts a word if it *terminates* in a final state. The labels of such an automaton are conditions over the propositions and fluents in a given state. We will illustrate how these conditions can be modeled using planning operators. There are some important observations to be made:

1. It is well known that an NFA can be transformed into an equivalent deterministic one using a power set construction (Hopcroft & Ullman 2000). This DFA, however, can become exponentially large, so that in most cases a simulation of the NFA is preferable.
2. Most state trajectory constraints are universally quantified. The quantified expressions can be unrolled. This is always possible as the scope of the quantified object variables is finite.
3. As the union of the conditions of all outgoing transitions is not always trivial, synchronizing the planning model with the automata of state trajectory constraint may also prune the exploration.

Examples

In PDDL3, the constraint *a fragile block can never have something above it* is expressed as

```
(always (forall (?b - block)
          (implies (fragile ?b) (clear ?b))
```

We call this condition an *always/every* constraint. The LTL formula for two selected blocks a and b is

```
[] ((fragile_a -> clear_a) &&
    (fragile_b -> clear_b))
```

The corresponding automaton is shown in Fig. 1[1]. The

[1]The automata in the figures are constructed automatically using the LTL to Büchi automaton converter. The in-

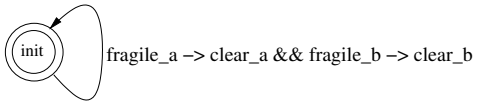

Figure 1: Automaton for the *always/every* constraint with a transition in clause form.

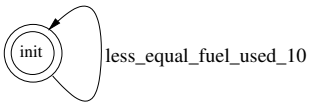

Figure 2: Automaton for numeric constraint.

automaton consists of only one state. Numeric conditions as generated, e.g., by `(always (<= (fuel-used) 10))` do not lead to additional expressiveness as for the translation process they are interpreted as integral propositions to be combined in transition labels. Fig. 2 provides a simple example. When applying such an automata, the construction has to be followed by a backward translation of edges to numeric conditions.

The assertion *each block should be put on the table at least once* corresponds to

```
(forall (?b - block)(sometime (ontable ?b)))
```

called an *every/sometime* constraint. For two blocks a Büchi automaton with respect to the LTL formula `(<> ontable_a) && (<> ontable_b)` is constructed. It is shown in Fig. 3 (top right).

The statement *in some state visited by the plan all blocks are on the table* is expressed as

```
(sometime (forall (?b - block) (ontable ?b))
```

denoted as *sometime/every* constraint. The according LTL formula is `<> (ontable_a && ontable_b)` with a Büchi automata shown in Fig. 3 (top left). It is much simpler than the previous one.

The expression *each truck should visit each city at most once* is given by the constraint

```
(forall (?t - truck ?c - city)
        (at-most-once (at ?t ?c)))
```

We use a simple instantiation with one truck and one city, yielding the LTL formula (cf. (Gerevini & Long 2005))

```
[](at_truck_a_city_a->
  (at_truck_a_city_a U ([]!at_truck_a_city_a)))
```

The corresponding Büchi-Automaton is displayed in Fig. 3 (bottom). The translation from a PDDL constraint to a Büchi automata is not lossless. This is due to the fact that PDDL constraints are defined over finite runs while Büchi automata are defined over infinite runs. It is not possible to capture the exact semantics

termediate LTL notation satisfies the syntax of *SPIN* model checker; *always* is denoted as `[]` and eventually as `<>`. We avoid hyphens in declaring the propositions as they are misinterpreted by the converter. They are re-introduced when generating the translated PDDL description.

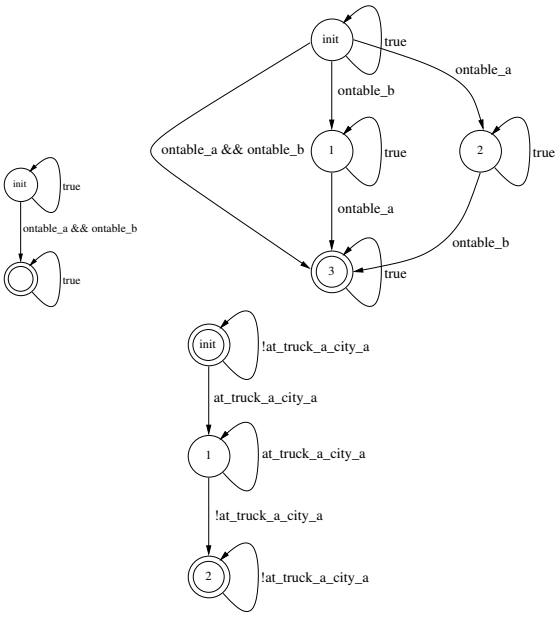

Figure 3: Büchi automata for the *sometime/every* constraint (top left), *every/sometime* constraint (top right) and *exactly-once* constraint (bottom).

of, e.g., `at-most-once` ϕ, when the plan consists of just one action and ϕ holds in the initial state as well as in the goal state. In our previous example, this implies that we will terminate at the state 1 of the automata and will claim that the constraint is not satisfied.

Instead of deriving one automata for both constraints in common, simulating two synchronized automata, one for each constraint, is equivalent. Hence, we observe a trade-off between the size of the automata for one constraint and the maintenance of several concurrent automata.

Preferences

Preferences model soft constraints that are desirable but do not have to be fulfilled in a valid plan. The degree of desirability of satisfying a preference constraint is specified in the plan metric.

Simple Preferences

Simple preferences refer to ordinary propositions (and are included to the planning goal). E.g., if we prefer block a to reside on the table during the plan execution, we write `(preference p (on-table a))` with a validity check `(is-violated p)` in the plan objective. Such checks are interpreted as natural numbers that can be scaled and combined with other variable assignments in the plan metric. To evaluate the costs for a given plan, we have to accumulate how often the stated preference condition is violated in the preconditions of the actions in the plan. The according numerical value is substituted in the metric for its evaluation. Quantified preference constraints like

```
(forall (?b - block) (preference p (clear ?b))
```

are flattened to multiple instantiated preference conditions (one for each block), while the inverse expression

```
(preference p (forall (?b - block) (clear ?b))
```

leads to only one constraint.

Preference on Temporal Plan Constraints

Preferences for state trajectory constraints like

```
(preference cleaned
   (forall (?t truck) (always (clean ?t))))
```

can, in principle, also be dealt with automata theory. Instead of requiring to reach an accepting state we *prefer* to be there, by means that not arriving at an accepting state incurs costs to the evaluation of the plan metric using the `(is-violated cleaned)` variable.

Language Compilation

Have the two language extensions enriched the PDDL language or is it possible to translate the new constructs away? Fortunately, we can show how to implement a language compilation from PDDL3 to PDDL2.

Temporal Plan Constraints

To encode the simulation of the synchronized automata, we devise a predicate `(at ?n - state ?a - automata)` to be instantiated for each automata state and each automata that has been devised. For detecting accepting states, we include instantiations of predicate `(accepting ?a - automata)`. The initial state of the planning problem includes the start state of the automata and an additional proposition if it is accepting. For all automata, the goal includes their acceptance.

Next, we have to specify allowed automata transitions in form of planning actions. This is done by declaring a grounded operator for each automata transition, with the current automaton state and the transition label as preconditions, as well as the current automaton state as the delete and the successor state as the add effect. For transition leading to an accepting state, we include the corresponding automata acceptance proposition to the add effects. As we require a tight synchronization between the constraint automaton transitions and the operators in the original planning space, we include *synchronization flags* that are flipped when an ordinary or a constraint automaton transition is chosen. An example for a grounded transition is

```
(:action sync-trans-a-0-init-a-0-accept-0
:precondition
     (and (at-a-0-init) (sync-automaton-a-0)
          (in-package1-truck2))
:effect
     (and (accepting-a-0) (not (at-a-0-init))
          (at-a-0-accept-0) (not (sync-automaton-a-0))
          (sync-ordinary)))
```

As said, the size of the Büchi automaton for a given formula can be exponential in the length of the formula[2]. In practice, the size of the automaton is often small compared to the size of the (grounded) model.

[2]In the notion of *essentiality* (Nebel 2000), which provides a complexity theory for domain compilations, the compilation is *essential*. Similar essential compilations have been proposed by (Gazen & Knoblock 1997) for ADL to STRIPS and by (Thiebaux, Hoffmann, & Nebel 2005) for domain axioms.

Metric Time Constraints

So far we have only seen how to derive automata for the *untimed plan constraints* `sometime`, `always`, `at-most-once`. Fortunately, (Gerevini & Long 2005) show that `sometime-before` and `sometime-after` can be expressed using standard LTL expressions, so these modalities easily fit into the above framework. For *metric time constraints* like `within`, `always-within`, `hold-during`, and `hold-after` we have to restrict actions to the execution time window specified in the constraints. Moreover, these constraints necessarily call for parallel/temporal planning, as they refer to absolute points in time for plan execution.

These expressions can be tackled using *timed initial literals* as already contained in the language PDDL2 (Hoffmann & Edelkamp 2005). Timed initial literals denote fixed *dates* in plan time in which an atom is *true* or *false*. As they are only allowed to be checked in operators' preconditions, they correspond to *action execution time windows*. The modalities `hold-after`, and `hold-during` immediately translate to timed initial literals for the operators, in which the stated conditions are satisfied in the preconditions. If the state formula is disjunctive the planner has to deal with multiple action windows.

For combined metric and temporal modalities as in `within` and `always-within` action execution time window are included in form of additional timed initial literal for the preconditions of the automata's transitions.

Preferences

For preference p we include numerical fluents `is-violated-p` to the grounded domain description. For each operator and each preference we apply the following reasoning. If the preferred predicate is contained in the *delete list* then the fluent is increased, if it is contained in the *add list*, then the fluent is decreased, otherwise it remains unchanged[3].

For preferences p on a state trajectory constraint that has been compiled to an automaton a, the fluents `(is-violated-a-p)` substitute the atoms `(is-accepting-a)` in an obvious way. If the automata accepts, the preference is fulfilled, so the value of `(is-violated-a-p)` is set to 0. In the transition that newly reaches an accepting state `(is-violated-a-p)` is set to 0, if it enters a non-accepting state it is set to 1. The `skip` operator also induces a cost of 1 and the automaton moves to a dead state.

[3]An alternative semantic to (Gerevini & Long 2005) would be to set the fluent to either 0 or 1. For rather complex propositional or numerical goal conditions in a preference condition, we can use *conditional effects*.

Implementation

We first transform PDDL3 files with preferences and state trajectory constraints to grounded PDDL3 files without them. For each state trajectory constraint, we parse its specification, flatten the quantifiers and write the corresponding LTL-formula to disk.

Then, we derive a Büchi-automaton for each LTL formula and generates the corresponding PDDL code to modify the grounded domain description[4]. Next, we merge the PDDL descriptions corresponding to Bčhi automata and the problem file. Given the grounded PDDL2 outcome, we apply efficient heuristic search forward chaining planner *Metric-FF* (Hoffmann 2003). Note that by translating plan preferences, otherwise propositional problems are compiled into metric ones. For temporal domains, we extended the *Metric-FF* planner to handle temporal operators and timed initial literals. The resulting planner is slightly different from known state-of-the-art systems of adequate expressiveness, as it can deal with disjunctive action time windows and uses an internal linear-time approximate scheduler to derive parallel (partial or complete) plans. The planner is capable of compiling and producing plans for all competition benchmark domains.

Due to the numerical fluents introduced for preferences, we are faced with a search space where cost is not neccessarily monotone. For such state spaces, we have to look at all the states to reach to an optimal solution. The issue then arises is if it is possible to reach an optimal solution fast. We propose to use a branch-and-bound like procedure on top of the best-first weighted heuristic search as offered by the extended *Metric-FF* planning system. Upon reaching a goal, we terminate our search and create a new problem file where the goal condition is extended to minimize the found solution cost. The search is restarted on this new problem description. The procedure terminates when the whole state space is looked at. The rationale behind this is to have improved guidance towards a better solution quality. If internal search fails to terminate within a specified amount of time, we switch to External Breadth-First search (BFS).

External Exploration

For complex planning problems, the size of the state space can easily surpass the main memory limits. Most modern operating systems provides a facility to use larger address spaces through *virtual memory* that can be larger than internal memory. When the program is executed, virtual addresses are translated into physical addresses. Only those portions of the program currently needed for the execution are copied into main memory; the rest stays on the harddisk. For the programs that do not exhibit any *locality of reference* for memory accesses, such general purpose virtual memory management can instead lower down their performances.

Algorithms that explicitly manage the memory hierarchy can lead to substantial speedups, since they are more informed to predict and adjust future memory access. The standard model for comparing the performance of external algorithms consists of a single processor, a small internal memory that can hold up to M data items, and an unlimited secondary memory. The size of the input problem (in terms of the number of records) is abbreviated by N. Moreover, the *block size* B governs the bandwidth of memory transfers. Only the number of block reads and writes are counted, computations in internal memory do not incur any cost. The single disk model for external algorithms has been devised by (Aggarwal & Vitter 1988).

It is convenient to express the complexity of external-memory algorithms using a number of frequently occurring primitive operations. Here D represents the number of disks that can be accessed simultaneously.

1. *Scanning*: *scan(N)* with an I/O complexity of $\Theta(\frac{N}{DB})$ that can be achieved through trivial sequential access.
2. *Sorting*: *sort(N)* with an I/O complexity of $\Theta(\frac{N}{DB}\log_{M/B}\frac{N}{B})$ that can be achieved through *Merge* or *Distribution Sort*

External Breadth-First Search in Undirected Graphs

Munagala and Ranade's algorithm (Munagala & Ranade 1999) for explicit Breadth-First Search has been adapted for implicit graphs. The new algorithm is known as *delayed duplicate detection* for *frontier search*. It assumes an undirected search graph. Let $\mathcal{I}$ be the initial state, and N be the implicit successor generation function. The algorithm maintains BFS layers on disk. Layer $Open(i-1)$ is scanned and the set of successors are put into a buffer of size close to the main memory capacity. If the buffer becomes full, internal sorting followed by a duplicate elimination scanning phase generates a sorted duplicate-free state sequence in the buffer that is flushed to disk.

In the next step, *external merging/sorting* is applied to remove duplicates in the flushed buffers. This results in a duplicates-free sorted file corresponding to $Open(i)$. One also has to eliminate $Open(i-1)$ and $Open(i-2)$ from $Open(i)$ to avoid re-expansions; that is, nodes extracted from the external queue are not immediately deleted, but kept until after the layer has been completely generated and sorted, at which point duplicates can be eliminated using a parallel scan. The process is repeated until $Open(i-1)$ becomes empty, or the goal has been found.

The corresponding pseudo-code is shown in Figure 4. A-sets in the algorithm correspond to temporary files. Termination is not shown, but imposes no additional overhead. As with the algorithm of Munagala and Ranade, delayed duplicate detection applies

[4] `www.liafa.jussieu.fr/~oddoux/ltl2ba`. Similar tools include *LTL→NBA* and the never-claim converter inherent to the SPIN model checker.

Procedure External-BFS
$Open(-1) \leftarrow \emptyset$, $Open(0) \leftarrow \{\mathcal{I}\}$
$i \leftarrow 1$
while $(Open(i-1) \neq \emptyset)$
$A(i) \leftarrow N(Open(i-1))$
$A'(i) \leftarrow$ *remove duplicates from* $A(i)$
$Open(i) \leftarrow A'(i) \setminus (Open(i-1) \cup Open(i-2))$
$i \leftarrow i+1$

Figure 4: Delayed duplicate detection in BFS.

$O(sort(|N(Open(i-1))|)+scan(|Open(i-1)|+|Open(i-2)|))$ I/Os. However, since no explicit access to the adjacency list is needed, by $\sum_i |N(Open(i))| = O(|E|)$ and $\sum_i |Open(i)| = O(|V|)$, the total execution time is $O(sort(|E|) + scan(|V|))$ I/Os.

Locality in Planning Domains

How many layers are sufficient for full duplicate detection in general is dependent on a property of the search graph called *locality*. For integer weighted problem graphs, it is defined as the maximum $\max\{\delta(s,u) - \delta(s,v), 0\}$ of all nodes u, v, with v being a successor of u and δ the shorted path distance. For undirected graphs we always have that $\delta(s,u)$ and $\delta(s,v)$ differ by at most one so that the locality is 1. The locality determines the *thickness* of the boundary slice of the graph needed to prevent duplicates.

Let l be the graphs locality, and k the number of stored layers. In breadth-first search, when layer m is expanded, all previous layers with g-value smaller than m have been closed, and are known by their optimal g-value. Thus, if a node u at level m is expanded, and its successor v has a shorter optimal distance to s, i.e., $m = \delta(s,v) < \delta(s,u) = m'$, then v must have been encountered earlier in the search, in the worst case at layer $m' = m - l$. The re-generation of v will be avoided if and only if it is contained in the stored layers $m - k \ldots m-1$; i.e., if and only if $k \geq l$. This is the basis of the following theorem due to (Zhou & Hansen 2004a)

Theorem 1 *(Locality Determines Boundary) The number of previous layers of a breadth-first search graph that need to be retained to prevent duplicate search effort is equal to the locality of the search graph.*

As a special case, in undirected graphs, the locality is 1 and we need to store the immediate previous layer only to check for duplicates.

The condition $\max\{\delta(s,u) - \delta(s,v), 0\}$ over all nodes u, v, with v being a successor of u is not a graph property. So the question is if we can find a sufficient condition or upper bound for it.

Theorem 2 *(Upper-Bound on Locality) The locality of a uniformly weighted graph for breadth-first search can be bounded by the minimal distance to get back from a successor node v to u, maximized over all u. In other words, with Γ representing the set of successors, we have*

$$\max_{u,v \in \Gamma(u)} \{\delta(v,u)\} \geq \max_{u,v \in \Gamma(u)} \{\delta(s,u) - \delta(s,v), 0\}$$

Proof: For any nodes s, u, v in a graph the triangular property of shortest path $\delta(s,u) \leq \delta(s,v) + \delta(v,u)$ is satisfied, in particular for s being the start node of the BFS and $v \in \Gamma(u)$. Therefore $\delta(v,u) \geq \delta(s,u) - \delta(s,v)$ and $\max_{u,v\in\Gamma(u)}\{\delta(v,u)\} \geq \max_{u,v\in\Gamma(u)}\{\delta(s,u) - \delta(s,v)\}$. In positively weighted graphs we have $\delta(v,u) \geq 0$ such that $\max_{u,v\in\Gamma(u)}\{\delta(v,u)\}$ is larger than the locality.

As for graphs without self-loops we have $\max_{u,v\in\Gamma(u)}\{\delta(v,u)\} = \max_u\{\delta(u,u)\} - 1$, in order to bound the locality we have to look for largest minimal cycles in the graph.

The question then arises is: How can we decide the condition in an implicitly given graph as they appear in action planning? In the following we provide an answer to this question based on the rules or operators involved in a state space. Without loss of generality, we consider STRIPS planning operators in the form of $\langle pre(O)\, add(O), del(O)\rangle$, representing preconditions, add, and delete lists for an operator O. A duplicate node in an implicit graph appears when a sequence of operators, applied to a state generate the same state again, i.e., they cancel the effects of each other. Hence the following definition:

Definition 1 *(*no-op *Sequence) A sequence of operators $O_1, O_2, \ldots, O_k$ is a* no-op sequence *if its application on a state produces no effects, i.e, $O_k \circ \ldots \circ O_2 \circ O_1 =$* no-op*,*

This definition provides us the basis to bound the locality of the implicit graphs in the following theorem. It generalizes undirected search spaces, in which for each operator O_1 we find an inverse operator O_2 such that $O_2 \circ O_1 = no\text{-}op$.

Theorem 3 *(*no-op *Sequence determines Locality) Let $\mathcal{O}$ be the set of operators in the search space and $l = |\mathcal{O}|$. If for all operators O_1 we can provide a sequence $O_2, \ldots, O_k$ with $O_k \circ \ldots \circ O_2 \circ O_1 =$* no-op*, where* no-op *is the identity mapping, then the locality of the implicitly generated graph is at most $k-1$.*

Proof: If $O_k \circ \ldots \circ O_2 \circ O_1 = no\text{-}op$ we can reach each state u again in at most k steps. This implies that $\max_u\{\delta(u,u)\} = k$. Theorem 2 shows that $\max_u\{\delta(u,u)\} - 1$ is an upper bound on the locality.

The condition $O_k \circ \ldots \circ O_2 \circ O_1 = no\text{-}op$ can be tested in $O(l^k)$ time. It suffices to check that the cumulative add effects of the sequence is equal to the cumulative delete effects. Using the denotation by (Haslum & Jonsson 2000), the cumulative add C_A and delete C_D effects of a sequence can be defined inductively as,

$$C_A(O_k) = A_k \qquad C_D(O_k) = D_k \text{ and,}$$

$$\begin{aligned} C_A(O_1, \ldots, O_k) &= (C_A(O_1, \ldots, O_{k-1}) - D_k) \cup A_k \\ C_D(O_1, \ldots, O_k) &= (C_D(O_1, \ldots, O_{k-1}) - A_k) \cup D_k \end{aligned}$$

Procedure Cost-Optimal-External-BFS
$U \leftarrow \infty$; $i \leftarrow 1$
$Open(-1) \leftarrow \emptyset$; $Open(0) \leftarrow \{\mathcal{I}\}$
while $(Open(i-1) \neq \emptyset)$
  $A(i) \leftarrow N(Open(i-1))$
  forall $v \in A(i)$
    if $v \in \mathcal{G}$ **and** $Metric(v) < U$
      $U \leftarrow Metric(v)$
      $ConstructSolution(v)$
  $A'(i) \leftarrow$ *remove duplicates from* $A(i)$
  for $loc \leftarrow 1$ **to** *locality*
    $A'(i) \leftarrow A'(i) \setminus Open(i-loc)$
  $Open(i) \leftarrow A'(i)$
  $i \leftarrow i+1$

Figure 5: Cost-Optimal External BFS Planning. $\mathcal{G}$ is the set of goals and U the best goal cost found.

Theorem 3 gives us the missing link to the successful application of external breadth first search in planning. Subtracting k previous layer *plus* the current layer from the successor list in an external breadth-first search guarantees its termination on finite planning graphs.

Cost-Optimal External BFS

In planning with preferences, we often have a monotone decreasing instead of a monotonic increasing cost function. Hence, we cannot prune states with an evaluation larger than the current one. Essentially, we are forced to look at all states.

Figure 5 displays the pseudo-code for external BFS exploration incrementally improving an upper bound U on the solution length. The state sets that are used are represented in form of files. The search frontier denoting the current BFS layer is tested for an intersection with the goal, and this intersection is further reduced according to the already established bound.

In an internal non memory-limited setting, a plan is constructed by backtracking from the goal node to the start node. This is facilitated by saving with every node a pointer to its predecessor. For memory-limited frontier search, a divide-and-conquer solution reconstruction is needed for which certain relay layers have to be stored in main memory. In external search divide-and-conquer solution reconstruction and relay layers are not needed, since the exploration fully resides on disk.

There is one subtle problem: predecessor pointers are not available on disk. This is resolved as follows. We propose to save predecessor together with every state. Once a goal is found, backtracking to the initial state along the stored files, and by looking for the matching predecessors constructs the whole solution. This results in a I/O complexity that is at most linear to the number of stored states. In the pseudo-codes this procedure is denoted by *ConstructSolution*.

Procedure External Enforced Hill-Climbing
$u \leftarrow \mathcal{I}$
$h = Heuristic(\mathcal{I})$
while $(h \neq 0)$
  $(u', h') \leftarrow$ *External-EHC-BFS*(u, h)
  if $(h' = \infty)$ **return** $\emptyset$
  $u \leftarrow u'$
  $h \leftarrow h'$
return $ConstructSolution(u)$

Figure 6: External Enforced Hill-Climbing.

Procedure External-EHC-BFS(u, h)
$Open(-1, h) \leftarrow \emptyset$, $Open(0, h) \leftarrow u$
$i \leftarrow 1$
while $(Open(i-1, h) \neq \emptyset)$
  $A(i) \leftarrow N(Open(i-1, h))$
  forall $v \in A(i)$
    $h' = Heuristic(v)$
    if $h' < h$
      return (v, h')
  $A'(i) \leftarrow$ *remove duplicates from* $A(i)$
  for $loc \leftarrow 1$ **to** *locality*
    $A'(i) \leftarrow A'(i) \setminus Open(i-loc)$
  $Open(i) \leftarrow A'(i)$
  $i \leftarrow i+1$

Figure 7: External-BFS for External Enforced Hill Climbing. u is the new start state with the heuristic estimate h.

External Enforced Hill Climbing

Enforced Hill Climbing (EHC) is an *enforced* form of hill climbing search. Starting from a start state, a breadth-first search is performed for a successor with a better heuristic value. As soon as such a successor is found, the hash tables are cleared and a fresh breadth-first search is started. The process continues until the goal is reached. Since EHC performs a complete breadth-first search on every state with a strictly better heuristic value, it is guaranteed to find a solution. The following theorem is due to (Hoffmann & Nebel 2001).

Theorem 4 *For directed graphs without dead-ends, Enforced Hill Climbing is complete and guaranteed to find a solution.*

Having external BFS in hand for planning domains, an external algorithm for enforced hill limbing can be constructed by utilizing the heuristic estimates. In Figure 6, we show the algorithm in pseudo-code format for external enforced hill-climbing. The externalization is embedded in the sub-procedure (Figure 7) that performs external breadth-first search for a state with better heuristic estimate. As heuristic guidance, we chose relax plan heuristics (Hoffmann 2003).

Conclusions

In this paper, we discussed a method to translate temporal and preference constraints into PDDL2. Temporal constraints are converted into Büchi automata in PDDL format, and are executed synchronously with the main exploration. Preferences are compiled away by a transformation into numerical fluents that impose a penalty upon violation. Incorporating better heuristic guidance, especially, for preferences is still an open research frontier.

We discuss two external algorithms in this paper: Cost-optimal external breadth-first search and external enforced hill climbing search for non-optimal planning. The crucial problem in external memory algorithms is the duplicate detection with respect to previous layers to guarantee termination. Using the locality of the graph calculated directly from the operators themselves, we provide a bound on the number of previous layers that have to be looked at.

Since states are kept on disk, external algorithms have a large potential for parallelization. We noticed that most of the execution time is consumed while calculating heuristic estimates. Distributing a layer on multiple processors can distribute the internal load without having any effect on the I/O complexity.

References

Aggarwal, A., and Vitter, J. S. 1988. The input/output complexity of sorting and related problems. *Journal of the ACM* 31(9):1116–1127.

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116:123–191.

Buchi, J. R. 1962. On a decision method in restricted second order arithmetic. In *Conference on Logic, Methodology, and Philosophy of Science*, 1–11.

DeGiacomo, G., and Vardi, M. Y. 1999. Automata-theoretic approach to planning for temporally extended goals. In *ECP*, 226–238.

Edelkamp, S., and Jabbar, S. 2006. Cost-optimal external planning. In *National Conference on Artificial Intelligence (AAAI)*. To Appear.

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

Gazen, B. C., and Knoblock, C. 1997. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In *ECP*, 221–233.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. Technical report, Department of Electronics for Automation, University of Brescia.

Haslum, P., and Jonsson, P. 2000. Planning with reduced operator set. In *AIPS*, 150–158.

Hoffmann, J., and Edelkamp, S. 2005. The deterministic part of IPC-4: An overview. *JAIR* 24:519–579.

Hoffmann, J., and Nebel, B. 2001. Fast plan generation through heuristic search. *JAIR* 14:253–302.

Hoffmann, J. 2003. The Metric FF planning system: Translating "Ignoring the delete list" to numerical state variables. *JAIR* 20:291–341.

Hopcroft, J. E., and Ullman, J. D. 2000. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

Jabbar, S., and Edelkamp, S. 2005. I/O efficient directed model checking. In *VMCAI*. 313–329.

Kabanza, F., and Thiebaux, S. 2005. Search control in planing for termporally extended goals. In *ICAPS*, 130–139.

Korf, R. E., and Schultze, P. 2005. Large-scale parallel breadth-first search. In *AAAI*.

Lago, U. D.; Pistore, M.; and Traverso, P. 2002. Planning with a language for extended goals. In *AAAI*, 447–454.

Munagala, K., and Ranade, A. 1999. I/O-complexity of graph algorithms. In *SODA*, 687–694.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *JAIR* 12:271–315.

Pistore, M., and Traverso, P. 2001. Planning as model checking for extended goals in non-deterministic domains. In *IJCAI*, 479–486.

Safra, S. 1998. On the complexity of omega-automata. In *Annual Symposium on Foundations of Computer Science*, 319–237. IEEE Computer Society.

Sistla, A. P.; Vardi, M. Y.; and Wolper, P. 1983. The complementation problem for Buchi automata with applications to temporal logic. *Theoretical Computer Science* 49(2–3):217–237.

Thiebaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artificial Intelligence* 168(1–2):38–69.

Wolper, P. 1983. Temporal logic can be more expressive. *Information and Control* 56:72–99.

Zhou, R., and Hansen, E. 2004a. Breadth-first heuristic search. In *ICAPS*, 92–100.

Zhou, R., and Hansen, E. 2004b. Structured duplicate detection in external-memory graph search. In *National Conference on Artificial Intelligence (AAAI)*. 683–689.

Preferences and Soft Constraints in PDDL3

Alfonso Gerevini[+] **and Derek Long**[*]

[+] Department of Electronics for Automation, University of Brescia (Italy), gerevini@ing.unibs.it
[*] Department of Computer and Information Sciences, University of Strathclyde (UK), derek.long@cis.strath.ac.uk

Abstract

In many read-world planning domains, generating good plan quality is a central issue. This is especially true for problems with many solutions, or with many goals that cannot be achieved altogether. We propose an extension to the PDDL language that aims at a better characterization of plan quality by allowing the user to express strong and soft state constraints about the structure of the desired plans, as well as strong and soft problem goals. In the plan quality evaluation, soft goals and constraints are evaluated according the their violation penalty weights, which are expressed by the user in the plan metric. The new language, PDDL3, allows us to distinguish alternative feasible plans (satisfying all strong constraints and goals), preferring plans that minimize the weighted violations for soft goals or constraints, possibly combined with other plan quality criteria. We describe the syntax and semantics of PDDL3.0 and we give several examples, including a domain from the very recent fifth International planning competition, which focused on soft trajectory constraints and goals.

1 Introduction

The notion of plan quality in automated planning is of great practical importance. In many real-world planning domains, we have to address problems with a large set of solutions, or with a set of goals that cannot all be achieved. In these problems, it is important to generate plans of good or optimal quality achieving as many goals as possible. When only a subset of goals can be achieved (because they conflict with each other, or because achieving all goals is computationally too expensive), the ability to distinguish the importance of different goals is critical.

PDDL is the standard planning language used in the International planning competitors (Ghallab *et al.* 1998; Fox & Long 2003; Edelkamp & Hoffmann 2004). The current version of PDDL, PDDL2.2, allows us to express some criteria for plan quality, such as the number of plan actions or parallel steps, and relatively complex plan metrics involving plan makespan and numerical quantities. These are powerful and expressive in domains that include metric fluents, but plan quality can still only be measured by plan size in the case of propositional planning. We believe that these criteria are insufficient, and we propose to extend PDDL with new constructs increasing its expressive power in specifying the plan quality metric.

The proposed extended language, PDDL3, allows us to express *strong and soft constraints on plan trajectories* (that is, constraints over possible actions in the plan and intermediate states reached by the plan), as well as *strong and soft problem goals* (that is, goals that must be achieved in any valid plan, and goals that we desire to achieve, but that do not have to be necessarily achieved).

Some informal examples of plan trajectory constraints and soft goals in a blocksworld domain are: *a fragile block can never have something above it, or it can have at most one block on it*; *we would like that the blocks forming the same tower always have the same colour*; *in some state of the plan, all blocks should be on the table*; *we would like that in the goal state there is only one block on the table*.

Some additional examples in a transportation domain are: *we would like that every airplane is used* (perhaps because it is better to distribute the workload among the available resources and limit heavy usage); *whenever a ship is ready at a port to load the containers it has to transport, all such containers should be ready at that port*; *we would like that at the end of the plan all trucks are clean and at their source location*; *we would like no truck to visit any destination more than once*.

Strong constraints and goals must be satisfied by any valid plan, while soft constraints and goals express desirable outcomes, some of which may be more preferred than others. Informally, in planning with soft constraints and goals, the best quality plan should satisfy "as much as possible" the soft constraints and goals according to the specified preference relation distinguishing alternative feasible plans (satisfying all strong constraints and goals).

While soft constraints have been extensively studied in the CSP literature (e.g., (Dubois, Fargier, & Prade 1996; Bistarelli, Montanari, & Rossi 1997; Rossi, Venable, & Yorke-Smith 2004)), only very recently has the planning community started to investigate them (Brafman & Chernyavsky 2005; Briel *et al.* 2004; Delgrande, Schaub, & Tompits 2005; Miguel, Jarvis, & Shen 2001; Smith 2004; Son & Pontelli 2004). A significant recent effort along this direction has been undertaken by the fifth International Planning Competition (IPC-5), which focuses on planning with soft goals and constraints using PDDL3.0 (Gerevini & Long 2005b), a first version of PDDL3 where we have imposed some simplifying restrictions to the language to make it more accessible for the competitors.

When we have soft constraints and goals, it can be useful to give different priorities to them, and this should be taken into account in the plan quality evaluation. While

there is more than one way to specify the importance of a soft constraint or goal, as a first attempt to tackle this issue, in PDDL3.0 we have chosen a simple quantitative approach: each soft constraint and goal is associated with a numerical weight representing the cost of its violation in a plan (and hence also its relative importance with respect the other specified soft constraints and goals). Weighted soft constraints and goals are part of the plan metric expression, and the best quality plans are those optimising such an expression (more details are given in the next sections).

Using this approach we can express that certain plans are more preferred than others. Some examples are: *I prefer a plan where every airplane is used, rather than a plan using 100 units of fuel less*, which could be expressed by weighting a failure to use all the planes by a number 100 times bigger than the weight associated with the fuel use in the plan metric; *I prefer a plan where each city is visited at most once, rather than a plan with a shorter makespan*, which could be expressed by using constraint violation costs penalising a failure to visit each city at most once very heavily; *I prefer a plan where at the end each truck is at its start location, rather than a plan where every city is visited by at most one truck*, which could be expressed by using goal costs penalising a goal failure of having every truck at its start location more heavily than a failure of having in the plan every city visited by at most one truck. Other formalised examples are given in following sections.

We also observe that the rich additional expressive power we propose to add for goal specifications allows the expression of constraints that are actually derivable necessary properties of optimal plans. By adding them as goal conditions, we have a way to express constraints that we know will lead to the planner finding optimal plans. Similarly, one can express constraints that prevent a planner from exploring parts of the plan space that are known to lead to inefficient performance.

2 State Trajectory Constraints

2.1 Syntax and Intended Meaning

State trajectory constraints assert conditions that must be met by the entire sequence of states visited during the execution of a plan. They are expressed through temporal modal operators over first order formulae involving state predicates. We recognise that there would be value in also allowing propositions asserting the occurrence of action instances in a plan, rather than simply describing properties of the states visited during execution of the plan, but we choose to restrict ourselves to state predicates in this extension of the language.

The basic modal operators we propose to use in IPC-5 are: `always`, `sometime`, `at-most-once`, and `at end` (for goal state conditions). We add `within` which can be used to express deadlines. In addition, rather than allowing arbitrary nesting of modal operators, we introduce some specific operators that offer some limited nesting. We have `sometime-before`, `sometime-after`, `always-within`. Other modalities could be added, but we believe that these are sufficiently powerful for an initial level of the sublanguage modelling constraints.

It should be noted that, by combining these modalities with *timed initial literals* (defined in PDDL2.2 (Edelkamp & Hoffmann 2004)), we can express further goal constraints. In particular, one can specify the interval of time when a goal should hold, or the lower bound on the time when it should hold. Since these are interesting and useful constraints, we introduce two modal operators as "syntactic sugar" of the basic language: `hold-during` and `hold-after`.

Trajectory constraints are specified in the planning problem file in a new field. In addition, we allow constraints to be specified in the action domain file on the grounds that some constraints might be seen as safety conditions, or operating conditions, that are not physical limitations, but are nevertheless constraints that must always be respected in any valid plan for the domain (say legal constraints or operating procedures that must be respected).

Note that none of the new temporal modal operators is allowed in preconditions of actions. That is, all action preconditions are with respect to a state (or time interval, in the case of `over all` action conditions). In effect, an action can only rely on "local" conditions at the point of execution — the state in which it is applied and the states that hold during its duration — never on conditions in the trajectory as a whole.

The specific BNF grammar of PDDL3.0 is given in (Gerevini & Long 2005a). The following is a fragment of the grammar concerning the new modalities of PDDL3.0 for expressing constraints (`con-GD`):

```
<con-GD> ::= (at end <GD>) | (always <GD>) |
             (sometime <GD>) | (within <num> <GD>) |
             (at-most-once <GD>) |
             (sometime-after <GD> <GD>) |
             (sometime-before <GD> <GD>) |
             (always-within <num> <GD> <GD>) |
             (hold-during <num> <num> <GD> |
             (hold-after <num> <GD> | ...
```

where `<GD>` is a goal description (a first order logic formula), `<num>` is any numeric literal (in STRIPS domains it will be restricted to integer values). There is a minor complication in the interpretation of the bound for `within` and `always-within` when considering STRIPS plans (and similarly for `hold-during` and `hold-after`): the question is whether the bound refers to sequential steps (in other words, actions) or to parallel steps. For STRIPS plans, the numeric bounds will be counted in terms of plan *happenings*. For instance, (`within 10` ϕ) would mean that ϕ must hold within 10 happenings. These would be happenings of one action or of multiple actions, depending on whether the plan is sequential or parallel.

2.2 Semantics

The semantics of goal descriptors in PDDL2.2 evaluates them only in the context of a single state (the state of application for action preconditions or conditional effects and the final state for top level goals). In order to give meaning to temporal modalities, which assert properties of trajectories rather than individual states, it is necessary to extend the semantics to support interpretation with respect to a finite trajectory (as it is generated by a plan). We propose a semantics for the modal operators that is the same basic interpretation as is used in TLPlan (Bacchus & Kabanza 2000)

$$
\begin{array}{lll}
\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{at end}\ \phi) & \text{iff} & S_n \models \phi \\
\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models \phi & \text{iff} & S_n \models \phi \\
\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{always}\ \phi) & \text{iff} & \forall i : 0 \leq i \leq n \cdot S_i \models \phi \\
\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{sometime}\ \phi) & \text{iff} & \exists i : 0 \leq i \leq n \cdot S_j \models \phi \\
\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{within}\ t\ \phi) & \text{iff} & \exists i : 0 \leq i \leq n \cdot S_i \models \phi \text{and } t_i \leq t \\
\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{at-most-once}\ \phi) & \text{iff} & \forall i : 0 \leq i \leq n \cdot \text{ if } S_i \models \phi \text{ then} \\
 & & \exists j : j \geq i \cdot \forall k : i \leq k \leq j \cdot S_k \models \phi \\
 & & \quad \text{and } \forall k : k > j \cdot S_k \models \neg\phi \\
\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{sometime-after}\ \phi\ \psi) & \text{iff} & \forall i \cdot \text{if } S_i \models \phi \text{ then } \exists j : i \leq j \leq n \cdot S_j \models \psi \\
\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{sometime-before}\ \phi\ \psi) & \text{iff} & \forall i \cdot \text{if } S_i \models \phi \text{ then } \exists j : 0 \leq j < i \cdot S_j \models \psi \\
\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{always-within}\ t\ \phi\ \psi) & \text{iff} & \forall i \cdot \text{if } S_i \models \phi \text{ then } \exists j : i \leq j \leq n \cdot S_j \models \psi \\
 & & \text{and } t_j - t_i \leq t
\end{array}
$$

Figure 1: Semantics of the basic modal operators in PDDL3.

for $\mathcal{LT}$ and other standard LTL treatments. Recall that a *happening* in a plan for a PDDL domain is the collection of all effects associated with the (start or end points of) actions that occur at the same time. This time is then the time of the happening and a happening can be "applied" to a state by simultaneously applying all effects in the happening (which is well defined because no pair of such effects may be mutex).

Definition 1 *Given a domain D, a plan π and an initial state I, π generates the trajectory*

$$\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle$$

iff $S_0 = I$ and for each happening h generated by π, with h at time t, there is some i such that $t_i = t$ and S_i is the result of applying the happening h to S_{i-1}, and for every $j \in \{1 \ldots n\}$ there is a happening in π at t_j.

Definition 2 *Given a domain D, a plan π, an initial state I, and a goal G, π is valid if the trajectory it generates, $\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle$, satisfies the goal: $\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models G$.*

This definition contrasts with the original semantics of goal satisfaction, where the requirement was that $S_n \models G$. The contrast reflects precisely this requirement that goals should now be interpreted with respect to an entire trajectory. We do not allow action preconditions to use modal operators and therefore their interpretation continues to be relative to the single state in which the action is applied. The interpretation of simple formulae, ϕ (containing no modalities), in a single state S continues to be as before and continues to be denoted $S \models \phi$. In the following definition we rely on context to make clear where we are using the interpretation of non-modal formulae in single states, and where we are interpreting modal formulae in trajectories.

Definition 3 *Let ϕ and ψ be atomic formulae over the predicates of the planning problem plus equality (between objects or numeric terms) and inequalities between numeric terms, and let t be any real constant value. The interpretation of the modal operators is as specified in Figure 1.*

Note that this interpretation exploits the fact that modal operators are not nested. A more general semantics for nested modalities is a straight-forward extension of this one. Note also that the last four expressions in Figure 1 are expressible in different ways if one allows nesting of modalities and use of the standard LTL modality until (more details on this in (Gerevini & Long 2005b)).

The constraint `at-most-once` is satisfied if its argument becomes true and then stays true across multiple states and then (possibly) becomes false and stays false. Thus, there is only at most one *interval* in the plan over which the argument proposition is true.

For general formulae (which may or may not contain modalities):

$\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{and}\ \phi_1 ... \phi_n)$ iff, for every i, $\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models \phi_i$

and similarly for other connectives.

Of the constraints `hold-during` and `hold-after`, `(hold-during` $t_1\ t_2\ \phi$`)` states that ϕ must be true during the interval $[t_1, t_2)$, while `(hold-after` $t\ \phi$`)` states that ϕ must be true after time t. The first can be expressed by using timed initial literals to specify that a dummy timed literal `d` is true during the time window $[t_1, t_2)$ together with the goal `(always (implies d` ϕ`))`.
A variant of `hold-during` where ϕ must hold *exactly* during the specified interval could be easily obtained in a similar way. The second can be expressed by using timed initial literals to specify that `d` is true only from time t, together with the goal `(sometime-after d` ϕ`)`.

3 Soft Constraints and Preferences

A soft constraint is a condition on the trajectory generated by a plan that the user would prefer to see satisfied rather than not satisfied, but is prepared to accept might not be satisfied because of the cost of satisfying it, or because of conflicts with other constraints or goals. In case a user has multiple soft constraints, there is a need to determine which of the various constraints should take priority if there is a conflict between them or if it should prove costly to satisfy them. This could be expressed using a qualitative approach but, following careful deliberations, we have chosen to adopt a simple quantitative approach for this version of PDDL.

3.1 Syntax and Intended Meaning

The syntax for soft constraints falls into two parts. Firstly, there is the identification of the soft constraints, and secondly there is the description of how the satisfaction, or lack of it, of these constraints affects the quality of a plan.

Goal conditions, including action preconditions, can be labelled as preferences, meaning that they do not have to be true in order to achieve the corresponding goal or precondition. Thus, the semantics of these conditions is simple, as far as the correctness of plans is concerned: they are all trivially satisfied in any state. The role of these preferences is apparent when we consider the relative quality of different plans. In general, we consider plans better when they satisfy soft constraints and worse when they do not. A complication arises, however, when comparing two plans that satisfy different subsets of constraints (where neither set strictly contains the other). In this case, we rely on a specification of the violation costs associated with the preferences.

The syntax for labelling preferences is simple:

`(preference [name] <GD>)`.

The definition of a goal description can be extended to include preference expressions. However, in PDDL3.0, we reject as syntactically invalid any expression in which preferences appear nested inside any connectives, or modalities, other than conjunction and universal quantifiers. We also consider it a syntax violation if a preference appears in the condition of a conditional effect. *Note that where a named preference appears inside a universal quantifier, it is considered to be equivalent to a conjunction (over all legal instantiations of the quantified variable) of preferences all with the same name.*

Where a name is selected for a preference it can be used to refer to the preference in the construction of penalties for the violated constraint. The same name can be shared between preferences, in which case they share the same penalty.

Penalties for violation of preferences are calculated using the expression

`(is-violated <name>)`

where `<name>` is a name associated with one or more preferences. This expression takes on a value equal to the number of distinct preferences with the given name that are not satisfied in the plan. Note that in PDDL3.0 we do not attempt to distinguish degrees of satisfaction of a soft constraint — we are only concerned with whether or not the constraint is satisfied. Note, too, that the count includes each separate constraint with the same name. This means that:

```
(preference VisitParis
    (forall (?x - tourist)
            (sometime (at ?x Paris))))
```

yields a violation count of 1 for `(is-violated VisitParis)`, if at least one tourist fails to visit Paris during a plan, while

```
(forall (?x - tourist)
    (preference VisitParis
        (sometime (at ?x Paris))))
```

yields a violation count equal to the number of people who failed to visit Paris during the plan. The intention behind this is that each preference is considered to be a distinct preference, satisfied or not independently of other preferences. The naming of preferences is a convenience to allow different penalties to be associated with violation of different constraints.

Plans are awarded a value through the plan metric, introduced in PDDL2.1 (Fox & Long 2003). The constraints can be used in weighted expressions in a metric. For example,

```
(:metric minimize
     (+ (* 10 (fuel-used))
        (is-violated VisitParis)))
```

would weight fuel use as ten times more significant than violations of the `VisitParis` constraint. Note that the violation of a preference in the preconditions of an action is counted multiple times, depending on the number of the action occurrences in the plan. For instance, suppose that `p` is a preference in the precondition of an action a, which occurs three times in plan π. If the plan metric evaluating π contains the term `(* k (is-violated p))`, then this is interpreted as if it were `(* v (* k (is-violated p)))`, where `v` is the number of separate occurrences of a in π for which the preference is not satisfied.

3.2 Semantics

We say that

$\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models (\texttt{preference}\ \Phi)$

is always true, so this allows preference statements to be combined in formulae expressing goals. The point in making the formula always true is that the preference is a soft constraint, so failure to satisfy it is not considered to falsify the goal formula. In the context of action preconditions, we say $S_i \models (\texttt{preference}\ \Phi)$ is always true, too, for the same reasons.

We also say that a preference `(preference` Φ`)` is *satisfied* iff $\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models \Phi$ and *violated* otherwise. This means that `(or` Φ `(preference` Ψ`))` is the same as `(preference (or` Φ Ψ`))`, both in terms of the satisfaction of the formulae and also in terms of whether the preference is satisfied. The same idea is applied to action precondition preferences. Hence, a goal such as:

```
(and (at package1 london)
     (preference (clean truck1)))
```

would lead to the following interpretation:

$\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models$

```
(and (at package1 london)
     (preference (clean truck1)
```

iff

$\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models$

```
(at package1 london)
```

and

$\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models$

```
(preference (clean truck1))
```

iff $S_n \models$ `(at package1 london)`

iff `(at package1 london)` $\in S_n$, since the preference is always interpreted as true. In addition, the preference would be *satisfied* iff:

$\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle \models$

```
(at end (clean truck1))
```

iff `(clean truck1)` $\in S_n$.

If the preference is not satisfied, it is violated.

Now suppose that we have the following preferences and plan metric:

```
(preference p1 (always (clean truck1)))
(preference p2 (and (at end (at package2 paris))
                    (sometime (clean track1))))
(preference p3 (...))
(:metric (+ (* 10 (is-violated p1)) (* 5 (is-violated p2))
            (is-violated p3))).
```

Suppose we have two plans, π_1, π_2, and π_1 does not satisfy preferences p1 and p3 (but it satisfies preference p2) and π_2 does not satisfy preferences p2 and p3 (but it satisfies preference p1), then the metric for π_1 would yield a value (11) that is higher than that for π_2 (6) and we would say that π_2 is better than π_1.

Formally, *a preference precondition is satisfied if the state in which the corresponding action is applied satisfies the preference*. Note that the restriction on where preferences may appear in precondition formulae and goals, together with the fact that they are banned from conditional effects, means that this definition is sufficient: the context of their appearance will never make it ambiguous whether it is necessary to determine the status of a preference. Similarly, *a goal preference is satisfied if the proposition it contains is satisfied in the final state*. Finally, *an invariant (*`overall`*) condition of a durative action is satisfied if the corresponding proposition is true throughout the duration of the action*.

In some case, it can be hard to combine preferences with an appropriate weighting to achieve the intended balance between soft constraints and other factors that contribute to the value of a plan (such as plan make span, resource consumption and so on). For example, to ensure that a constraint takes priority over a plan cost associated with resource consumption (such as make span or fuel consumption) is particularly tricky: a constraint must be weighted with a value that is higher than any possible consumption cost and this might not be possible to determine. With non-linear functions it is possible to achieve a bounded behaviour for costs associated with resources. For example, if a constraint, C, is to be considered always to have greater importance than the make span for the plan then a metric could be defined as follows:

```
(:metric minimize (+ (is-violated C)
                     (- 1 (/ 1 (total-time))))).
```

This metric will always prefer a plan that satisfies C, but will use make span to break ties.

Nevertheless, for the competition, where it is important to provide an unambiguous specification by which to rank plans, the use of plan metrics in this way is clearly very straightforward and convenient. We leave for later proposals the possibilities for extending the evaluation of plans in the face of soft constraints.

4 Some Examples

In this section, we give some examples from well known domains and also from one of the PDDL3 domains that have recently been developed by the organizers of IPC-5: the Travelling and Purchase Problem (TPP).[1]

[1]For IPC-5 several new domains involving preferences have been defined. A detailed description of them is outside the scope of this paper.

4.1 Blocks World and Rovers

The following state trajectory constraints could be stated either as strong constraints or soft constraints.

"A fragile block can never have something above it":

```
(always (forall (?b - block)
          (implies (fragile ?b) (clear ?b))))
```

"A fragile block can have at most one block on it":

```
(always (forall (?b1 ?b2 - block)
          (implies (and (fragile ?b1) (on ?b2 ?b1))
                   (clear ?b2))))
```

"The blocks forming the same tower always have the same color":

```
(always (forall (?b1 ?b2 - block ?c1 ?c2 - color)
          (implies (and (on ?b1 ?b2) (color ?b1 ?c1)
                        (color ?b2 ?c2))
                   (= ?c1 ?c2))))
```

"Each block should be picked up *at least* once":

```
(forall (?b - block) (sometime (holding ?b)))
```

"Each block should be picked up *at most* once":

```
(forall (?b - block) (at-most-once (holding ?b)))
```

"In some state visited by the plan all blocks should be on the table":

```
(sometime (forall (?b - block) (on-table ?b)))
```

This constraint requires all the blocks to be on the table in the *same* state. In contrast, if we only require that every block should be on the table in *some* state we can write:

```
(forall (?b - block) (sometime (on-table ?b)))
```

The following two examples use the IPC-3 Rovers domain involving numerical fluents. "We would like that the energy of every rover should always be above the threshold of 5 units":

```
(always (forall (?r - rover) (> (energy ?r) 5)))
```

"Whenever the energy of a rover is below 5, it should be at the recharging location within 10 time units":

```
(forall (?r - rover)
   (always-within 10 (< (energy ?r) 5)
                     (at ?r recharging-point)))
```

4.2 TPP

TPP is a relatively recent planning domain that has been investigating in Operation Research (OR) for several years. TPP is a known generalization of the Travelling Salesman Problem, and is defined as follows. We have a set of different types of goods and a set of markets. Each market is provided with a limited amount of each type of goods at a known price. The TPP consists in selecting a subset of markets such that a given demand of each type of goods can be purchased, minimizing the routing cost and the purchasing cost. This problem arises in several applications, mainly in routing and scheduling contexts, and it is known to be NP-hard. In OR, computing optimal or near optimal solutions for TPP instances is an active research topic.

For IPC-5, several variants of this domain have been formalized in PDDL3.0. One of them is equivalent to the original TPP, while the others are simplified or extended formulations.[2]

[2]A description of each of these variants is available from the website of IPC-5: `http://ipc5.ing.unibs.it`.

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

(:action load
 :parameters (?g - goods ?t - truck ?m - market ?l1 ?l2 ?l3 ?l4 - level)
 :precondition (and (at ?t ?m) (loaded ?g ?t ?l3) (ready-to-load ?g ?m ?l2) (next ?l2 ?l1) (next ?l4 ?l3))
 :effect (and (loaded ?g ?t ?l4) (not (loaded ?g ?t ?l3)) (ready-to-load ?g ?m ?l1) (not (ready-to-load ?g ?m ?l2))))

(:action unload
 :parameters (?g - goods ?t - truck ?d - depot ?l1 ?l2 ?l3 ?l4 - level)
 :precondition (and (at ?t ?d) (loaded ?g ?t ?l2) (stored ?g ?l3) (next ?l2 ?l1) (next ?l4 ?l3))
 :effect (and (loaded ?g ?t ?l1) (not (loaded ?g ?t ?l2)) (stored ?g ?l4) (not (stored ?g ?l3))))

(:action buy
 :parameters (?t - truck ?g - goods ?m - market ?l1 ?l2 ?l3 ?l4 - level)
 :precondition (and (at ?t ?m) (on-sale ?g ?m ?l2) (ready-to-load ?g ?m ?l3) (next ?l2 ?l1) (next ?l4 ?l3))
 :effect (and (on-sale ?g ?m ?l1) (not (on-sale ?g ?m ?l2)) (ready-to-load ?g ?m ?l4) (not (ready-to-load ?g ?m ?l3))))
```

Figure 2: Propositional version with preferences of the TPP domain developed for IPC-5.

Examples from the Propositional Version of TPP

Figure 2 shows the operators of the "Propositional" version of TPP (with preferences) developed for IPC-5. This is a simplified version of the original TPP, where the amounts of goods that we can buy are discrete and are modeled by a certain number of qualitative levels, that are specified in the problem initial state. Moreover, goods have no price. Note that each `drive` action has a soft precondition

```
(preference p-drive (forall (?g - goods)
                      (ready-to-load ?g ?from level0))))
```

expressing the preference that "a truck can move from a market only if it leaves at that market no amount of (`ready-to-load`) goods". The goods that are ready to load are those that have been purchased at the corresponding market, so this implies that it is prefered that a truck should never leave a location in which there are still purchased goods.

In the following, we illustrate several preferences over goals and state trajectory constraints that are included in a IPC-5 test problem for the propositional version of TPP with preferences The first three sets of goal preferences, together with their penalty weights (see below), encode the more global preference of "maximising the level of purchased goods that are stored in a depot", for each type of goods specified in the initial state (in this problem, goods have four levels):

```
(forall (?g - goods)
   (preference G1 (exists (?l - level)
       (and (not (= ?l level0)) (not (= ?l level1))
            (stored ?g ?l)))))

(forall (?g - goods)
   (preference G2 (exists (?l - level)
       (and (not (= ?l level0)) (not (= ?l level2))
            (stored ?g ?l)))))

(forall (?g - goods)
   (preference G3 (exists (?l - level)
       (and (not (= ?l level0)) (not (= ?l level3))
            (stored ?g ?l)))))
```

Moreover, we prefer that "everything we buy is then stored in a depot", that is, that the level of the goods that we have bought (that are `ready-to-load`), and that have been left at a market or on a truck, is zero:

```
(forall (?g - goods)
   (preference G4
       (and (forall (?m - market)
                    (ready-to-load ?g ?m level0))
            (forall (?t - truck) (loaded ?g ?t level0)))))
```

The soft state trajectory constraints in TPP Propositional are the following ones. "Each market should be visited at most once by a truck":

```
(forall (?m - market ?t - truck)
   (preference C1 (at-most-once (at ?t ?m))))
```

"Each type of goods should be loaded at most once in a truck" (we want to buy and load the whole amount of the goods before storing them in a depot).

```
(forall (?t - truck ?g - goods)
   (preference C2 (at-most-once (exists (?l - level)
                       (and (loaded ?g ?t ?l)
                            (not (= ?l level0)))))))
```

"There should be at most one truck at a market at the same time":

```
(forall (?m - market ?t1 ?t2 - truck)
   (preference C3 (always (imply (and (at ?t1 ?m)
                                      (at ?t2 ?m))
                                 (= ?t1 ?t2)))))
```

"Each truck should be used":

```
(forall (?t - truck)
   (preference C4 (sometime (exists (?g - goods ?l - level)
                       (and (loaded ?g ?t ?l)
                            (not (= ?l level0)))))))
```

"A particular type of goods (`goods5`) should be stored at some level before another particular type of goods (`goods4` or `goods3`) is stored at that same level" (in other words, the level of `goods5` should always be greater than or equal to the level of `goods3` and `goods4`):

```
(forall (?l - level)
   (preference C5 (sometime-before
                    (and (stored goods4 ?l) (not (= (?l 0)))
                         (stored goods5 ?l))))

(forall (?l - level)
   (preference C6 (sometime-before
                    (and (stored goods3 ?l) (not (= (?l 0)))
                         (stored goods5 ?l))))
```

The following is an example of plan metric for the previous preferences. Note that preference `p-drive` is a soft precondition of each `drive` action appearing in the plan:

```
(:metric minimize
  (+ (* 1 (is-violated p-drive))
     (* 12 (is-violated G1)) (* 10 (is-violated G2))
     (* 8 (is-violated G3)) (* 4 (is-violated G4))
     (* 1 (is-violated C1)) (* 1 (is-violated C2))
     (* 2 (is-violated C3)) (* 3 (is-violated C4))
     (* 13 (is-violated C5)) (* 13 (is-violated C6))))
```

Assuming that in the initial state there are four possible levels for each type of goods, the decreasing penalty weights associated with goal preferences `G1`–`G3` encode the desire that we maximise the level of stored goods (for each type of goods, there is no penalty if we have four levels of stored goods, and the higher is the level of the stored goods, the less is the penalty we get).

In the particular TPP problem which the plan metric above is associated with, in general the way we achieve the goals is less important than achieving them. Thus, the penalty for violating the preferences over trajectory constraints (`C1`–`C4`) and action preconditions (`p-drive`) are lower than the penalties for violating soft goals. Preferences `C5` and `C6` are exceptions and have the highest penalty weights. This is because in this specific problem it is very important to keep a balance between the pair of goods involved by these constraints (that is, the levels of `goods3` and `goods4` should never exceed the level of `goods5`).

Finally, note that constraints `C5` and `C6` may interfere with goal preferences `G1`–`G3`, because the maximum levels of available goods at the markets may be different for different types of goods (as specified in the problem initial state). For example, if the maximum level for `goods5` is 2, then constraints `C5` and `C6` impose that `goods3` and `goods4` can never exceed this level (even if the an extra level of these goods could be purchased).

Examples from the Metric-Time Version of TPP

The Metric-Time version of TPP is significantly more complex than the Propositional one, and it is similar to the original formulation of TPP, with some extensions. The hard numeric goals are that the amounts of the stored goods are not less than the corresponding amounts of requested goods, which are specified in the initial state.

Concerning the goal preferences and soft constraints for this domain version, most of them are similar to some of the preferences for the propositional version of TPP, except that here we use numerical fluents for expressing the amounts of goods. Examples of additional soft constraint are
"whenever `goods3` are loaded in a truck, `goods3` should be in a depot within 2390 time units" (because, for instance, these goods get deteriorated by longer travels):

```
(forall (?t - truck)
   (preference C7 (always-within 2390
      (>= (loaded goods3 ?t) 0) (= (loaded goods3 ?t) 0))))
```

"The amount of certain goods stored in a depot should always be less than the amount of other particular goods":

```
(preference C8 (always
                 (> (stored goods13) (stored goods6))))
```

"We start storing some particular goods only after we have stored the requested amount of other particular goods":

```
(preference C9 (sometime-before (> (stored goods12) 0)
                 (>= (stored goods2) (request goods2))))
```

"We start storing some particular goods only after we have bought the requested amount of some other particular goods":

```
(preference C10 (sometime-before (> (stored goods10) 0)
                 (>= (bought goods8) (request goods8)))).
```

5 Extensions and Generalization

There is considerable scope for developing the proposed extension. First, and most obviously, modal operators could be allowed to nest. This would allow a rich expressive power in the specification of modal temporal goals. Nesting would allow constraints to be applied to parts of trajectories, as is usual in modal temporal logics. In addition, we could introduce propositions representing that an action appears in a plan.

Other modal operators could be added. We have excluded them PDDL3.0 because we have found that many interesting and challenging goals can be captured without them, and we do not wish to add unnecessarily to the load on potential competitors. The modal operator until would be an obvious one to add. Without nesting, a related always-until and sometime-until would allow expression of goals such as "every time a truck arrives at the depot, it must stay there until loaded" or "when the truck arrives at the depot, it must stay there until cleaned and fully refuelled at least once in the plan". The formal semantics of always-until and sometime-until can be easily derived from the one of until in LTL. By combining always-until and other modalities we can express complex constraints such as that "whenever the energy of a rover is below 5, it should be at the recharging location within 10 time units and remain there until recharged":

```
(and (always-until (charged ?r) (at ?r rechargepoint))
     (always-within 10 (< (charge ?r) 5)
                       (at ?r rechargingpoint)))
```

Another modality that would be an useful extension of the expressive power is a complement for `within`, such as persist, with the semantics that a proposition once made true must persist for at least some minimal period of time. Without nesting, a related always-persist and sometime-persist would allow expression of goals such as "I want to spend at least 2 days in each of the cities on my tour", or "every time the taxi goes to the station it must wait for at least 10 without a passenger".
The formal semantics of always-persist and sometime-persist is given in Figure 3. A generalisation that would allow `within` and persist to be combined would be to allow the time specification to be associated with a comparison operator to indicate whether the bound is an upper or lower bound.

$\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{always-persist}\ t\ \phi)$ iff $\forall i : 0 < i \leq n \cdot$ if $S_i \models \phi$ and $S_{i-1} \models \neg\phi$ then $\exists j : j - i \geq t \cdot \forall z : i \leq z \leq j \cdot S_z \models \phi$ and if $S_0 \models \phi$ then $\forall z : z \leq t \cdot S_z \models \phi$

$\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle \models (\texttt{always-persist}\ t\ \phi)$ iff $\exists i : 0 < i \leq n \cdot$ if $S_i \models \phi$ and $S_{i-1} \models \neg\phi$ then $\exists j : j - i \geq t \cdot \forall z : i \leq z \leq j \cdot S_z \models \phi$, or if $S_0 \models \phi$ then $\forall z : z \leq t \cdot S_z \models \phi$

Figure 3: Semantics of always-persist and sometime-persist.

We have deliberately not introduced the operator next, which is common in modal temporal logics. This is because concurrent fragments of a plan might cause a state change that is not relevant to the part of the state in which the next condition is intended to apply. Furthermore, the fact that PDDL plans are embedded on a real time line means that the intention behind next is less obviously relevant. We realise that next has been particularly useful in expressing control rules for planners like TALPlanner (Kvarnström & Magnusson 2003) and TLPlan (Bacchus & Kabanza 2000), but our intention in developing this extension is to focus on providing a language that is useful for expressing constraints that govern plan quality, rather than for control knowledge. We believe that the use of `always-within` captures a much more useful concept for plan quality that is actually a far more realistic constraint in modelling planning problems.

Extensions to the use of soft constraints include the definition of more complex preferences, such as conditional preferences, and a possible qualitative method for expressing priorities over preferences. Moreover, the evaluation of the soft constraints could be extended by considering a degree of constraint violation, such as the amount of time when an `always` constraint is violated, the delay that falsifies a `within` constraint, or the number of times an `always-after` constraint is violated.

6 Conclusions

Planning has been tackling increasingly difficult problems with greater success over recent years. An objective for the community is to move the focus of research towards the solution of problems with increasing relevance to application. In many application areas, the quality of plans is central to their usefulness. It is essential to consider the quality in terms of constraints across the trajectories and in terms of preferences that are imposed by the users. To manage these problems, planning algorithms must have access to this information and we have proposed an extension to PDDL that will provide this. The role of PDDL in forming a common foundation for the extension of existing planning technology has been proven repeatedly over the past 8 years. Although the concepts of constraints, both hard and soft, are not new, even to planning, the adoption of a common language and the basis for benchmarks will play a central role in promoting research into these areas. The interest in the 5th IPC is already a clear demonstration of the way in which the research agenda can be moved forward through the vehicle of PDDL.

Acknowledgments

We would like to thank Y. Dimopoulos, C. Domshlak, S. Edelkamp, M. Fox, P. Haslum, J. Hoffmann, A. Jonsson, D. McDermott, A. Saetti, L. Schubert, I.Serina, D. Smith and D. Weld for some very useful discussions about PDDL3.

References

Bacchus, F., and Kabanza, F. 2000. Using temporal logic to express search control knowledge for planning. *Artificial Intelligence* 116(1-2):123–191.

Bistarelli, S.; Montanari, U.; and Rossi, F. 1997. Semiring-based constraint solving and optimization. *Journal of ACM*.

Brafman, R., and Chernyavsky, Y. 2005. Planning with goal preferences and constraints. In *Proc. of ICAPS-05*.

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

Delgrande, P. J.; Schaub, T.; and Tompits, H. 2005. A general framework for expressing preferences in causal reasoning and planning. In *Proc. of the 7^{th} International Symposium on Logical Formalizations of Commonsense Reasoning*.

Dubois, D.; Fargier, H.; and Prade, H. 1996. Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty. *Applied Intelligence* 6:287–309.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classic part of the 4th international planning competition. Technical Report 195, Institut für Informatik, Freiburg, Germany.

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

Gerevini, A., and Long, D. 2005a. Bnf description of pddl3.0. Technical report, `http://ipc5.ing.unibs.it`.

Gerevini, A., and Long, D. 2005b. Plan constraints and preferences in PDDL3. Technical Report RT-2005-08-47, Departimento di Elettronica per l'Automazione, Universitá di Brescia.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - the planning domain definition language. Technical Report CVC TR98-003/DCS TR-1165.

Kvarnström, J., and Magnusson, M. 2003. Talplanner in the 3rd international planning competition: Extensions and control rules. *Journal of AI Research* 20.

Miguel, I.; Jarvis, P.; and Shen, Q. 2001. Efficient flexible planning via dynamic flexible constraint satisfaction. *Engineering Applications of Artificial Intelligence* 14(3):301–327.

Rossi, F.; Venable, K.; and Yorke-Smith, N. 2004. Controllability of soft temporal constraint problems. In *Proc. of CP*.

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

Son, T., C., and Pontelli, E. 2004. Planning with preferences using logic programming. In *Proc. of LPNMR-04*. Springer-Verlag. LNAI 2923.

Handling Soft Constraints and Goals Preferences in SGPlan*

Chih-Wei Hsu and Benjamin W. Wah
Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
Urbana, IL 61801, USA
{*chsu,wah*}*@manip.crhc.uiuc.edu*

Ruoyun Huang and Yixin Chen
Department of Computer Science and Engineering
Washington University in St Louis
St Louis, MO 63130, USA
rh11@cec.wustl.edu
chen@cse.wustl.edu

Abstract

In this paper, we present the partition-and-resolve strategy in SGPlan (hereafter called SGPLan_5) for fully supporting all language features in PDDL3.0. Based on the architecture of SGPlan that supported PDDL2.2 (hereafter called SGPLan_4), SGPLan_5 partitions a large planning problem into subproblems, each with its own subgoal, and resolves those inconsistent solutions using our extended saddle-point condition. Subgoal partitioning is effective for solving large planning problems because each partitioned subproblem involves a substantially smaller search space than that of the original problem. In SGPLan_5, we generalize subgoal partitioning so that the goal state of a subproblem is no longer one goal fact as in SGPLan_4, but can be any fact with loosely coupled constraints with other subproblems. We have further developed methods for representing a planning problem in a multi-valued form and for carrying out partitioning in the transformed space. The multi-valued representation leads to more efficient heuristics for resolving trajectory and temporal constraints and goal preferences.

INTRODUCTION

In this paper, we present the partition-and-resolve strategy in SGPLan_5 for fully supporting all language features in PDDL3.0 (Gerevini & Long 2005). By extending the architecture of SGPLan_4 (Chen, Wah, & Hsu 2006) that supports PDDL2.2 (Edelkamp & Hoffmann 2004), SGPLan_5 partitions a large planning problem into subproblems, each with its own subgoal, and resolves those inconsistent solutions of subgoals using our extended saddle-point condition.

Inspired by real applications, Smith recently introduced the over-subscription planning problem (Smith 2004) that has a number of soft goals with different violation costs. Unlike PDDL2.2 domains whose goal state is a conjunctive list of facts, the planning task in an over-subscription planning problem entails the selection of an appropriate subset of soft goals when it is infeasible to achieve the entire set of goals. The idea has been extended in PDDL3.0 in such a way there are soft constraints over intermediate states.

*Research supported by the National Science Foundation Grant IIS 03-12084.

To develop a planner that fully supports the PDDL3.0 planning models and to accommodate its new features on trajectory constraints and goal preferences, we have transformed each planning problem into a multi-valued domain formulation and have revised our partitioning strategy in SGPLan_4 based on the new representation. We have developed new search techniques, both at the global and the subproblem levels, for optimizing goal preferences and for resolving trajectory and temporal constraints.

DESIGN GOALS

PDDL3.0 extends the previous PDDL2.2 specifications by introducing several new features: a) simple preferences over only action preconditions or goals, b) qualitative preferences that are logical preferences over trajectory constraints, c) complex constraints that are trajectory constraints with metric time and possibly numeric fluents, and d) complex preferences that are preferences over trajectory constraints with metric time and possibly numeric fluents. We have developed new components in SGPLan_5 to support these features.

Given a plan π, an initial state I, a sequence of actions and possibly their schedule, we can derive the trajectory of π under the domain definition. We can compute for this trajectory its violated constraints, which include mutex constraints as in PDDL2.2, inconsistent state-variable assignments, and trajectory constraints introduced in PDDL3.0. The objective is to satisfy all the hard constraints or goals as well as to optimize the soft-constraint violations and the plan quality.

SGPLan_5 uses a *multi-valued domain formulation (MDF)* based on the SAS+ formalism. MDF has been used in several planners, including Fast Downward (Helmert 2004) and the IP planner (van den Briel, Vossen, & Kambhampati 2005). Its advantage is that it allows a more compact representation of facts and their dependencies. For example, in the traditional representation of binary facts, the location of *truck1* in the *TPP* domain is represented in binary facts, such as *at(truck1, location1)*, ..., *at(truck1, location8)*. These facts can be denoted more compactly in MDF by one variable *location(truck1)* that takes multiple values: *location1*, ..., *location8*. The MDF variables further allows us to derive the possible transitions among their values. For instance, *location1* $\longrightarrow$ *location2* represents the connection between two locations.

Based on the MDF formulation, we can derive causal

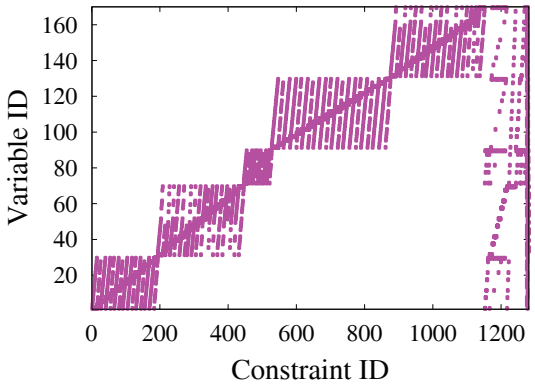

Figure 1: The locality of the constraint-variable structure in the fifth instance of the *TPP-SimplePreferences* domain.

dependencies among the variables, where variable a has a causal dependency on another variable b if a transition in a has a state requirement on b. For example, in order for *truck1* to move *crate* from *location1* to *location2*, it has to be at *location1*. In this case, the transition for *crate* has a causal dependency on the location of *truck1*.

We have implemented our preprocessing engine for translating a PDDL3.0 problem into MDF. There are several reasons for using MDF.

a) It provides a more compact representation than a binary-valued representation and leads to a more effective partitioning of the constraints.

b) MDF facilitates the analysis of the transition graphs of variables and causal dependencies among the variables. These can be used to derive a much more accurate heuristic guidance than the previous Metric-FF heuristic.

c) Using the new heuristic function, high-quality approximate plans can be extracted for efficiently resolving temporal constraints in PDDL3.0 planning problems.

CONSTRAINT LOCALITY

In PDDL2.2 planning, we have shown that constraint partitioning by subgoals leads to localized constraints when we partition the constraints in such a way that a majority of them are within a partition. This approach, however, cannot be directly applied in PDDL3.0 domains because there are some soft goals that are never present in an optimal solution. Constraint locality will also be different when problems are represented in MDF.

Because it is possible to have inconsistencies among soft constraints, our previous subgoal partitioning strategy that aims to satisfy a conjunctive list of conditions on the final state cannot be applied directly. To address this issue, we have extended our partitioning approach. Using multi-valued domain analysis, we first eliminate a number of mutual exclusions as well as inconsistencies among the soft constraints. Moreover, we have found that constraint locality is associated with some state variables with causal dependencies to other state variables. Conceptually, these

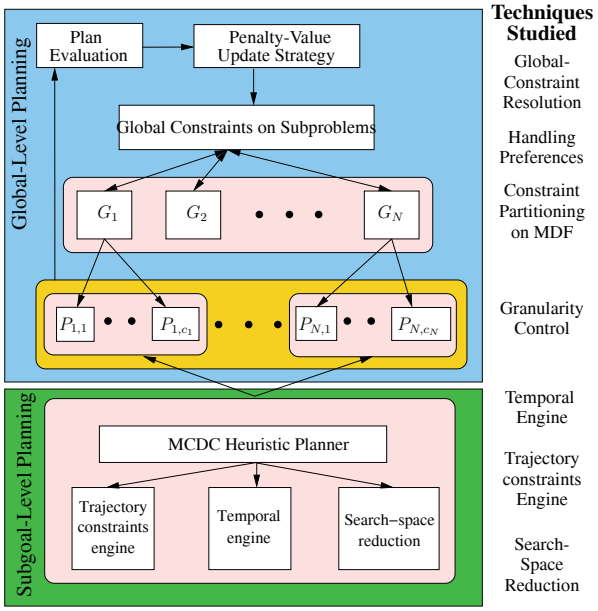

Figure 2: Architecture of SGPLan$_5$.

variables can be treated as goals because their transitions incur changes on other variables. As a result, we partition the problem by these variables and formulate global constraints as those that involve variables across different partitions.

We have evaluated our revised partitioning method on some IPC5 benchmarks. Figure 1 illustrates the constraint-variable structure in the fifth instance of the *TPP-SimplePreferences* domain that aims to plan the purchase and storage of five products. In this problem, each variable represents an action and its schedule in the plan, and a constraint can be an active mutual exclusion, an inconsistent state variable assignment, or a violated trajectory constraint. By partitioning those variables that represent the status of products (five in this instance) into five subproblems, each involving one product, it is obvious that a majority of the constraints can be localized. In this form, a global constraint involves variables that are across two partitions, such as a constraint on the equality of the quantity of two products.

Note that the difficulty of resolving constraints is domain dependent, although constraint locality is common in all IPC5 benchmarks. For instance, all the subproblems in the *OpenStacks* domain are trivial to solve, but the major challenge is to enforce the consistency of its shared variables.

ARCHITECTURE OF SGPLan$_5$

By formulating a subproblem in such a way that each has one goal state, SGPLan$_5$ partitions a planning problem into subproblems and finds a feasible plan for each goal fact (Figure 2). In the global level, it partitions the problem by its multi-valued state variables and resolves its violated global constraints using the theory of extended saddle points (Wah & Chen 2006). In the local level, it calls a basic planner for solving each partitioned subproblem, using the violated global constraints and the global preferences as biases.

Global-Level Search

Partitioning strategy. We have observed that the remaining constraints have a strong locality if we can first eliminate those that involve a large number of state variables. From the causal graph, we can extract those (low-level) state variables that influence many other state variables. Dependencies due to these variables would cause active mutual exclusions across subproblems, regardless of how the constraints are partitioned. On the other hand, there are (high-level) variables whose state transitions involve a set of low-level variables. Since constraint locality is associated with high-level state variables, we can formulate constraints that involve variables across partitions as global constraints. Also, we have chosen an optimal grain size that minimizes the number of shared variables in order to reduce the number of global constraints. For example, in the *TPP* domain, there will be a lot of active mutual exclusions if we use the same truck to generate two subplans for buying two products, whereas the number of mutual exclusions will be minimal when two trucks are used.

Resolution of global constraints. A planning problem solved by SGPLan$_5$ is defined in mixed space with a nonlinear objective and one or more constraints. By formulating a penalty function that consists of the sum of the objective and the transformed constraint functions weighted by penalties, SGPLan$_5$ implements a search to find extended saddle points (ESPs) of the penalty function (Wah & Chen 2006). Here, an ESP is a local minimum of the penalty function with respect to the original variables and a local maximum with respect to the penalties for all penalties larger than a threshold. The algorithm is based on the ESP condition (ESPC), which states the one-to-one correspondence between the ESPs and those feasible local optima in mixed space. To implement the ESPC, the search consists of two loops: an inner loop that looks for a local minimum of the penalty function and an outer loop that looks for any penalty value larger than the threshold. The ESPC also allows the search of ESPs to be partitioned into multiple searches, each looking for a local ESP in a subproblem, and an outer loop that resolves the inconsistencies among the subproblems.

A direct implementation of ESPC in a search algorithm may get stuck in an infeasible region when the objective is too small or when the penalty values and/or constraint violations are too large. To address this issue, SGPLan$_5$ performs backtracking to escape from infeasible local traps.

Handling local constraints. Since the *minimum causal dependency-cost* (MCDC) heuristic can generate a highly accurate approximate plan from a state, we can use it as a tight lower bound on the makespan when resolving temporal constraints. For temporal constraints in the form of deadlines, we prune any state whose MCDC value exceeds the deadlines. For trajectory constraints, we evaluate the approximate MCDC plan and add penalty for those that are violated in the approximate MCDC plan.

Handling preferences. There are three classes of methods developed for handling soft goals in a planning problem. The first class solves the problem by encoding it as an integer program, as in OptiPlan (van den Briel *et al.* 2004). The second class first heuristically selects a subset of goals and then applies an existing planner to achieve them (Nigenda & Kambhampati 2005). Last, the third class does not select a subset of goals upfront but treats soft goals as soft planning constraints (Benton, Do, & Kambhampati 2005). It then derives new heuristics and reachability analysis for each soft goal in order to guide the search to an optimal final state. The success of these approaches based on heuristic functions, however, depends on the assumption that either all the soft goals are independent or their interactions have been addressed in the heuristic function (Smith 2004).

We have classified all trajectory preferences into two categories. The first class of preferences consists of those soft constraints on the final state and the persistent soft constraints (model operator *always*). We consider them with the original goal definitions because they have temporal overlaps on the final state. Although it is not easy to find an optimal set of soft constraints to be satisfied, it is trivial to compute their violation cost, when given an assignment of all state variables involved in the goal preferences. Therefore, we enumerate all reachable elements of each state variable involved and choose an optimal combination of facts to achieve. These enumerations can be decomposed because those constraints on the final state also have strong localities. It is still possible to make an unreachable assignment, even though the MDF analysis can detect many implicit mutual exclusions. For those unreachable assignments, we perform backtracking to find alternative assignments. When the cost of the assignment (such as a weighted sum of preference violations and plan quality) is unknown until the end of planning, we also perform backtracking to find better solutions.

The second class of preferences are those with insufficient information on their satisfiability. This may happen because the related soft constraints are not always active. To address this issue, we have devised a *relax-and-tighten* strategy that ignores initially all those preferences belonging to the second class and that penalizes those unsatisfied preferences to generate a solution. As is done earlier in resolving constraints, we have developed a number of heuristics for estimating the reachability of preferences and have applied iterative refinements until no better solutions can be found.

Note that the penalties on constraints are independent of the preference weights. For those soft constraints in the first class, they are enforced by the basic planner because they are local constraints when we partition by state variables. On the other hand, we update the penalties of constraints in the second class based on their violation but not their weights. Because we rely on changes in penalty values in order to find a better solution in terms of the plan metrics, our strategy does not guarantee plan optimality.

Local-Level Basic Planner

Our basic planner follows the heuristic search algorithm used in Metric-FF (Hoffmann 2003), but employs a new heuristic based on MDF.

MCDC heuristic planner. Using MDF, we have implemented a new search heuristic by exploring the value transition graph of each variable and the causal dependencies between the transition graphs. The general idea is inspired by and similar to the heuristic used in the Fast Downward planner (Helmert 2004). However, our MCDC heuristic is very different from the Fast Downward heuristic in a number of aspects.

First, our MCDC heuristic employs a complete recursive depth-first search for generating a heuristic plan with the minimum cost without pruning the causal graphs. In contrast, the Fast Downward heuristic is incomplete since it performs strongly-connected-component analysis and removes nodes with low connectivity. We have found that our complete recursive search leads to much less node expansions during planning.

Second, we have developed a set of strong necessary conditions for pruning infeasible or dominated paths when searching for the best approximate plan. We have also developed an algorithm for detecting symmetric objects in a given state to further reduce the cost of evaluating the MCDC heuristic. These rules can help reduce the average computing time of our heuristic by one to two orders of magnitude.

Third, in addition to sequential propositional planning supported by the Fast Downward heuristic, MCDC supports parallel temporal planning and can generate estimates of makespans for temporal plans.

The heuristic plan found by MCDC is approximate because the transition graphs found are not complete and some of its actions may not be supported. Moreover, numerical and trajectory constraints are ignored in MCDC.

MCDC is not admissible because, when computing an approximate plan, we consider each subgoal individually and sum the costs of all subgoals in order to estimate the overall heuristic value. Thus, MCDC ignores the positive interactions among the subgoals and is not admissible.

Search-space reduction. Before solving a partitioned subproblem, we can often eliminate many irrelevant actions in its search space. We identify those relevant actions by traversing the causal graphs in MDF and by ignoring actions that are not useful for achieving the current subgoal state variables. We also prioritize actions that do not cause an inconsistent assignment of multi-valued state variables. This is done by following our partitioning setting to compute a set of local state variables for each subproblem, and by applying the helpful action idea introduced in FF (Hoffmann & Nebel 2001) in order to defer those actions that change the value of non-local state variables.

Preliminary Experimental Results

We have defined predicates and functions for capturing the violation and the corresponding cost of each constraint. We have modified the parser and the pre-processor of SGPLan$_4$ in such a way that all the constraints are grounded and all ADL features in their conditions are compiled away. We have also integrated all the techniques for handling soft constraints in the search engine of SGPLan$_5$, both at the top-level of the search and in the local-level basic planner. At this time, we have evaluated SGPLan$_5$ on four IPC5 benchmarks: *TPP*, *OPENSTACKS*, *TRUCKS*, and *STORAGE* and have obtained promising results and solutions with good quality. Further, for the *SimplePreferences* domain, if the objective value is only determined by the final state, then we can compute the optimal assignment of the final state by only considering those constraints on the final state and by ignoring all other constraints. Since the solution is the optimal assignment of a relaxed problem, it is an optimal solution to the original problem when the assignment is actually achieved.

References

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

Chen, Y. X.; Wah, B. W.; and Hsu, C. W. 2006. Temporal planning using subgoal partitioning and resolution in SGPlan. *J. of Artificial Intelligence Research*.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classic part of the 4th International Planning Competition. Technical report, Tech. Rep. 195, Institut für Informatik, Freiburg, Germany.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences for PDDL3. Technical report, R.T. 2005-08-07, Dept. of Electronics for Automation, U. of Brescia, Brescia, Italy.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *ICAPS*, 161–170.

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

Hoffmann, J. 2003. The Metric-FF planning system: Translating ignoring delete lists to numeric state variables. *J. of Artificial Intelligence Research* 20:291–341.

Nigenda, R. S., and Kambhampati, S. 2005. Planning graph heuristics for selecting objectives in oversubscription planning problems. In *ICAPS*, 192–201.

Smith, D. E. 2004. Choosing objectives in oversubscription planning. In *ICAPS*, 393–401.

van den Briel, M.; Nigenda, R. S.; Do, M. B.; and Kambhampati, S. 2004. Effective approaches for partial satisfaction (over-subscription) planning. In *AAAI*, 562–569.

van den Briel, M.; Vossen, T.; and Kambhampati, S. 2005. Reviving integer programming approaches for AI planning: A branch-and-cut framework. In *ICAPS*, 310–319.

Wah, B., and Chen, Y. X. 2006. Constraint partitioning in penalty formulations for solving temporal planning problems. *Artificial Intelligence* 170(3):187–231.

Temporal Preference Optimization as Weighted Constraint Satisfaction

Michael D. Moffitt and **Martha E. Pollack**
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109, USA
{mmoffitt, pollackm}@eecs.umich.edu

Abstract

We present a new efficient algorithm for obtaining utilitarian optimal solutions to Disjunctive Temporal Problems with Preferences (DTPPs). The previous state-of-the-art system achieves temporal preference optimization using a SAT formulation, with its creators attributing its performance to advances in SAT solving techniques. We depart from the SAT encoding and instead introduce the Valued DTP (VDTP). In contrast to the traditional semiring-based formalism that annotates legal tuples of a constraint with preferences, our framework instead assigns elementary costs to the constraints themselves. After proving that the VDTP can express the same set of utilitarian optimal solutions as the DTPP with piecewise-constant preference functions, we develop a method for achieving *weighted constraint satisfaction* within a meta-CSP search space that has traditionally been used to solve DTPs without preferences. This allows us to directly incorporate several powerful techniques developed in previous decision-based DTP literature. Finally, we present empirical results demonstrating that an implementation of our approach consistently outperforms the SAT-based solver by orders of magnitude.

Introduction

Several recent studies have addressed the topic of *preferential optimization* in constraint-based temporal reasoning (Khatib *et al.* 2001). In this line of research, traditional temporal constraints (Dechter, Meiri, & Pearl 1991) are augmented with local preference functions that express how well a particular assignment satisfies the corresponding constraint. For instance, these functions might convey that a certain activity should be as long as possible, or that it is desirable for a pair of activities to be scheduled very close to one another. 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 (Kumar 2004; Morris *et al.* 2004; Peintner & Pollack 2005), where the sum of the individual preference values is maximized. As researchers continue to incorporate temporal constraints into their planning systems (Fox & Coddington 2002; Schwartz & Pollack 2004; Gerevini, Saetti, & Serina 2005), it is likely that these ongoing efforts in the CP community will be used extensively when integrating preferences into the design of expressive languages and efficient algorithms for planning.

Recently, Sheini et al. (2005) introduced a highly efficient approach to finding utilitarian optimal solutions to Disjunctive Temporal Problems with Preferences (DTPPs), a powerful representation that subsumes many common temporal preference optimization problems. In their work, the logical structure of the DTPP is decomposed into a Mixed Logical Linear Program, where the Boolean literals are linked to simple temporal constraints. The resulting problem is solved by a system named ARIO, in which a top-level SAT solver invokes a special-purpose temporal constraint engine. Experimental results showed that ARIO is several orders of magnitude faster than alternative methods for solving DTPPs (Peintner 2005), and the creators of ARIO credit advances in SAT techniques as the principal reason for this success.

In this paper, we present a new efficient algorithm for obtaining utilitarian optimal solutions to Disjunctive Temporal Problems with Preferences. To facilitate our approach, we introduce the Valued DTP (VDTP). In contrast to the traditional semiring-based formalism that annotates legal tuples of a constraint with preferences, our framework instead assigns elementary costs to the constraints themselves. While this reformulation provides no increase in expressive power, it simplifies some of the computational difficulties related to temporal optimization, since search strategies for disjunctive temporal reasoning typically view constraints as meta-level variables. After proving that the VDTP can express the same set of utilitarian optimal solutions as the DTPP with piecewise-constant preference functions, we develop a method for achieving *weighted constraint satisfaction* within a meta-CSP search space that has traditionally been used to solve DTPs without preferences. This allows us to directly incorporate techniques developed in previous decision-based DTP literature in order to make preferential optimization particularly efficient. Finally, we present empirical results demonstrating that an implementation of our approach consistently outperforms the SAT-based solver by orders of magnitude.

Background

Disjunctive Temporal Problems A *Disjunctive Temporal Problem* (DTP) (Stergiou & Koubarakis 1998) is a constraint satisfaction problem 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: $c_{i1} \vee c_{i2} \vee ... \vee c_{in_i}$

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 $a_{ij}, b_{ij} \in \Re$ (we will refer to the interval $[a_{ij}, b_{ij}]$ as the *feasible region* for c_{ij}). DTPs are thus a generalization of Simple Temporal Problems (STPs), in which each constraint is limited to a single disjunct.

There are generally two ways of defining a solution to a DTP. 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 $\{c_{i1}, c_{i2}, ..., c_{in_i}\}$, representing the various disjuncts one can choose to satisfy that disjunctive constraint. A complete assignment in the meta-CSP thus involves a selection of a single disjunct for each constraint, commonly referred to as a *component STP*.

Within the meta-CSP formulation, the constraints are implicitly defined by the underlying semantics of the disjuncts: the values (disjuncts) assigned to each meta-variable must be mutually consistent. The consistency of a set $\mathcal{S}$ of such inequalities can be determined by first constructing its *distance graph*, a graph that includes a node for each time point and an arc with weight b from y to x whenever $x - y \leq b$ is in $\mathcal{S}$. Then $\mathcal{S}$ is consistent if and only if its distance graph contains no negative cycles, which can be determined in polynomial time by computing its all-pairs shortest path (APSP) matrix and checking the entries along the main diagonal (Dechter, Meiri, & Pearl 1991).

Disjunctive Temporal Problems with Preferences To extend a DTP to a DTP with Preferences (DTPP) (Peintner & Pollack 2004), each disjunct $c_{ij} : a_{ij} \leq x_{ij} - y_{ij} \leq b_{ij}$ is augmented with a preference function $\langle f_{ij} : t \in [a_{ij}, b_{ij}] \rightarrow \{0, \Re^+\} \rangle$ that maps every allowable temporal difference to a *preference value* expressing its relative utility (Khatib *et al.* 2001).[1] DTPPs subsume STPPs (Simple Temporal Problems with Preferences) in the same way that DTPs subsume STPs. Given a solution S to the DTPP D, the preference value of a disjunctive constraint C_i in C is defined to be the maximum value achieved by any of its disjuncts:

$$val_D(S, C_i) = \max_{c_{ij} \in D(C_i)} f_{ij}(x_{ij} - y_{ij})$$

With the addition of preferences, we are no longer concerned with simply finding a feasible solution; we also want a solution of high quality. This requires us to define an *objective function* with respect to each of the individual preference functions. In this paper, we will consider the useful *utilitarian* objective, where the global value of a solution S is equal to the sum of the preference values of the individual constraints (Morris *et al.* 2004):

$$val_D(S) = \sum_i val_D(S, C_i)$$

Solving DTPPs Two approaches have been developed in previous literature for performing utilitarian optimization of a DTPP. Both can handle problems containing complex preference functions, requiring only that they be piecewise-constant in shape.[2]

The first is based on a SAT reformulation of a DTPP (Sheini *et al.* 2005). It involves the creation of a Mixed Logical Linear Programming (MLLP) problem composed of two types of constraints: logical constraints over Boolean variables, and Unit-Two-Variable-Per-Inequality (UTVPI) integer constraints of the form $ax - by \leq d$, where $a, b \in \{-1, 0, 1\}$. The disjuncts in the DTPP are converted to a set of UTVPI constraints, a Boolean *indicator variable* is created for each constraint, and a SAT problem is constructed in which these indicator variables are used to represent the logical structure of the DTPP. The reformulated problem is then solved by system named ARIO, which is composed of a tightly integrated UTVPI engine and SAT solver. Since this approach can handle only the decision variant of the DTPP, optimization is achieved by repeatedly calling the combined constraint engine on a sequence of satisfaction problems with increasingly higher objectives until no feasible solution can be found. Efficient SAT-solving techniques (Moskewicz *et al.* 2001) make the approach taken by ARIO particularly attractive.

The second approach, named GAPD (*Greedy Anytime Partition algorithm for DTPPs*) was designed exclusively for the purpose of solving DTPPs (Peintner 2005). It is based largely on the GAPS algorithm (Peintner & Pollack 2005) for finding utilitarian optimal solutions to STPPs. It begins by first searching for a consistent component STP S to the DTP D induced by fixing all constraints in the DTPP at their bottommost preference level of 0. It then either uses the GAPS algorithm to find an optimal solution to the STPP S' corresponding to S, or computes another solution S'' to the DTP D, and repeats. In this way, the disjunctive search for feasible solutions is decoupled from the process of optimization. Unfortunately, the memory requirements of GAPS (and GAPD) are exponential in the size of the STPP (and DTPP). Furthermore, they have both been shown to be several orders of magnitude slower than ARIO for finding optimal solutions. However, as the names suggest, these algorithms have desirable anytime properties, and are indeed complete algorithms.

Valued Disjunctive Temporal Problems

The DTP with Preferences has typically been regarded as a type of *semiring* formulation (Bistarelli, Montanari, & Rossi 1997), in which preference values are attributed to the (infinitely many) legal object-level tuples that comprise a constraint. While being expressive, this modeling of preferences presents some computational challenges, since search strategies for disjunctive temporal reasoning operate on the meta-CSP rather than invoking object-level assignments directly. In response, we will introduce a variation of the DTP where the disjunctive constraints themselves are associated with costs, making our representation comparable to early versions of the Valued CSP formalism for finite-domain constraints (Schiex, Fargier, & Verfaillie 1995).

Definition: A *Valued Disjunctive Temporal Problem* is a tuple $\langle X, C, S, \varphi \rangle$, where X and C are as in a DTP, S is a

[1] Note that zero is the minimum preference value that can be obtained by an assignment that satisfies the constraint.

[2] DTPPs containing other preference function shapes can be approximated by piecewise-constant functions via discretization.

valuation structure $(E, \circledast, \succ)$, and φ is a mapping from C to E. $\square$

In the valuation structure, E is a set whose elements are called *valuations*, and are totally ordered by $\succ$. The symbol $\circledast$ denotes a commutative, associative closed binary operation on E, and expresses how to aggregate the individual valuations. The function φ simply maps each constraint C_i to an element of E, indicating how important it is for the constraint to be satisfied.

Rather than explore the entire class of problems that the Valued DTP can represent, we will focus our attention on a special *weighted* case, where $E = \Re^+ \cup \{\infty\}$ and $\circledast = +$ (i.e., arithmetic sum), using the usual ordering $<$. In other words, each constraint C_i is associated with a positive numeric weight (which we subsequently refer to as w_i instead of $\varphi(C_i)$ to improve readability), and the objective will be to find an assignment S that imposes the minimal cost, where the *cost* is defined to be the weighted sum of violated constraints in the VDTP D:

$$cost_D(S) = \sum_i \{w_i | violates(S, C_i)\}$$

As an example, we present the following (very small) instance of our weighted VDTP:

C_1: $\{c_{11}: 1 \leq x - y \leq 2\}$	$w_1 = 1$
C_2: $\{c_{21}: 3 \leq x - y \leq 4\} \vee \{c_{22}: 5 \leq x - z \leq 6\}$	$w_2 = 2$
C_3: $\{c_{31}: 1 \leq y - z \leq 2\}$	$w_3 = 4$
C_4: $\{c_{41}: 0 \leq x - z \leq 7\}$	$w_4 = \infty$

Clearly there is no assignment that will satisfy all the constraints of this problem, since c_{21} conflicts with c_{11}, and c_{22} conflicts with the constraint induced by the composition of c_{11} and c_{31}. In addition, C_4 is a hard constraint having infinite weight, and therefore must be satisfied in any solution.

Once again, we can consider object-level and meta-level solutions to our VDTP. For instance, the object-level assignment $(x, y, z) \leftarrow (6, 3, 1)$ violates only constraint C_1. This has a cost of 1, and since we know that no solution exists with a cost of 0, it is an optimal solution. Importantly, when we move to the meta-CSP, a solution is no longer necessarily a total assignment; instead a (meta)-variable may be left unassigned, signifying that none of the disjuncts associated with it should be enforced.

The Relationship Between DTPPs and VDTPs

The previous example illustrates an important distinction between DTPPs and VDTPs. Each constraint in a DTPP plays a dual role, serving both as a hard constraint that requires assignments to be within the feasible region, and as a soft constraint expressing the preference values that the assignments within that region will receive. In contrast, each VDTP constraint is either strictly hard (i.e., having an infinite weight, and thus requiring satisfaction) or strictly soft (i.e., having finite weight, and permitting violation).

Nonetheless, we can show that for purposes of utilitarian optimality, any DTPP D' has an equivalent VDTP D in the following sense: (i) an assignment S is a solution to D' iff it is a solution to D, and (ii) solution S_1 is at least as preferred as S_2 in D' ($val_{D'}(S_1) \geq val_{D'}(S_2)$), iff S_1 is also at least as preferred as S_2 in D ($cost_D(S_1) \leq cost_D(S_2)$). In fact, we prove something stronger: we show that in addition, given any VDTP D, there is a DTPP D' that is equivalent in exactly the same sense, and hence the VDTP and DTPP formalisms have equivalent expressive power. This relationship requires only that the preference functions be piecewise-constant, an assumption commonly made in prior DTPP research.

Converting a VDTP into a DTPP

We begin by showing how to convert a VDTP into an equivalent DTPP. Let D be a VDTP with constraints $\{C_1, ..., C_n\}$ where $C_i = c_{i1} \vee ... \vee c_{in_i}$ and $c_{ij} = a_{ij} \leq x_{ij} - y_{ij} \leq b_{ij}$ and w_i is the weight of C_i. Create the *derived DTPP* D' by constructing a *derived constraint* C'_i from each constraint C_i in D:

- If C_i is hard (i.e., $w_i = \infty$), let $C'_i = c'_{i1} \vee ... \vee c'_{in_i}$ where $c'_{ij} = a_{ij} \leq x_{ij} - y_{ij} \leq b_{ij}$ with preference function $\langle f_{ij} : t \in [a_{ij}, b_{ij}] \rightarrow 0 \rangle$. Here, C'_i is a *hard-derived constraint*.
- Otherwise, if C_i is soft (i.e., $w_i \neq \infty$), let $C'_i = c'_{i1} \vee ... \vee c'_{in_i}$ where $c'_{ij} = -\infty \leq x_{ij} - y_{ij} \leq \infty$ with preference function $\langle f_{ij} : t \in [a_{ij}, b_{ij}] \rightarrow w_i, t \notin [a_{ij}, b_{ij}] \rightarrow 0 \rangle$. Here, C'_i is a *soft-derived constraint*.

In what follows, we distinguish the *constraint component* (i.e., the disjunctive constraints themselves) from the weights or the preference functions. It will also be useful to refer to the *worth* of a VDTP solution S, which is the weighted sum of soft constraints that are satisfied by S:

$$worth_D(S) = \sum_i \{w_i | w_i \neq \infty \wedge satisfies(S, C_i)\}$$

Note that the worth of a solution is the inverse of its cost, hence we wish to maximize this metric. We now show that every VDTP has an equivalent DTPP.

Theorem 1. VDTP D and its derived DTPP D' are equivalent.

Lemma 1. An assignment S is a solution to D if and only if it is a solution to D'.

Let $S : X \rightarrow \Re$ be an object-level assignment. First, assume S is a solution to the VDTP D. Then, for any arbitrary hard constraint C_i in D, S satisfies C_i. Since the constraint component of each hard-derived constraint C'_i in D' is identical to that of the hard constraint in D from which it was derived, S satisfies all hard-derived constraints in D'; it also satisfies all soft-derived constraints, since each of those contain only disjuncts with infinite feasible regions, and are thus satisfied by any assignment. So S is a solution to D'. Similarly, assume that S is a solution to the DTPP D'. Since it satisfies all hard-derived constraints C'_i in D', it must satisfy every hard constraint in D, because again, the corresponding constraints have identical constraint components. Since S need not satisfy the soft constraints in D, it is thus also a solution to D. $\square$

Lemma 2. If S_1 and S_2 are solutions to D (and D'), then $worth_D(S_1) \geq worth_D(S_2)$ iff $val_{D'}(S_1) \geq val_{D'}(S_2)$.

The worth of the VDTP D is as defined above. For a given solution S, the value of the DTPP D' is:

$$\begin{aligned} val_{D'}(S) &= \textstyle\sum_i val_{D'}(S, C'_i) \\ &= 0 + \textstyle\sum_i \{val_{D'}(S, C'_i) | soft\_derived(C'_i)\} \\ &= \textstyle\sum_i \{w_i | w_i \neq \infty \wedge satisfies(S, C_i)\} \end{aligned}$$

The second equality follows from the fact that the preference function for all hard-derived constraints is a constant zero. The third is because, by construction, for every soft-derived constraint C'_i, $val_{D'}(S, C'_i) = 0$ if the constraint from which it was derived (C_i) is not satisfied, and $val_{D'}(S, C'_i) = w_i$ if it is satisfied. Consequently, $worth_D(S) = val_{D'}(S)$, and so the lemma follows trivially. □

Theorem 1 follows directly from Lemmas 1 and 2.

Converting a DTPP into a VDTP

The conversion of a DTPP to a VDTP is slightly more complicated, and relies on the notion of *preference projections* (Peintner & Pollack 2005). An STPP preference projection "slices" an STPP constraint into a set of intervals that produce a preference value greater than or equal to some specified level l. A DTPP preference projection generalizes this notion to all intervals (disjuncts) in a DTPP constraint.

Definition (STPP Preference Projection). *Given an STPP constraint $C_{ij} = \langle a_{ij} \leq x_{ij} - y_{ij} \leq b_{ij}, f_{ij} \rangle$, the* **preference projection at level** *l for C_{ij} is $\mathcal{P}_{ij}[l] = \{c_1, c_2, ..., c_n\}$, where $c_k = \langle a_k \leq x_{ij} - y_{ij} \leq b_k \rangle$, $b_k < a_{k+1}$ for $1 \leq k < n$ and $\bigcup_{k=1}^{n} [a_k, b_k] = \{t | f_{ij}(t) \geq l\}$.*

Definition (DTPP Preference Projection). *Given a DTPP constraint $C_i = c_{i1} \vee c_{i2} \vee ... \vee c_{in}$, the* **preference projection at level** *l for C_i is $\mathcal{P}_i[l] = \bigcup_{j=1}^{n} \mathcal{P}_{ij}[l]$.*

In converting a DTPP into an equivalent VDTP, the basic idea will be to create multiple VDTP constraints for each individual DTPP constraint: one for each distinct preference level. Weights will be assigned in such a way that satisfying all projected constraints through level k will result in a *total* weight of k. The procedure is as follows:

Let D' be a DTPP with constraints $\{C'_1, ..., C'_n\}$. Then create the *derived VDTP* D as follows. For each constraint C'_i in D':

- Create a hard constraint $C_{<i,0>}$ in D, where $C_{<i,0>} = \bigvee \mathcal{P}_i[0]$ and $w_{i,0} = \infty$. Set l to zero.
- Find the smallest $l' > l$ such that $\mathcal{P}_i[l'] \neq \mathcal{P}_i[l]$.
- Create a soft constraint $C_{<i,l'>}$ in D, where $C_{<i,l'>} = \bigvee \mathcal{P}_i[l']$ and $w_{i,l'} = (l' - l)$. Set l to l'.
- Iterate until an l' is reached such that $\mathcal{P}_i[l'] = \oslash$.

Theorem 2. DTPP D' and its derived VDTP D are equivalent.

Lemma 3. An assignment S is a solution to D' if and only if it is a solution to D.

Suppose S is a solution to D'. By definition, it must necessarily satisfy every C'_i in D'. As in Lemma 1, we note that every hard constraint $C_{<i,0>}$ in D has a constraint component identical to that of the C'_i from which it was derived; thus, these must be satisfied as well, as must D as a whole (since the soft constraints may be violated by any solution). Now assume that S is a solution to D. It then necessarily satisfies every hard constraint $C_{<i,0>}$ (and observe that these are the only hard constraints). It must then also satisfy every C'_i in D' since the constraint components are again identical. Hence, S is a solution to D'. □

Lemma 4. If S_1 and S_2 are solutions to D' (and D), then $worth_D(S_1) \geq worth_D(S_2)$ iff $val_{D'}(S_1) \geq val_{D'}(S_2)$.

For an arbitrary constraint C'_i in D', let $val_{D'}(S, C'_i) = k_i$. Then by construction, S will satisfy any constraint $C_{<i,l>}$ in D for which $l \leq k_i$. Furthermore, S will not satisfy any constraint $C_{<i,m>}$ in D for which $m > k_i$. Also, by construction we have:

$$\sum_l \{w_{i,l} | l \neq 0 \wedge l \leq k_i\} = k_i$$

Since this holds for all constraints C'_i in D', we again get $worth_D(S) = val_{D'}(S)$, and the lemma follows immediately.[3] □

Theorem 2 follows directly from Lemmas 3 and 4.

Theorem 3. For the objective of utilitarian optimality, DTPPs and VDTPs are equivalent in expressive power.

This follows directly from Theorems 1 and 2. □

The fact that a utilitarian aggregation of preference values can be obtained by a Valued DTP should come as little surprise, since this objective function is commonly found in Valued CSP literature (Freuder & Wallace 1992; Schiex, Fargier, & Verfaillie 1995). In addition, similar proofs of equivalence have been drawn between the Valued CSP and Semiring CSP for finite-domains (Bistarelli *et al.* 1999). However, much of the prior work on preferences in TCSPs has had no need to move to the valued representation, in part because they have exposed tractable cases of the semiring that require either weaker objective functions (e.g., Weakest Link Optimality (Khatib *et al.* 2003)) or different shapes of preferences (e.g., convex piecewise-linear functions (Morris *et al.* 2004)). In contrast, our focus is on a broad range of intractable instances where this alternative modeling of preferences can be more easily exploited.

Solving Valued DTPs

We have just shown that for utilitarian optimization, any DTPP with piecewise-constant preference functions can be translated into an equivalent VDTP. Using this reformulation, we will describe a branch-and-bound algorithm for finding utilitarian optimal solutions to VDTPs; pseudocode is given in Figure 1. The input variable A is the current set of assignments to meta-variables, and is initially $\oslash$; variable U is the set of unassigned meta-variables (initially the entire set C); *cost* is the total weighted sum of violated constraints (initially zero); and *upperbound* is the stored cost of the best solution found so far (initially set to ∞). Note that unlike ARIO, we require no MLLP module or SAT-solving component, and unlike GAPD, our memory requirements are polynomial in the size of the problem.

This algorithm, which takes an approach similar to that in (Moffitt & Pollack 2005), resembles the meta-CSP backtracking search commonly used for solving traditional DTPs with two notable differences. First, backtracking occurs only when the combined weight of the violated constraints

[3] One interesting consequence of this construction is that several disjunctive constraints may be generated from a single non-disjunctive STPP constraint, if its preference function is not semi-convex.

Solve-VDTP(A, U, *cost*, *upperbound*)
If (*cost* $\geq$ *upperbound*) return
If ($U = \emptyset$)
    best-solution-so-far $\leftarrow A$
    upperbound $\leftarrow$ *cost*
    return
EndIf
$C_i \leftarrow$ *select-variable*(U), $U' \leftarrow U - \{C_i\}$
For each disjunct c_{ij} of $D(C_i)$
    $A' \leftarrow A \cup \{C_i \leftarrow c_{ij}\}$
    If (**consistent**(A'))
        Solve-VDTP(A', U', *cost*, *upperbound*)
    EndIf
EndFor
$A' \leftarrow A \cup \{C_i \leftarrow \epsilon\}$
Solve-VDTP(A', U', *cost* + w_i, *upperbound*)

Figure 1: A branch-and-bound algorithm for solving VDTPs

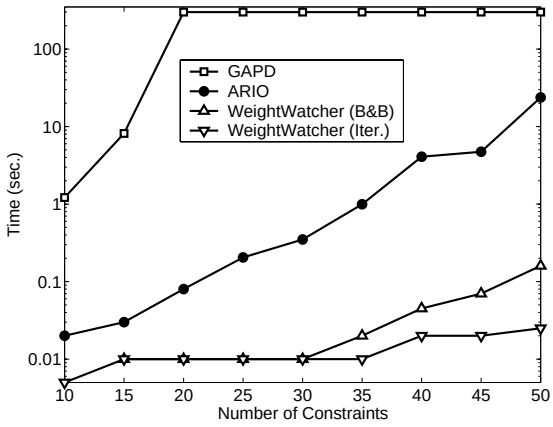

Figure 2: Median running times for GAPD, ARIO, and WEIGHTWATCHER for DTPPs of varying sizes (Timeout set at 300s)

(*cost*) equals or exceeds that of the current best solution (*upperbound*); in a standard DTP solver, backtracking would occur whenever *cost* became nonzero (i.e., when any constraint had been violated). Second, in addition to the values in the original domains of the meta-variables, there is the possibility of an empty assignment ('ϵ'), and so the branching factor increases by exactly one. This latter modification, in combination with the meta-CSP search space employed in temporal reasoning, sets our algorithm apart from previous applications of weighted constraint satisfaction to classical CSPs.

The key advantages to our meta-CSP reformulation of DTPP optimization are that (1) it performs optimization without the need to solve a sequence of satisfaction problems, and (2) it allows the direct incorporation of several powerful techniques previously developed in decision-based DTP literature (Stergiou & Koubarakis 1998; Armando, Castellini, & Giunchiglia 1999; Oddi & Cesta 2000; Tsamardinos & Pollack 2003), including incremental forward checking, semantic branching, removal of subsumed variables, conflict-directed backjumping, no-good recording, and several heuristics. Previous CSP-based algorithms applied these only to the small portion of the search space corresponding to the DTPP's lowest preference level, a possible explanation of their poor performance compared to the SAT-based solver.

In addition to our branch-and-bound algorithm, an alternative *iterative weakening* approach to optimization (Provost 1993) is also available, which begins with infeasibly high objective values and works downwards until consistency is achieved; see (Moffitt & Pollack 2005) for details.

Experimental Results

In this section, we describe the results of a set of experiments that were performed to compare the weighted constraint satisfaction approach of our solver (which we name WEIGHTWATCHER) against the two previous systems for solving DTPPs: ARIO[4] (Sheini *et al.* 2005) and the GAPD solver (Peintner 2005). All tests were performed on a 2.26GHz Pentium 4 processor with 1 GB of RAM.

Unfortunately, there remains an absence of real-world benchmarks in temporal preference literature with which to provide an empirical comparison of solvers. Consequently, we must employ the same problem generator used in prior DTPP studies, which takes as parameters $\langle E, C, D_-, D_+, L, R_-, R_+ \rangle$. The DTPP is constructed by generating a set of E events $\{x_1, x_2, ..., x_E\}$ and a set of C constraints, where each constraint C_i consists of exactly 2 disjuncts. Each disjunct c_{ij} is assigned a pair of events x_{ij} and y_{ij}, and the lower and upper bounds a_{ij} and b_{ij} on the feasible difference between those events are selected from the interval $[D_-, D_+]$. To define the preference function for each temporal difference, the values a_{ij} and b_{ij} serve as the endpoints for preference level 0. To construct preference level l, a reduction factor is chosen from the interval $[R_-, R_+] \subset [0, 1]$ with uniform probability, and is applied to the interval at preference level $l-1$; the resulting (smaller) interval is placed randomly between its endpoints. This process is repeated until preference level L is reached or an interval having zero length is created.

We ran three sets of experiments, in which we varied the problem size, number of preference levels, and constraint density. For all experiments, we generate 30 trials for each setting of parameters, and report the median running time for each solver over the 30 trials. A timeout of 300 seconds is enforced on all problems. The time required to convert the DTPP into the VDTP is included in these runtimes, but is extremely negligible.

Varying Problem Size In our first experiment, we explore the abilities of WEIGHTWATCHER, ARIO, and GAPD to scale with the size of the problem. Our ability to perform well on this set of tests is especially important, since unlike other

[4] At the time of this writing, ARIO is the only participant of the recent SMT (satisfiability modulo theories) competition (Barrett, de Moura, & Stump 2005) whose difference-logic engine has been augmented with a reasoning module (in ARIO's case, an MLLP solver) that allows it to handle the DTPP objective function and solve optimization (as opposed to satisfaction) problems.

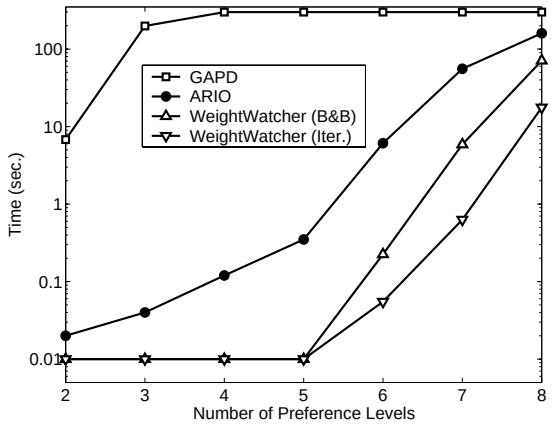

Figure 3: Median running times for GAPD, ARIO, and WEIGHTWATCHER for DTPPs with varying numbers of preference levels (Timeout set at 300s)

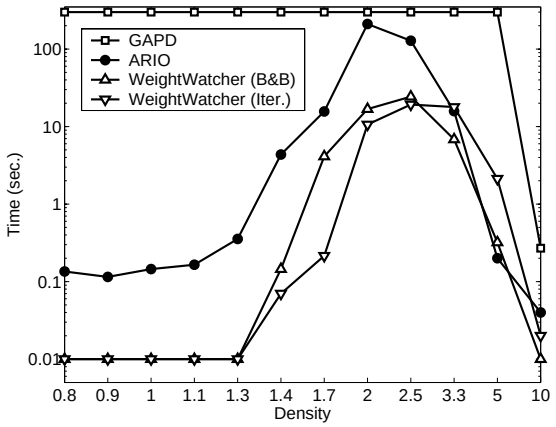

Figure 4: Median running times for GAPD, ARIO, and WEIGHTWATCHER for DTPPs of varying density (Timeout set at 300s)

problem parameters (such as the number of preference levels), the size of the problem is often difficult for a knowledge engineer to control directly. We hold fixed the following parameters: $\{D_- = -50, D_+ = 100, L = 5, R_- = 0.5, R_+ = 0.9\}$. These settings are identical to those used in (Sheini *et al.* 2005). We vary the number of constraints C from 10 to 50, and set the number of events E to $\frac{4}{5}C$ to maintain a constant constraint density.

Figure 2 displays the results of this experiment. The number of constraints in the problem is shown on the x-axis, and on the y-axis is the median running time (note the logarithmic scale). GAPD quickly reaches the timeout limit of 300 seconds once the number of constraints equals $C = 25$. The median runtime of ARIO consistently remains far below the cutoff threshold, and reaches 23.8 seconds when $C = 50$. Recall that the presence of an efficient SAT solver has been labeled as the key ingredient in achieving this substantial improvement. Yet, the branch-and-bound and iterative versions of WEIGHTWATCHER surpass both GAPD and ARIO on all problem sizes, without the aid of SAT techniques. For $C = 50$ (the largest set of problems), the median runtime of the iterative version is 0.025 seconds, three orders of magnitude faster than ARIO.

Varying the Number of Preference Levels In our second experiment, we examine the effect of the number of preference levels on the performance of these solvers. We hold fixed the parameters $\{E = 24, C = 30, D_- = -50, D_+ = 100, R_- = 0.5, R_+ = 0.9\}$, and vary the number of preference levels L between 2 and 8.

Figure 3 provides the results of this experiment. Once again, GAPD tends to be much slower than ARIO, and ARIO is in turn considerably slower than either incarnation of WEIGHTWATCHER. A difference of between one and two orders of magnitude is observed between ARIO and our iterative solver for cases where the number of preference levels is larger than 3.

Varying the Constraint Density In our third and final experiment, we explore the abilities of these solvers to scale with constraint density, which is the ratio of constraints to events (C/E). We hold fixed the parameters $\{C = 30, D_- = -50, D_+ = 100, L = 5, R_- = 0.5, R_+ = 0.9\}$, and vary the number of events E between 3 and 36.

In Figure 4 we present the results. For problems having constraint densities less than or equal to 2.5, both WEIGHTWATCHER solvers typically achieve a performance improvement between one and two orders of magnitude over ARIO; for densities larger than this, the difference in speed becomes less evident. It is also near this point that the branch-and-bound version of WEIGHTWATCHER begins to overtake the iterative version. This is likely because the costs of the optimal solutions for these extremely constrained problems are quite large, and thus several iterations are required before a solution is discovered.

In summary, WEIGHTWATCHER consistently outperforms the previous DTPP solvers, including the SAT-based ARIO system, on several dimensions. This suggests that, despite recent successes of SAT in solving decision variants of temporal reasoning (Armando *et al.* 2004; Nieuwenhuis & Oliveras 2005), there remains considerable room for improvement in the application of these techniques to optimization. Of course, by no means does this imply that the SAT approach is inherently flawed; however, our results clearly demonstrate that well-established CSP-based methods should not be ignored or discounted when constructing algorithms for optimization of temporal preferences.

Conclusion and Future Work

In this paper, we have presented a new efficient algorithm for obtaining utilitarian optimal solutions to Disjunctive Temporal Problems with Preferences (DTPPs). To facilitate our approach, we have introduced the Valued DTP, and have shown that it can express the same set of utilitarian optimal solutions as the DTPP with piecewise-constant preference functions. Using this relationship, we have developed a method for achieving *weighted constraint satisfaction* within a meta-CSP search space that has traditionally been used to solve DTPs without preferences, allowing us to leverage several

techniques employed in previous decision-based DTP literature. Experimental results demonstrate that our solver consistently outperforms the state-of-the-art SAT-based approach by orders of magnitude.

This line of research opens the door to several promising avenues of continued progress. For instance, while our focus has been on generating utilitarian optimal solutions, we believe our algorithm can be generalized to handle other objective functions as well. Furthermore, one may wish to combine this approach with our recent addition of conditional bounds and finite-domain constraints to DTPs (Moffitt, Peintner, & Pollack 2005), allowing the encoding of non-temporal constraints that the SAT-based formulation can express quite easily. Finally, integration of solvers such as ours into modern temporal planning systems – especially those that exploit DTP solving techniques (Schwartz & Pollack 2004; Gerevini, Saetti, & Serina 2005) – appears to be a logical next step toward efficient reasoning of preferences in planning domains.

Acknowledgements

The authors thank the anonymous reviewers for their many helpful comments. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHD030010 and the Air Force Office of Scientific Research under Contract No. FA9550-04-1-0043. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of DARPA, the Department of Interior-National Business Center, or the United States Air Force.

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)*, 16–29.

Armando, A.; Castellini, C.; and Giunchiglia, E. 1999. SAT-based procedures for temporal reasoning. In *Proceedings of the 5th European Conference on Planning (ECP-1999)*, 97–108.

Barrett, C.; de Moura, L.; and Stump, A. 2005. SMT-COMP: Satisfiability Modulo Theories Competition. In *Proceedings of the 17th International Conference on Computer Aided Verification (CAV-2005)*, 20–23.

Bistarelli, S.; Montanari, U.; Rossi, F.; Schiex, T.; Verfaillie, G.; and Fargier, H. 1999. Semiring-based CSPs and valued CSPs: Frameworks, properties, and comparison. *Constraints* 4(3):199–240.

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.

Fox, M., and Coddington, A. M., eds. 2002. *AIPS 2002 Workshop on Planning for Temporal Domains*.

Freuder, E. C., and Wallace, R. J. 1992. Partial constraint satisfaction. *Artificial Intelligence* 58(1-3):21–70.

Gerevini, A.; Saetti, A.; and Serina, I. 2005. Integrating planning and temporal reasoning for domains with durations and time windows. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-2005)*, 1226–1231.

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.

Kumar, T. K. S. 2004. A polynomial-time algorithm for simple temporal problems with piecewise constant domain preference functions. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-2004)*, 67–72.

Moffitt, M. D., and Pollack, M. E. 2005. 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.; 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.

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 Practice of Constraint Programming (CP-2004)*, 408–422.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 530–535.

Nieuwenhuis, R., and Oliveras, A. 2005. DPLL(T) with exhaustive theory propagation and its application to difference logic. In *Proceedings of the 17th International Conference on Computer Aided Verification (CAV-2005)*, 321–334.

Oddi, A., and Cesta, A. 2000. Incremental forward checking for the disjunctive temporal problem. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI-2000)*, 108–112.

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.

Provost, F. J. 1993. Iterative weakening: Optimal and near-optimal policies for the selection of search bias. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-1993)*, 749–755.

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.

Schwartz, P. J., and Pollack, M. E. 2004. Planning with disjunctive temporal constraints. In *ICAPS-04 Workshop on Integrating Planning into Scheduling*.

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.

Elicitation and Application of Narrative Constraints Through Mixed-Initiative Planning

James M. Thomas
Liquid Narrative Group
Department of Computer Science
Box 7535, NC State University
Raleigh, NC 27695 USA
jmthoma5@ncsu.edu

R. Michael Young
Liquid Narrative Group
Department of Computer Science
Box 7535, NC State University
Raleigh, NC 27695 USA
young@csc.ncsu.edu

Abstract

This paper describes an approach for the elicitation and application of authorial goals for interactive narrative. A framework is presented for using author input to elaborate descriptions of planning problems and domains. The authors describe a mixed-initiative graphical user interface that exploits this framework to allow authors to express narrative goals and understand how these goals affect user experiences of an interactive narrative.

Motivation

The headlamps of the rented Aston Martin illuminated more fog than motorway as JB raced smoothly toward Windermere on the M6 an hour before dawn. Although he had not seen another vehicle since junction 36, JB's eyes jerked quickly to the mirrors when a rhythmic thumping from the right front wheel shattered the stillness. He could not afford to encounter any roadside "assistance" this morning; he had no key for the titanium brief case handcuffed to his wrist, and no license for the Walther PPK holstered to his chest. This was a particularly bad time to enter "Spare-Tyre World".

Automated techniques have proven useful for generating plans from the blocks world, to the spare tire world, to worlds as difficult as Mars. Traditional automated planners require goals to be represented as a conjunction of atomic logical propositions. A successful plan guarantees satisfaction of each of the "hard" constraints of the goal state. As planners have been applied to increasingly diverse domains, they have encountered more "over-subscription problems" (Smith 2004), where available resources may be insufficient to satisfy all the desirable goals, but a plan that achieves a useful subset of these goals is still useful.

The domain of interactive narrative, where human users interact with one or more computer controlled agents to produce stories, is another where plan goals often contain a mix of hard and soft constraints. Furthermore, these hard and soft constraints can be seen as comprising three distinct competing motivations fundamental to the genre: character believability, user control, and narrative coherence. Ensuring a preferred balance between these goals has been described as 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).

These conflicts are illustrated in the example narrative that introduces this section of the paper. JB's goal of changing the tire may be less important than the story goal of having JB arrive at the destination at a particular time. Having JB reach the destination unseen may be more important to the narrative structure than reaching it quickly. It may be seen as out of character for JB to change the tire if a more exotic alternative mode of transportation is known to be available, just as plans where JB uses *mix-martini-by-shaking* are preferable to those that employ stirring. How should the system detect and respond to the human user's attempt to steal the kite surfer that would satisfy JB's goals and the story goals?

This paper investigates a particular strategy for obtaining a more elaborate description of the interactive narrative domain and applying this information to make more preferable planning decisions.

Background: Interactive Narrative

A fundamental challenge the domain of interactive poses to automated planning is finding a heuristic that quantifies the relative "goodness" a story plan (Riedl and Young 2004). We lack mature computational models for narrative components such as "suspense", or "romance" and we lack automated mechanisms to draw associations between such generalized ideals and the semantics of a specific problem domain and plan space. In fact, not only are the heuristics unproven, one can argue that we have yet to compile comprehensive heuristics that can be used by skilled and motivated humans to ensure narrative quality. As author Somerset Maugham quipped, "There are three rules for writing the novel. Unfortunately, no one knows what they are". Regardless, authors are able to express goals for

particular narrative elements of particular stories in terms relevant to those particular domains.

Perhaps the difficulty in defining the quality of narrative is a reason many interactive systems restrict run-time behaviors of the user and automated characters to avoid conflicts with a narrowly defined plot. Unfortunately, limiting the user's control reduces the coveted quality of "immersion" in the story, and limiting the behaviors of computer-control characters can marginalize their importance to the story. On the other hand, systems that strive only for character authenticity often lack the control to have these emergent creatures contribute to the plot.

Interactive narrative domains have some advantages that could be more fully exploited to confront these challenges. The virtual worlds in which these narratives take place are both fully knowable and fully malleable. In fact, the plan authors may be responsible not only for the domain representation, but also may have a role in the construction of the domain itself. Because interactive narrative planning is a component within this larger creative process, the set of allowable actions and the propositions used to describe initial and goal states for use by the planner are subject to changes large and small.

If planning is situated advantageously within this larger process, human creativity can be leveraged to solve problems of increasing complexity. The authors are in the process of creating a mixed-initiative planning interface called Bowman. Bowman is a component of the Zócalo suite of planning tools available through NCSU at http://zocalo.csc.ncsu.edu. Zócalo is a web-services extension of the Mimesis architecture described by Riedl, Saretto, and Young (2003). Mimesis and Zócalo use a planner is based on Longbow, a decompositional (HTN) partial order causal link planner described by Young, Pollack and Moore (1994), that automatically detects user actions that could threaten the goals of a hierarchical story plan. The planner responds to these threats by creating a set of policies to be applied at run-time if the user chooses a threatening action.

Bowman provides a GUI that allows authors to describe types, objects, operators, conditions and the initial and goal state of a planning problem. Bowman represents the planning problem in a XML structure that is sent to the web-service to generate plans. The planner interface supports requests for the next N plans, planning for N seconds, or simply until a complete plan is found. The plan space and individual plan nodes are sent back to Bowman, also in an XML wrapper. Integrated, scalable vector graphic rendering of both the plan space, and individual plans allow authors to navigate from plan to plan to explore structural differences.

Background: Domain Elaboration

To obtain a broader and deeper set of the plan author's preferences requires a richer understanding of the planning domain. This has not been a traditional focus of automated planning. As Nau (2005) noted in a recent historical survey, much of the work in classical, domain-independent planners has taken place in highly abstract and tightly restricted domains. The domains are simple enough that the limited expressiveness of STRIPS (Fikes and Nilsson 1971), or languages that are slight extensions to STRIPS in a direction of immediate concern to the research can sufficiently describe problems of interest. Nau argues that the restrictions demanded by many supposedly "domain-independent" classical planners admit a class of domains that is so small and so far removed from reality that it can be effectively considered a single specialized world. In contrast, most planners that work in the real world are customized to a specific domain, fine-tuning their input, output, and world model to the environment in which they operate. Nau points in the direction of richer domain descriptions by advocating a shift toward what he calls "configurable planners".

A configurable planner combines a domain-independent planning engine with configuration information describing how to solve problems in that domain. Configurable planners often employ higher-level abstractions like hierarchical task networks to capture and exploit domain knowledge. Nau claims recent planning competitions have shown that configurable planners rival the efficiency of domain-specific planners and yet work in a broader class of domains than either the domain-specific or the ostensibly domain-independent planners.

Nau's configurable planners require more elaborate domain descriptions to represent the knowledge that enable the domain-independent planning engines to achieve better performance. Similarly, research efforts in advisable and mixed-initiative planning have led to a richer language for expressing the characteristics of planning domains. Advisable planning (Myers 1996) attempts to shape the behavior of the planner by adding additional constraints to extend the definitions of the domain and the problem prior to the invocation of the planner. Mixed initiative systems 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 to extend the domain description with preferences in the form of constraints, then by the planning system, which uses this additional information in the construction of plans.

The success of an advisable planning system is perhaps more strongly tied to the effectiveness of its domain

knowledge representation than is the case for mixed-initiative systems where ambiguities can be iteratively resolved. Myers has demonstrated the value of a "domain metatheory" (Myers 2000a) 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).

Myers' domain metatheory serves two roles 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. Her model is built on three constructs: *roles, features,* and *measures*. *Roles* describe the function of an object within a particular operator. *Features* are author-defined modifiers that differentiate operators. *Measures* are user-defined criteria that determine partial orderings of features. For example, "By Air" and "By Land" could be two features that are applied to operators. A measure of "Affordability" may order operators with the "By Air" feature ahead of operators with the "By Land" feature. Conversely, the measure of "Speed" may provide the reverse ordering. Myers extends measures to apply to object instances, and allows for reasoning about which specific object-instances can be *role-fills* for particular operators in the plan. For example, in a travel domain the role of <Accommodation> may be constrained for certain operators in certain circumstances to a *role-fill* of "3-star hotel". The authors provide a more complete review of Myers' work and mixed initiative planning can be found in (Thomas and Young, 2006).

Bowman's "Domain Elaboration Framework"

Rather than exposing all elements of the domain metatheory directly to the plan author, Bowman translates the constructs presented to human authors into a more compact representation for use by the planner. This separation allows for some preprocessing of the human input to gain computational efficiencies on the one hand, and it allows for richer interactions with the human on the other. Bowman is not unique in representing the planning problem solving state at multiple levels of abstractions. Allen and Ferguson (2002) employ a four-layer model. Their model allowed the user to move from high-level hierarchical objectives, through task structures that summarized classes of concrete solutions, to more primitive descriptions of particular plan fragments and world states.

The lowest layer internal representation used by Bowman is called the Domain Elaboration Framework, or DEF. The basis of DEF is a STRIPS-style planning domain characterized by objects, conditions and operators. Operators are either primitive or abstract compositions of one or more primitive or abstract 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 **types**. This hierarchy has a predefined unique root node named "***anyThing***". The author creates and names all the other nodes of the type hierarchy. Every operator, parameter, and object instance has at least one associated type and zero or more associated **measurements**. There is no restriction on the maximum number of types or measurements that can be associated with a single entity, to help mitigate inheritance conflicts that would otherwise arise in even marginally deep hierarchies.

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 normalized over the interval [-1, 1]. This interval was chosen to facilitate application of a neutral default value ('0'), in cases where author preferences have yet to be discerned.

DEF does not appear to be the most ambitious knowledge representation scheme. The intended strategy is for DEF to be general and simple enough to be applied fairly orthogonally to the constituent elements of the planning domain. This in turn should to provide the human users of Bowman the power to experiment with the creation of higher-level abstractions that may be useful in interactive narrative. As noted in the background sections, there is not yet a strong consensus of what those higher-level abstractions ought to be so it seems prudent to avoid building particular guesses into the foundations of the system. Bowman attempts to allow for abstractions of arbitrary complexity at the interface level, while preserving an underlying representation in DEF that facilitates efficient reasoning about the qualities of individual plans and the qualitative differences between plans.

Expressivity of DEF

An example that illustrates some of the synergy between Bowman and DEF is the simple application of **type**. Because every object instance in the domain and every parameter of an operator or a condition that expresses an initial or goal state has a type, Bowman can reference the type hierarchy to ensure that authorial intentions for bindings are maintained. For example, in the case of an operator whose semantics are to move something from an object of type *location* specified in a parameter named *loc1*, the author would associate the type *location* with parameter *loc1* through the Bowman interface. Behind the scenes, when the author first creates the type *location,* Bowman automatically creates a condition *isalocation(?x)* that is added to the list of conditions available to describe problems in this planning domain. Then, when the author

makes the association between the *loc1* parameter and the *location* type, Bowman automatically adds the condition *isalocation(loc1)* to the list of preconditions of the associated move operator. Note that the default type of "anyThing" is automatically supplied wherever the plan author has not taken the time to be more specific. Thus, this is an optional mechanism to allow the plan author to communicate a planning preference with minimal effort. Furthermore, this preference is expressed to the planner without any extension to the most basic STRIPS implementation.

Figure 1 shows how Bowman uses the types expressed in DEF to guide the human author in editing the literal preconditions of an operator. Note that a color-coding of the type hierarchy is automatically generated and applied to the object constants (depicted in the tree view in the upper right subwindow), operator parameters, and terms of the literals (lower left subwindow). As the author drags and drops objects or operator parameters to make associations with the literals, corresponding conditions are automatically added to the description of the operator.

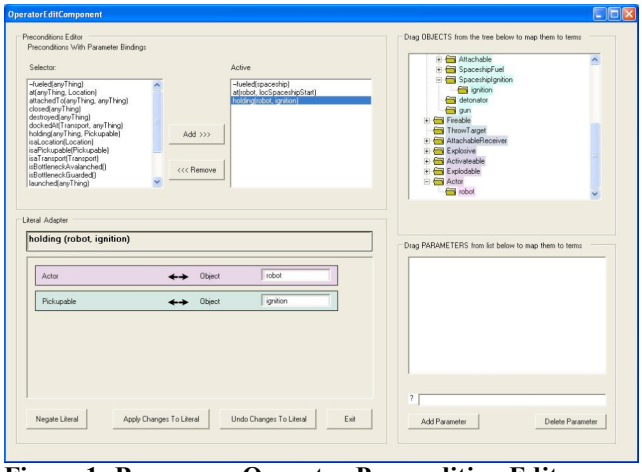

Figure 1: Bowman – Operator Precondition Editor

Making types and measurements applicable to every operator, operator parameter, and condition parameter is an expressive advantage of DEF. This advantage may be exploited to express the goals of an author of interactive narrative. Authorial goals can often be articulated in terms of the types of actions contained in a story. A poignant example of this can be found in a commercially successful narrative: the film The Princess Bride. The movie begins by introducing a boy who is home from school with an illness. His grandfather comes to visit him to read him the book The Princess Bride. He overcomes the boy's initial reluctance by advertising that the story will contain such narrative preferences as "Fencing. Fighting. Torture. Revenge. Giants. Monsters. Chases. Escapes. True Love. Miracles." When an explicitly romantic scene appears early in the Grandfather's reading, the boy suspiciously interrogates him with "What is this? Are you trying to trick me? Where's the sports? Is this a… *kissing* book?"

Many of these preferences can be translated to quantifications over DEF measurements applied to classical planning constructs such as operators and object instances. For example, the author can define "*kissing*" as a dimension assign measurements like (*kissing*, .95) to every operator associated with the act of kissing. Operators that have no connection to interpersonal relationships or romance can be interpreted to have neutral weights for *kissing*, requiring no action on the part of the plan author. If the plan author chooses, she may encode antagonistic operators (perhaps fighting, torture) with negative weights for the kissing dimension. Similarly, dimensions to classify actions consistent with fencing, fighting, and torture can be easily specified by the author.

Another kind of preference shown in this example is an implication of the *types* of objects that populate the story, for example: giants, monsters, and swords. Elicitation of the author's preferences may thus involve the addition of new types (like "monster") to the type hierarchy and to the list of types for the object instances of "Shrieking Eel", "Rodent of Unusual Size", or "Zaphod Beeblebrox".

However, preferences for more complex narrative structures, such as those implied by "Chases", "Escapes", "True Love" and "Revenge" cannot be specified through static elaboration of the domain. For example, is attempt-to-kill(?intended-victim, ?attacker) a 'revenge' action? It depends on how the action and the bindings to the objects in the action relate to other actions and other objects *within a particular concrete plan.* It is impossible to affix these more complex narrative qualities to static elements of the domain description. In the case of "*attempt-to-kill*", there must have been an earlier action, explicitly articulated in the plan or inferred from the initial state, where the victim negatively affected the attacker. Clearly, to identify the degree to which a construct like "revenge" is satisfied in a story requires context-dependent evaluations of actions and bindings over an un-circumscribable span of the narrative plan.

In fact, most of the constructs used to describe narrative since Aristotle's early exploration of dramatic structure in Poetics, require significant story context. For example, Aristotle highlighted "peripeteia", or reversal of circumstance, as a key abstraction for characterizing complex plots. While it would be possible to have a reversal of circumstance realized completely within a single operator of a narrative plan, not all reversals can be defined in a single action. Narrative structures seem to beg for greater expressivity than what is offered by DEF.

A tool that is particularly well suited to the expression of context-dependent conditions is that of "Ceteris Paribus" or "CP" preference statements. CP statements take the form of conditional expressions linked by logical connectives. For example, it seems that CP statements can

define revenge: "Use *attempt-to-kill(?intended-victim, ?attacker)* WHEN *wronged(?intended-victim, ?attacker)* AND *^wrong-avenged?(intended-victim, ?attacker)"*. Human users are accustomed to supplying CP preference statements in search dialogs and in any number of other computer-assisted tasks. Brafman and Chernyavsky (2005), have shown a method to add CP preference statements in solving planning problems that have been converted to equivalent constraint satisfaction problems, or CSPs. Prestwich (et. al., 2005) have shown algorithms to find optimal solutions for CP nets through progressive relaxation of CP preference statements.

However, such methods do not provide a general solution for narrative preferences. Even in this small example, two literals ("*wronged*", "*wrong-avenged*") had to be contrived to represent the history of the relationship between victim and attacker. The problem is that revenge is a property of the plan, not an abstract truth-value of the final state. As Brafman and Chernyavsky note, their initial work concentrated on goal constraints - the methodology for expressing preferences over plan structure has not yet been shown. For the CP statements to have the intended effect, "*wronged*" and "*wrong-avenged*" must be added as effects to all the appropriate operators. But even this is insufficient. More context information must be taken into account in deciding whether or not an actor was wronged. Actors have partial knowledge. What if one of the characters does not know they have been wronged, or someone has framed a third party for the injury?

What is actually required is a model of the beliefs, desires, and intentions (BDI) of each of the characters, and an encoding of the domain that expresses how different actions may affect character BDI. As introduced in (Thomas and Young, 2006) DEF can be used to construct narrative "macros" that mimic a BDI model. Narrative "macros" are relatively general sets of conditions and operators that can be used to model dynamic relationships between entities in the problem domain. An example of such a relationship is the "attitude" one agent in the domain has for another agent in the domain. A set of general "attitude-maintenance" macros (e.g., an "attitude-up" operator, an "attitude-neutral" condition), can be built up from the DEF constructs in Bowman and shared between authors of interactive narrative. In the optimal case, such a library is created with suitably general parameters like "?attitude-name" so authors can use a single set of macros to model "friendship", "jealousy", or "revenge". Bowman users, i.e., actual working authors, can use macros to experiment with the construction of reusable narrative constructs. This differs from the approach suggested for incorporation of the CP statements because the macros need not be implicitly tied to particular statements of preference. It might be an interesting exercise to see how many of the state trajectory constraints proposed for PPDL3 (Gerevini and Long, 2005) could be mapped to similar STRIPS-based macro constructs.

As in the early days of computer programming, where useful macros pointed the way toward high-level language constructs that caused quantum leaps in expressivity and productivity, perhaps the approach advocated here can lead to a similar bootstrapping of knowledge descriptions in the planning domain, at least for interactive narrative.

To summarize, DEF associates a set of one or more types and zero or more measurements with every operator, operator parameter, condition parameter, and object instance. DEF is a domain-independent representation intended to by leveraged by a user interface for use with any planner that can work with a STRIPS-style domain description. Planning heuristics that takes these measurements as an input can offer a high-degree of fidelity to discrimination between candidate plans, but will not necessarily provide sufficient expressivity for arbitrary narrative constructs.

Using DEF to Reason About Qualities of Plans

One of the motivations behind Myers' advisable planning work that is shared by interactive narrative is the desire to create multiple plans for a given problem that differ according to author-defined qualities. Specifically, the plan author can nominate a subset of measures from the domain metatheory to serve as the evaluation criteria by which the qualities of different plans are to be compared. For each evaluation criterion *k,* an evaluation function returns the extent to which that measure is realized in a given plan fragment expressed as value from the interval [0, 1]. A set of *k* evaluation criteria defines a *k*-dimensional evaluation space. Myers uses this evaluation function to construct formulae that provide several types of measurements. A formula gives the Euclidean distance between two plans across the *k*-dimensional space. A dispersion formula provides the average distance between plans in a plan set. A proximity formula measures the degree to which a solution set is 'near' all the other points in the evaluation space. Because DEF requires explicit normalization of weights over an interval [-1, 1], it is simple to create an evaluation function that harnesses for the qualitative measurements Myers has demonstrated.

Another mechanism for reasoning about types recently introduced by Myers (2005) is also easily applicable in the DEF context. Myers defines a function MinSuperType(V) which finds the most specific super-type common to a set of elements V . This allows the author to characterize the differences between plans or parts of plans, through the five distinct set relationships that correspond to particular subsets of their typed elements. Set-arithmetic functions are described that help pinpoint key strategic differences between plans and show areas where plans are not as different as they might seem. Again, DEF is well-suited to make use of this reasoning because of its extensive use of a type hierarchy.

Relaxable Custom Heuristics

Heuristics are rules used to measure the desirability of one plan compared to another. The planner uses heuristics to rank the order in which nodes of the search space will be expanded. Thus, heuristics are a special case of preferences applied to the analysis of a plan. Typical uses of heuristics are to favor plans that have fewer steps, plan that have fewer flaws, plans that have more abstract (hierarchical) as opposed to primitive steps, or plans where particularly problematic causal links are established early. The idea of different treatment for conditions derived from different sources is not new, as it was featured prominently in the description of the NONLIN algorithm (Tate 1976).

With Bowman, the plan author can construct heuristics based on any of the attributes described in DEF: types, dimensions, or measurements 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.

Furthermore, the Bowman GUI can allow the author to differentiate the weighting of different heuristic factors through the assignment of an enforcement priority to each precondition and constraint on through sliders that read "Ignored" at the bottom of the scale and "Required" at the top.

This strategy could be implemented without changing existing planning algorithms by exploiting Bowman's iterative problem solving environment and let the author determine a global narrative rigidity value, also via a slider. Each individual preference value whose slider is lower than the global cutoff value would be automatically redacted from the problem definition sent to the planner. The author could then perform a manual version of simulated annealing, progressively tweaking the relative preference values repeatedly requesting plans at different levels or the slider to see how varying degrees of their authorial preferences affect the diversity of the plan space.

Our hope is that by exploiting the expressivity and efficiency of a suitably tuned GUI and the creativity and insight of a suitably engaged human author, we can facilitate the creation of more interesting and complex interactive narratives. We believe that the architecture of the system will allow the experimentation and exploration of human authors to create artifacts will be of near term usefulness and will also inform future research efforts.

Alternative Strategies For Advice Conflicts

Myers (2000b) has researched two techniques for partial satisfaction of user-specified advice. The first, minimize introduced local advice violations (MILAV) guarantees that some plan will be produced but may ignore some advice that could be satisfiable. During search, MILAV performs a one-step look-ahead for each task node ranking the impact of all possible choices (i.e. operator selection, variable instantiations) according to the number of new advice violations that each will introduce. Search control then uses this ranking to select options that minimize introduced advice violations. Unfortunately, locally optimal decisions made via this heuristic may not be globally optimal as there is only a one-step lookahead.

The second technique for partial advice satisfaction is Local Maxima Search (LMS). This algorithm selects a seed solution that satisfies a selected subset of user advice and employs a hill-climbing algorithm to iteratively improve on the current best solution by adding a single new piece of advice at a time. This continues until no more advice can be added that will lead to a complete plan. At that point a new seed node is chosen and the process repeats. Thus, LMS embodies an anytime approach that will yield plans of increasing advice satisfaction quality as more time is allotted. Several optimizations for seed selection have been tested. Either of these approaches may be workable with the types of narrative advice solicited by Bowman.

Research Status

A first phase of Bowman has been implemented. It is part of the Zócalo suite of planning tools available through NCSU at http://zocalo.csc.ncsu.edu. Bowman is used with Zócalo to define planning domains and planning problems using classical constructs, generate plans, and navigate through plan spaces and individual plans. We are in the process of elaborating the domain representations employed by Bowman and Zócalo to address some of the unmet challenges of interactive narrative. These extensions include many of the constructs in DEF, narrative macro support, support for relaxable custom heuristics, and some additional techniques unique to the Mimesis strategies of narrative mediation.

This paper has advocated a "generate-and-test" approach to exploring the preferences authors of interactive may want to use to describe plan quality. To lessen the burden of the "generate-and-test" methodology we have focused a lot of attention on methods to expedite the author's understanding of plan qualities, and plan space characteristics. We are using scaled vector graphic (SVG) images of individual plan the plan space and individual plans that can be navigated and queried interactively.

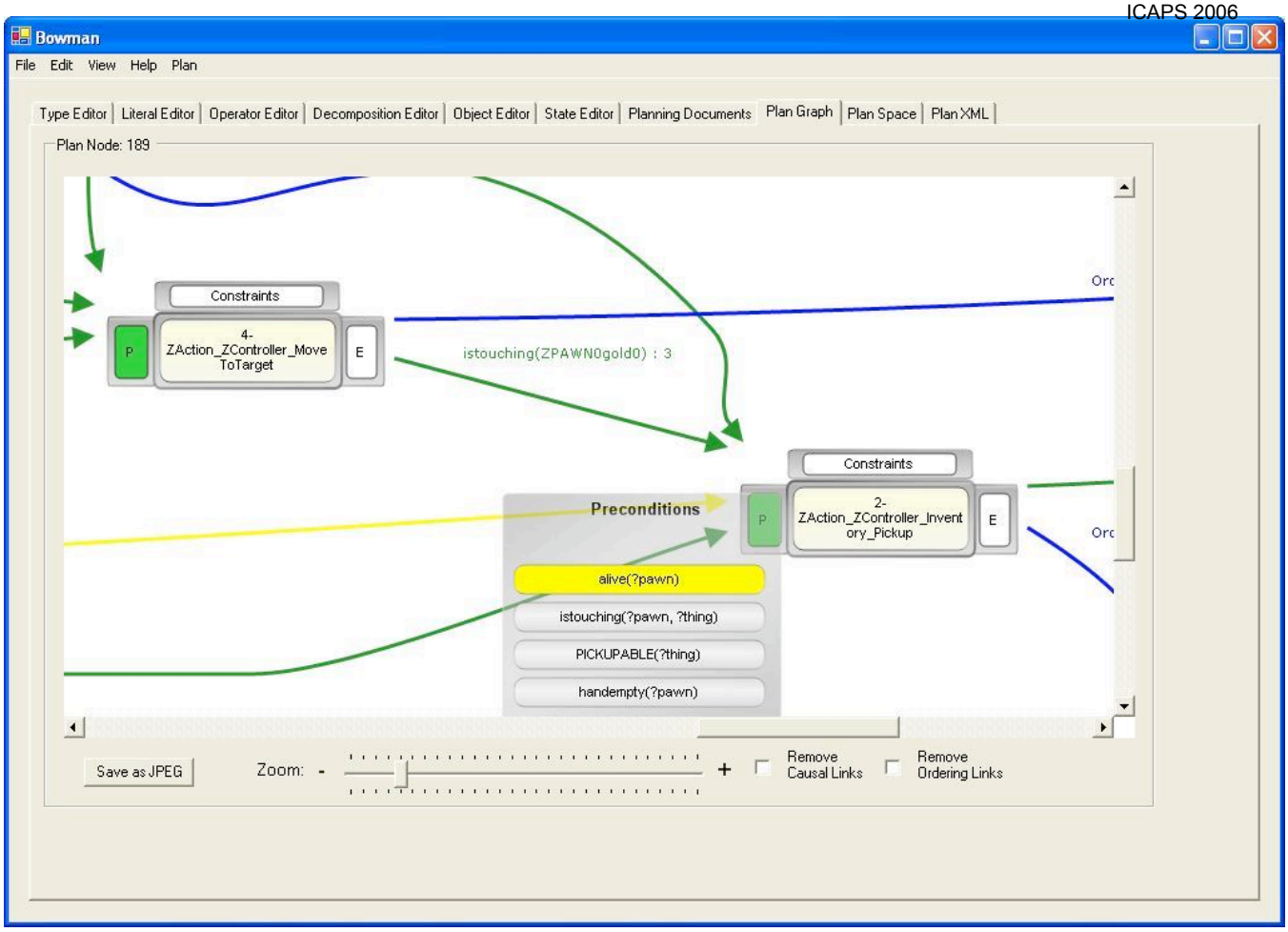

Figure 2: Bowman – Plan Node Analysis

For example, as shown in figure 2, each step in the plan is represented by a collection of four rounded rectangles. The center rectangle is labeled with the step number and operator name. The smaller rectangle on the left side of each step labeled with the letter "P" contains the preconditions of the step, and the rectangle on the right labeled with an "E" contains the effects. 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 (shown) containing each of the preconditions in the plan. As the user moves the mouse down over each precondition, it is highlighted in yellow ("alive(?pawn)" in the figure), as is the causal link which establishes that condition.

Authors can enter this plan node analysis mode from the plan space view simply by clicking on a rectangle for a individual plan node. We continue to add features to enhance authors' understanding of the relationship between the domain, the plan space and individual plans.

Conclusion

This paper introduced a general planning domain metatheory called DEF and a general plan-authoring interface called Bowman, currently under development at North Carolina State University. These tools are being used to support author-preference realization in interactive narrative. We have significant work yet to do including an evaluation of the effectiveness of the tools for non-technical authors, and potential expansion to work with other planners and PDDL input and output descriptions.

Acknowledgements

The authors wish to thank the members of the Liquid Narrative Group at NC State University for their assistance in developing Zócalo. This is material is based upon work supported under a National Science Foundation Graduate Research Fellowship and National Science CAREER award 0092586.

References

Allen, J. F., and Ferguson, G. 2002. Human-Machine collaborative planning. *In Proceedings of the Third International Workshop on Planning and Scheduling for Space*.

Aylett, R.S. 2000. Emergent Narrative, Social Immersion And "Storification" *Proceedings, Narrative Interaction for Learning Environments*, Edinburgh.

Brafman, R. and Chernyavsky, Y. 2005. Planning with Goal Preferences and Constraints. In *Proceedings of ICAPS-05.*

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.

Gerevini, A., and Long, D. 2005. Plan Constraints and Preferences in PDDL3. T*echnical Report R.T. 2005-08-47*, Dipartimento di Elettronica per l'Automazione, Università degli Studi di Brescia, via Branze 38, 25123 Brescia, Italy.

Myers, K. 2000a. 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, 2000b. K. Planning With Conflicting Advice. In *Proceedings of the Fifth International Conference on AI Planning Systems* (AIPS 2000).

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.

Myers, K. L. 2005. Metatheoretic Plan Summarization And Comparison. In *Proceedings of the ICAPS-05 Workshop on Mixed-initiative Planning and Scheduling* Monterey, CA.

Nau, D. S. 2005. May All Your Plans Succeed! *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, July 2005. Invited talk.

Prestwich, S., Rossi, F., Venable, K. B., and Walsh, T. Constraint-based Preferential Optimization* In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05).* Morgan Kaufmann, 2005.

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)*.

Tate, A. 1977. Generating Project Networks. In *Proceedings of IJCAI-77*, pp. 888-893.

Thomas, J. and Young, R. M. 2006. Author in the Loop: Using Mixed-Initiative Planning to Improve Interactive Narrative. In *Proceedings of the ICAPS-06 Workshop on AI Planning for Computer Games and Synthetic Characters,* Cumbria, UK.

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 Soft Regular Constraints

Alessandro Zanarini and **Gilles Pesant**
Département de génie informatique
École Polytechnique de Montréal
C.P. 6079, succ. Centre-ville, Montreal, Canada H3C 3A7
{azanarini,pesant}@crt.umontreal.ca

Michela Milano
D.E.I.S., Università di Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
mmilano@deis.unibo.it

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 give a brief background on Constraint Programming and the regular constraint. In Section 3 we introduce the new encoding and the planner. In Section 4 we show the soft version of the planner that can handle preferences on the goals and Section 5 shows the benefits that the soft planner can bring also to speed up the search. In Section 6 we give some experimental results. Finally, in Section 7 conclusions are given.

Background

Constraint Programming

Let $X = x_1, x_2, \ldots, x_k$ be a sequence of variables with respective domains $D_1, D_2, \ldots, D_k$. We denote $D_X = \bigcup_{1 \leq i \leq n} D_i$. A *constraint* C on X is defined as a subset of the Cartesian product of the domains of the variables in X, i.e. $C \subseteq D_1 \times D_2 \times \cdots \times D_k$. A tuple $(d_1, \ldots, d_k) \in C$ is called a *solution* to C. A value $d \in D_i$ for some $i = 1, \ldots, k$ is *inconsistent* with respect to C if it does not belong to a tuple of C, otherwise it is *consistent*. C is *inconsistent* if it does not contain a solution. Otherwise, C is called *consistent*. A constraint is called a *binary constraint* if it is defined on two variables. If it is defined on more than two variables, we call C a *global constraint*.

The solution process of constraint programming interleaves *constraint propagation* and *search*. The search process essentially consists of enumerating all possible variable-value combinations, until we find a solution or prove that none exists. We say that this process constructs a *search tree*. To reduce the exponential number of combinations, *constraint propagation* is applied to each node of the search tree: Given the current domains and a constraint C, remove domain values that do not belong to a solution to C. This is repeated for all constraints until no more domain value can be removed. Very efficient algorithms have been developped to remove such values from constraints, exploiting their structure (see (Apt 2003) and (Milano & Trick 2003) for further explanation).

Automaton and Regular Constraint

A deterministic finite automaton (DFA) may be described by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ is an alphabet, $\delta : Q \times \Sigma \rightarrow Q$ is a partial transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final (or accepting) states. Given an input string, the automaton starts in the initial state q_0 and processes the string

one symbol at a time, applying the transition function δ at each step to update the current state. The string is accepted if and only if the last state reached belongs to the set of final states F. The languages recognized by DFA's are precisely regular languages.

A $Regular$ constraint (Pesant 2004) is specified using a deterministic finite automaton that describes the regular language to which the sequence of values taken by the sequence of variables must belong. That automaton is then unfolded into a layered directed graph where vertices of a layer correspond to states of the automaton and arcs represent variable-value pairs. This graph has the property that paths from the first layer to the last are in one-to-one correspondence with solutions of the constraint. The existence of a path through a given arc thus constitutes a support for the corresponding variable-value pair.

As the cost-variant of the $Regular$ constraint, $CostRegular(X, \Pi, z, C)$ holds if the values taken by the sequence of finite domain variables X spell out a word belonging to the regular language associated to the deterministic finite automaton Π, and if z, a bounded-domain continuous variable, is equal to the sum of the variable-value assignment costs given by cost matrix C. Instead of simply maintaining paths, the filtering algorithm for $CostRegular$ must consider the length of these paths, defined as the sum of the costs of individual arcs, representing $\langle$ variable, value, state $\rangle$ tuples, whose costs are given by cost matrix C. Supports do not come from just any path but rather from a path whose length falls within the domain of z. To check this efficiently, it is sufficient to compute and maintain shortest and longest paths from the first layer to every vertex and from every vertex to the last layer: if the shortest way to build a path through a given arc is larger than the upper limit of the interval for z, the arc cannot participate in a solution and can thus be removed; if the longest way to build a path through a given arc is smaller than the lower limit of that interval, the arc can again be removed. In this way, domain consistency is achieved for the variables of X. The domain of z can also be trimmed using the shortest and longest paths from the first to the last layer.

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 a, b, c shows respectively the automata for block a, b and c (note that the automata slightly differ from the following formal definition for a reason that will be clearer later).

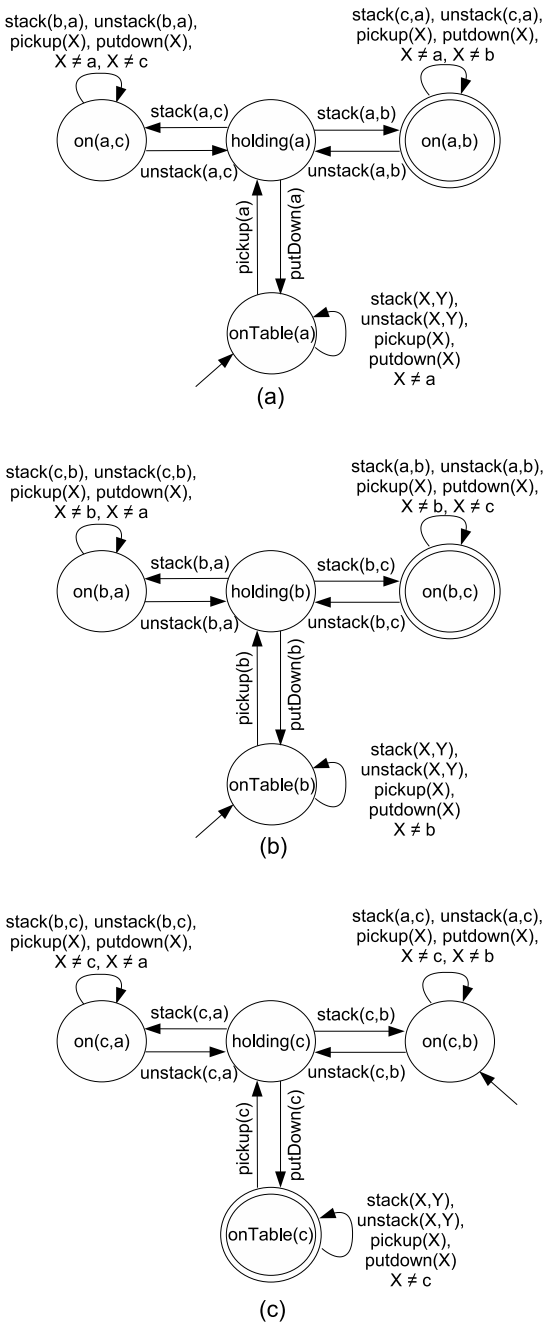

Figure 1: Automata of blocks world example.

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 repre-

sents 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.

The example presented above about blocks world shows the automata already minimized. Following the example, in the first time step actions like $\texttt{stack}(\texttt{b}, \texttt{X})$ are not allowed because they are pruned by the automaton c. Suppose now that $X_1 = unstack(c, b)$: the automaton c will reach the state $\texttt{holding}(\texttt{c})$ while the automata a and b will remain in the initial state. After this instantiation, the automaton c will filter from the domain of the variable X_2 all the actions except $\texttt{stack}(\texttt{c}, \texttt{X})$ and $\texttt{putdown}(\texttt{c})$. As you can note, the sequence $\texttt{unstack}(\texttt{c}, \texttt{b}), \texttt{putdown}(\texttt{c}), \texttt{pickup}(\texttt{b}), \texttt{stack}(\texttt{b}, \texttt{c})$, $\texttt{pickup}(\texttt{a}), \texttt{stack}(\texttt{a}, \texttt{b})$ is correctly recognized by the three 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).

Consider again the example about the blocks world domain. Figure shows the automata with the associated fake actions; on the transition T_{fake} the two different violation costs are shown. Suppose that we are searching for a plan of length 4 and X_1, X_2, X_3, X_4 are instantiated respectively to $\texttt{unstack(c, b)}, \texttt{putdown(c)}, \texttt{pickup(b)}, \texttt{stack(b, c)}$. The automaton of the block a will be in the state $\texttt{onTable(a)}$, the one for the block b in the state $\texttt{on(b, c)}$ and the one for the block c in $\texttt{onTable(c)}$. The final fake action will not change the state for the automata b and c but it will allow to find a solution since it will lead the automaton a to the final state. According to the two proposed violation functions we will get a violation equal to 1 (for PDDL-like violation) or equal to 2 if we consider the distance (we have to perform two more actions to reach the final state).
Once the violation functions are defined, then it is possible to define preferences among the goals using the objective function and/or posting constraints directly on the single violation variables. In the example, let V_a, V_b and V_c be the three violation variables and let the upper bound on the plan length be equal to 5; if we define the total violation TV as $TV = V_a + 2 * V_b + 2 * V_c$ (it is preferable to have $\texttt{on(b, c)}, \texttt{onTable(c)}$ satisfied than $\texttt{on(a, b)}$) then we will get as the final plan $\texttt{unstack(c, b)}, \texttt{putdown(c)}, \texttt{pickup(b)}, \texttt{stack(b, c)}, \texttt{pickup(a)}$. Note that $V_a = 1$, $V_b = 0$ and $V_c = 0$ hence $TV = 1$; not satisfying the goals for the block c or b will bring an increase of the total violation by at least 2 hence the solution found is optimal.

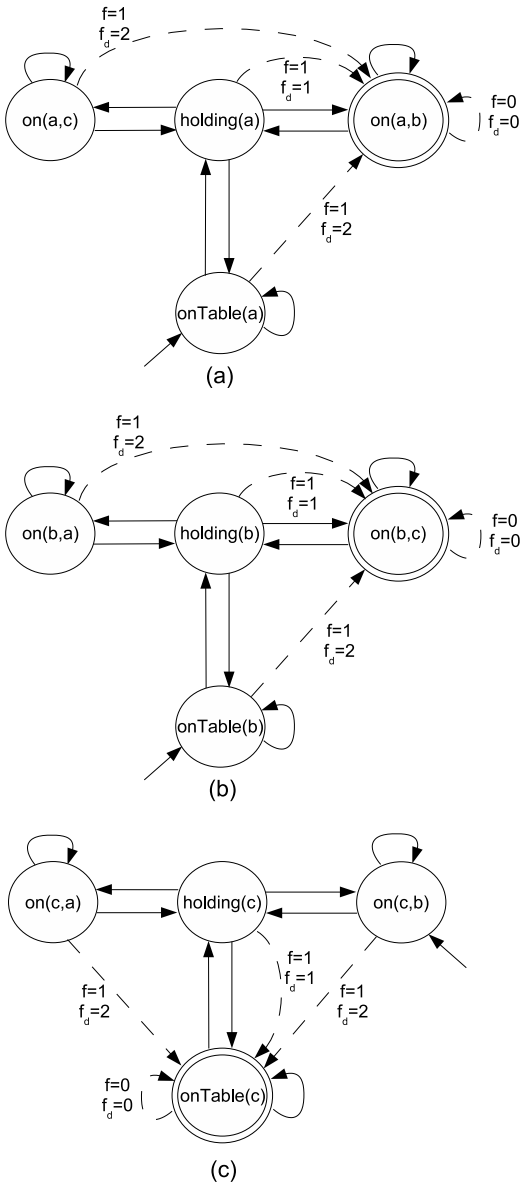

Figure 2: Soft automata for blocks world. The fake actions are shown with dashed lines.

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 itera-

tion, 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:

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.

	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

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

Apt, K. 2003. *Principles of Constraint Programming*. Cambridge University Press.

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.

Milano, M., and Trick, M. 2003. *Constraint and Integer Programming - Toward a Unified Methodology*. Kluwer Academic Publishers.

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.

Pesant, G. 2004. A Regular Language Membership Constraint for Finite Sequences of Variables. In *Principles and Practice of Constraint Programming – CP-2004: Proceedings of the Tenth International Conference*. Springer-Verlag LNCS 3258. 482–495.

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.