AIPS Workshop on Planning in Temporal Domains Foreword

Maria Fox and Alex Coddington
University of Durham, UK
maria.fox@durham.ac.uk a.m.coddington@durham.ac.uk

Temporal Planning has recently become a major focus of research activity in AI Planning. There are now a number of different approaches to handling domains in which time is an important, and sometimes scarce, resource. These approaches, which include Graphplan extensions, model checking techiques, hierarchical decomposition, heuristic strategies and reasoning about temporal networks, are capable of planning with durative actions, temporally extended goals, temporal windows and other features of time-critical planning domains.

This workshop considers a range of these approaches and some of the important technical problems addressed in their implementation. These include:

1. Modelling time – how are temporal aspects of a domain best modelled?
2. Handling concurrency – what semantic constraints are imposed, on exploitation of concurrent activity?
3. Validating plans – automated verification of plan correctness is potentially much harder in temporal domains.
4. Managing continuous change.

The workshop complements the Third International Planning Competition, which is reaching its conclusion at AIPS in parallel with the workshop. Temporal planning has been the key theme of the competition this year, and several of the presenters at the workshop are also participants in the competition. We hope that the workshop will help to focus attention on what has been achieved so far as well as on some of the unresolved challenges of temporal planning.

Planning for Temporal Domains

Maria Fox and Alex Coddington
University of Durham, UK

Contents

Temporal Planning through Mixed Integer Programming

Yannis Dimopoulos
Computer Science Department
University of Cyprus, Nicosia, Cyprus
yannis@cs.ucy.ac.cy

Alfonso Gerevini
Dipartimento di Elettronica per l'Automazione
Università degli Studi di Brescia, Brescia, Italia
gerevini@ing.unibs.it

Abstract

Temporal planning is an important problem, as in many real world planning domains actions have different durations and the goals should be achieved by a specified deadline, or as soon as possible. This paper presents a novel approach to temporal planning that is based on Mixed Integer Programming. In the new framework, a temporal planning domain is modeled by two sets of linear inequalities. The first set involves integer variables and is a Graphplan-like encoding of a simplification of the original problem where the duration of the actions is ignored. The second set involves both integer and real valued variables, and models the temporal aspects of the problem. The two sets interact through the common integer variables, and their combination can be solved by using known Mixed Integer Programming techniques. The new method aims at generating good solutions quickly, under different minimization objectives. Preliminary experimental results illustrate the effectiveness of our approach.

Introduction

Over the last years there has been a remarkable progress in solving STRIPS planning problems (Weld 1999). However, for many interesting applications the STRIPS language is inadequate. We need to solve problems that involve actions that have different durations, consume resources and must be executed by certain deadlines. We need to generate plans that optimize complex combinations of different criteria, including completion time, resource utilization, action costs and others.

Such advanced application domains involve *numeric* variables, constraints, and complex objective functions. Mixed Integer Programming (MIP), and its language of linear inequalities, can easily accommodate these key features and therefore seems to provide a rich representational framework for such applications. However, there are relatively few works that apply IP to planning problems with numeric constraints (Wolfman & Weld 1999), (Kautz & Walser 1999).

In the original STRIPS language actions are instantaneous and time is implicitly represented. Several domain-independent systems have been proposed to handle a richer notion of time (e.g., (Allen 1991; Tsang 1986; Penberthy & Weld 1994)). However, these approaches scale up poorly, and can deal with only very simple problems. The success of recent approaches to STRIPS planning, such as planning graphs and heuristic search, has motivated the application of these techniques to temporal planning. For instance, TGP (Smith & Weld 1999) uses a generalization of Graphplan mutual exclusion reasoning to handle actions with durations, while TP4 (Haslum & Geffner 2001) applies heuristic search to solve problems with action durations and resources.

In this paper we apply MIP to temporal planning, by developing models for domains that contain actions with different durations. Our approach decomposes a planning domain into two interacting sets of linear inequalities referred to as the *logical* and the *temporal* part respectively.

The logical part is an encoding of the planning graph of the STRIPS problem that is obtained from the original problem by ignoring action durations. For this encoding we use the method developed by (Vossen *et al.* 1999) and improved by (Dimopoulos 2001). This approach formulates the planning graph of a STRIPS planning domain as an Integer Programming problem and then uses branch-and-bound for solution extraction.

The temporal part associates with every action a real-valued variable that represents the start time of the action. The linear inequalities of this part ensure the correctness of the start times that are assigned to the actions taking into account their durations.

The combination of the logical and temporal parts can be solved in a uniform, integrated way by using well-known MIP techniques like CPLEX. Since the two sets of constraints interact, this is much more effective than a naive approach in which first we iteratively solve the logical part, and then we check whether this potential solution satisfies the constraints of the temporal part, until we find a valid temporal plan (i.e., a solution for both the sets of constraints).

In order to increase the efficiency of the representation, the structure of the domain is exploited. In particular, we propose some techniques that use ideas from domain analysis tools (Fox & Long 1998; Gerevini & Schubert 1998; 2000) to reduce the number of constraints and variables of the temporal part of a planning problem, leading to stronger MIP formulations.

In contrast to TGP and TP4 that generate plans of minimum duration, the new approach does not provide optimality guarantees. However, apart from the overall duration of the plan, the MIP models can easily accommodate differ-

ent optimization criteria and any constraint that can be expressed as linear inequalities.

The rest of the paper is organized as follows. First we briefly give the necessary background; then we present our basic temporal model, i.e., the set of inequalities forming the temporal part of the problem encoding (while for the logical part we will use some known encoding); then we describe how planning problems can be solved in the new approach; then we give some preliminary experimental results; finally, we give our conclusions and mention future work.

Preliminaries

The planning language we consider is propositional STRIPS extended with time. Actions have (positive) preconditions, (add and delete) effects and constant duration that can be any real number. Our assumptions for the execution of actions are the same as those used in (Smith & Weld 1999) and (Haslum & Geffner 2001):

- The preconditions of an action must hold in the beginning and during the execution of the action.
- Add and delete effects take place at some point during the execution of an action and can only be used only at the end of the execution of the action.

The above assumptions require that the preconditions and effects of an actions are protected during their execution. Therefore, the linear inequalities of the MIP models we develop, enforce that actions with contradictory effects or with contradictions between their effects and preconditions do not overlap in time.

A MIP problem (Wolsey 1998) comprises of a mixture of real-valued and integer variables, a set of linear inequalities on these variables, and an objective function. The models developed in the paper are 0/1 MIP models i.e., integer variables can only assume the values 0 and 1. We assume that the reader is familiar with the basics of MIP.

Our modeling techniques utilize some ideas developed in the context of the domain analysis tool DISCOPLAN (Gerevini & Schubert 2000). In particular, they exploit *single-valuedness* (sv) constraints and binary XOR-constraints, which are automatically inferred by DISCOPLAN. An sv-constraint states that the value of a certain predicate argument is unique for any given values of the remaining arguments. An example of an sv-constraint in blocks-world is $on(x, *y)$, stating that any object is on at most one thing ("$*$" indicates the single-valued argument). An example of XOR-constraint is (XOR $on(x, y)$ $clear(y)$) stating that any object is either clear or has something on it.

The Basic Temporal Model

Let P be a temporal planning problem and let P^S be its STRIPS simplification (i.e., actions are instantaneous). Assume that P^S is solved by an algorithm that builds and searches its planning graph. For each action (instantiated operator) A of the problem and each level l of the graph, there is a corresponding node in the planning graph, denoted by A^l, that can be understood as a binary variable. Assume that plans are generated in the form of value assignments to the action variables such that the value 1 is assigned to variable A^l iff action A at level l is included in the plan.

Our goal now is to find a set of linear inequalities for problem P that, given a plan for P^S, assign start times to the actions in the plan. The inequalities that model the temporal part of P involve, apart from the binary action variables A^l, a set of real valued variables as follows. For every action A and level l of the graph, we introduce a variable A^l_{st} that represents the start time of action A at level l, i.e., the time when the execution of the action starts. Similarly, for every fluent f of the domain, and every level l of the graph, we introduce a variable f^l_{st} that represents the time at which fluent f becomes true. In the following, $dur(A)$ denotes the duration of action A, which is a real number.

The first set of inequalities of the temporal model is used to enforce the constraint that actions can not start before their preconditions become true. If f is a precondition of A, the following set of inequalities is included in the model, for each level l of the planning graph

$$(1) \quad A^l_{st} \geq f^l_{st}$$

The next set of inequalities represents the contraint that a fluent can become true after the execution of an action that adds it. Therefore, if f is an add effect of action A the model includes (A^l is a 0/1 variable)

$$(2) \quad f^{l+1}_{st} \geq A^l_{st} + dur(B) \cdot A^l$$

Note that if fluent f was true at level l before the execution of action A, the above constraint causes f^{l+1}_{st} to take a value that can not be smaller than the end time of A. In combination with the previous constraint (1) this causes all actions that have f as their precondition, and appear in levels higher than l, to start after the end time of A. Although there is a way to overcome this restriction, in this paper we assume that there is no reachable state S such that a fluent f is true in S, and an action that adds f can be executed in S (the blocks world and Rocket are examples of such domains).

The temporal model prevents contradictory actions from overlaping in time. For every pair of actions A and B such that A deletes the preconditions of B, the following constraints are added to the model

$$(3) \quad A^{l+1}_{st} \geq B^l_{st} + dur(B) \cdot B^l$$

For every pair of actions A and B such that A deletes an add effect of B, the model includes the following inequalities

$$(4.1) \quad A^{l+1}_{st} \geq B^l_{st} + dur(B) \cdot B^l$$

$$(4.2) \quad B^{l+1}_{st} \geq A^l_{st} + dur(B) \cdot A^l$$

For every fluent f, the following constraints propagate the start time of f through the levels of the planning graph

$$(5) \quad f^{l+1}_{st} \geq f^l_{st}$$

Similarly, for each action A, its start time is propagated through the following constraints

$$(6) \quad A^{l+1}_{st} \geq A^l_{st}$$

Finally, plans start at time 1, which is stated by

$$(7) \quad f^1_{st} \geq 1$$

The following theorem states that any valid Graphplan-style plan satisfying constraints (1)–(7) is temporally sound.

Theorem 1 (Soundness) *For every action A that is in a plan and satisfies the constraints above, the following holds: If p is precondition of A, then there exists an action A_p such that p is an add effect of A_p, and*

a) $st(A_p) + dur(A_p) \leq st(A)$;

b) *if B is an action that has p as a delete effect, $st(B) + dur(B) \leq st(A_p)$ or $st(A) + dur(A) \leq st(B)$.*

Furthermore, if q is an add effect of A, then

c) *for every action C in the plan that deletes q, $st(C) + dur(C) \leq st(A)$ or $st(A) + dur(A) \leq st(C)$.*

Proof: The correctness of the underlying non-temporal planning algorithm guarantees that if A is an action in the plan and p is a precondition of A, there will be another action A_p in the plan that has p as an add effect. Moreover, if l is the level of A and l' the level of A_p, it must be the case that $l' < l$, and there is no action that deletes p and occurs in any level l'' with $l'' \geq l'$ and $l'' \leq l$.

In order to prove (a), observe that because of inequality (2), $st(p^{l'+1}) \geq st(A_p^{l'}) + dur(A_p)$. Moreover, the set of inequalities (5) will enforce $st(p^l) \geq st(A_p^{l'}) + dur(A_p)$ (because $l' < l$). Finally, because of inequality (1), we have $st(A^l) \geq st(p^l)$, which together with the previous constraint give $st(A^l) \geq st(A_p^{l'}) + dur(A_p)$.

In order to prove (b), let B be any action that has p as delete effect, and let l'' be its level. As noted earlier, the correctness of the non-temporal plan implies that either $l'' < l'$ or $l'' > l$ must hold. Assume $l'' < l'$. Since B deletes an add effect of A_p, inequality (4.2) will enforce the constraint $st(A_p^{l''+1}) \geq st(B^{l''}) + dur(B)$. This constraint, together with the set of inequalities (6), impose $st(A_p^{l'}) \geq st(B^{l''}) + dur(B)$. Assume that $l'' > l$. Since B deletes a precondition of A, inequality (3) will enforce $st(B^{l+1}) \geq st(A^l) + dur(A)$ which together with inequalities (6) ensure that $st(B^{l''}) \geq st(A^l) + dur(A)$. Thus it is indeed the case that $st(B) + dur(B) \leq st(A_p)$ or $st(A) + dur(A) \leq st(B)$ is true.

In order to prove (c), assume that an action $C^{l'}$ in the plan deletes q, and that q is an add effect of A^l. Clearly $l' < l$ or $l' > l$ (because of the correctness of the non-temporal plan). If $l' < l$, since constraints (4) will enforce $st(A^{l'+1}) \geq st(C^{l'}) + dur(C)$, by constraints (6) we have $st(A^l) \geq st(C^{l'}) + dur(C)$. Similarly, if $l' > l$, since constraints (4) ensure that $st(C^{l+1}) \geq st(A^l) + dur(A)$, by constraints (6) we have $st(C^{l'}) \geq st(A^l) + dur(A)$. Therefore, it is indeed the case that $st(C) + dur(C) \leq st(A)$ or $st(A) + dur(A) \leq st(C)$ is true. □

Improved Temporal Modeling

When we model a problem in terms of a set of linear inequalities, the number of constraints and variables that are present in the set is a practically important issue, since this can significantly affect the performance of the solver. The above model of temporal planning problems can generate a large number of constraints and variables, but it can be substantially improved if certain features of the domain structure are taken into account. The improvements that we will discuss aim at reducing the number of temporal constraints, as well as the number of temporal variables that are required to correctly model a planning domain. The reduction of the number of temporal constraints is based on the notions of *argument persistence*, *persistent pair of fluents* and *strong interference* that we give for binary fluents.[1]

Definition 2 (Argument persistence) *Let f be a binary fluent such that $f(x, *y)$ holds, and every action (instantiated operator) that has an instance of f in its add effects has another instance of f with the same first argument but different second argument in its preconditions. We say that f is* persistent *on its first argument.*

Persistence on the second argument of a fluent is defined similarly. We now define argument persistence on the first argument for a pair of binary fluents. Persistence on other arguments, or between a binary and a unary fluent can be defined in a similar way. In the following, X, Y, Z and W indicate any constant, x, y and z universally quantified variables, x, y and z operator parameters. Moreover, we assume that no action has any literal as both precondition and add effect.

Definition 3 (Persistent pair of fluents) *Let $f1$ and $f2$ be two binary fluents such that $f1(x, *y)$, $f2(x, *z)$ and* (XOR *$f1(x, y)$, $f2(x, z)$) hold. We say that $f1$ and $f2$ is a* persistent pair *of fluents on their first argument, if every action that has $f1(X, Y)$ or $f2(X, Z)$ as an add effect, also has $f1(X, W)$ or $f2(X, W')$ as a precondition, where $W \neq Y, Z$ and $W' \neq Y, Z$.*

Now we can define the notion of *strong interference* between actions that will be used to improve the temporal model.[2]

Definition 4 (Strong interference) *A pair of actions A and B* strongly interfere *if*

- *f is a fluent such that $f(x, *y)$ holds, A has a precondition $f(X, Y)$, B has a precondition $f(X, Z)$, and either $Y \neq Z$, or $Y = Z$ and A and B have an instance of f as add effect with X as the same first argument and different second argument;* or
- *$f1, f2$ is a persistent pair of fluents on their first argument, and A has a precondition $f1(X, Y)$ and B a precondition $f2(X, Z)$, or (a) they have a common precondition $f1(X, Y)$ or $f2(X, Y)$ which they both delete and (b) they have an instance of $f1$ or $f2$ as add effect with X as the first argument and different second argument (if they are instances of the same fluent).*

For instance, under the assumptions of the previous definition, A and B strongly interfere when

- $f1(a, b)$ is a precondition of A and $f2(a, c)$ is a precondition of B, or

[1] We restrict our analysis to binary fluents, which are the most common in many existing domain formalizations. Work on an extension to fluents of higher arity is in progress.

[2] For the sake of clarity, the definition is given for the case of persistence on the first argument of binary fluents, but it can be easily generalized to the cases where persistence is on the second argument, and f, $f1$, $f2$ are unary fluents.

- when $f1(a,b)$ is a precondition of A and B, $\neg f1(a,b)$ is an effect of A and B, $f2(a,b)$ is an effect of A, and $f1(a,c)$ an effect of B.

It turns out that for pairs of actions that strongly interfere, all constraints of the form (3) and (4) can be omitted from the temporal model, because actions that strongly interfere can not overlap in time. We call such models *reduced temporal models*. An example of reduced model is given after the following theorem stating soundness of reduced temporal models.

Theorem 5 *For any two actions A and B that strongly interfere and are both included in a plan, a reduced temporal model satisfies either $st(A) \geq st(B) + dur(B)$ or $st(B) \geq st(A) + dur(A)$.*

Proof (sketch): Assume that both actions A and B are included in a plan generated by the underlying non-temporal planning algorithm, and let l be the level of A and l' the level of B. Since by Definition 4 the two actions have mutually exclusive preconditions or mutually exclusive effects the soundness of the non-temporal algorithm implies $l' \neq l$. We first consider the case in which $l < l'$, and $f(X,a)$ is a precondition of A and $f(X,b)$ a precondition of B, where f is a persistent fluent on its first argument, and a, b any pair of different constants.[3] Assume that A has $f(X,b)$ as an add effect. Consequently, because of constraint (2), $st(f(X,b)^{l+1}) \geq st(A^l) + dur(A)$ will hold. Then the set of constraints (5) will enforce that $st(f(X,b)^{l'}) \geq A^l_{st} + dur(A)$. Since $f(X,b)$ is a precondition of B, constraints (1) will enforce $st(B^{l'}) \geq st(f(X,b)^{l'})$ and therefore $st(B^{l'}) \geq st(A^l) + dur(A)$.
Assume now that either A does not have any add effect of the form $f(X,Y)$, or it has an add effect $f(X,Y)$ with $Y \neq b$. Then there will be a sequence of actions $A_1, ..., A_n$ in the plan, at levels $l_1, ..., l_n$ respectively, with $n > 1$, $l_i > l$ and $l_i < l'$, such that $f(X,b_i)$ is an add effect of A_i and a precondition of A_{i+1} for some constant b_i, $b_n = b$ and $l_{i+1} > l_i$. Since f persists on its first argument, if $f(X,b_i)$ is an add effect of A_i, then, for some d, $f(X,d)$ must be a precondition of A_i. Hence, it must be the case that $d = b_{i-1}$, where b_{i-1} is an add effect of action A_{i-1}. Therefore, the combination of constraints (1), (2) and (5) implies, along the sequence of actions $A_1, .., A_n$, a set of constraints $st(A^{l_{i+1}}_{i+1}) \geq st(A^{l_i}_i) + dur(A_i)$, for every $i \geq 1$, which implies that $st(A^{l_n}_n) \geq st(A^{l_1}_1) + \sum_{1 \leq i \leq n-1} dur(A_i)$. Then, by constraints (6) $st(B^{l'}) \geq st(A^{l_1}_1) + \sum_{1 \leq i \leq n-1} dur(A_i)$.

We now consider the temporal relation between actions A and A_1. Assume first that action A has an add effect $f(X,c)$ for some constant $c \neq b$. Then, $f(X,c)$ must be a precondition of A_1. Therefore $st(A^{l_1}_1) \geq st(A^l) + dur(A)$. Assume now that action A does not have any instance of fluent f in its add effects with X as its first argument. Then, by Definition 2 and the assumed soundness of the non-temporal plan, $f(X,a)$ must be a precondition of A_1, and A, A_1 do not strongly interfere, provided that they do not have other preconditions or effects that could cause strong interference. Moreover, note that $f(X,a)$ must be a delete effect of A_1. Since A_1 deletes a precondition of A, constraint (4) applies, and $st(A^{l_1}_1) \geq st(A^l) + dur(A)$ must hold. Hence, it is again the case that $st(A^{l_1}_1) \geq st(A^l) + dur(A)$. This constraint, together with $st(B^{l'}) \geq st(A^{l_1}_1) + \sum_{1 \leq i \leq n-1} dur(A_i)$, give us $st(B^{l'}) \geq st(A^l) + dur(A)$.

Assume now that actions A and B, have both precondition $f(X,c)$ and that A adds $f(X,a)$ and B adds $f(X,b)$, for some different constants a, b and c. Then again there will be a sequence of actions $A_1, ..., A_n$ in the plan, such that A_n adds $f(X,c)$, and by an analogous argument the constraints $st(A_1) \geq st(A^l) + dur(A)$ and $st(B^{l'}) \geq st(A^{l_1}_1) + \sum_{1 \leq i \leq n-1} dur(A_i)$ will hold, enforcing $st(B^{l'}) \geq st(A^l) + dur(A)$.

Now let again $l < l'$, and assume that A has a precondition $f1(X,a)$ and B has a precondition $f2(X,b)$, where $f1$ and $f2$ is a persistent pair on the first argument, and a and b is any pair of different constants. If A has $f2(X,b)$ as an add effect, then using arguments similar to the one given above we can prove that $st(B^{l'}) \geq st(A^l) + dur(A)$. Otherwise, there will be a sequence of actions $A_1, ..., A_n$ in the plan, at levels $l_1, ..., l_n$ respectively, with $n > 1$, $l_i > l$ and $l_i < l'$, such that $f1(X,b_i)$ or $f2(X,b_i)$ is an add effect of A_i for some constant b_i, and $f_2(X,b)$ is an add effect of A_n. By arguments similar to those used above, we can prove that $st(B^{l'}) \geq st(A^{l_1}_1) + \sum_{1 \leq i \leq n-1} dur(A_i)$ and $st(A^{l_1}_1) \geq st(A^l) + dur(A)$ will hold. Thus $st(B^{l'}) \geq st(A^l) + dur(A)$ will also hold.

Finally, if $l > l'$, we can use symmetric arguments to those above to prove $st(A^l) \geq st(B^{l'}) + dur(B)$. □

We now discuss an improvement that reduces the number of temporal variables in the model of a problem. More importantly, it achieves more effective propagation of the start times of actions and fluents.

Let f be a binary fluent for which the sv-constraint $f(x, *y)$ holds. We can replace in the model all temporal variables $st(f(X,Y)^l)$ that refer to the different values of Y and same value of X with one new variable $st(fn(X)^l)$, for each level l. Similarly, if $f(*x, y)$ holds. Moreover, if $f1(x,y)$ and $f2(x,z)$ are two fluents related with a XOR constraint (XOR $f1(x,y)$ $f2(x,z)$), we can replace their corresponding temporal variables referring to the different values of y and z by, but same value X for x, with a single variable $st(f12(X)^l)$. Similarly, if (XOR $f1(y,x)$ $f2(z,x)$) or (XOR $f1(x,y)$ $f2(x)$) holds. We call this new set of variables *abstract temporal fluent variables*. Note that it can be the case that several different sv or XOR constraints hold for the same fluent, giving rise to different models depending on the particular abstract variables that are choosen. We handle such cases in an ad-hoc manner, but here we do not discuss this issue further.

In domains that involve operators with more than two arguments that can be instantiated by many objects, the technique of splitting action start time variables can be used.

[3] For clarity the proof is given considering only persistence on first arguments; generalization to persistence on different arguments is straightforward.

Let $A(\mathtt{x},\mathtt{y},\mathtt{z})$ be an operator such that, for every possible value of parameter $\mathtt{x}$, all actions that have different values for the pair of parameters $\mathtt{y}$ and $\mathtt{z}$ are mutually exclusive. We denote such an operator by $A(\mathtt{x},*\mathtt{y},*\mathtt{z})$. Moreover, assume that all preconditions and effects of $A(\mathtt{x},*\mathtt{y},*\mathtt{z})$ are unary or binary fluents, none of which has the pair $\mathtt{y},\mathtt{z}$ in its parameters. If this is the case, we can split each variable $st(A(X,Y,Z)^l)$ into two variables $st(A_1(X,Y)^l)$ and $st(A_2(X,Z)^l)$, and add the constraint $st(A_1(X,Y)^l) = st(A_2(X,Z)^l)$ to the model. In the constraints (1) and (2) of the temporal model, in which $st(A(X,Y,Z)^l)$ occurs along with a start time variable that refers to a fluent of the form $f(X)$, $f(Y)$ or $f(X,Y)$, variable $st(A(X,Y,Z)^l)$ is replaced by $st(A_1(X,Y)^l)$. Similarly, if the start time fluent variables that occurs in such a constraint is on a fluent of the form $f(X,Z)$, variable $st(A(X,Y,Z)^l)$ is replaced by $st(A_2(X,Z)^l)$. In constraints (3) and (4) one of $st(A_1(X,Y)^l)$ and $st(A_2(X,Z)^l)$ replaces variable $st(A(X,Y,Z)^l)$, depending on the fluent that gives rise to the conflict.

Let again $A(\mathtt{x},*\mathtt{y},*\mathtt{z})$ be an operator as defined above. It may be the case that the duration of the instances of $A(\mathtt{x},*\mathtt{y},*\mathtt{z})$ does not depend on all its parameters, but only on a subset of them. In the blocks world for example, it is possible that the duration of the move actions depends on the block that moves and the destination of the block, but not the origin of the block. Assume that the duration of a given operator $A(\mathtt{x},*\mathtt{y},*\mathtt{z})$ does not depend on the values of parameter $*\mathtt{z}$. Then, we can replace all occurences of $dur(A(X,Y,Z)) \cdot A(X,Y,Z)^l$ in constraints (1)–(4) of the temporal model by $dur(A(X,Y,Z)) \cdot \sum_z A(X,Y,z)^l$, where $\sum_z$ denoted the sum over all possible values of parameter $\mathtt{z}$. If the duration of $A(\mathtt{x},*\mathtt{y},*\mathtt{z})$ depends only on the values of parameter $\mathtt{x}$ we can replace $dur(A(X,Y,Z)) \cdot A(X,Y,Z)^l$ by $dur(A(X,Y,Z)) \cdot \sum_y \sum_z A(X,y,z)^l$. We call this technique, *compact duration modeling*. In some domains this technique combined with start time variable splitting may lead to tight MIP formulations.

Example: Consider the Rocket domain with the usual ld (load) and ul (unload) actions for packages and fl (fly) for airplanes. Assume that these actions have different durations. The basic temporal model for this domain includes all constraints that have been described in the previous section. Consider for example $ld(p1,pl1,l1)$, which represents the action of loading package $p1$ to plane $pl1$ at location $l1$. The constraints of type (1) for this action are $st(ld(p1,pl1,l1)^l) \geq st(at(p1,l1)^l)$ and $st(ld(p1,pl1,l1))^l \geq st(at(pl1,l1)^l)$ for each level l. There is one constraint of type (2) for each level l, namely $st(at(p1,l1)^{l+1}) \geq st(ld(p1,pl1,l1))^l + dur(ld(p1,pl1,l1)) \cdot x^l_{ld}$, where x^l_{ld} is a 0/1 variable that takes the value 1 if the action $ld(p1,pl1,loc1)$ is included in the plan, and 0 otherwise.

The temporal overlap of actions deleting preconditions of $ld(p1,pl1,l1)$ is prohibited by constraints (3), namely, $st(ld(p1,X,l1)^{l+1}) \geq st(ld(p1,pl1,l1))^l + dur(ld(p1,pl1,l1)) \cdot x^l_{ld}$ and $st(fly(pl1,l1,Y)^{l+1}) \geq st(ld(p1,pl1,l1))^l + dur(ld(p1,pl1,l1)) \cdot x^l_{ld}$, where X stands for any plane different from $pl1$ and Y for any location different from $l1$. Since $at(p1,l1)$ and $in(p1,Z)$ is a persistent pair for any plane Z, and the two load-actions of the first inequality strongly interfere (they share the precondition $at(p1,l1)$, but add different instances of in with $p1$ as first argument), by Theorem 5 the first constraint is not included in the reduced model. However, the second constraint is included, since the load and fly actions that are involved do not strongly interfere. The only action that deletes the add effect of $ld(p1,pl1,l1)$ is "blocked" through a pair of constraints of type (4), which are $st(ul(p1,pl1,l1)^{l+1}) \geq st(ld(p1,pl1,l1))^l + dur(ld(p1,pl1,l1)) \cdot x^l_{ld}$ and its symmetric. Note that by Theorem 5 these constraints are not included in the reduced model, because of the persistent pair $in(p1,pl1)$, $at(p1,l1)$ appearing in the preconditions of the two interfering actions.

Since $at(x,*y)$ holds for any plane x and location y, all occurrences of $st(at(p1,l1)^l)$ can be replaced by the variable $st1(at(p1)^l)$. Moreover, since $(\mathsf{XOR}\ at(x,y)\ in(x,y))$ holds, $st(at(p1,X)^l)$ and $st(at(p1,Y)^l)$ can be replaced by $st(atin(p1)^l)$.

Solving Planning Problems

When considered alone, the temporal model that we have described could find feasible start times for the actions of a plan that is produced by any algorithm solving planning graphs. This STRIPS planner would ignore completely the duration of the actions, and the temporal part would not need to know how the planner generates the plans. The two parts would be "glued" together through the 0/1 action variables that are shared by the two parts.

This seperation of the logical and the temporal part of a planning problem facilitates the use of a different algorithm for each of these parts, e.g., propositional satisfiability for the first and linear programming for the second, in an architecure similar to LPSAT (Wolfman & Weld 1999). However, in the approach taken here we represent both parts by a set of linear inequailities and use standard branch-and-bound on the *union* of the two parts. The potential benefit of such a unified algorithmic framework is the possibility of exploiting the strong interaction between the two parts, which may lead to extensive value propagation.

For the formulation of a logical part of a temporal planning problem as a set of linear inequalities, we use the method developed by (Vossen *et al.* 1999) and improved by (Dimopoulos 2001). This approach essentially translates the planning graph of a STRIPS problem into an Integer Programming model, and then uses branch-and-bound for solution extraction.

The overall duration of the plan is represented by the variable mks (for makespan) and a set of constraints of the form $(a^l_{st} + dur(a) \cdot a^l \leq mks)$, for every action a and level l. The objective function that is used in the problem formulation depends on the optimization objective. If the objective is the minimization of the makespan of the plan the objective function is $min(mks)$. If there is a deadline for the overall execution of the plan, variable mks takes this value, and in the objective function any other quantity can be used. For instance, if each action a_i has an associated cost c_i the overall

Pr.	TP4 t/d/a	TGP t/d/a	MIP t/d/a/l	MIP tt
bw1	3/11/11	-/10/-	37/11/11/5	817
bw2	51/11/14	-/9/-	32/11/12/5	653
bw3	423/8/10	1231/8/9	9/8/9/4	276
bw4	-/13/-	-/12/-	65/14/13/6	-
r1	288/14/30	140/14/24	54/14/24/7	1083
r2	-/14/-	3306/15/28	79/15/28/7	949
r3	6252/9/36	5692/9/29	432/10/29/6	-
r4	-/9/-	-/9/-	263/11/34/6	-

Table 1: TP4, TGP and MIP on makespan minimization problems.

cost of the plan is minimized through the objective function $min(\sum_{a_i^l} c_i \cdot a_i^l)$.

The algorithm starts with the encoding of the planning graph of length 1 (i.e, with one level), and then it extends the encoding by increasing the number of levels in the underlying graph, until a feasible solution for both the logical and the temporal parts of the encoding is found. Let l be the level of the first solution, and let opt^l denote the value of the optimal solution for that level, under the optimization objective. After the problem for the l levels is solved to optimality, the encoding is extended by considering an extra level of the underlying graph, and a new search starts for a solution with an objective function value less than opt^l.[4] If a new, improved, solution is found, the procedure repeats by extending again the encoding (i.e, by considering an extra level for the underlying planning graph). If at some level no better solution is found, the algorithm terminates. Of course, the solutions found are not guaranteed to be optimal, as it is possible that better solutions can be extracted if the underlying planning graph is extended further.

Experimental Results

We ran some initial experiments with the new temporal planning approach. The models were generated by hand, using the algebraic modeling system PLAM (ProLog and Algebraic Modeling) (Barth & Bockmayr 1998), and solved with CPLEX 7.1. In order to gain some insight about the difficulty of the problems, and the quality of the solution that are generated by the new method, the makespan minimization problem were also solved with TP4 and TGP. TP4 and CPLEX were run on a Sun Ultra-250 with 1 GB RAM, and an UltraSparcII 400MHz processor. TGP was run on a Pentium 500MHz machine running under Linux. Table 1 presents some of the experimental results. The bw rows refer to blocks world problems, and the r rows refer to Rocket problems. All blocks world problems are instances with 8 blocks, while the rocket problems involve 4 locations, 2 or 3 planes, and 9 to 11 packages. All run times reported are in seconds. A time limit of 7200 seconds was used for all systems.

For TP4 and TGP, the entries t/d/a in the table are respectively the run time (t), plan duration (d) and number of actions (a) of the generated plan. A -/d/- entry indicates that the system was searching for a solution of duration d, when the time limit was reached, and execution was aborted.

The data for the MIP method are presented in the t/d/a/l format, with the following meaning. The number in position l is the first level at which a feasible solution is found. The data t/d/a refer to the optimal solution of the problem that corresponds to graph length l. The duration of this optimal solution is presented in d, and the number of actions in the solution in a. The number in position t is the run time, and includes both the time needed for proving the infeasibility of the problems associated with planning graphs of length less than l, as well as finding the optimal solution for the graph of length l. The last column in the table, labelled with tt, presents the overall run time needed for solving the problem on all different levels, up to the level where the solution does not improve further. A dash in this column denotes that CPLEX reached the time limit before completing the search of the last level.

We note that in all problems, except bw4, the best solution that was found by the method, was at the same planning graph level with the first feasible solution. In problem bw4, the first solution was found at level 6. The optimal solution for this level is 14, the graph was expanded, and a better solution with duration 13 was found after 3032 secs.

It seems that the new MIP temporal planning method performs well in providing good solutions early in the computation. We have obtained similar results for other optimization criteria, including minimization that combines actions cost and makespan with different weights, that will be reported in an extended version of this paper. Of course there can be cases, as problem r3, where the method fails to find the optimal solution within a reasonable time. Nevertheless, in all problems considered it quickly found high quality solutions.

Conclusions and Future Work

We have presented a novel approach to temporal planning that relies on an explicit representation of the temporal constraints present in a planning problem. We have showed how the structure of a domain can be exploited in the MIP formulations, and presened some encouranging preliminary results from an experimental analysis of the new method.

We are currently extending our experiments and investigating ways of improving the new method. One promising direction is to exploit further the separation of the logical and temporal part of a planning problem by relaxing the former and tightening the latter. Another direction concerns handling a more expressive planning language capable, for instance, of dealing with level 3 of PDDL 2.1, the official language of the AIPS-2002 planning competition. Such features include actions with preconditions/effects involving numerical quantities, resources, and temporal conditions that are required to hold at some point during the execution of an action, at its beginning or at its end. It appears that all such features can be accomodated by a simple extension of our model. Finally, we are investigating the use of our techniques for generating plans of good quality under different (possibly competing) criteria.

[4] When we extend the encoding, both the logical and the temporal parts of encoding are extended.

References

Allen, J. 1991. Temporal reasoning and planning. In *Reasoning about Plans*. Morgan Kaufmann.

Barth, P., and Bockmayr, A. 1998. Modelling discrete optimisation problems in constraint logic programming. *Annals of Operations Research* 81.

Dimopoulos, Y. 2001. Improved integer programming models and heuristic search for AI planning. In *Proceedings of ECP-01 (to appear)*.

Fox, M., and Long, D. 1998. The automatic inference of state invariants in TIM. *JAIR* 9:367–421.

Gerevini, A., and Schubert, L. 1998. Inferring State Constraints for Domain-Independent Planning. In *Proceedings of AAAI-98*.

Gerevini, A., and Schubert, L. 2000. Discovering state constraints in DISCOPLAN: Some new results. In *Proceedings of AAAI-00*.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In *Proceedings of ECP-01 (to appear)*.

Kautz, H., and Walser, J. 1999. State-space planning by integer optimization. In *Proceedings of AAAI-99*.

Penberthy, J., and Weld, D. 1994. Temporal planning with continuous change. In *Proceedings of AAAI-94*.

Smith, D., and Weld, D. 1999. Temporal planning with mutual exclusive reasoning. In *Proceedings of IJCAI-99*.

Tsang, E. 1986. Plan generation in a temporal framework. In *Proceedings of ECAI-96*, 479–493.

Vossen, T.; Ball, M.; Lotem, A.; and Nau, D. 1999. On the use of integer programming models in ai planning. In *Proceedings of IJCAI-99*.

Weld, D. 1999. Recent advances in AI planning. *AI Magazine* 20(2).

Wolfman, S., and Weld, D. 1999. The LPSAT engine and its application to resource planning. In *Proc. of IJCAI-99*.

Wolsey, L. 1998. *Integer Programming*. John Wiley and Sons.

Fast Temporal Planning in a Graphplan Framework

Maria Fox and Derek Long
University of Durham, UK
D.P.Long@dur.ac.uk,Maria.Fox@dur.ac.uk

Abstract

Graphplan (Blum & Furst 1995) has been successfully extended to plan with actions with durations (Smith & Weld 1999) (Garrido, Onaindía, & Barber 2001; Garrido, Fox, & Long 2001). Existing approaches treat durative actions as spanning several layers in the plan graph, with fact layers corresponding to points in the flow of time. A simple model of time is used which prohibits much of the concurrency available for exploitation in an interesting problem. In this paper we describe an alternative approach, in which the fact layers of a plan graph are used to represent periods of time elapsing between the instantaneous start and end points of actions. The extents of these periods of time in a successful plan are determined using a linear constraint solver to ensure that actions can be temporally arranged in a way that avoids logical conflicts between them. The model of time used, embodied in the language PDDL2.1, is more expressive than that of earlier systems, enabling the exploitation of increased concurrency. We describe the planning algorithm, LPGP (Linear Programming GraphPlan), the model of time used and some of the results obtained for LPGP in temporal planning domains. We also present some indicative comparisons with other temporal planners.

Introduction

Graphplan (Blum & Furst 1995) has proved an influential planning system providing a clean foundation for the development of a number of scaling extensions. One of the most interesting developments of Graphplan is the TGP system (Smith & Weld 1999) - one of the first domain-independent planners to manage temporal planning without the aid of heuristic control rules.

The approach taken in TGP is to associate real-valued durations with the action schemas and then to allow action layers to span several fact layers in the graph construction process. TGP does not build an explicit representation of the graph. Instead TGP uses a highly optimised representation that exploits the monotonicities available in the Graphplan plan graph. However, the implicit graph can be understood in terms of the Graphplan plan graph with an extended binary mutex relation enabling mutexes between actions and propositions to be inferred from interactions between actions. The extra mutex relation is necessary because actions can overlap the time points at which facts appear and there can be interference between them. TGP implements the strong mutex requirement that prevents any pairs of actions, or actions and propositions, from overlapping *in any way* if there is any potential for interaction between them. This strong requirement prohibits much of the interesting concurrency in planning problems.

However, it is often reasonable for actions that refer to the same facts to be executed concurrently. For example, two durative actions which interact only at their end points can be successfully overlapped, provided that they do not end at the same moment. This is the observation exploited by LPGP (Linear Programming GraphPlan). The language used by LPGP represents actions with durations in terms of the local pre- and post-conditions of their end points, as well as invariant conditions that must hold over the interval of execution. The language corresponds to level 3 of the durative actions component of PDDL2.1 (Fox & Long 2001).

The key idea implemented in LPGP is that, although it might make sense to view an action as having a *delayed* effect (Bacchus & Ady 2001), logical change must be instantantaneous at the end of the delay. This is because logical change is by nature discrete (actions might have other, continuous, effects in addition). An action might also have immediate logical effects which are available as soon as the action starts executing. This view makes it natural to treat an action with a delayed effect in terms of its two endpoints separated by a period of time over which invariant conditions might be required to hold. As we describe in this paper, invariant conditions can be maintained across the part of a plan graph between the start and end points of actions using a mechanism by which invariant conditions are checked at every fact layer that occurs between these two points.

This paper describes the temporal planning approach of LPGP and presents some preliminary results suggesting that its performance can be comparable to that of other non-

heuristic temporal planning systems in the literature. Because LPGP uniquely interprets fact layers as having duration some of the guarantees that Graphplan offers cease to apply in LPGP. In particular, LPGP does not guarantee parallel optimality because it is possible for plans that are long in terms of action duration, but short in terms of the number of fact layers visited, to be found before plans with shorter temporal makespan but a larger number of visited fact layers. In order for LPGP to find parallel optimal plans it is necessary for it to search beyond the layer at which the first successful plan is extracted. This necessity is explained in the following sections.

LPGP currently plans with the simple durative actions level of PDDL2.1. In order to enable a Graphplan strategy to exploit the view of durative actions as two connected instantaneous end points it is necessary to transform PDDL2.1 actions into a particular format which can be seen as an intermediate domain description language. We first describe the way in which a PDDL2.1 domain is converted into such a form. We then go on to describe the modifications to the Graphplan algorithm that allow correct temporal planning behaviour to be achieved and the mechanisms by which the temporal durations of states are introduced into the planning structure. Finally, we present some results and discuss our plans for future development.

Treatment of PDDL2.1 domains

PDDL2.1 extends PDDL in several important ways. In this paper we consider only the temporal extension, and only a restricted part of that. The treatment of numeric values has been explored in the Graphplan framework (Koehler 1998), but we have not considered it further in this work. PDDL2.1 offers the opportunity to use different kinds of durative actions: the simplest are those in which the durations are fixed, possibly as a function of the parameters of the action. PDDL2.1 represents durative actions by describing the transitions that occur at the end points of the interval of activity, using an essentially classical pre- and post-condition model of these transitions, together with a collection of invariant conditions that must hold over the duration of the action.

A straightforward conversion of PDDL2.1 actions into actions that can be used by a standard Graphplan planner is to create the simple actions representing the end points of the durative actions. This is a good starting point, although we shall see that there are some complications that must be addressed. The first of these is that we want to ensure that the start and end point actions are always managed as a pair. To achieve this, we add a new effect to the start action that is required by, and deleted by, the end action. Thus, the end action cannot be executed without also executing the start action.

To implement the requirement that invariant conditions hold between the end points of a durative action it is necessary to ensure that they are maintained between each pair of happenings executed in the plan within the interval of the durative action. A happening is a collection of (instantaneous) actions (or end points of durative actions) executed at the same time. If we model durative actions with only the pair of end point actions then the invariant is effectively ignored. To correctly account for the invariant we introduce a new action with the invariant as its precondition. In order to force this action to sit between the end points of the durative action from which it is derived, we give the action a precondition achieved by the start action and an effect required as precondition by the end action. We want it to be possible for multiple happenings to occur in the interval between the end points, and in that case the invariant should be rechecked following each such happening. This means that we must force the invariant checking action to be reapplied at each layer in the graph between the layer containing the start action and the layer containing the corresponding end action. Left to its own devices Graphplan will attempt to exploit *noops* to make the effect of the invariant action persist until the end point at which it is required, or the effect of the start action persist until the invariant action requires it, placing a single instance of the invariant checking action at whichever intermediate layer is least inconvenient. To prevent this we require two mechanisms, one being a modification of the Graphplan machinery itself and the other being an addition to the domain encoding. The latter is the requirement to add an additional effect to the invariant checking action, which is the special proposition achieved by the start action and used as a precondition of the invariant-checking action itself. It can be seen that this action then behaves like a *noop* with additional preconditions — the invariant conditions of the durative action to which it corresponds. The modification in the Graphplan engine is not to generate the standard *noop* for either the special effect of the invariant-checking action or for the effect of the start action that acts as precondition for the invariant-checking action and the end action.

The way in which this collection of actions now fits together to model the enactment of a durative action can be seen in Figure 1. An example of the actions generated for a durative action from PDDL2.1 can be seen in Figure 2.

Because the operators that result from the translation process are instantaneous the standard Graphplan mutex relation is used. There is no need to extend the mutex relation to take account of intervals containing points, because no intervals arise in the graph construction process. The transformation can be performed automatically from a PDDL2.1 input, so should not be seen as introducing a new language, but simply as a compilation into an internal representation format.

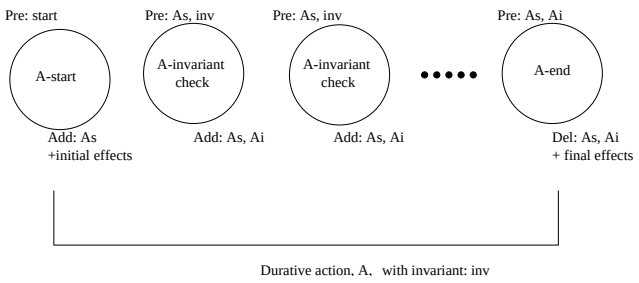


Figure 1: Modelling a durative action with a collection of simple instantaneous actions.

```
(:durative-action debark
 :parameters (?p - person ?a - aircraft ?c - city)
 :duration (= ?duration debarking-time)
 :condition (and (at start (in ?p ?a))
                 (over all (at ?a ?c)))
 :effect (and (at start (not (in ?p ?a)))
              (at end (at ?p ?c))))

(:action debark-start
 :parameters (?p -person ?a -aircraft ?c -city)
 :duration (debarking-time)
 :precondition (in ?p ?a)
 :effect (and (not (in ?p ?a))
              (debarking-inv ?p ?a ?c)))

(:action debark-inv
 :parameters (?p -person ?a -aircraft ?c -city)
 :precondition (and (debarking-inv ?p ?a ?c)
                    (at ?a ?c))
 :effect (and (idebarking-inv ?p ?a ?c)
              (debarking-inv ?p ?a ?c)))

(:action debark-end
 :parameters (?p -person ?a -aircraft ?c -city)
 :duration (debarking-time)
 :precondition (idebarking-inv ?p ?a ?c)
 :effect (and (not (idebarking-inv ?p ?a ?c))
              (not (debarking-inv ?p ?a ?c))
              (at ?p ?c)))
```

Figure 2: The result of converting a PDDL2.1 durative action (at the top) into linked instantaneous actions. Note the introduction of the duration field in both the start and end actions.

Modification to Graphplan

The main differences in the approach to introducing time into Graphplan that we describe in this paper, and the approach used in TGP, are that we invert the way in which time is attached to states and actions, we do not use the graph layers to measure time in uniform increments and we do not require an extended mutex relation. In TGP states are instantaneous, while time flow is attached to the actions. Actions can span several layers of the graph between the point at which their preconditions must be achieved and the end point at which they have their effects. TGP actions do not have initial effects and actions are mutex with any other actions that might attempt to access the propositions used or changed by them. This is a strong mutex relationship, and prevents any attempt to model, for example, executing an action to wash ones hands while a sink is being filled — the sink-filling action must end before the water is accessible.

TGP models the flow of time in uniform increments associated with the graph layers, each of length equal to the GCD of the action durations. This has an important advantage which is that the optimality of the plan length in terms of graph layers is equivalent to the optimality in terms of execution time. However, the price is very high: any plan that has a long execution time relative to the lengths of any of its actions will require a large number of fact layers to be considered at plan extraction time, even if the plan requires relatively few actions.

In our planner we attach duration to states, so each fact layer is associated with a duration. The layers are used only to capture the points at which events occur within the execution trace of the plan, rather than uniform passage of time. By separating the graph structure from the flow of time in this way we gain the benefit that plans with few events only require short graph structures. However, it is not always true that a plan that requires fewest distinct points of activity will be the shortest in duration. For example, if two goals can be achieved by the parallel execution of actions A and B, with durations 3 and 5 time units respectively, or by the single action C with duration 100 units, the plan in which A and B are used will require more distinct levels of activity (the simultaneous start of A and B, the end of A and then the end of B) than the plan using C alone. In fact, this example is slightly simplified because of the need to insert the special actions to check invariants. The complete plan structure is illustrated in Figure 3. We discuss this issue further, below.

The implementation of the machinery in the Graphplan algorithm is achieved by modifying the basic algorithm as follows.

Graph construction

The graph construction phase is modified so that no *noops* are constructed for the facts that have an `-inv` suffix. This

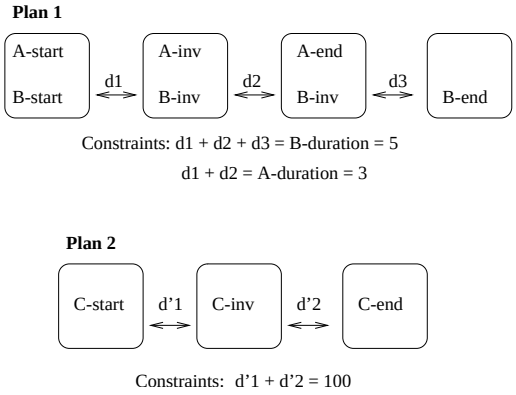


Figure 3: Two alternative plan structures showing how a temporally longer plan can have a simpler activity structure, being represented in fewer plan graph layers.

forces the invariant checking actions to be used to propagate these facts between layers, ensuring that the invariants are checked as the propagation is carried out. Apart from this minor change no modifications are needed to the standard Grpahplan graph construction process.

Graph search

This phase of the Graphplan algorithm is the one most affected. When an end action is selected to act as the achiever for a goal fact we introduce a temporal constraint. This constraint will assert that the total duration of the fact layers between the start and end actions of the durative action must equal the duration of the action. However, when the end action is first introduced we cannot yet know when the start action will appear. Therefore, the constraint is initially an assertion that the layers between the current layer and the layer containing the end action must have total duration less than the duration of the associated action. The two forms of constraints, then, are simply linear constraints on the durations of the fact layers and the equations must be solved for these durations, minimizing the total duration of the plan. This means that it is possible to use a linear programming algorithm (such as the simplex algorithm) to solve the equations. The form of the constraints for such a solver is best given as a matrix of the coefficients for the linear combinations of the variables (which are the durations attached to the fact layers). The matrix contains as many columns as there are fact layers in the graph and as many rows as there are durative actions in the (current) plan. New columns are added to the matrix as the graph is extended, prior to searching from each new layer. As an end action is introduced into the plan a new row is added to the matrix. When an invariant-checking action is added, the column denoting the fact layer succeeding the action layer containing the invariant check is set to 1 in the row corresponding to the end action coupled to this invariant check. When a start action is introduced, the correct entry is set to 1, just as for the invariant check, but also the constraint is switched from an inequality to an equality. Backtracking through the choice of any of these action types causes the exact reversal of these activities, resetting matrix entries to 0 where they were set to 1. The matrix associated with a simple example developing plan structure is shown in Figure 4.

The result of these activities is that the plan, as it is constructed, always has an associated linear programming problem. If the problem is ever unsolvable then the plan is invalid and search must backtrack. An important decision is when to check the equations. One possibility would be to check them whenever the matrix is modified. However, the inequality constraints are typically less difficult to satisfy than the equality constraints, so we choose to carry out checks only when start actions are added to the plan, which convert inequality constraints to equality constraints. This has the benefit of reducing the number of calls to the linear constraint solver, but the cost of not always discovering that the equations are unsolvable until several choices after the point of failure. Other schemes would be possible, such as checking the constraints at each layer in order to avoid developing bad choices into the next layer.

This approach is very similar to that taken in Zeno (Penberthy 1993; Penberthy & Weld 1994), but in that planner the underlying architecture was a partial order planner. In the Graphplan framework we gain all of the benefits that have been associated with Graphplan in comparison with partial order planners (and all of the weaknesses), and we are able to construct a complete collection of constraints at all points in the planning process. In contrast, Zeno was unable to invoke the numeric constraint solver until the constraints were properly instantiated, preventing it from identifying flawed plans as early as might be hoped. Of course, Zeno was handling a richer language, including numeric effects, presenting a harder problem than a treatment of duration constraints alone.

A very important question arises in determining the treatment of start actions as possible achievers. When an *end* action is used to achieve a goal the corresponding start action will be forced into the plan in order to satisfy the preconditions of the end action. On the other hand, if a *start* action could satisfy a goal then the corresponding end action *should already have been placed in the plan*. This organisation follows from the backward sweep search that is used to construct a plan in Graphplan. Unfortunately, it is very difficult to return to a previously visited layer in the search and insert additional actions, so using a start action to achieve a goal is very problematic. This problem does not arise in TGP because the action representation precludes durative actions achieving anything at the start of their execution.

To handle this problem we allow start actions to achieve effects, but never *introduce* them into the plan as achievers

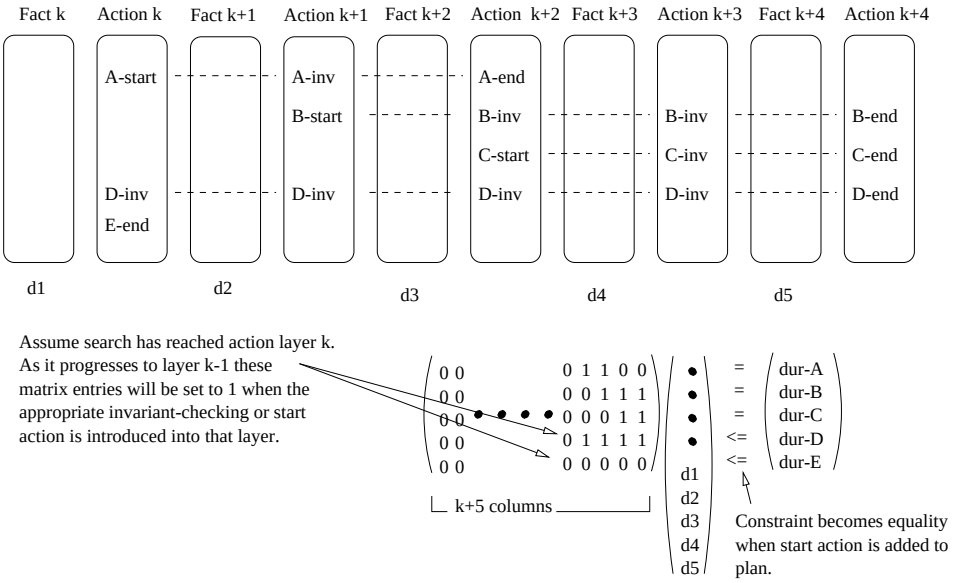


Figure 4: The matrix of constraints associated with an example partially complete graph search.

unless the corresponding end action is already in the plan. This means that if a start action is introduced into a plan because its end effect has been exploited then we can make use of any of the fortuitous side effects of the start action. This is not a complete solution to the problem, since it is possible to construct examples of durative actions in which it is the initial effects that are sought, rather than final effects. For example, the durative action of burning a match is useful for the existence of the heat and light created at the start of the action, rather than for the creation of the burnt stub at the end of the action. LPGP cannot currently handle such actions properly, but we are working on extensions that will enable exploitation of such start effects.

An interesting additional factor in our treatment is connected to an important consequence of the mutex relationships that we use to govern the validity of plans in PDDL2.1. The semantics of PDDL2.1 forbids actions from being executed simultaneously if they could possibly interfere with one another's pre- or post-conditions. Therefore, actions that do interact must be separated by a small, but non-zero, interval. Typically, the only constraint we have to satisfy is that the duration of separation must be positive. This gives rise to the need to introduce very small values into a plan. In the validation of plans (Long & Fox 2001) we introduce a small constant that dictates the minimum degree of separation allowed between actions, in order to avoid the problem that one plan might be judged better than another simply because it used smaller separations than the second, possibly otherwise identical, plan. This bound must be introduced into the equations we construct as a lower bound on the values of the variables (the fact layer durations). It will be observed that the first fact layer can never be constrained by any constraint other than this lower bound, since it can never appear *between* a start and end action. This leads the constraint solver to assign the minimum duration to the first fact layer in every case, which is a direct reflection of the decision in the semantics of PDDL2.1 to begin the initial state at time 0, while insisting that states are always associated with intervals that are half-open on the right. That decision prevents the first actions in a plan from being executed at 0 and forces them to begin at a small, non-zero time after 0. The value of the small, non-zero time that is used is selected by the programmer in the current implementation (we set it at 0.001), but it would be easy to set if from the command line, or, as we discuss in (Long & Fox 2001), from a value communicated in a problem description.

Results

We implemented the system in several stages: the first stage is a simple translator, transforming PDDL2.1 domains into the action sets described above. This is a stand-alone program, built using the tool-kit associated with the PDDL2.1 parser we have released, and represents one of a collection of translation tools for conversion of PDDL2.1 domain and problem descriptions. The second stage is the adaptation of a Graphplan implementation, to create a system we call LPGP (Linear-Programming GraphPlan).

To solve the linear constraints we used the `lp_solve` library originally developed by Michel Berkelaar (Berkelaar 2000). This we connected as a library to the LPGP code, and used its API to manage the constraint matrix. This solver attempts to solve modified problems from the same basis that solved the problem before the modification,

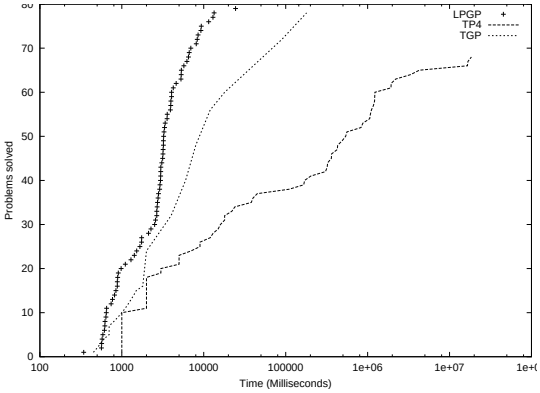


Figure 5: Plot showing relative performance of TGP, TP4 and Temporal LPGP on temporal logistics problems. The collection is the same as used in (Haslum & Geffner 2001): problems range from 4 packages in 3 cities to 5 packages in 4 cities. Note that the time axis is log-scaled. The TGP performance data was taken from (Haslum & Geffner 2001).

which is an excellent strategy in the context of our exploitation: most often new constraints do not have a dramatic impact on the constraint solution, since most durative actions span few fact layers.

The results depicted in Figure 5 show performance of LPGP compared with that of TGP and TP4 (Haslum & Geffner 2001). TP4 is a temporal planner that uses the HSP architecture (Bonet, Loerincs, & Geffner 1997; Bonet & Geffner 1997) and follows the same route as TGP in adopting a model of durative actions in which concurrent activity is constrained to avoid consulting propositions that are in use within another action. The data set is taken from (Haslum & Geffner 2001) and was generated on a 900MHz PC. The TGP data is for the version with EBL/DDB. It should be noted that LPGP does not use EBL/DDB (the reasons for this are discussed further, below). The domain is a simple extension of the Logistics benchmark, allowing an additional action by which trucks can drive between locations in different cities, but at the cost of a longer action. TP4 and TGP are generating (temporal makespan) optimal plans. In contrast, LPGP is generating plans that minimize the *activity makespan*, by which we mean the number of distinct time points at which activity occurs within the plan. Activity can include simultaneous initiation or termination of durative actions, and the special invariant checking actions. Minimizing the activity makespan can lead to plans with sub-optimal temporal makespan in the temporal logistics domain, so it should be noted that TGP and TP4 are solving a harder problem than is LPGP.

Figures 6 and 7 show LPGP's performance in a temporal planning domain called Mars Rover. This domain features a network of locations and a collection of rover vehicles each

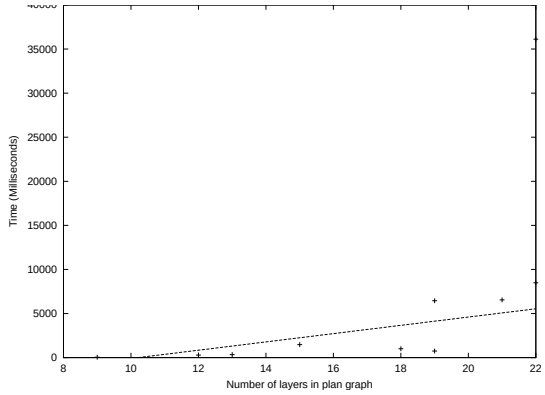


Figure 6: Plot showing effect of plan graph length on LPGP performance for small sample of Mars Rover problems.

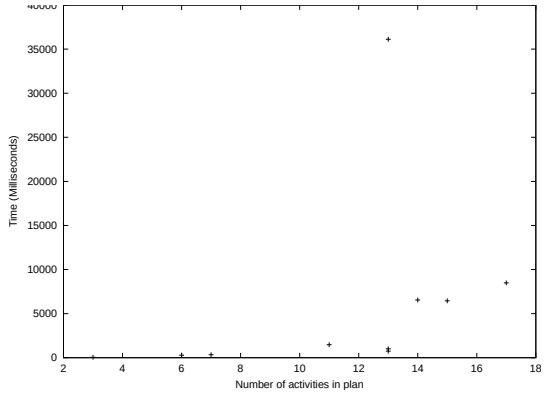


Figure 7: Plot showing the number of activities in each plan against the time taken to produce it, using same sample of Mars Rover problems as used in Figure 6

able to access different portions of the network (for example, rough-terrain vehicles can navigate mountainous areas, whilst other rovers can only navigate relatively flat areas). At different locations on the network there are tasks to be completed, such as collecting and analysing soil and rock samples and recording photographic images. The rovers are equipped with different capabilities – some have cameras on board and some have spectrometers and other equipment. The planner's task is to allocate suitable rovers to the different tasks so that required data can be communicated to a lander accessible to the rovers. The figures indicate the strong effect of graph construction and search on the time to plan and the weaker effect of number of activities in the plan on the performance of the planner. It should be remembered that each activity in the plan is represented by at least three steps in the graphplan plan (the start, the end and at least one invariant step).

A feature of the domain encoding is that whilst it is necessary to remain located at a single spot whilst communicating image and geological data, the nature of the communication link between the lander and rovers equipped for soil analysis means that soil analysis data can be communicated while the rover is on the move. However, a restriction on this is that, due to spikes generated when the motors start, communication is not allowed over an interval that includes the *start* of navigation. Thus, navigation and communication can overlap, once navigation has started. This means that rovers carrying soil analysis data can communicate and navigate, or collect other samples, concurrently. The plan shown in figure 8 contains concurrency that it would not be possible to exploit using TGP.

Although activity makespan is different from temporal makespan it remains to be demonstrated how much longer than optimal the makespans of activity-optimal plans are likely to be. We have observed that in pathological cases (where there are both sequences of very short duration actions and sequences of very long duration actions available in the domain for achieving the same goal collections) there can be wide variation. However, this domain structure does not seem to be arise commonly and we have observed that LPGP typically constructs plans about twice the length of those constructed by TP4. On the basis of our experiments to date LPGP seems to be comparable to Sapa (Kambhampati 2001) in terms of plan lengths.

A very interesting possibility for an improvement in the behaviour of LPGP, which we are currently exploring, exploits the fact that LPGP can continue searching for better plans after it has found the first plan. We can use the temporal duration of the first plan as a bound for other possible plans, and continue to search for alternatives up to that bound. This would give a form of *anytime* behaviour, in which LPGP, after generating its first plan, could always return the best plan it has available at any time. Such an approach can still have termination properties (since the best plan so far acts as a bound on the duration for any other plans), but it could also be able to find the plan with optimal temporal makespan if given enough time.

Conclusions and further work

This paper has described a successful attempt to exploit the Graphplan architecture to construct temporal plans, but using a different approach to that used in TGP or TPSys (in either of its versions (Garrido, Onaindía, & Barber 2001; Garrido, Fox, & Long 2001)). Where those systems use the graph itself to represent the flow of time, and to solve the associated constraints on the ways in which the durations of actions must interlock in a successful plan, we use the graph to capture only the distinct points of activity and the logical relationships between them, while handling the duration constraints in a separate linear constraint solver. This offers the significant benefit of reducing the necessary graph size for most problems. It has the disadvantage that optimisation of temporal duration is then separated from the optimisation of graph length and this prevents the planner from claiming temporal-optimality. However, an important benefit of our treatment is that it provides an acceptably accurate representation of the PDDL2.1 semantics, enabling the exploitation of interesting concurrency in temporal planning domains.

Many extensions and modifications to Graphplan-based planners have been explored in the past, including extensions to the language to include ADL features (Nebel, Dimopoulos, & Koehler 1997), filtering to remove irrelevant information (Koehler *et al.* 1997), an efficient search beyond the fix-point (Long & Fox 1999), exploitation of symmetry (Fox & Long 1999) and handling sensory actions (Anderson & Weld 1998). The modifications we have explored in this version of temporal Graphplan planning seem to be orthogonal to many of those extensions, since the underlying Graphplan behaviour is largely unchanged. It remains a possible direction for future work to explore which of these extensions could be successfully integrated with the mechanisms discussed in this paper. The EBL/DDB modification proposed in (Kambhampati 1999) is an obvious integration to try, as this yields interesting performance improvements for Graphplan-based planners. The key problem to be addressed is in constructing a conflict set at a layer. In the extended algorithm we have described it can happen that an action will fail because it causes a violation of the temporal constraints, but this does not lead to identification of a single point of blame in the current layer. If the temporal constraints cannot be solved it is potentially expensive to go back through them, determining which is the most recent constraint that could be modified to make the temporal constraints satisfiable. This is an interesting search problem that we have not yet ad-

```
0.001: (sample_rock rover2 rover2store waypoint1) [8]
2.002: (navigate rover3 waypoint1 waypoint0) [6]
2.002: (navigate rover1 waypoint2 waypoint5) [6]
8.002: (navigate rover2 waypoint1 waypoint7) [6]
9.003: (calibrate rover3 camera2 objective1 waypoint0) [5]
14.002: (sample_soil rover1 rover1store waypoint5) [8]
14.003: (communicate_rock_data rover2 general waypoint1 waypoint7 waypoint1) [10]
22.002: (take_image rover3 waypoint0 objective2 camera2 high_res) [7]
24.003: (navigate rover1 waypoint5 waypoint4) [6]
29.002: (communicate_soil_data rover1 general waypoint5 waypoint4 waypoint1) [10]
39.003: (communicate_image_data rover3 general objective2 high_res
                                                    waypoint0 waypoint1) [12]
```

Figure 8: Sample plan for Rover domain. The labels at the start of each action show when the action starts execution, and the bracketed values following each action shows its duration. Note the concurrent transmission of soil data with navigation at time 29.002-30.003.

dressed.

A critical extension to the PDDL language introduced in PDDL2.1 is the ability to express plan metrics. Most real planning problems require solutions to be judged by qualities other than simply the number of steps or even their temporal makespan. One of the most significant challenges for the Graphplan architecture, if it is to remain relevant to future developments of planning, is to find ways to modify the search to take into account such plan metric information. The first step in addressing this challenge is to find a convincing means by which to combine efficient behaviour with, at least heuristically, minimising the temporal makespan of the plan. The idea we discuss above, in which we continue generating plans after the first has been generated using the best plan so far as a bound on the continued search, is a particularly interesting one and we intend to make this a priority in our future work on LPGP.

References

Anderson, C., and Weld, D. 1998. Conditional effects in Graphplan. In *AIPS-98*, 44–53.

Bacchus, F., and Ady, M. 2001. Planning with resources and concurrency: a forward chaining approach. In *Proc. of 17th IJCAI*.

Berkelaar, M. 2000. Lp-solve. Technical report, ftp://ftp.es.ele.tue.nl/pub/lp_solve/.

Blum, A., and Furst, M. 1995. Fast Planning through Plan-graph Analysis. In *IJCAI*.

Bonet, B., and Geffner, H. 1997. Planning as heuristic search: new results. In *Proceedings of the European Conference on Planning (ECP)*.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *AAAI*.

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

Fox, M., and Long, D. 2001. PDDL2.1: An extension to PDDL for expressing temporal planning domains. Technical report, Available at: www.dur.ac.uk/d.p.long/competition.html.

Garrido, A.; Fox, M.; and Long, D. 2001. A temporal planning system to manage level 3 durative actions of PDDL2.1. In *Proceedings of the 20th UK Planning and Scheduling SIG*.

Garrido, A.; Onaindía, E.; and Barber, F. 2001. Time-optimal planning in temporal problems. In *Proc. European Conference on Planning (ECP-01)*.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In *Proc. of European Conf. on Planning, Toledo*.

Kambhampati, S. 1999. Improving Graphplan's search with EBL and DDB techniques. In *Proceedings of IJCAI'99*.

Kambhampati, S. 2001. Sapa: a domain independent heuristic metric temporal planner. In *Proc. of the European Conference on Planning, Toledo*.

Koehler, J.; Nebel, B.; Hoffmann, J.; and Dimopoulos, Y. 1997. Extending planning graphs to an ADL subset. In *ECP-97*, 273–285.

Koehler, J. 1998. Planning under resource constraints. In *Proc. of the 15th European Conference on AI*.

Long, D., and Fox, M. 1999. The efficient implementation of the plan-graph in STAN. *JAIR* 10.

Long, D., and Fox, M. 2001. Encoding temporal planning domans and validating temporal plans. In *Proceedings of the 20th UK Planning and Scheduling SIG*.

Nebel, B.; Dimopoulos, Y.; and Koehler, J. 1997. Ignoring irrelevant facts and operators in plan generation. In *ECP-97*, 338–350.

Penberthy, J., and Weld, D. 1994. Temporal planning with continuous change. In *Proc. of the 12th National Conference on AI*.

Penberthy, J. S. 1993. Planning with Continuous Change. Technical Report 93-12-01, University of Washington Department of Computer Science & Engineering.

Smith, D., and Weld, D. 1999. Temporal Graphplan with mutual exclusion reasoning. In *Proceedings of IJCAI-99, Stockholm*.

Encoding Temporal Planning as CSP

Amol D. Mali
Electrical Engg. & Computer Science
University of Wisconsin, Milwaukee, WI 53211
Phone: 1-414-229-6762, Fax: 1-414-229-2769, mali@miller.cs.uwm.edu

Abstract

Recent advances in constraint satisfaction and heuristic search have made it possible to solve classical planning problems significantly faster. There is an increasing amount of work on extending these advances to solving more expressive planning problems which contain metric time, quantifiers and resource quantities. One can broadly classify classical planners into two categories: (i) planners doing refinement search and (ii) planners iteratively processing a representation of finite size like a SAT encoding or planning graph or a constraint satisfaction problem (CSP). One key challenge in the development of planners casting planning as SAT or CSP is the identification of constraints which are satisfied if and only if there is a plan of k steps. This task is even more complex for planners handling metric time and/or resource quantities and/or quantifiers. In this paper we show how such a SAT encoding can be synthesized for temporal planning. This encoding contains twenty kinds of constraints. We show how this encoding can be simplified. Though solving a SAT encoding may not be the best approach to temporal planning (especially when there are too many actions and/or too much variation in durations of actions), the set of constraints we identify makes it easier to develop temporal planners casting planning as a constraint satisfaction problem other than SAT, like integer linear programming (ILP). The SAT encoding we present can be easily adapted to more complex cases of temporal planning like the one in which different pre-conditions and effects of an action may be true at different times during its execution.

1 Introduction

Recent advances in classical planning have made it possible to solve larger planning problems than before. Many of the recently developed planners cast planning as a constraint satisfaction problem. The planner in [Do & Kambhampati 2000] solves a planning problem by solving a CSP generated based on the planning graph built by Graphplan [Blum & Furst 1997]. The CPlan planner [van Beek & Chen 1999] solves planning as a constraint programming problem. Graphplan [Blum & Furst 1997] builds a representation called planning graph and identifies mutual exclusion constraints (mutexes) between actions. Graphplan propagates these mutexes to identify more mutexes. These inferred mutexes significantly improve the backward search over planning graph for solution extraction. SAT is a specific kind of CSP in which all variables are boolean and the constraints involve variables connected with operators from boolean logic. Some of the recently developed efficient planners cast planning as propositional satisfiability. These include SAT-plan [Kautz & Selman 1996], [Kautz et al 1996], MEDIC [Ernst et al 1997] and the planner which casts hierarchical task network planning as satisfiability [Mali 1999],[Mali 2000]. SAT encodings of a large number of planning problems in benchmark domains contain a significant number of binary clauses. Simplification techniques for binary clauses have been shown to improve the performance of planning as SAT [Brafman 2001]. Advances in SAT solving like better branching heuristics [Li & Anbulagan 1997] can be exploited to further improve the performance of SAT-based planners. Some of the recently developed efficient planners cast planning as 0-1 integer linear programming (ILP) in which all variables are boolean and all constraints are linear. These planners include the planner from [Vossen et al 2000].

More expressive planning problems contain quantifiers, conditional effects, metric time and resource quantities. Examples of such problems include many NASA planning applications [Smith & Weld 1999]. In these applications, both spacecraft and planetary rovers use heaters to warm up various components and these actions may span several other actions or experiments [Smith & Weld 1999]. There is an increasing interest in extending/adapting advances in classical planning to solving more expressive planning problems. Temporal Graphplan (TGP) is an extension of Graphplan [Blum & Furst 1997] to handle time durations of actions. The LPSAT planner [Wolfman & Weld 1999] solves planning problems involving resource quantities by transforming them into problems containing linear constraints and propositional clauses. Some of the constraints solved by LPSAT have a logical and a mathematical part, e.g. the constraint $((a > 3) \Rightarrow (x \vee y))$. Other more expressive planners include [Tsamardinos et al 2000].

Development of more expressive planners involves sev-

eral challenges. These include verification of soundness and completeness, besides getting optimal plans in a shorter time. The tasks of ensuring soundness and completeness are trivial for refinement planners. This is because refinement planners maintain a different representation for different partial plans in the form of nodes in a search tree. In progression, an action sequence is executed starting at initial state and it is checked whether goal is true at the end of its execution. A partial plan is a set of constraints which may or may not be a plan. Partial plans generated by forward state-space planners and backward state-space planners are sequences of actions whose goal achieving capability can be verified by simple methods like progression. A partial plan generated by partial order planner contains a set of constraints like step-action bindings and partial orderings over steps. The goal achieving capability of such a partial plan can be verified simply by finding if there exists a total order over the steps which is consistent with the partial order constraints in the partial plan such that the goal is achieved when the steps are executed in the order specified by the total order. The verification of soundness and completeness of more expressive planners which cast planning as CSP is non-trivial since a set of constraints needs to be identified such that these are solved if and only if there is a plan of k steps. This set needs to be loose enough to allow generation of all action sequences that are plans and tight enough to exclude generation of any action sequence that is not a plan. The correctness of the temporal planner TGP [Smith & Weld 1999] is difficult to verify.

The adaptation of SAT-plan to handle metric time is challenging because of task of identifying all constraints that must be satisfied if and only if there is a plan of bounded length. Temporal planning problem is the problem of finding a set of $< \ action_i, \ start\_time_i \ >$ tuples for achieving a given goal, starting with a given completely described initial state using actions that have time durations. $start\_time_i$ is the start time of action $action_i$. This is the time at which the execution of $action_i$ starts. We use the following assumptions in temporal planning from [Smith & Weld 1999] when setting up the SAT encoding. Effects of actions are undefined during their execution. Action durations are integers. Pre-conditions of an action in a plan should all be true at its start time. The effects of an action hold only at the end of its execution. The SAT encoding is such that it has a solution if and only if there is a plan of k time steps. In particular, we identify twenty kinds of constraints which together form the encoding when translated into clauses. We also show how the encoding can be simplified so that the number of clauses and variables are reduced without losing soundness and completeness. We show how the encoding can be adapted to more complex case of temporal planning in which it is not necessary for different pre-conditions of an action to be true at same time and it is not necessary for different effects to hold at the same time. We show how temporal planning can be cast as a CSP other than SAT.

2 Background

In this section, we explain how SAT-based planners work and describe the state-space SAT encoding from [Kautz et al 1996] for classical planning where all actions have unit duration. An action is a ground instance of an operator. For example, if $move(x, y, z)$ is an operator for moving block x from top of block y or table to top of block z or table, $x \neq y, y \neq z, x \neq z, x \neq Table$, then there are $O(n^3)$ actions when there are n blocks.

SAT-plan [Kautz & Selman 1996] works in the following manner. Based on initial state, number of steps assumed to exist in plan (k), goal state and action description, it generates an encoding (SAT instance) such that the instance has a model if and only if there is a plan of k steps. The steps range from 0 to $(k-1)$. The encoding is simplified by rules of inference of boolean logic, e.g. $a \wedge (\neg a \vee b)$ can be simplified to $(a \wedge b)$ (this simplification step is optional but most SAT planners carry this out). The encoding is then passed to a SAT solver. If it is solved, the solution (truth assignment) is interpreted and plan is output by reading the truth values assigned to step-action binding and step ordering variables. If it cannot be solved, then value of k is increased and the process of encoding generation, simplification and solving is repeated. Variables representing an occurrence of actions at various steps are step-action binding variables.

The explanatory frame axiom-based state-space encoding [Kautz et al 1996] contains the following constraints: (i) All propositions true in true in initial state are true at time 0 and all propositions false in the initial state are false at time 0. (ii) If an action occurs at time t, its pre-conditions are true at time t and its effects are true at time $(t+1)$. (iii) If an action o_i needs proposition p true and action o_j deletes p, then o_i and o_j cannot occur at same time. (iv) All propositions from goal are true at time k. (v) If a proposition p is true at time t and false at time $(t+1)$, some action deleting p must occur at time t. If a proposition p is false at time t and true at time $(t+1)$, some action making p true must occur at time t. These constraints are included to ensure that truth of a proposition cannot change unless an action causing the change occurs. These constraints are known as explanatory frame axioms.

3 SAT Encoding for Temporal Planning

To generate a SAT encoding, we first need to bound the number of steps in plan at which actions occur. We denote this bound by k. O denotes the set of all actions in domain and U denotes the set of all ground fluents in domain. A fluent is a proposition whose truth can change. Propositions that are not made true or false by any

action are considered to be invariants whose truth remains unchanged. Before explaining the constraints appear in the propositional encoding of a temporal planning problem, we explain some relevant notation for various boolean variables in the encoding. $pt(t)$ is a boolean variable which denotes the truth of true state of fluent p at time t. If $pt(t)$ is assigned true, it means that fluent p is true at time t. If $pt(t)$ is assigned false, it means that the fluent p is not true at time t (so it is either false or undefined/unknown). $pu(t)$ denotes that the truth of the undefined state of fluent p at time t. If $pu(t)$ is assigned true, it means that the fluent p is undefined at time t. If $pu(t)$ is assigned false, it means that the fluent p is not undefined at time t. In this case p is either true or false at time t. $pf(t)$ denotes that the truth of the false state of fluent p at time t. If $pf(t)$ is assigned true, it means that the fluent p is false at time t. If $pf(t)$ is assigned false, it means that the fluent p is not false at time t. In this case p is either true or undefined at time t. $o_i(t)$ denotes the occurrence of the ground action o_i at time t. $o_i(t)$ is an action variable. $pf(t), pt(t)$ and $pu(t)$ are fluent variables. If $o_i(t)$ is assigned true, it means that o_i occurs at time t. If $o_i(t)$ is assigned false, it means that o_i does not occur at time t. d_i denotes the duration of the ground action o_i. The time steps in the encoding at which actions may occur range from 0 to $(k-1)$. d_{max} is the maximum of durations of all actions. The following constraints appear in the state space encoding of a temporal planning problem. The encoding is a state-space encoding since it refers to states of all fluents at all time steps. We assume that actions do not have negated pre-conditions.

1. The constraints of this kind state that each fluent is either false or true or undefined at every time step. For all fluents p, for all times $t \in [0, k]$, $(pt(t) \vee pf(t) \vee pu(t))$. There are $(k+1).\mid U \mid$ clauses of this kind. These contain $3.(k+1).\mid U \mid$ variables.

2. The constraints of this kind state that at a time, a fluent cannot be in more than one of the following states: true, false and undefined. For all fluents p, for all times $t \in [0, k]$, $(pt(t) \Rightarrow \neg pf(t))$, $(pt(t) \Rightarrow \neg pu(t))$, $(pf(t) \Rightarrow \neg pu(t))$. There are $3.(k+1).\mid U \mid$ clauses of this kind.

3. Fluents true in initial state are true at time 0. Fluents false in initial state are false at time 0. These constraints contribute $\mid U \mid$ unit clauses to the encoding. If the initial state is $(a \wedge b \wedge \neg c \wedge \neg d)$, the encoding contains $(at(0) \wedge bt(0) \wedge cf(0) \wedge df(0))$.

4. Fluents from goal are true at time k. If goal is a conjunction of s fluents, these constraints contribute s unit clauses to the encoding.

5. If an action o_i occurs (starts) at time $t, 0 \leq t \leq (k - d_i)$, its pre-conditions are true at t and its effects are true at time $(t + d_i)$. These constraints contribute $O(k. \mid O \mid .m)$ clauses to the encoding, m being the maximum sum of the number of pre-conditions and the number of effects of an action. These constraints contribute $k. \mid O \mid$ variables to the encoding. For example, if x and y are pre-conditions of action o_3, and $\neg x$ and z are its effects, this constraint generates $(o_3(j) \Rightarrow (xt(j) \wedge yt(j) \wedge xf(j + d_3) \wedge zt(j + d_3)))$, where $j \in [0, k - d_3]$. Note that we require $t \leq (k - d_i)$ because if o_i starts at $t > (k - d_i)$, it will finish at time $(k + 1)$ or later and the encoding has only $(k + 1)$ time steps ranging from 0 to k.

6. If an action o_i does not delete any of its pre-conditions, then if o_i starts at time t, where $0 \leq t \leq (k - d_i)$, then o_i cannot start at any time $t' \in [t + 1, (t + d_i - 1)]$, d_i being duration of o_i. These constraints prevent an occurrence of an action during its execution, when the action does not delete any of its pre-conditions. These can be considered as mutual exclusion relationships of actions with themselves. These constraints contribute $O(k.q.d')$ clauses to the encoding, where q is the number of actions which do not delete any of their own pre-conditions and d' is the maximum of the durations of these actions.

7. If an action o_i does not delete its pre-condition x, then if o_i starts at time t, x remains true over the closed interval $[t + 1, t + d_i]$ when $d_i > 1$. If there are m actions that do not delete any of their own pre-conditions, these constraints contribute $O(m.d_{max}.k.m')$ clauses to the encoding, where m' is the maximum number of pre-conditions of an action not deleted by the action.

8. The pre-conditions of an action o_i which are deleted by o_i are undefined over the closed interval $[t + 1, t + d_i - 1]$ if o_i starts at t. If there are m_1 actions that delete some of their own pre-conditions, these constraints contribute $O(m_1.d_{max}.k.m'')$ clauses to the encoding, where m'' is the maximum number of pre-conditions of an action deleted by the action. For example, if effects of action o_4 are $\neg p$ and q, such that p is one of its pre-conditions, and duration of o_4 is 3, then this constraint yields $(o_4(j) \Rightarrow (pu(j + 1) \wedge pu(j + 2)))$.

9. Effects of an action o_i (except pre-conditions of o_i which o_i deletes) are undefined from time $(t + 1)$ until time $(t + d_i - 1)$, including both these times, if o_i starts at t. These constraints contribute $O(\mid O \mid .d_{max}.k.m''')$ clauses to the encoding, where m''' is the maximum number of effects of an action, excluding the pre-conditions deleted by it. For example, if effects of action o_4 are $\neg p$ and q, such that p is one of its pre-conditions, and duration of o_4 is 3, then this constraint yields $(o_4(j) \Rightarrow (qu(j + 1) \wedge qu(j + 2))$.

Constraints 10, 11, 12, 13, 14 and 15 are all explanatory frame axioms. These contribute $O(k. \mid U \mid)$ clauses to the encoding. These constraints prevent changes in states of fluents without occurrences of actions causing these changes.

10. If fluent p is true at time t and false at time $(t + 1)$, some action of duration 1 which deletes p must occur (start)

at time t. For example, if actions o_4, o_6 and o_9 are the only actions of duration 1 which delete fluent x, this constraint generates $((xt(t) \wedge xf(t+1)) \Rightarrow (o_4(t) \vee o_6(t) \vee o_9(t)))$.

11. If fluent p is false at time t and true at time $(t+1)$, some action of duration 1 which makes p true must occur (start) at time t.

12. If fluent p is true at time t and undefined at time $(t+1)$, some action of duration greater than 1 which deletes p must occur (start) at time t. For example, if k = 20 and the only actions of duration more than 1 which delete fluent x are o_7, o_8 and o_{11} such that $d_7 = 4, d_8 = 12$ and $d_{11} = 7$, then we have $((xt(t) \wedge xu(t+1)) \Rightarrow (o_7(t) \vee o_8(t) \vee o_{11}(t))), t \in [0, 8]$. We have $((xt(t) \wedge xu(t+1)) \Rightarrow (o_7(t) \vee o_{11}(t))), t \in [9, 13]$. We have $((xt(t) \wedge xu(t+1)) \Rightarrow o_7(t)), t \in [14, 16]$. This constraint does not lead to any clauses for the change in state of x from true to unknown for $t \in [17, 19]$. This is because any action changing x in this fashion will end after $t = 20$ in case it starts at time 17 or later.

13. If fluent p is false at time $t \in [0, k-2]$ and undefined at time $(t+1)$, some action of duration greater than 1 which makes p true must occur (start) at time t.

14. If fluent p is undefined at time $t \in [0, k-1]$ and false at time $(t+1)$, some action o_i of duration greater than 1 which makes p false must start at time $(t + 1 - d_i)$. For example, if o_5, o_7 and o_{10} are the only actions which delete fluent y and $d_5 = 2, d_7 = 5$ and $d_{10} = 7$, then this constraint is represented by $((yu(t) \wedge yf(t+1)) \Rightarrow (o_5(t-1) \vee o_7(t-4) \vee o_{10}(t-6)))$.

15. If fluent p is undefined at time $t \in [0, k-1]$ and true at time $(t+1)$, some action o_i with duration greater than 1 which makes p true must start at time $(t + 1 - d_i)$.

All of the following constraints represent mutual exclusion relations between actions. These contribute $O(k. \mid O \mid^2)$ clauses to the encoding. These constraints are identified after taking into account the all possible kinds of temporal relationships between two actions (seven types) from Figure 1 and all possible effects two actions may have on a fluent p. Given two actions o_i and o_j and a fluent p, the actions can affect the fluent in the following ways: (i) both o_i and o_j need p and both delete p, (ii) only o_i needs p and both o_i and o_j delete p, (iii) neither o_i nor o_j needs p and both delete p, (iv) neither o_i nor o_j needs p and both make p true, (v) o_i makes p true and o_j only deletes p, (vi) o_i only needs p and o_j only deletes p and, (vii) o_i makes p true and o_j both needs p and deletes p.

16. If two actions o_i and o_j have same duration such that o_i deletes pre-condition of o_j, then o_i and o_j cannot start at same time t. The clauses $\neg(o_i(t) \wedge o_j(t)), t \in [0, k-1]$ are generated for all pairs of such actions to represent this constraint in the encoding.

17. If o_i with duration greater than 1 deletes pre-condition p of action o_j which has duration 1 such that o_j does not delete its own pre-condition p, then o_i and o_j cannot start at same time t. The clauses $\neg(o_i(t) \wedge o_j(t)), t \in [0, k-1]$ are generated for all pairs of such actions to represent this constraint in the encoding.

18. If o_i has pre-condition p which it also deletes and action o_j only deletes p (o_j does not need p), then if o_i starts at t, then o_j cannot start at a time $t' \geq t$ if $(t'+d_j) = (t+d_i)$. To represent this in the encoding, the clauses $\neg(o_i(t) \wedge o_j(t'))$ are generated for all pairs of such actions, for all times t and t' such that $(t + d_i) = (t' + d_j), t' \geq t, t \in [0, k-1], t' \in [0, k-1]$.

19. If actions o_i and o_j both only delete some fluent p and both do not need p, then it cannot be the case that o_i ends at starting time of o_j. This avoids consecutive deletions of a fluent without making the fluent true. To represent this in the encoding, the clauses $\neg(o_i(t) \wedge o_j(t'))$ are generated for all pairs of such actions, for all times t and t' such that $(t + d_i) = t', t \in [0, k-1], t' \in [0, k-1]$.

20. If two actions o_i and o_j both only delete some fluent p and both do not need it, then it cannot be the case that o_i ends at ending time of o_j. To represent this in the encoding, the clauses $\neg(o_i(t) \wedge o_j(t'))$ are generated for all pairs of such actions, for all times t and t' such that $(t + d_i) = (t' + d_j), t \in [0, k-1], t' \in [0, k-1]$.

The encoding contains $O(k.(\mid O \mid + \mid U \mid))$ variables and $O(k.(\mid O \mid + \mid U \mid) + k. \mid O \mid^2)$ clauses. The number of variables and clauses in the SAT encoding of a temporal planning problem are respectively higher than the the number of variables and clauses in the encoding of the same problem with all action durations set to 1. This is mainly because of the action durations and the three variables needed to model three different states of each fluent.

4 Discussion

We showed how temporal planning can be encoded as a SAT problem and found the asymptotic number of variables and clauses in the encoding. In this section, we show how the size of this encoding can be reduced without losing soundness and completeness. We also show how this encoding can be adapted to the more complex case of temporal planning in which it is not necessary for all pre-conditions of an action to be true at the same time and different effects of an action may become true at different times during its execution. We also show how the encoding is useful in casting temporal planning as a CSP other than SAT.

4.1 Reducing Encoding Size

Though a smaller encoding is not always easier to solve, smaller encodings have been shown to be solvable faster [Kautz & Selman 1996], [Ernst et al 1997], [Mali 1999]. The size of an encoding can be reduced by propagating the truth assigned to unit clauses. Such unit clauses are available in the specification of initial state and goal in the en-

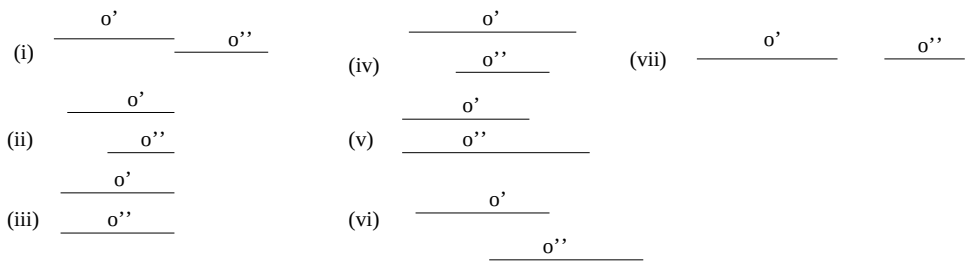


Figure 1: Temporal relations between two actions o' and o''

coding. Almost all SAT simplification techniques which apply to the case where all actions have duration 1 also apply to the case where actions have unequal durations and we will not discuss these.

One can select an accurate value of k to avoid solving several encodings generated with unacceptable lower values of k. The planning graph [Blum & Furst 1997] can be used to wisely choose the value of k. Graphplan works in 2 phases. The first involves growing a **planning graph** and is called the **plangraph** construction phase. This is a forward phase, beginning with the initial state. The second phase is a solution extraction phase. This is backward search phase starting with the goal. Plangraph, or planning graph (PG), has two kinds of levels called **action levels** and **proposition levels**. The 0th proposition level is the same as the initial state. The 0th proposition level occurs before the 0th action level which in turn occurs before the 1st proposition level which precedes the 1st action level, etc. In general, the i th proposition level is immediately succeeded by the i th action level. And, the i th action level immediately precedes $(i+1)$ th proposition level. A proposition level and an action level can be considered as sets whose members are the same as the contents of these levels. The i th action level in the plangraph contains all actions whose all pre-conditions appear in the i th proposition level. There is also a dummy action called a **no-op**, **maintenance action**, or **persistence action** in the i th action level for each proposition in the i th proposition level. The pre-condition and effect of this action is the proposition for which the action was created. This action is included in the plangraph because if no action changing the truth of the proposition occurs, the truth of the proposition remains same. The $(i+1)$ th proposition level is the union of the i th proposition level and the effects of the actions in the i th action level. Thus, proposition level i is a superset of proposition level $(i-1)$. Similarly, action level i is a superset of action level $(i-1)$.

There are three kinds of edges in plangraph: **(i)** edges from propositions in proposition level i to the same propositions in proposition level $(i+1)$ (for no-ops), **(ii)** edges from propositions in proposition level i to actions (whose pre-condition list contains these propositions) in action level i and **(iii)** edges from actions in action level i to propositions (which are effects of these actions) in proposition level $(i+1)$.

A key to the efficiency of Graphplan is the inference of binary **mutex** (mutually exclusive) relations. Two kinds of mutexes are found: (i) mutexes between actions, and (ii) mutexes between propositions. Each of these two kinds of mutexes could be static or dynamic. Static mutexes are found by examining pre-conditions and effects of actions. Dynamic mutexes are found by propagating static mutexes using truths of conditions in the initial state. Note that dynamic mutexes may be permanent or temporary. Two actions at action level i are mutex if (i) their effects are inconsistent (the effect of one action is the negation of some effect of another action), or (ii) one action deletes some pre-condition of another action, or (iii) the actions have pre-conditions that are mutex at proposition level i. For (iii) one needs to understand the definition of mutex propositions given next. Two propositions p, q in proposition level i are mutex if (i) p is the negation of q, or (ii) all ways (actions) of achieving p are mutex with all ways (actions) of achieving q. Note that while considering all ways of achieving a proposition, no-ops are also considered.

If the plangraph has a proposition level that contains all conditions from the goal such that no two of these are mutex, Graphplan starts a backward search for a plan. For each condition in the goal, it chooses a source of support (an action) in the immediately preceding action level. The pre-conditions of these actions become subgoals to be achieved. If no two pre-conditions of the chosen actions are mutex, it chooses sources of support for these subgoals from the immediately preceding action level and continues this process. In case subgoals are found to be mutex, it backtracks, chooses different sources of support, and repeats this process. If all combinations of the supporting actions fail for each subgoal at each proposition level, then Graphplan grows plangraph with one more action level and proposition level and tries the backward solution extraction process again. If no solution is found, Graphplan extends planning graph by one action level and 1 proposition level and tries the solution extraction again. Graphplan is guaranteed to report unsolvability of a problem. A plangraph is said to **level off** if the none of the following change when

the plangraph is grown further: (i) the number of actions in last action level, (ii) the number of propositions in last proposition level, (iii) the number of pairs of actions that are mutex in last action level, and (iv) the number of pairs of mutex propositions in last proposition level. In the planning graph in Fig. 2, false propositions are not shown in proposition levels for readability. Because of same reason, mutex relations are not shown. The planning graph has 3 proposition levels and 2 action levels. M1, M2, M3 and M4 are actions in the domain. The pre-conditions and effects of these actions are shown on the left and right sides of the boxes respectively. Here is the planning graph figure. The goal is achievable with the plan Step 0: M2, Step 1: M1 & M3. The actions M1 and M2 are static mutex and the actions M3, M4 are dynamic mutex in the first action level. Kautz & Selman have shown that planning graph can be used to reduce the size of a SAT encoding for planning with actions of unit duration [Kautz & Selman 1999].

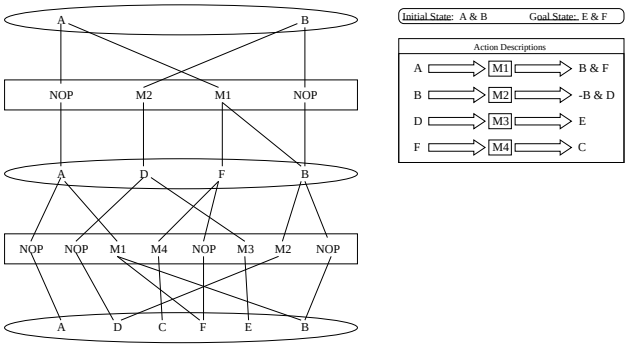


Figure 2: A planning graph

It is clear that actions that do not appear in the last action level of the leveled off planning graph are not relevant to solving the problem. Such actions need not be represented in the SAT encoding. Let us consider a leveled off planning graph with four action levels which are $\{o_2, o_4\}$, $\{o_2, o_4, o_5\}$, $\{o_2, o_4, o_5, o_8, o_9\}$, and, $\{o_2, o_4, o_5, o_8, o_9, o_{11}\}$ respectively, excluding no-ops from the action levels. This planning graph is constructed by Graphplan [Blum & Furst 1997]. If $O = \{o_i \mid 1 \leq i \leq 20\}$, then it can be concluded that actions $o_1, o_3, o_6, o_7, o_{10}, o_{12}, o_{13}, o_{14}, o_{15}, o_{16}, o_{17}, o_{18}$, o_{19} and o_{20} are not relevant to solving the problem. These need not be represented in the encoding for temporal planning. Since o_5 occurs in the first action level of the planning graph, it is clear that in case o_5 occurs in temporal plan, its start time will be greater than or equal to $min(d_2, d_4)$. This is because the planning graph shows that o_5 can occur only after one or more of the actions from $\{o_2, o_4\}$ occur. This means that we need not create variables $o_5(t), t \in [0, min(d_2, d_4) - 1]$. Similar argument shows that we need not create the following variables: $o_8(t_1), t_1 \in [0, (min(d_2, d_4) + min(d_2, d_4, d_5)) - 1]$, $o_9(t_2), t_2 \in [0, (min(d_2, d_4) + min(d_2, d_4, d_5)) - 1]$ and, $o_{11}(t_3), t_3 \in [0, (min(d_2, d_4) + min(d_2, d_4, d_5) + min(d_2, d_4, d_5, d_8, d_9)) - 1]$. This significantly reduces the number of variables and clauses in the encoding. This also reduces the number of literals in various clauses. The number of steps k can then be chosen so that $\theta' \leq k \leq \theta''$ where $\theta' = (min(d_2, d_4) + min(d_2, d_4, d_5) + min(d_2, d_4, d_5, d_8, d_9) + min(d_2, d_4, d_5, d_8, d_9, d_{11}))$ and $\theta'' = (max(d_2, d_4) + max(d_2, d_4, d_5) + max(d_2, d_4, d_5, d_8, d_9) + max(d_2, d_4, d_5, d_8, d_9, d_{11}))$

4.2 More Complex Temporal Planning

We assumed that all pre-conditions of an action are true at the same time which is same as the time of start of execution of the action. We also assumed that all effects of an action hold at the same time which is same as the time of end of execution of the action. In reality, different effects may hold at different times and it may not be necessary for all pre-conditions of an action to hold at the same time. Consider the action $move(A, B, C)$ which moves block A from top of block B to top of block C. Its pre-conditions are $clear(A), on(A, B)$ and $clear(C)$. Let us assume that there are several grippers and one does not need the pre-condition $hand - empty$. The effects of this action are $on(A, C), \neg clear(C), clear(B)$ and, $\neg on(A, B)$. $clear(A)$ and $on(A, B)$ have to be true at same time. However $clear(C)$ can become true later during the execution of the action since some other action may move block from the top of C before A is put on C. The effect $\neg on(A, B)$ holds immediately and the effect $on(A, C)$ becomes true much later. Consider the action $fly(P, London, Paris)$ which flies plane P from $London$ to $Paris$. Its effects are $at(P, Paris), \neg at(P, London)$. It is clear that $\neg at(P, London)$ becomes true much earlier than $at(P, Paris)$. Such an action may be specified with times at which various pre-conditions are needed true relative to s and various effects become true, relative to s, where s is start time of the action.

Let us see whether one needs to change constraints from section 3 to handle such actions and if so how. Constraints 1 through 4 do not need any change to handle such actions. Constraint 5 needs a minor change. The new constraint specifies that different pre-conditions and effects are true at different times. Constraints 6 and 7 do not need any change. Constraint 8 needs a minor change. The new constraint states that pre-condition of an action deleted by itself is undefined over the interval $[t + 1, t + r - 1]$, when the pre-condition is deleted r time units after the start of the action, t being start time of the action. Constraint 9 needs a minor change to specify that different effects of an action are undefined over different time intervals. Constraints 10 and 11 do not need any change. Constraint 12 needs a mi-

nor change. The new constraint states that if a fluent p is true at time t and undefined at time $(t+1)$, some action of duration greater than 1 which deletes p at two time units or more later than the time of start of its execution must occur at time t. The remaining constraints can be easily adapted to handle such actions.

4.3 Temporal Planning as CSP other than SAT

Since SAT can be easily transformed into 0-1 ILP, an ILP encoding of temporal planning can be directly created using our SAT encoding. For example, the clause $(x \vee y \vee \neg z \vee \neg b)$ can be translated into the linear constraint $(x' + y' + (1 - z') + (1 - b')) \geq 1$ where the domain of the integer variables x', y', z', b' is $[0, 1]$. The direct translation of our SAT encoding into 0-1 ILP encoding leads to $O(k.(\mid O \mid + \mid U \mid))$ integer variables and $O(k.(\mid O \mid + \mid U \mid) + k. \mid O \mid^2)$ linear constraints among these. It has been empirically shown in [Vossen et al 2000] that the ILP formulations need to be stronger to be easily solvable. The strength of an ILP formulation obtained by a direct translation of SAT can be improved by adding constraints so as to reduce the number of non-integer solutions of the ILP formulation.

One can create a CSP using the constraints that lead to the SAT encoding. Specifically, one can have $k. \mid O \mid$ boolean variables whose values represent occurrence/absence of actions at various time steps. One can have $(k+1). \mid U \mid$ variables with the domain $\{t, f, u\}$ to represent various states of the fluents at all time steps. Note that in SAT encoding we need $3.(k+1). \mid U \mid$ boolean variables to represent various states of all fluents at all time steps. In the CSP formulation, we need only $(k+1). \mid U \mid$ variables. Though the domains of these variables contain 3 values, the worst-case size of the space of assignments to the fluent variables is lower in the CSP formulation. Since by definition each variable in CSP is assigned a unique value, we do not need constraints to specify that a fluent cannot be in more than one state at any time. Note that in the SAT encoding we need $3.(k+1). \mid U \mid$ binary clauses to specify that a fluent cannot be in more than one state at any time. In SAT encoding, we need $(k+1). \mid U \mid$ clauses to specify that each fluent is true or false or undefined at each time step. No constraints are needed in the CSP to specify this requirement. Remaining constraints leading to the SAT encoding can be translated to complete the CSP formulation. This shows how the constraints we developed to generate a SAT encoding are useful in casting temporal planning as a CSP. Note that no extra effort is needed to verify the correctness of the CSP formulation.

5 Summary

There is an increasing amount of work on extending/adapting the recent advances in plan synthesis under classical assumptions to more expressive planning. More expressive planning involves dealing with metric time, conditional effects, quantifiers and resource quantities. Verifying the soundness and completeness of such planners which cast planning as some kind of CSP is non-trivial. We developed a SAT encoding for temporal planning such that that the encoding is solvable if and only if there is a plan whose execution time is less than or equal to chosen bound $(k+1)$. We showed how the size of the encoding can be significantly reduced using information in planning graph. We showed how the SAT encoding can be adapted to handle actions all of whose pre-conditions need not be true at same time and/or whose effects become true at different times. We also showed how the SAT encoding is useful in casting temporal planning as a 0-1 ILP and as a CSP different from SAT and 0-1 ILP.

Acknowledgement - This work is supported in part by NSF grant IIS-0119630 to Mali. Mali also thanks students in his graduate course "Artificial Intelligence Planning Techniques" (CS-790-001) taught in Fall 2000 and Fall 2001 for useful discussions.

References

[van Beek & Chen 1999] Peter van Beek and Xinguang Chen, CPlan: A constraint programming approach to planning, Proceedings of National Conference on Artificial Intelligence (AAAI), 1999.

[Blum & Furst 1997] Avrim Blum and Merrick Furst, Fast planning through planning graph analysis, Artificial Intelligence 90, 1997, 281-300.

[Brafman 2001] Ronen I. Brafman, A simplifier for propositional formulas with many binary clauses, Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 2001.

[Do & Kambhampati 2000] Minh B. Do and Subbarao Kambhampati, Solving planning graph by compiling it into CSP, Proceedings of international conference on artificial intelligence planning and scheduling (AIPS), 2000.

[Ernst et al 1997] Michael Ernst, Todd Millstein and Daniel Weld, Automatic SAT compilation of planning problems, Proccedings of International Joint Conference on Artificial Intelligence (IJCAI), 1997.

[Kautz & Selman 1996] Henry Kautz and Bart Selman, Pushing the envelope: Planning, propositional logic and stochastic search, Proceedings of the National Conference on Artificial Intelligence (AAAI), 1996.

[Kautz et al 1996] Henry Kautz, David McAllester and Bart Selman, Encoding plans in propositional logic, Proceedings of Knowledge Representation and Reasoning conference (KR), 1996.

[Kautz & Selman 1999] Henry Kautz and Bart Selman, Unifying SAT-based and graph-based planning, Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 1999.

[Li & Anbulagan 1997] Chu Min Li and Anbulagan, Heuristics based on unit propagation for satisfiability problems, Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 1997.

[Mali 1999] Amol Mali, Plan merging and plan reuse as satisfiability, Proceedings of European Conference on Planning (ECP), Durham, UK, 1999.

[Mali 2000] Amol Mali, Enhancing HTN planning as satisfiability, Proceedings of the International Conference on Artificial Intelligence and Soft Computing (ASC), Banff, Alberta, Canada, 2000.

[Mali & Kambhampati 1998] Amol Mali and Subbarao Kambhampati, Encoding HTN planning in propositional logic, Proceedings of the international conference on Artificial Intelligence Planning Systems (AIPS), 1998.

[Smith & Weld 1999] David E. Smith and Daniel S. Weld, Temporal planning with mutual exclusion reasoning, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1999.

[Tsamardinos et al 2000] Ioannis Tsamardinos, Martha E. Pollack and John F. Horty, Merging plans with quantitative temporal constraints, temporally extended actions and conditional branches, Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling (AIPS), 2000.

[Vossen et al 2000] Thomas Vossen, Michael Ball, Amnon Lotem and Dana Nau, Applying integer programming to AI planning, Knowledge Engineering Review, 2000.

[Wolfman & Weld 1999] Steven Wolfman and Daniel Weld, The LPSAT engine and its application to resource planning, Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 1999.

An Incremental Temporal Partial-Order Planner*

Eliseo Marzal and **Eva Onaindia** and **Laura Sebastia**
Dpto. Sistemas Informaticos y Computacion
Universidad Politecnica de Valencia, 46071 Valencia, Spain
{emarzal,onaindia,lstarin}@dsic.upv.es.

Abstract

In this paper we present TANDOR, a domain-independent temporal planner that handles durative actions and returns a set of solution plans ordered by increasing temporal cost. TANDOR guarantees the first solution to be the minimal duration plan; next solution will be a longer plan and so on. This functionality is quite relevant in many domains as the user might be interested in obtaining a good plan in terms of execution cost rather than the minimal duration solution. TANDOR applies a heuristic extracted from a problem relaxation to guide a regression search in an plan space which encodes the different alternative solutions to a problem. Unlike state space temporal planners, the branching factor in TANDOR is vastly reduced thus allowing to obtain a very good performance in a the experimental evaluation.

INTRODUCTION

Classical planning models are not suitable for temporal planning domains where actions take different times, actions can be executed concurrently and plans need to achieve goals within given deadlines. The main goal in temporal planning is to compute the minimal duration plan. This optimal solution usually entails a high cost in the plan execution as more resources or more expensive resources are needed to carry out the plan. This way, a trip from city A to city B will be faster by plane but also more expensive than driving between both cities. Or using a single vehicle to transport a set of objects rather than several vehicles might be a preferable solution if the user wants to use the minimum number of resources as possible.

TANDOR is a domain-independent temporal planner capable to efficiently compute several solution plans for a given temporal problem. TANDOR guarantees the minimal duration plan. Afterwards, it progressively computes other solution plans in increasing temporal cost. TANDOR behaves very much like POCL planners. Unlike standard partial-order planning, TANDOR is a propositional temporal planner which uses the information extracted from a problem relaxation to guide a regression search in a plan space.

*This work has been partially supported by projects DPI2001-2094-C03-03 (MCyT), UPV n. 20010017 and UPV n. 20010980.

The main contribution of TANDOR is its excellent performance and scaling-up as shown in the experimental results for several temporal domains. Unlike most of the recent heuristic temporal planners, namely TP4 (Haslum & Geffner 2001) or SAPA (Do & Kambhampati 2001), which are state space planners, TANDOR works on a space of temporal plans, what allows to vastly reduce the branching factor during the search process. At each plan node TANDOR applies a complete process of conflict checking to obtain all possible temporal solutions.

This paper is organized as follows: next section shows the structure of durative actions used in TANDOR, the following one is a brief revision on the main concepts of POCL planning, section *Components of* TANDOR explains the main components of the temporal planner, section *Conflicts* shows the conflict resolution process, section TPOP *algorithm* gives a complete description of the algorithm implemented in TANDOR; last but one section shows some experimental results obtained with TANDOR and last section concludes.

INTERNAL MODEL OF TIME: DURATIVE ACTIONS

TANDOR handles a very simple model of durative actions. Unlike conservative models of actions (Smith & Weld 1999), durative actions allow to include local conditions and effects to be satisfied at different times during the execution of the action. This approach allow actions to overlap even when their preconditions or effects refer to the same propositions because now all literals are annotated with time points. TANDOR has been adapted to the new version of PDDL language, PDDL 2.1 (Fox & Long 2001). From all the components cited in (Fox & Long 2001), TANDOR makes use of all but *end conditions*.

The two basic components in TANDOR are temporal propositions and temporal actions. A temporal proposition, π, is a tuple $< p, t >$ where p is the proposition and t is the time instant at which p is produced. A temporal action α is a tuple $< a, s, e >$ where a represents the action itself, s is the start time and e the end time of the action ($e = s + \mathsf{dur}(a)$).

Let $\alpha =< a, s, e >$ be a temporal action:

- $\mathsf{Cond}(\alpha) = \mathsf{SCond}(\alpha) \cup \mathsf{Inv}(\alpha)$ where $\mathsf{SCond}(\alpha)$ is the set of conditions to be guaranteed at the start of the action

and $\mathsf{Inv}(\alpha)$ is the set of invariant conditions that must be guaranteed over the execution of the action $[s, e]$.

- $\mathsf{Eff}(\alpha) = \mathsf{SEff}(\alpha) \cup \mathsf{EEff}(\alpha)$ where $\mathsf{SEff}(\alpha) = \mathsf{SAdd}(\alpha) \cup \mathsf{SDel}(\alpha)$ is the set of effects to be asserted at time s, and $\mathsf{EEff}(\alpha) = \mathsf{EAdd}(\alpha) \cup \mathsf{EDel}(\alpha)$ is the set of positive and negative effects to be asserted at time e. Given $\alpha =< a, s, e >$, $\mathsf{SEff}(\alpha) = \{< p, t >: p \in \mathsf{Add(a)} \cup \mathsf{Del(a)} \wedge t = s\}$ and $\mathsf{EEff}(\alpha) = \{< p, t >: p \in \mathsf{Add(a)} \cup \mathsf{Del(a)} \wedge t = e\}$.

TANDOR deals with a discrete model of time and we will assume that conditions and negative effects of a same action can simultaneously occur at the same time point without being this a cause of inconsistency in the model. In the same way, positive effects are available since the first instant of time at which they are produced. That is, we will assume closed temporal intervals for the effects of an action.

REVIEWING SOME POCL CONCEPTS

POCL planners work on a plan space by progressively refining an initial empty plan. Each node in the plan space encodes a partial-order plan which is constructed by incrementally adding all plan components. The basic POPLAN algorithm is shown in Table 1 (Yang 1997).

Algorithm POPLAN (Θ, Π_{init});
Input: A set of planning operators Θ, and an initial plan Π_{init} consisting of a start step and a finish step and a constraint that the start step be before the finish step;
Output: A correct plan if a solution can be found.

```
OpenList := Π_init
repeat
   Π := SelectPlan (OpenList);
   remove Π from OpenList;
   if Correct (Π)= TRUE then return(Π);
   else
      flaw := SelectFlaw(Π);
      if flaw is a threat then
        Succ := Resolve-Threat (flaw, Π);
      else
        Succ := Establish-Precond (Π);
      add all nodes in Succ to OpenList;
until OpenList is empty;
return (fail);
```

Table 1: POPLAN algorithm

The search-control problem in POCL planners occurs in two dimensions. In the first, a decision has to be made as to which node among the set of all frontier nodes should be selected next for expansion. A second dimension of search control is defined as the problem of selecting a next flaw from the selected node to work on.

Threats. A threat represents a potential conflict between an effect of a step S_i in the plan and a causal link when S_i can nullify the link (Peot & Smith 1993). Threats can be solved by applying demotion of S_i, promotion of S_i or by introducing a variable-binding constraint to *separate* the potential values that two variables can be assigned to.

COMPONENTS OF TANDOR

TANDOR behaves very much like POCL planners. Unlike standard partial-order planning, TANDOR is a propositional temporal planner which uses the information extracted from a problem relaxation to guide a regression search in a plan space.

The goal of TANDOR is to compute the solution plans in increasing order of temporal cost. First stage of the algorithm builds a relaxed planning graph which is later used for the calculation of admissible heuristics. During the second stage TANDOR incrementally builds the plan space where each node represents a different alternative to solve the problem.

Creating the Temporal Planning Graph

The Temporal Planning Graph (TPG) is a relaxed graph where delete effects of actions are ignored. The TPG is a directed, layered graph alternating proposition and action levels.

Definition 1 (proposition level) *A proposition level $P[t]$ is made up of all temporal propositions generated at time t. $P[0]$ consists of all the literals in the initial situation.*

$P[t] = \{< p, t >: t = 0 \vee \exists \alpha =< a, s, e > \wedge < p, t > \in \{\mathsf{SAdd}(\alpha) \cup \mathsf{EAdd}(\alpha)\}\}$

Definition 2 (action level) *An action level $A[t]$ is formed by all temporal actions that can be executed at time t and have not appeared in a previous action level.*

$A[t] = \{\alpha =< a, t, e >: \ \forall p \in \mathsf{Cond}(\alpha) \ \exists < p, t' > / \ t' \leq t \ \wedge \forall \beta =< a', s', e' >, s' < t \rightarrow a' \neq a\}$

First level in the TPG is $P[0]$ which contains all literals in the initial situation. Given a proposition level $P[t]$, TANDOR checks whether it is possible to create $A[t]$ according to definition 2. Notice that no special checking is required for invariant conditions as delete effects of actions are ignored in the TPG . Once $A[t]$ is created, the effects of actions in $A[t]$ are added in the corresponding proposition level according to definition 1. Following, TANDOR moves forward in time to the next $P[t]$ and the process is repeated again.

The TPG creation terminates when no new temporal actions can be added in the graph, that is when the last proposition level has been analyzed. Notice the TPG creation does not stop when a proposition level $P[t]$ containing all top-level goals is reached. This is so because we can not guarantee that the solution comprised from $P[0]$ to $P[t]$ is the minimal duration plan as delete effects have been ignored and conflicts among actions too. Consequently, the TPG is a fully extended graph which contains at least one temporal instance of each different action and encodes all different temporal plans for the problem at hand. The value t of the first temporal proposition level $P[t]$ where all top-level goals have appeared is used as a lower bound during the second

stage and will represent the duration of the optimal plan if no harmful interactions occur among actions.

The set of fully instantiated actions extracted from the TPG is used in two ways:

a) in the unification process at the time of finding an action whose effects achieve a precondition

b) to compute the estimated duration of the partial plan comprised in a node

Creating the space of temporal plans

The second stage of TANDOR is a regression search process to create the space of temporal plans. Each node (temporal plan) in the search space is an AND graph, (AG), and represents a partial solution plan. From the information contained in the TPG, TANDOR computes the estimated duration of the partial plan comprised in the AG .

The plan comprised in an AG may contain conflicts or harmful interactions among the actions in the node. The conflict resolution usually entails an updating of the AG estimated value. When this value is higher than the values of the remaining solutions (frontier nodes of the plan space), TANDOR selects another AG and proceeds in the same way.

Following, we detail the composition of AND graphs. Next section explains the types of conflicts and their resolution and the following one shows the complete algorithm of the temporal planner (TPOP).

Structure of AND graphs

Definition 3 (AND graph) *An AG is a tuple $< N, E, h >$ where:*

- *N is a set of temporal actions where each node is the producer of a temporal proposition*
- *E is a set of temporal ordering relations among actions in N*
- *h is the estimated overall duration of the partial solution plan comprised in the AG*

Definition 4 (temporal ordering relation) *Let $\alpha =< a, s, e >$ and $\beta =< a', s', e' >$. A temporal ordering relation between α and β (represented as $\alpha \xrightarrow{<t,d>} \beta$) is a tuple $< t, d >$ where:*

- *t is the type of ordering relation and represents the two time points (start/end times of α and β) between which the relation is posed*
- *d is the minimal temporal distance between both time points*

The four types of ordering relations are:

- $\alpha \xrightarrow{<s-s,d>} \beta$ indicates that β starts its execution after d time units have elapsed from the start time of α, i.e. $s' \geq s + d$
- $\alpha \xrightarrow{<s-e,d>} \beta$ indicates that execution of β will finish after d time units have elapsed from the start time of α, i.e. $e' \geq s + d$

Table 2: Transitiviness property for temporal ordering relations

$\beta \rightarrow \gamma$ / $\alpha \rightarrow \beta$	$s-s, d'$	$s-e, d'$	$e-s, d'$	$e-e, d'$
$s-s, d$	$< s-s, d+d' >$	$< s-e, d+d' >$	$< s-s, d+d'+\mathsf{dur}(\beta) >$	$< s-e, d+d'+\mathsf{dur}(\beta) >$
$s-e, d$	$< s-s, d+d'-\mathsf{dur}(\beta) >$	$< s-e, d+d'-\mathsf{dur}(\beta) >$	$< s-s, d+d' >$	$< s-e, d+d' >$
$e-s, d$	$< e-s, d+d' >$	$< e-e, d+d' >$	$< e-s, d+d'+\mathsf{dur}(\beta) >$	$< e-e, d+d'+\mathsf{dur}(\beta) >$
$e-e, d$	$< e-s, d+d'-\mathsf{dur}(\beta) >$	$< e-e, d+d'-\mathsf{dur}(\beta) >$	$< e-s, d+d' >$	$< e-e, d+d' >$

- $\alpha \xrightarrow{<e-s,d>} \beta$ indicates that execution of β starts after d time units have elapsed from the end time of α, i.e. $s' \geq e + d$
- $\alpha \xrightarrow{<e-e,d>} \beta$ indicates that execution of β will finish after d time units have elapsed from the end time of α, i.e. $e' \geq e + d$

Temporal ordering relations fulfill the transitiviness property. Table 2 shows the results of combining the different types of temporal orderings. Rows represent $\alpha \rightarrow \beta$ and columns $\beta \rightarrow \gamma$.

Ordering relations are also used to set causal links between a producer and a needer action. There are two types of causal links according to the production time of effects:

- $\alpha \xrightarrow{<s-s,d>} \beta$, $d \geq 0$, if $\exists < p, t > \in \mathsf{SAdd}(\alpha) \wedge p \in \mathsf{Cond}(\beta) \wedge t \leq s'$
- $\alpha \xrightarrow{<e-s,d>} \beta$, $d \geq 0$, if $\exists < p, t > \in \mathsf{EAdd}(\alpha) \wedge p \in \mathsf{Cond}(\beta) \wedge t \leq s'$

to represent that a start or end effect of α is used to satisfy a condition of β. Notice that no matter the type of condition of β as both start conditions and invariants require the effect to hold at the beginning of the needer action. We will simplify the representation of causal links by using the notation $\alpha \xRightarrow{<t-s,d>} \beta$, which denotes either of the two above ordering relations.

Obviously, transitiviness is also applied to causal links. For example, if $\alpha \xrightarrow{<e-s,d>} \beta$ and $\beta \xRightarrow{<e-s,d'>} \gamma$, a causal link between α and γ would be represented as $\alpha \xRightarrow{<e-s,d+d'+\mathsf{dur}(\beta)>} \gamma$.

Property 1 (correctness and consistency of an AND graph) *$AG =< N, E, h >$ is correct and consistent if $\forall \beta =< a, s, e > \in N$ $p \in \mathsf{Cond}(\beta)$:*

1) $\exists \alpha \in N : \alpha \xRightarrow{<t-s,d>} \beta \in E$ **and**

2) if $p \in \mathsf{SCond}(\beta) \rightarrow$
$\forall \gamma \in N :< \neg p, t' >\in \mathsf{SDel}(\gamma) \cup \mathsf{EDel}(\gamma) \rightarrow$
$t' < t \leq s \vee t \leq s < t'$
if $p \in \mathsf{Inv}(\beta) \rightarrow$
$\forall \gamma \in N :< \neg p, t' >\in \mathsf{SDel}(\gamma) \cup \mathsf{EDel}(\gamma) \rightarrow$
$t' < t \leq s \vee t \leq e < t'$

Basically, property 1 states that an AG is correct and consistent if there exists a producer action for every condition of all actions in the AG and there is not a contradictory temporal proposition between the production time point and the required time interval.

Definition 5 (temporal solution plan) *A valid temporal plan for a planning problem is the set of nodes N of a correct and consistent* AG.

CONFLICTS

Conflicts arise when property 1 is not satisfied. Particularly, if part *1)* of property 1 holds and part *2)* fails then a *definite* conflict is found in the graph. Otherwise, when the causal link is not established yet, there exists a *potential* conflict in the AG. Conceptually, conflicts are very similar to *threats* in partial-order causal-link planning.

TANDOR is aimed to detect potential conflicts in AND graphs. When a potential conflict is found in an AG, TANDOR creates all possible conflict-free combinations of actions.

Definition 6 (potential conflict) *Let $AG = (N, E, h)$ where $\alpha =< a, s, e >, \beta \in N$. $c(\alpha, \beta)$ is a potential conflict if one of the two following expressions holds:*

1) $p \in \mathsf{SCond}(\alpha) \wedge < p, t' >\in \mathsf{SDel}(\beta) \cup \mathsf{EDel}(\beta) \wedge t' \leq s$

2) $p \in \mathsf{Inv}(\alpha) \wedge < p, t' >\in \mathsf{SDel}(\beta) \cup \mathsf{EDel}(\beta) \wedge t' \leq e$

Solving (avoiding) a potential conflict implies to create a conflict-free AG before solving the conditions of the actions involved in the conflict. Let $c(\alpha, \beta)$ be a potential conflict as defined in 6. There are two general ways of solving potential conflicts.

Method 1. Choose a producer action γ for α such that $< p, t >\in \mathsf{SAdd}(\gamma) \cup \mathsf{EAdd}(\gamma)$ and post the temporal ordering relations $\beta \overset{<t'-t,1>}{\longrightarrow} \gamma$ and $\gamma \overset{<t-s,0>}{\Longrightarrow} \alpha$.

Let $\gamma =< a', s_1, e_1 >$ and $t = s_1$ if $< p, t >\in \mathsf{SAdd}(\gamma)$ or $t = e_1$ if $< p, t >\in \mathsf{EAdd}(\gamma)$:

- if $t' < t \leq s$ then none of the actions has to be moved
- if $t \leq t'$ then the temporal ordering relation $\beta \overset{<t'-t,1>}{\longrightarrow} \gamma$ implies to move γ forward in time a distance of $t' - t + 1$ time units
- if $t > s$ then the causal link $\gamma \overset{<t-s,0>}{\Longrightarrow} \alpha$ implies to move α forward in time a distance of $t - s$ time units.

This is a general method to solve any type of conflict. If the producer action γ is already established in the AG then the conflict becomes a definite conflict.

A common characteristic of this solving method is that it makes necessary the use of two different producer actions for α and β. Figure 1 shows the four possible cases of application of method 1. Dashed lines indicate ordering relations and solid lines causal links. Conditions are placed above the corresponding time point of the action and positive and neagtive effects below the action. Clearly, in the four cases β needs another producer action for its condition p. Notice that this resolution method is also valid in case $< p, s >\in \mathsf{SDel}(\alpha)$ or $< p, e >\in \mathsf{EDel}(\alpha)$.

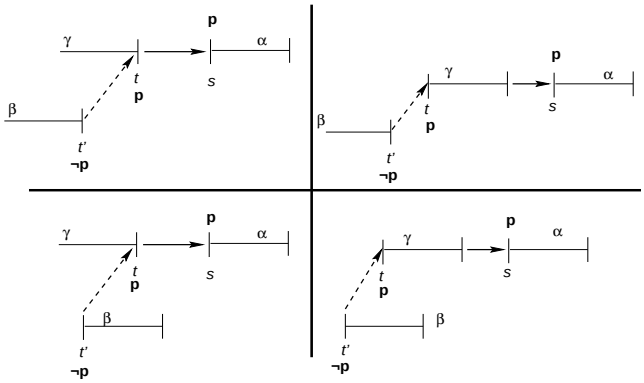


Figure 1: Conflict resolution: method 1

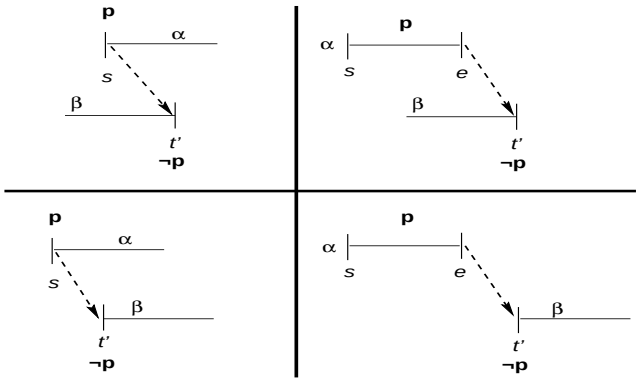


Figure 2: Conflict resolution: method 2

Method 2. The second method is only applicable if it is feasible to set a direct temporal ordering relation between α and β, regardless the producer actions of α and β.

- if $p \in \mathsf{SCond}(\alpha)$ then establish the temporal ordering $\alpha \overset{<s-t',1>}{\longrightarrow} \beta$, which moves β forward in time a distance of $s - t' + 1$ time units
- if $p \in \mathsf{Inv}(\alpha)$ then establish the temporal ordering $\alpha \overset{<e-t',1>}{\longrightarrow} \beta$, which moves β forward in time a distance of $e - t' + 1$ time units

Notice that this resolution method is not valid if $p \in \mathsf{SCond}(\alpha) \wedge < p, s >\in \mathsf{SDel}(\alpha)$ or $p \in \mathsf{Inv}(\alpha) \wedge < p, e >\in \mathsf{EDel}(\alpha)$. In the four cases shown in Figure 2 both α and β can share the same producer action.

Once described the two general conflict resolution methods, we can give a definition of unsolvable conflict.

Definition 7 (unsolvable conflict) *$c(\alpha, \beta)$ is an unsolvable conflict if both α and β share the same producer action and method 2 of resolution cannot be applied, that is it is not feasible to establish a direct temporal ordering between α and β.*

TPOP ALGORITHM

The algorithm to build the temporal POP (TPOP) is very similar to the standard POPLAN algorithm shown in table 1. There are some differences though:

- **Unification process**. While POP planners use a establish − precond procedure to find all possible choices to achieve a goal, TANDOR makes use of the already instantiated actions in the TPG. This way, the pattern matching process is less costly as actions are instantiated only once.
- **Precondition resolution**. When solving a precondition p of an action a_i, TANDOR might use an action in the TPG (new step in POCL planners) or an already existing action in the AG itself (old step in POCL planners). If both choices come from the same operator, TANDOR ignores the creation of a successor node which introduces a new action from the TPG and only keeps one child node with the old action from the AG . This apparent lack of completeness is recovered during the conflict resolution process when applying Method 1. Actually, Method 1 is a technique based on the white-knight concept developed by Chapman (Chapman 1987) to restore a precondition. The use of this so called *restoration* technique has been successfully applied in POCL planners for solving threats (Sebastia, Onaindia, & Marzal 2000). In summary, if using an existing action in the AG to achieve a precondition is a wrong choice then a conflict will eventually arise in the AG and it will be solved by means of the methods exposed in the previous section.

The TPOP algorithm is input the TPG of a problem and returns the minimal duration temporal plan. If the algorithm keeps on being executed it will return other solutions ordered by increasing temporal cost. The algorithm consists of the following steps:

1) TPOP **initialization**. Initially, the TPOP and the first AG , $\mathsf{AG}_0 = (N, E, h)$, is composed of two ficticious actions $\alpha =< a_0, 0, 0 >$, $\beta =< a_n, h, h >\in N$ which represent the initial and final state respectively; $\alpha \overset{<e-s,h>}{\longrightarrow} \beta \in E$ and h is the time t of the first proposition level in the TPG at which all top-level goals are found simultaneoulsy (this will be, in the best case, the optimal plan duration). $\mathsf{Cond}(\beta)$ represents the top-level goals and $\mathsf{Eff}(\alpha)$ the literals in the initial situation. $\mathsf{SetAG} = \{AG_0\}$.
2) **Select AG**.
   2.1 if SetAG is empty then execution of TANDOR is finished.
   2.2 $\mathsf{AG} = min_h \mathsf{SetAG}$.
3) **Select flaw from AG**.
   3.1 if AG contains potential conflicts
   - select a conflict c and create all possible ways of solving c by applying **method 1** and **method 2.**
   - for each method application create an AG : $\{\mathsf{AG}_1, \mathsf{AG}_2, \ldots\}$.
   - $\mathsf{SetAG} = \mathsf{SetAG} - \mathsf{AG} \cup \{\mathsf{AG}_1, \mathsf{AG}_2, \ldots\}$.
   - go to 2

   3.2 select an action $\alpha =< a, s, e >\in N$ in AG such that $\mathsf{Cond}(\alpha)$ are unsolved
   - if none of the actions in N has unsolved conditions then return AG as a temporal solution plan. If another plan solution is required go to 2; otherwise, TANDOR execution is finished.
   - TANDOR searches in the TPG for the set of actions A to generate each $p \in \mathsf{Cond}(\alpha)$
   - remove $\beta \in A$ if β is the inverse action of α
   - remove $\gamma \in A$ if the introduction of γ creates a loop with a predecessor action of α
   - for each combination of valid producer action for conditions in $\mathsf{Cond}(\alpha)$ create an $\mathsf{AG} : \{\mathsf{AG}_1, \mathsf{AG}_2, \ldots\}$.
   - $\mathsf{SetAG} = \mathsf{SetAG} - \mathsf{AG} \cup \{\mathsf{AG}_1, \mathsf{AG}_2, \ldots\}$
   - go to 2

The function to compute the heuristic value of AG is very simple. The estimated duration of the partial plan comprised in AG (start or end time of the final ficticious action β) is determined by the start/end times of the actions in AG . If no conflicts appear in AG then start times of actions are given by the TPG and consequently the value of h will remain the same as the initial value. If some conflicts arise in the graph then the start/end times of actions may change and these new values will be propagated through the graph to update the value of h accordingly.

The propagation process computes the new start/end times of actions according to the changes provoked by the conflict resolution (see section). Then for a given node $\alpha =< a, s, e >$ whose conditions are all solved ($\forall p \in \mathsf{Cond}(\alpha)\ \exists \beta : \beta \overset{<t-s,d>}{\Longrightarrow} \alpha \in E$), the start time of α is computed as $s = max_{t,p \in Cond(\alpha)} < p, t >\in \mathsf{SAdd}(\beta) \cup \mathsf{EAdd}(\beta)$. This process is repeated successively up to reaching the final ficticious action.

There are several criteria that TANDOR applies to prune a node in the search space:

1) an unsolvable conflict is found in the AG
2) the only choice to achieve a condition of an action α is the inverse action of α. This does not apply if β and α share the same producer action and $\beta \overset{<e-s,0>}{\longrightarrow} \alpha$.
3) the only choice to achieve a condition of an action α is by using an action which generates a loop in a branch of the AG .

By applying these criteria as well as the use of some local heuristics as "select firstly the confict with least number of resolution alternatives" or "select firstly the action with the least number of solving choices", we achieve plan spaces with very low branching factors. In order to accomplish efficient temporal planning, branching is as important as lower-bound heuristics.

Table 3: Duration of actions for the first temporal domain

loading	20	unloading	20
fly C1 C2	60	drive C1 C2	200
drive C1 C3	180	drive C2 C3	30

Table 4: Results for the first temporal domain

instance	TPG	Sol. 1	Sol. 2	Sol. 3	TOTAL
1	0.051	0.061	0.085	0.107	0.304
2	0.051	0.082	0.027	0.155	0.315
3	0.063	0.086	0.06	0.144	0.353
4	0.052	0.118	0.092	0.020	0.282

SOME EXPERIMENTAL RESULTS

In this section we show some experimental results obtained with TANDOR. All the tests were run ten times on a SUN Ultra 10 machine. We evaluate the performance of TANDOR on problems from two metric temporal domains.

The first domain is a similar version of the temporal logistics domain. The setting is formed by three cities C1, C2 and C3 with airports in C1 and C2. There is one object to transport from C1 to C3. Table 3 shows the durations of actions (symmetrical actions have the same duration). We have tested different instances of this problem, changing the number and location of planes and trucks among cities. For each instance there are several plans which TANDOR returns in order of increasing duration. We describe only the first three plans for instance 1.

- **instance 1**. One plane in C1; a truck T1 in C1 and another truck T2 in C2. The optimal plan for this instance is to fly from C1 to C2 and then drive from C2 to C3 with truck T2. The total plan duration is 170 time units. Next solution takes 220 time units and corresponds to the plan which drives directly from C1 to C3 with the truck T1. Next plan takes 270 t.u. and consists of driving from C1 to C2 with truck T1 and then drive from C2 to C3 with the same truck.
- **instance 2**. One plane in C1; a truck in C1 and another truck in C3.
- **instance 3**. Two planes and two trucks, one plane and one truck in C1, the other plane and truck in C2.
- **instance 4**. One plane in C2, two trucks in C1 and C2 respectively.

Table 4 shows the results in seconds. First column is the time for the TPG construction, second column shows the time necessary to obtain the first solution, third and fourth column is the additional time to get the second and third plan. Last column shows the total time.

The different times for each solution depends on the number of conflicts found in the resolution process as well as the number of ways of solving those conflicts. As we can see, times are very similar for all the problem instances.

Table 5: Results for the zeno-travel domain

instance	TPG	Sol.	TOTAL
zeno-travel 1	0.002	0.003	0.005
zeno-travel 2	0.007	0.019	0.026
zeno-travel 3	0.009	0.048	0.057
zeno-travel 4	0.012	0.104	0.116

The second domain is the *zeno-travel* domain (Penberthy & Weld 1994) in which it is necessary to transport one, two, three or four objects from one place to another by using only one vehicle. Table 5 shows the running times of TANDOR for the zeno-travel domain. The most significant aspect of these results is not the times themselves but that TANDOR scales up very well to reasonable sized problems. As we can see in table 5 the time for solving one problem is roughly twice as much as the previous instance. In SAPA (Do & Kambhampati 2001) the results obtained for zeno1 to zeno4 in a Sun Ultra 5 machine with 256 MB RAM go from 0.35 to 7.76 seconds when using the *sum-duration* heuristics. And the results of TGP (Smith & Weld 1999) published in (Garrido, Fox, & Long 2001) show that TGP needs less than 0.1 seconds to solve zeno1 and more than 100 seconds to solve either of the other instances.

CONCLUSIONS AND FUTURE WORK

In this paper we have presented TANDOR, an admissible heuristic temporal planner which is able to return different solution plans ordered by increasing temporal cost. This functionality is very relevant is practical domains as the user might be interested in obtaining a good quality plan rather than the minimal duration one.

TANDOR performs a regression search in a plan space and applies a heuristic derived from a relaxed temporal planning graph. TANDOR performs very well in typical temporal domains as shown in the preliminary experimental results. The application of lower-bound heuristics in a search process with very low branching factors is aimed at scability with reasonable plan quality.

Currenly, we are investigating on the application of POP heuristics on threats and goal selection to TANDOR. In the long term, our objective is to incorporate resource management in TANDOR to integrate planning and scheduling.

References

Chapman, D. 1987. Planning for conjuntive goals. *Artificial Intelligence* 32:333–377.

Do, M., and Kambhampati, S. 2001. Sapa: a domain-independent heuristic metric temporal planner. In *Proc. European Conference on Planning (ECP-01)*.

Fox, M., and Long, D. 2001. PDDL2.1: an extension to PDDL for expressing temporal planning domains. Technical report, University of Durham, UK.

Garrido, A.; Fox, M.; and Long, D. 2001. A temporal planning system to manage level 3 durative actions of pddl2.1. In *PLANSIG 2001, (submitted)*.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In *Proc. European Conference on Planning (ECP-01)*, 121–132.

Penberthy, J., and Weld, D. 1994. Temporal planning with continuous change. *Proc. 12th Nat. Conf. on AI*.

Peot, M., and Smith, D. 1993. Threat-removal strategies for partial-order planning. In *Proc. 11th Nat. Conf. on AI, AAAI-93*.

Sebastia, L.; Onaindia, E.; and Marzal, E. 2000. A graph-based approach for pocl planning. In *Horn, W. ed., Proc. 14th European Conf. on AI (ECAI-2000)*, 531–535.

Smith, D., and Weld, D. 1999. Temporal planning with mutual exclusion reasoning. In *Proc. 16th Int. Joint Conf. on AI (IJCAI-99)*, 326–337.

Yang, Q. 1997. *Intelligent Planning. A Decomposition and Abstraction Based Approach*. Berlin, Heidelberg: Springer-Verlag.

Handling Durative Actions in a Continuous Planning Framework

Alexandra Coddington
University of Durham, Science Laboratories, Durham DH1 3LE, UK

Abstract

This paper describes a continuous planning framework to be used by a planning agent which is situated within an environment, in which goals have priorities and deadlines while actions have duration. In contrast to much work in planning in which a set of goals is posed to a planner which then generates a complete plan to achieve such goals, this framework assumes that goals may be generated continuously, which requires the interleaving of planning and execution. Constraints upon time may mean it is not possible for all goals to be achieved – as a consequence the planning agent must be able to prioritise its goals. A crucial component of this framework is a temporal manager which enables the planner to reason about whether or not there is sufficient time available to achieve all goals, and to calculate deadlines for actions and outstanding subgoals. In this paper, the way in which the partial order planning paradigm could be extended to reason with PDDL2.1 level 3 durative actions is also investigated with a view to incorporating this paradigm within the continuous planning framework. The issue of plan metrics is discussed as the continuous planning framework offers the possibility of evaluating the degree to which plans support the planning agent's *motivations* as well as taking into account the number of high priority goals that have been achieved.

Introduction

The focus of much work on planning is primarily concerned with developing planners which are efficient and which produce plans that are both sound and complete (Blum & Furst 1995; Bacchus & Ady 2001; Smith & Weld 1999). In these systems, goals are presented by an external user, and planning ceases either once a plan has been generated which achieves these goals or if the planner fails to generate a plan. There is no facility for the achievement of new goals that are presented to the planner once the planning process has commenced. In addition, once a plan has been created it remains unexecuted. The framework introduced in this paper has been designed for an agent situated within an environment which requires continuous planning. It is assumed that the agent has a set of *motivations* which enable it to generate and prioritise goals. New goals may be posed to the system while it is planning, which requires planning and execution to be interleaved. Goals have deadlines and actions take time to execute which means a core component of the framework is responsible for reasoning about whether or not goals may be achieved by their deadlines, and for assigning deadlines to actions and subgoals. Goals also have an associated priority so that instead of simply failing to return a plan if there is insufficient time to satisfy one or more goals by their deadlines, the system is able to abandon the achievement of some goals in favour of achieving others. Finally, the agent's *motivations* may be used as part of a heuristic to enable the planner to select the most promising plan for further refinement.

This paper presents the continuous planning framework in section and then describes *motivations* and how they enable the planning agent to generate and prioritise goals, and select good plans for subsequent refinement. The core component, the *Plan to achieve goal* component which uses an extended partial order planning paradigm is presented and discussed. The remainder of the paper then describes how the partial order planning paradigm could be extended to reason with the extra expressive power of PDDL2.1 level 3 durative actions. Finally, conclusions and further work are discussed.

The Continuous Planning Framework

The continuous planning framework is illustrated in figure 1 and was designed for an autonomous agent (such as a robot) situated within an environment. An underlying assumption is that the agent (via the *Generate/update goals* component) can generate new goals in response to its current context (this is encapsulated within the current initial state model, the plan, as well as the agent's motivations). For the purpose of this paper it is assumed that this component can be emulated by a user keying in new goals. Solid rectangular boxes represent the various processes in the framework that are the focus of this research – these processes have been implemented using Allegro Common Lisp. The dashed boxes represent two components responsible for updating the agent's motivations and and for generating goals.

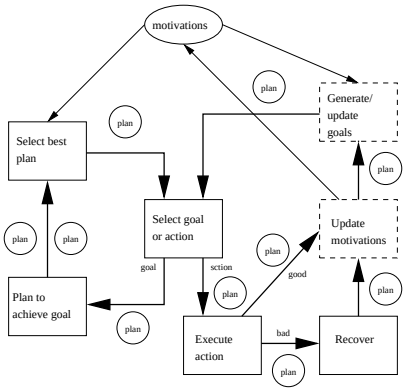


Figure 1: The Continuous Planning/Execution Framework

These have not been implemented and will not be discussed in this paper – details can be found in (Norman 1997; Coddington 2001). The ovals represent knowledge sources – these represent the planning problem, (including the initial and goal states as well as the current plan) and the agent's motivations.

This framework can be viewed as a dynamic system in which the agent continually generates goals in response to its perceived current and predicted future states of the environment as well as in response to its motivations. Each newly generated goal has a deadline by which it must be achieved, as well as a value indicating its importance or priority. Newly generated goals are added to the representation of the planning problem. The process *Select goal or action* determines whether one of the goals should be achieved or whether one of the actions (belonging within a plan) should be executed. When this process chooses to achieve a goal, the goal is passed to a planner which plans to achieve the goal. An important part of the planning process involves determining whether or not goals may be achieved by their deadlines as well as assigning deadlines to actions and subgoals. The planner generates a search space of alternatives when planning, which requires a plan evaluation metric to select the most promising plan for further refinement (*Select best plan*). This metric is designed to take into account the total duration, the number of actions, the number of high priority goals that have been achieved, as well as the degree to which the plan supports the agent's motivations.

When a decision is made to execute an action, the *Execute action* component updates the plan and the model of the current state to reflect the changes that have occurred following execution. If the actual outcome differs significantly (i.e. enough to undermine the plan in some way) from the predicted outcome, the component *Recover* is responsible for repairing the plan. In addition, as a consequence of changes to the environment and plan following execution, the agent's motivations may change (these are updated by the component *Update motivations*), which in turn may cause new goals to be generated or existing goals to be updated.

The continuous planning framework is similar to Sage (Knoblock 1995) – an extension of UCPOP (Penberthy & Weld 1994) which supports simultaneous actions execution and which integrates planning and execution. Sage however, does not reason about time or take into account the context of the planning agent when choosing plans for subsequent refinement. The remainder of the paper will focus mostly on the component responsible for generating plans. A full description of the remaining components can be found in (Coddington 2001).

Motivations

Motivations may be thought of as long term higher level drives or emotional states which direct an agent which is situated within an environment to both plan and act. Associated with each motivation is a value indicating its current weight – this value changes over time and provides a driving force directing the generation of goals to satisfy the motivation. For example, a truck-driving agent might have a motivation concerned with conserving fuel. The weight associated with this motivation depends upon the level of fuel in the truck – as the level of fuel decreases and falls below some threshold, a goal to replenish the fuel supply will be generated. The weight associated with motivations has a direct effect upon the priority of goals generated to satisfy those motivations. Finally, whilst acting to achieve goals, the agent may cause changes to its motivations. Through executing an action, an agent may support or undermine its motivations. For example, the truck-driving agent will undermine the motivation to conserve fuel when it drives from one location to another, and will support that motivation when it refuels. It is therefore possible to evaluate the degree to which the actions in a plan (and therefore the plan itself) support or undermine the agent's motivations. This is exploited as part of a plan evaluation metric. In summary, motivations, together with the agent's context (the perceived current state and predicted future states captured in the plan), enable the agent to generate and prioritise goals with deadlines, and enable the agent to evaluate plans, favouring plans which best support the motivations. Full details concerning motivations and the way in which they cause goals to be generated can be found in (Norman 1997).

Planning to achieve goals

Figure 2 shows the subcomponents of the planner (*Plan to achieve goal*) in more detail. A goal (selected by the component *Select goal or action*) is presented to the planner which generates a plan to achieve that goal. Currently, the planner uses an extended partial order planning paradigm (for example SNLP (McAllester & Rosenblitt 1991)). Once

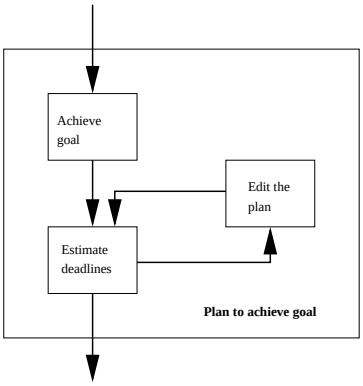


Figure 2: Planning to achieve a goal

a plan has been generated, the component *Estimate deadlines* is responsible for both determining whether there is sufficient time available to achieve all goals in the plan as well as for assigning deadlines to actions (and therefore subgoals). If this process fails it means there is insufficient time available to achieve all of the goals within the plan in which case the plan is edited to remove a goal, together with its associated actions and constraints. Once a plan has been edited, the *Estimate deadlines* component reestimates deadlines. When the temporal component has successfully assigned deadlines to actions, the process is complete.

The *Plan to achieve goal* component of the continuous planning framework was implemented using an extended partial order planning paradigm for several reasons. Firstly, partial order planners output plans that offer a higher degree of execution flexibility than those generated by Graphplan (Blum & Furst 1995) and state search planners and are arguably better suited for frameworks in which planning and execution are interleaved (Nguyen & Kambhampati 2001). In particular, the set of persistence constraints (which maintain the truth of preconditions), may be used when monitoring the outcome of execution to see whether the remaining plan is still valid. In addition, partial order planning is arguably more suited than Graphplan or state search planners to the requirement that new goals may be generated during the planning process as the new goals may simply be added to the set of outstanding goals without affecting the planning process. Graphplan style planners, in contrast, would have to recommence the plan extraction process to take into account the new goals. Smith (Smith, Frank, & Jonsson 2000) argues that partial order planners offer a more promising approach for handling domains with durative actions, temporal and resource constraints. The main drawback of partial order planning has been the lack of a good heuristic for selecting plans for further refinement; search control is of fundamental importance for partial order planning. However, recent work by (Nguyen & Kambhampati 2001) challenges the prevailing pessimism about the scalability of partial order planning by presenting novel heuristic control techniques.

Achieving a goal

The partial order planning paradigm has been extended in two ways to support the fact that goals have an associated deadline and a value indicating their importance, while actions have duration.

- When creating a new action or further instantiating an existing action in order to achieve a goal it is necessary to estimate the duration of that action. The duration of an action may depend upon the values assigned to its parameters. For example, the duration assigned to an instance of the operator schema *drive-to(?x, ?y)* will depend upon the values assigned to the variables *?x* and *?y*. In this case, the exact duration can only be determined when both *?x* and *?y* have been instantiated. In the current implementation of the framework, a worst case estimate of the duration of each incomplete action instantiation is provided which requires a degree of domain knowledge. For example, if the domain contains a network of locations within a town, the worst case estimate of the duration associated with instances of the *drive-to(?x, ?y)* would be the time taken to travel between the two furthest apart locations.

- In order to edit the plan, a record must be kept of the dependencies that exist between actions and goals. Currently, each action contains a list of the goals to which they contribute.

- Goals have a value indicating their priority. Actions which contribute to those goals are also assigned a value indicating their priority – if an action contributes to a single goal, the action inherits the value indicating the importance of that goal, if the action contributes to more than one goal it is assigned the sum of the values indicating the importance of each goal. Preconditions of actions have the same priority as their associated action.

- Actions are assigned values indicating the degree to which they support the agent's motivations (for further details see (Coddington 2001)).

Estimating Deadlines

Actions and their associated preconditions must be assigned a deadline in order that the planner can ensure that the goal to which they contribute will be met by its deadline. The purpose of the *Estimate deadlines* component is twofold.

- To enable the *Select goal or action* component to determine which goal is to be achieved or whether an action is to be executed.

- To reason about whether there is sufficient time available to achieve all goals by their respective deadlines.

The algorithm adopts a pessimistic approach when estimating deadlines – if actions are only partially ordered with respect to each other those actions are each assigned the earliest possible deadline. For example, if three actions a_1, a_2 and a_3, each with a duration of 3 minutes and which remain unordered with respect to each other, are chosen to achieve a goal with the deadline 17:00, the algorithm estimates the deadline for each action to be 16:51. This means that during the early stages of plan refinement the deadlines estimated for actions and subgoals are likely to be too early. This is in contrast to DEVISER (Vere 1983) which estimates execution windows bounded by the earliest possible execution time (which is too early) as well as the latest possible execution time (which is too late). If there is insufficient time available to achieve a goal, the algorithm fails and returns the goal.

Editing the plan

If there is insufficient time available to achieve a goal, the *Edit the plan* component removes that goal together with its associated actions and constraints. This requires a record of the dependencies between actions and goals to be maintained as described in section . Once a plan has been edited, deadlines are reestimated for the actions in a plan. Should there still be insufficient time available to achieve all goals, the plan is again edited. This process continues until there is sufficient time available to execute the remaining plan.

Evaluating Plans

A plan evaluation heuristic has been implemented which takes into account the degree to which a plan supports the agent's motivations, the number of higher priority achieved goals, the total execution time, as well as the number of actions. Further details of the heuristic may be found in (Coddington 2001).

Discussion

The decision to edit the plan once it has been determined that there is insufficient time available to achieve all of the goals, was based upon the desire to emulate human decision making – humans tend to abandon some goals in favour of others if there is insufficient time available to achieve all goals. It seems preferable to preserve as much of the original plan as possible as opposed to replanning from scratch. However, this approach has various associated costs – in particular, it is necessary during the planning process to maintain a record of the dependencies between actions and goals to facilitate plan editing. In addition, once a plan has been edited, deadlines have to be reassigned to the remaining actions in the plan, and, if there is still insufficient time available, the cycle of editing and reassigning deadlines is repeated. An alternative approach would be to simply replan from scratch, once it is established that there is insufficient time available to achieve all goals, by presenting only a subset (selected by taking into account the priorities and deadlines of each goal) of the original set of goals (generated by the component *Generate/update goals*) to the planner. In the future it is intended to perform a set of experiments to determine whether or not the decision to edit the plan is more or less efficient than replanning from scratch. If replanning from scratch proves to be less costly, some of the main benefits of partial order planning such as being able to plan to achieve goals using a skeletal partial order plan, will be lost.

Partial order planning for PDDL2.1 level 3 durative actions

The partial order planning paradigm used as the basis for the *Plan to achieve goal* component of the continuous planning framework makes the classical temporal planning assumption whereby actions with duration are viewed as a black box – the preconditions of durative

actions must be true at the starting point of execution and remain true throughout the interval during which the action is executed, while effects become true at the end point of execution but are undefined during the interval of execution. Both temporally extended GraphPlan based systems such as TGP (Smith & Weld 1999) and TPSYS (Garrido, Onaindía, & Barber 2001) and partial order planners such as DEVISER (Vere 1983) make this assumption about durative actions. However, this assumption excludes many valid plans as there is no way of syntactically distinguishing between preconditions (propositions that are required to be true only until the starting point of the action) and invariant conditions (propositions that are required to remain true throughout the interval of execution), while effects are only defined at the end point of execution.

In contrast, PDDL2.1 level 3 durative actions (Fox & Long 2001) provide greater expressive power by allowing the domain modeller to specify local pre and postconditions of the end-points of the interval over which execution of the durative action takes place, as well as any invariant conditions that must hold throughout that interval (Fox & Long 2001). This is achieved by using temporally annotated conditions and effects: the annotation of a condition states whether the associated proposition must be asserted at the *start* of the interval, the *end* of the interval or *over* the interval; the annotation of an effect asserts whether the proposition occurs immediately or at the end point of the interval. Figure 3 shows how a PDDL2.1 level 3 durative action models a person boarding an aeroplane. A number of researchers have recently developed extensions of GraphPlan or state search planners capable of reasoning with PDDL2.1 level 3 durative actions (Do & Kambham-

```
(:durative-action board
   :parameters (?p - person ?a - airplane ?c - city)
   :condition (and (at start (at ?p ?c))
                   (at start (at ?a ?c))
                   (over all (at ?a ?c)))
   :effect (and (at start (not (at ?p ?c)))
                (at end (in ?p ?a))))
```

Figure 3: A PDDL2.1 `board` operator.

pati 2001; Garrido, Fox, & Long 2001).

Since the advent of GraphPlan and the many GraphPlan inspired successors, partial order planning has been neglected due to its comparatively poor performance. However, the extra burden of reasoning about time even making the simplistic black-box assumptions about durative actions (stated above) causes the performance of temporally extended GraphPlan based systems (such as (Smith & Weld 1999)) or state search planners to deteriorate in comparison to their performance in non-temporal domains. The extra expressive power of PDDL2.1 with regard to modelling durative actions seems likely to lead to an even greater deterioration in performance if only because of the additional constraint reasoning that must be done to ensure temporal consistency (Coddington, Fox, & Long 2001).

Partial order planning (McAllester & Rosenblitt 1991; Penberthy & Weld 1994), on the other hand, is more suitable for modelling concurrency between actions with different durations, and, because it avoids full instantiation, may be more efficient when reasoning with planning domains specified using PDDL2.1. It was therefore decided to investigate extending the partial order planning algorithm to solve problems specified using the level 3 durative action specification of PDDL2.1 level 3 with a view to incorporating it in the continuous planning framework described in section . In the remainder of this paper, the algorithm is described in further detail.

Extending partial order planning

One way of planning with PDDL2.1 level 3 durative actions (see (Fox & Long 2001)) is to decompose them into their instantaneous start and end actions, taking care to maintain the relationship between start and end and also between those points and any invariant conditions specified by the durative action. The meaning of a durative action is obtained by the construction of two instantaneous *end-point* actions, with a standard STRIPS semantics (assuming there are no non-atomic conditions or effects), and a collection of instantaneous *monitoring* actions responsible for maintaining invariant conditions over the specified duration. This approach provides a simple basis for handling durative actons in a partial order framework.

If no invariant conditions (of the form `(over all p)`) are specified, a durative action DA can simply be transformed into two STRIPS actions, one for each of the endpoints, *start* and *finish* associated with DA. When invariants are specified it is necessary to ensure that the invariant remains true over the interval that occurs between the *start* and *end* point associated with DA. In a partial order planner this may be achieved by modelling invariant conditions as special causal links which may be protected using standard partial order planning threat resolution procedures. The only aspect of durative actions which is not captured in this transformation using standard partial order planning machinery is the duration associated with DA. This may be modelled by minor modifications to both the representation of temporal constraints as well as to the temporal constraint consistency checker. The extension that is required is to ensure that if an action a_{start} is temporally constrained to come after b_{start}, and a_{end} is constrained to come before b_{end}, then the duration of a is strictly less than the duration of b. In the partial order framework, checking that this requirement is satisfied will help to prune inconsistent alternatives early. Alternatively it might be left to the final partial plan linearisation process to ensure that sufficient time elapses between the start and end points of durative actions, but this is a less efficient solution because of the failure to identify some temporally inconsistent plans.

The process is as follows: When instantiated, a durative action DA is converted into two actions DA_{start} and DA_{end} which are added to the actions A belonging to the partial plan P.

1. The *name* field of DA_{start} (DA_{end}) is the *name* field of DA appended with the suffix *start* for DA_{start} (or *end* for DA_{end}).

2. The *parameters* field of DA_{start} (DA_{end}) contains the set of typed variables belonging within the precondition and effect propositions of DA_{start} (DA_{end}).

3. The *precondition* of DA_{start} (DA_{end}) is equal to the conjunction of the set of all propositions *p*, such that `(at start p)` (`(at end p)`) is a condition of DA.

4. The *effect* of DA_{start} (DA_{end}) is equal to the conjunction of the set of all simple effect propositions *e*, such that `(at start e)` (`(at end e)`) is an effect of DA.

Should the durative action DA contain invariant conditions of the form `(over all p)`, causal links are created of the form $DA_{start} \xrightarrow{p} DA_{end}$ and added to the set of causal links L. The implications of modelling invariant conditions as causal links is discussed further in section below.

In addition, the temporal constraint $DA_{start} \prec DA_{end}$ is added to the set of constraints O.

When the level 3 durative action template *board* described in figure 3 is selected to achieve the goal $(in\ ernie\ plane)$ it is instantiated and transformed into the actions $board_{start}$ and $board_{end}$ of figure 4.

The temporal constraint $board_{start} \prec board_{end}$ is added to the set of temporal constraints O.

The invariant condition *(over all (at plane ?c1))* is transformed into a causal link

$$board_{start} \xrightarrow{(at\ plane\ ?c1)} board_{end}$$

and added to the set of causal links L. The justification for this is discussed in the following section.

Modelling Invariant Conditions

The decision to model invariant conditions of the form `(over all p)` as causal links of the form $DA_{start} \xrightarrow{p} DA_{end}$ assumes (if the causal link is interpreted in the traditional partial order planning manner) that the action DA_{start} establishes the invariant condition p (i.e. DA_{start} contains p as an effect proposition) while the action DA_{end} consumes p (i.e. DA_{end} contains p as a condition proposition). In fact, there are three cases to consider:

- DA_{start} contains p as a precondition;
- DA_{start} contains p as an effect;
- DA_{start} does not contain p either as a precondition or as an effect.

In the first of these cases the causal link $DA_{start} \xrightarrow{p} DA_{end}$ is added, and the goal (p, DA_{start}) is added to the open conditions of the plan. This expresses the requirement that p be maintained over the whole interval of the action, but it is a non-standard use of causal links to use them to promise that a condition will be maintained before it has actually been achieved. On the other hand, the plan will be invalidated if p cannot be achieved for DA_{start}, and the sooner it is known that p must be maintained the less wasted search will be incurred.

The second case is a simple one - the causal link $DA_{start} \xrightarrow{p} DA_{end}$ is added to the plan when DA_{start} and DA_{end} are added, because DA_{start} is itself the achiever of its own invariant condition.

The last case, in which p is an invariant but neither a precondition or a start effect, is slightly more subtle. In fact, because the temporal relations between time points in the plan are all strict precedence relations (partial order planners typically do not reason about synchronous activity) this case can be treated as equivalent to the first. That is, the causal link $DA_{start} \xrightarrow{p} DA_{end}$ is added to the plan, along with the goal (p, DA_{start}). The goal ensures that an achiever will be found for p, the causal link ensures that p will be preserved until the end point. The subtlety exists because, if we could exploit synchronicity it would be necessary to be precise about the exact point at which p needs to be asserted to satisfy the invariant. Because we cannot, we are forced to ensure that p is asserted strictly before the point at which it is required (which is immediately after application of the action).

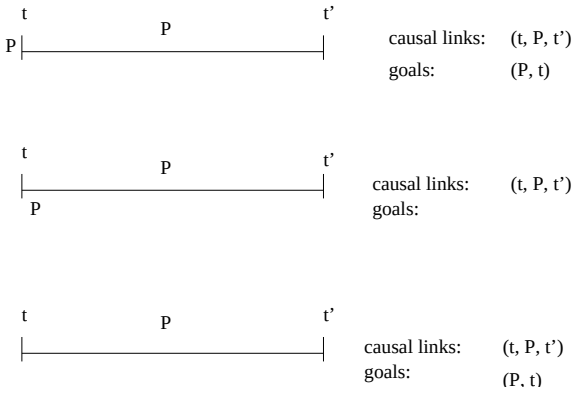


Figure 5: Three Invariant Situations

Figure 5 describes the three cases and the causal links that must be added.

The fill-bath operator, shown in figure 6, is an example of the second case above. The invariant condition *(tap-on ?b)* is achieved by the fill-bath action itself and has to be maintained throughout the filling interval. The *board* operator, given in section 3, is an example of the first case (the plane must be at the city as a condition of board and throughout the duration of boarding).

Conclusions and Further Work

In this paper a continuous planning framework was presented in which it is assumed that a situated planning agent is able to generate and prioritise goals taking into account its context and *motivations*. Goals have deadlines and actions have duration which means it may not be possible to achieve goals by their deadlines. In addition, planning and execution must be interleaved while the agent's motivations may be used as part of a heuristic to select the most promising plan for further refinement. The component responsible for generating plans (*Plan to achieve goal*) was based upon the partial order planning paradigm which was extended to reason about whether or not there is sufficient time available

```
(:action board-start                          (:action board-end
   :parameters (ernie plane ?c1 - city)          :parameters (ernie plane)
   :condition (and (at ernie ?c1)                :condition ()
                   (at plane ?c1))
   :effect (not (at ernie ?c1)))                 :effect (in ernie plane))
```

Figure 4: The PDDL2.1 `board` operator is converted into two simple action instances, `board-start` and `board-end`.

```
(:durative-action fill-bath
   :parameters (?b)
   :duration (= ?duration (/ capacity flow))
   :conditions (and (at start (plug-in ?b)
    (at start (= (level ?b) 0))
    (at start (tap-off ?b))
    (at start (in-bathroom))
    (at end (in-bathroom))
    (over all (plug-in ?b))
    (over all (tap-on ?b))
    (over all (bath-filling ?b))))
   :effect (and (at start (tap-on ?b))
                (at start (not (tap-off ?b)))
                (at start (bath-filling ?b))
                (at end (not (tap-on ?b)))
                (at end (tap-off ?b))
                (at end (not (bath-filling ?b)))
                (at end (assign (level ?b) capacity))))
```

Figure 6: A PDDL2.1 fill-bath operator.

to achieve all goals, to estimate deadlines for actions, and to maintain a record of the dependencies between actions and goals to facilitate plan editing. The degree to which plans support the motivations of an agent, together with the number of goals of high priority offer an extra plan metric when selecting the best solution to a planning problem – the use of plan metrics other than the number of actions in a plan were discussed by the developers of PDDL2.1 (Fox & Long 2001).

Because the original implementation of this component made the classical black-box assumption concerning durative actions it was decided to investigate the feasibility of extending the partial order paradigm to reason about PDDL2.1 level 3 durative actions which give the domain modeller greater expressive power. The required extensions described in this paper are simple and elegant, but more work is needed to confirm whether or not this elegance is bought at the expense of an unmanageable search problem. A partial order planner capable of reasoning with PDDL2.1 durative actions is currently being implemented and future work will involve experimenting with search control in this system by investigating the heuristic techniques described in (Nguyen & Kambhampati 2001), symmetry-breaking techniques (Fox & Long 1999) and goal-ordering strategies (Porteous, Sebastia, & Hoffmann 2001).

Finally, once the partial order planning component *Plan to achieve goal* has been adapted to reason with PDDL2.1 durative actions, minor modifications will have to be made to the component responsible for assigning deadlines to actions *Estimate deadlines* to cope with the fact that durative actions are now represented using two consecutive start and end point instantaneous STRIPS actions. Such extensions will enable the continuous planning framework described in this paper to reason about goals with deadlines and durative actions with greater expressive power.

References

Bacchus, F., and Ady, M. 2001. Planning with resources and concurrency: a forward chaining approach. In *Proc. of 17th IJCAI*.

Blum, A., and Furst, M. 1995. Fast planning through plan-graph analysis. In *Proceedings of IJCAI-95*.

Coddington, A. M.; Fox, M.; and Long, D. 2001. Handling durative actions in classical planning frameworks. In *Proceedings of the 20th Workshop of the UK Planning and Scheduling SIG*, 44–58. ISSN 1368-5708.

Coddington, A. M. 2001. *Self-motivated Planning in Autonomous Agents*. Ph.D. Dissertation, University of London.

Do, M. B., and Kambhampati, S. 2001. Sapa: A domain-independent heuristic metric temporal planner. In *Proceedings of the European Conference on Planning*.

Fox, M., and Long, D. 1999. The detection and exploita-

tion of symmetry in planning domains. In *Procs. of the 15th Int. Joint Conf. on Artificial Intelligence*.

Fox, M., and Long, D. 2001. PDDL2.1 : An extension to PDDL for expressing temporal planning domains. Technical report, Department of Computer Science, University of Durham, Durham, DH1 3LE, UK.

Garrido, A.; Fox, M.; and Long, D. 2001. A temporal planning system to manage level 3 durative actions of PDDL2.1. In *Proceedings of the 20th Workshop of the UK Planning and Scheduling SIG*, 127–138. ISSN 1368-5708.

Garrido, A.; Onaindía, E.; and Barber, F. 2001. Time-optimal planning in temporal problems. In *Proceedings of the European Conference on Planning (ECP-01)*.

Knoblock, C. A. 1995. Planning, execution, sensing and replanning for information gathering. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of the 9th National Conference on AI*, 634–639.

Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-2001)*.

Norman, T. J. F. 1997. *Motivation-based Direction of Planning Attention in Agents with Goal Autonomy*. Ph.D. Dissertation, University of London.

Penberthy, J. S., and Weld, D. S. 1994. Temporal planning with continuous change. In *Proceedings of AAAI-94*, volume 2, 1010–1015.

Porteous, J.; Sebastia, L.; and Hoffmann, J. 2001. On the extraction, ordering and usage of landmarks in planning. In *Proceedings of the European Conference on Planning*.

Smith, D. E., and Weld, D. S. 1999. Temporal planning with mutual exclusion reasoning. In *Proceedings of IJCAI-99, Stockholm*, 325–337.

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

Vere, S. A. 1983. Planning in time: Windows and durations for activities and goals. *Pattern Analysis and Machine Intelligence* 5:246–267.

Improving the Temporal Flexibility of Position Constrained Metric Temporal Plans

Minh B. Do & Subbarao Kambhampati
Department of Computer Science and Engineering
Arizona State University, Tempe AZ 85287-5406
{binhminh,rao}@asu.edu

Abstract

In this paper we address the problem of post-processing position constrained plans, output by many of the recent efficient metric temporal planners, to improve their execution flexibility. Specifically, given a position constrained plan, we consider the problem of generating a partially ordered (aka "order constrained") plan that uses the same actions. Although variations of this "partialization" problem have been addressed in classical planning, the metric and temporal considerations bring in significant complications. We develop a general CSP encoding for partializing position-constrained temporal plans, that can be optimized under an objective function dealing with a variety of temporal flexibility criteria, such as makespan. We then present some greedy value ordering strategies that are designed to efficiently generate solutions with good makespan values for these encodings. We demonstrate the effectiveness of our greedy partialization approach in the context of a recent metric temporal planner that produces p.c. plans. We also briefly discuss and evaluate an extension of our partialization approach for temporal plans with resource constraints.

1 Introduction

Of late, there has been significant interest in synthesizing and managing plans for metric temporal domains. Plans for metric temporal domains can be classified broadly into two categories–"position constrained" (p.c.) and "order constrained" (o.c.). The former specify the exactly start time for each of the actions in the plan, while the latter only specify the relative orderings between the actions. The two types of plans offer complementary tradeoffs *vis a vis* search and execution. Specifically, constraining the positions gives complete state information about the partial plan, making it easier to control the search. Not surprisingly, several of the more effective methods for plan synthesis in metric temporal domains search for and generate p.c. plans (c.f. TLPlan[1], Sapa[3], TGP [18]).

At the same time, from an execution point of view, o.c. plans are more advantageous than p.c. plans –they provide better execution flexibility both in terms of makespan and in terms of "scheduling flexibility" (which measures the possible execution traces supported by the plan [20; 15]). They are also more effective in interfacing the planner to other modules such as schedulers (c.f. [19; 11]), and in supporting replanning and plan reuse [21; 9].

A solution to the dilemma presented by these complementary tradeoffs is to search in the space of p.c. plans, but post-process the resulting p.c. plan into an o.c. plan. Although such post-processing approaches have been considered in classical planning ([10; 21; 2]), the problem is considerably more complex in the case of metric temporal planning. The complications include the need to handle the more expressive action representation and the need to handle a variety of objective functions for partialization (in the case of classical planning, we just consider the least number of orderings)

Our contribution in this paper is to first develop a Constraint Satisfaction Optimization Problem (CSOP) encoding for converting a p.c. plan in metric/temporal domains into an o.c. plan. This general framework allows us to specify a variety of objective functions to choose between the potential partializations of the p.c. plan. We then develop a greedy algorithm for partialization, which can be seen as specific variables and value ordering strategies over the CSOP encoding. We will demonstrate the effectiveness of these partialization algorithm in the context of a recent metric/temporal planner called Sapa[3]. Our results show that the temporal flexibility measures, such as the makespan, of the plans produced by Sapa can be significantly improved while retaining Sapa's efficiency advantages.

The paper is organized as follows. In Section 2, we discuss the background on action representation that we assume for the temporal planning problem and the definitions related to the partialization problem. Then, in Section 3 we discuss the CSOP encoding for the partialization problem. Section 4 focuses on how the CSOP encoding can be solved. In Section 4, we also provide a greedy variable and value ordering strategies for solving the encoding. The empirical results for this greedy ordering strategy are provided in Section 5. In Section 6, we show how the partialization encoding can be extended to handle temporal planning problems with continuous changes. Section 7 discusses the related work and Section 8 presents our conclusions.

2 Preliminaries

2.1 Action representation

The representation of actions that we assume in this paper is similar to that used on [1], and [3]. Here, we shall review the temporal aspects of the representation, postponing the discussion of resource consumption aspects to Section 6. Each action A has a duration D_A, starting time st_A, and an end time $et_A = st_A + D_A$. The preconditions of an action may either be instantaneous or durative and their effects may occur at any time point during their execution. Action A has preconditions $p \in Pre(A)$ that may be required to be true for duration $[st^p_A, et^p_A]$ such that $st_A \leq st^p_A \leq et^p_A \leq et_A$. Figure 1 shows graphically the action $A = load(p, t, l)$ ($load(package, truck, location)$). In this action, precondition $p_1 = at(p, l)$ only needs to be true at the starting point of A ($st^{p_1}_A = et^{p_1}_A = st_A$), precondition

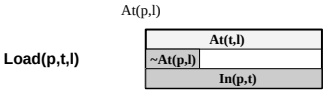


Figure 1: Action Example

$p_2 = at(t,l)$ needs to be true throughout the duration of A ($st_A^{p_2} = st_A, et_A^{p_2} = et_A$)). We need a period of time $d < D_A$ to achieve the effect $e_1 = \neg At(p,l)$, which indicates that the package is not on the ground ($st_A^{e_1} = st_A < et_A^{e_1} < et_A$), and need the whole action duration to achieve the effect $e_2 = In(p,t)$ or having the package inside the truck ($st_A^{e_2} = st_A < et_A^{e_2} = et_A$). For each effect e of action A that occurs at et_A^e, there is a duration $[st_A^e, et_A^e] : st_A \leq st_A^e \leq et_A^e \leq et_A$ in which we do not allow any process that leads to the event that causes $\neg e$ to occur. Note that unlike preconditions, an effect e can both be positive (add) or negative (delete).

An important issue in converting a p.c. plan into an o.c. plan is to ensure that actions that are not ordered with respect to each other are free of any interference. In general, two actions A and A' interfere if $\exists p : \neg p \in E(A) \wedge (p \in P(A') \bigcup E(A')$. Unlike classical planning, the temporal concurrency between A and A' depends on the exact temporal constraints (values of st^p, et^p in A and A'). Thus, the temporal relations between two interfering actions A and A' depend on the exact proposition p that relates them and it is possible to have more than one interference relation between two actions, each one enforcing different set of temporal constraints. Therefore, we use the notation $\bigotimes_{A,A'}^{p}$ to denote a specific interference relation between A, A' as it holds if $\neg p \in E(A)$ and $p \in P(A') \bigcup E(A')$. Each interference relation $\bigotimes_{A,A'}^{p}$ constrains the temporal orders between A and A' - specifically with the constraint $(et_A^{\neg p} < st_{A'}^{p}) \vee (et_{A'}^{p} < st_A^{\neg p})$. In our example in Figure 1, if any action A' that uses the proposition $At(p,l)$ of having the effect of causing $At(p,l)$, then A' is interference upon p with action $A = load(p,t,l)$.

2.2 Problem Definition

Position and Order constrained plans: *A position constrained plan (p.c.) is a plan where the execution time of each action is fixed to a specific time point. An order constrained (o.c.) plan is a plan where only the relative orderings between the actions are specified.*

There are two types of position constrained plans: *serial* and *parallel*. In a serial position constrained plan, no concurrency is allowed. In a parallel position constrained plan, actions are allowed to execute concurently. Examples of the serial p.c. plans are the ones returned by classical planners such as GRT [17], MIPS [5] and their temporal cousins. The parallel p.c. plans are the ones returned by Graphplan-based planners and their temporal cousins such as Sapa[3], TGP[18], TP4[7]. Examples of planners that output order constrained (o.c.) plans are Zeno[16], HSTS[14], IxTexT[11].

Figure 2 shows a valid p.c. parallel plan consisting of four actions A_1, A_2, A_3, A_4 with their starting time points fixed to T_1, T_2, T_3, T_4 and an o.c plan consisting of the same set of actions and achieving the same goals. For each action, the shaded regions show the durations in which each precondition or effect should hold during each action's execution time. The darker ones represent the effect and the lighters represent preconditions. For example, action A_1 has a precondition Q and effect R; action A_3 has no preconditions and two effects $\neg R$ and S.

It should be easy to see that o.c. plans provide more execution flexibility than p.c. plans. In particular, an o.c. plan can be "dispatched" for execution in any way consistent with the relative ordering among the actions. In other words, for each valid o.c. plan P_{oc}, there may be multiple valid p.c. plans that satisfy the orderings in P_{oc}, which can be seen as different ways of dispatching the o.c. plan.

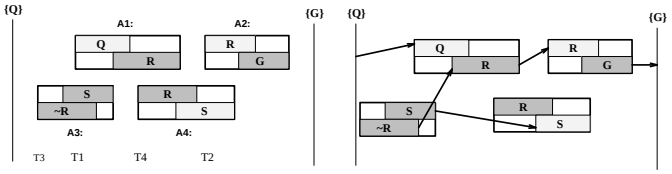


Figure 2: Examples of p.c. and o.c. plans

A measure of the temporal quality of a plan is its "makespan." The **makespan** of a plan is the minimum time needed to execute a plan. For a p.c. plan, the makespan is the duration from the earliest starting time until the latest ending time among all actions. In the case of serial p.c. plans, it is easy to see that the makespan will be greater than or equal to the sum of the durations of all the actions in the plan. For the o.c. plan, the makespan is the minimum makespan of any of the p.c. plans that are consistent with it. Given an o.c. plan P_{oc}, there is a polynomial time algorithm based on topological sort of the orderings in P_{oc}, which outputs a p.c. plan P_{pc} where all the actions are assigned earliest possible start time point according to the ordering in P_{oc}. The makespan of that p.c. plan P_{pc} is then used as the makespan of the original o.c. plan P_{oc}.

While generating a p.c. plan consistent with an o.c. plan is easy enough, in this paper, we are interested in the reverse problem–that of generating an o.c. plan given a p.c. plan. Thus, for a given p.c. plan P_{pc}, we want to find the optimal o.c. plan according to some criterion of temporal/execution flexibility such as smallest makespan or smallest number of orderings. In the next section, we shall provide a general CSP encoding for this "partialization problem." Finding optimal solution for this encoding turns out to be NP-hard even for classical planning (i.e., non-durative actions)[2]. Consequently, we shall develop value ordering strategies that are able to find a reasonable solution for the encoding in polynomial time.

3 Formulating a CSOP encoding for the partialization problem

Suppose that P_{pc}, containing a set of actions $\mathcal{A}$, and their starting times st_A^{pc}, is a valid p.c. plan for some temporal planning problem $\mathcal{P}$. Let P_{oc} be a partialization of P_{pc} for the problem $\mathcal{P}$. P_{oc} must then satisfy the following conditions:

1. P_{oc} contains the same actions $\mathcal{A}$ as P_{pc}.
2. P_{oc} is executable. This requires that the preconditions of all actions are satisfied, and no pair of interfering actions are allowed to execute concurrently.
3. P_{oc} is a valid plan for $\mathcal{P}$. This requires that P_{oc} satisfies all the top level goals (including deadline goals) of $\mathcal{P}$.
4. (Optional) The orderings on P_{oc} are such that P_{pc} is a legal dispatch (execution) of P_{oc}.
5. (Optional) The set of orderings in P_{oc} is minimal (i.e., no ordering is redundant)

Given that P_{oc} is an order constrained plan, ensuring goal and precondition satisfaction involves ensuring that (a) there is a causal support for the condition and that (b) the condition, once supported, is not violated by any possibly intervening action. The fourth constraint ensures that P_{oc} is in some sense an *order generalization* of P_{pc} [10]. This is not strictly needed if our

interest is only to improve temporal flexibility.[1] Finally, the fifth **c**onstraint above is optional in the sense that any objective function defined in terms of the orderings anyway ensures that P_{oc} contains no redundant orderings.

In the following, we will develop a CSP encoding for finding P_{oc} that captures the constraints above. This involves specifying the variables, their domains, and the inter-variable constraints.

Variables: The encoding will consist of both continuous (temporal) and discrete variables. The continuous variables represent the temporal aspects of actions in the plan, and the discrete variables represent the logical causal structure and orderings between the actions in the plan. Specifically, for the set of actions in the p.c. plan P_{pc} and two additional actions A_i and A_g representing the initial and final dummy actions,[2] the set of variables are as follows:

Temporal variables: For each action A, the encoding has one variable st_A to represent the time point at which we can start executing A. The domain for this variable is $Dom(st_A) = [0, +\infty)$.

Discrete variables: There are several different types of discrete variables representing the causal structure and qualitative orderings between actions:

- *Causal effect:* We need variables to specify the causal links relationship between actions. Specifically, for each fact $p \in P(A)$ and a set of actions $\{B_1, B_2.....B_n\}$ such that $p \in E(B_i)$, we set up one variable: S^p_A where $Dom(S^p_A) = \{B_1, B_2....B_n\}$.
- *Supportively related:* Two actions A and A' are supportively related if $\exists p \in (E(A) \bigcap P(A')$. For each such pair, we introduce one variable $\bigcirc^p_{AA'}$: $Dom(\bigcirc^p_{AA'}) = \{\prec, \perp\}$ ($A\ before_p\ A$', or *no-order* between A & A'). In our example in Figure 2, some ordering variables are: $\bigcirc^R_{A_1A_2}$, $\bigcirc^R_{A_3A_2}$, $\bigcirc^S_{A_3A_4}$.
- *Destructively related (interference):*Two actions A and A' are destructively related if they interfere with each other. For each such pair, using the same notation introduced at the end of Section 2.1, we introduce one variable $\bigotimes^p_{AA'}$: $Dom(\bigotimes^p_{AA'}) = \{\prec, \succ\}$ ($A\ before_p\ A'$, or $A\ after_p\ A'$). For the plan in Figure 2, the ordering variables are: $\bigotimes^R_{A_1A3}$ and $\bigotimes^R_{A_2A_3}$.[3]

Following are the necessary constraints to represent the relations between different variables:

1. If B supports the condition p for A, then there should be a supportive ordering between B and A w.r.t. p:
   $S^p_A = B \Rightarrow \bigcirc^p_{BA} = \prec$
2. Causal link protections: If B supports p to A, then every other action A' that has an effect $\neg p$ must be prevented from coming between B and A:
   $S^p_A = B \Rightarrow \forall A',\ \neg p \in E(A') : (\bigotimes^p_{A'B} = \prec) \vee (\bigotimes^p_{A'A} = \succ)$
3. Constraints to prevent non-minimal orderings (optional): If B does not support p to A, then there is no need for a supportive ordering between B and A w.r.t. p:
   $S^p_A \neq B \Rightarrow \bigcirc^p_{BA} = \perp$.
4. There are constraints between ordering variables and action start time variables (as per the discussion in Section 2.1). Specifically, we want to enforce that if $A \prec_p A'$ then $et^p_A < st^p_{A'}$. However, because we only maintain one continuous variable st_A in the encoding for each action, the constraints are as follow:
   $\bigcirc^p_{AA'} = \prec \Leftrightarrow\ st_A + (et^p_A - st_A) < st_{A'} + (st^p_{A'} - st_{A'})$.
   $\bigotimes^p_{AA'} = \prec \Leftrightarrow\ st_A + (et^{\neg p}_A - st_A) < st_{A'} + (et^p_{A'} - st_{A'})$.
   $\bigotimes^p_{AA'} = \succ \Leftrightarrow\ st_{A'} + (et^p_{A'} - st_{A'}) < st_A + (st^{\neg p}_A - st_A)$.
   Notice that all values $(st^p_A - st_A)$, $(et^p_A - st_A)$ are constants for all actions A and propositions p.
5. Deadlines and other temporal constraints: These model any deadline type constraints in terms of the temporal variables. For example, if all the goals need to be achieved before time t_g, then we need to add a constraint: $st_{A_g} \leq t_g$. Other temporal constraints, such as those that specify that certain actions should be executed before/after certain time points, can also be handled by adding similar temporal constraints to the encoding.
6. Constraints to make the orderings on P_{oc} consistent with P_{pc} (optional): Let T_A be the fixed starting time point of action A in the original p.c plan P_{pc}. To guarantee that P_{pc} is consistent with the set of orderings in the resulting o.c plan P_{oc}, we add a constraint to ensure that the value T_A is always present in the live domain of the temporal variable st_A.

Given the presence of both discrete and temporal variables in this encoding, the best way to handle it is to view it as a leveled CSP encoding where in the satisficing assignments to the discrete variables activate a set of temporal constraints between the temporal variables. These temporal constraints, along with the deadline and order consistency constraints are represented as a temporal constraint network [4]. Solving the network involves making the domains and inter-variable intervals consistent across all temporal constraints [20]. The consistent temporal network then represents the o.c. plan. Actions in the plan can be executed in any way consistent with the temporal network (thus providing execution flexibility).

Objective Function: Each satisficing assignment for the encoding above will correspond to a possible partialization of P_{pc}, i.e., an o.c. plan that contains all the actions of P_{pc}. However, some of these assignments (o.c. plans) may have better execution properties than the others. We can handle this by specifying an objective function to be optimized, and treating the encoding as a Constraint Satisfaction Optimization (CSOP) encoding. The only requirement on the objective function is that it be specifiable in terms of the variables of the encodings. Objective functions such as makespan minimization and order minimization readily satisfy this requirement.

4 Solving the partialization encoding

As mentioned above, the encoding, once setup, can be solved by a coupled framework (such as the one used in LPSAT [22]) where in a discrete CSP solver is used to handle the discrete variables, and a temporal CSP solver is used to handle the temporal variables. Every assignment to the discrete variables will activate a set of constraints between the temporal variables, which, in conjunction with the constraints of type 4 and 5 can

[1] In the terminology of [2], the presence of fourth constraint ensures that P_{oc} is a de-ordering of P_{pc}, while in its absence P_{oc} can either be a de-ordering or a re-ordering.

[2] A_i has no preconditions and has effects that add the facts in the initial state. A_g has no effect and has preconditions representing the goals.

[3] Sometimes, we will use the notation $A \prec_p A'$ to represent $\bigcirc^p_{AA'} = \prec$ and $\bigotimes^p_{AA'} = \prec$.

be solved by the temporal CSP solver. All the temporal constraints are "simple" [4] and can thus be handled in terms of a simple temporal network. Optimization can be done using a branch and bound scheme on top of this.

Notwithstanding the foregoing discussion, solving the CSOP encoding will be NP-hard problem (this follows from [2]). Consequently, we focus on developing variable and value ordering strategies for the encoding, which can ensure that the very first satisficing solution found will have a high quality in terms of the objective function. Clearly, these strategies will depend on the specific objective function. In the following, we will develop strategies that are suited to objective functions based on minimizing the makespan.

4.1 Greedy value ordering strategies for solving the encoding

In this section, we discuss a value ordering strategy that finds an assignment to the CSOP encoding such that the corresponding o.c plan P_{oc} is biased to have a reasonably good makespan. The strategy depends heavily on the positions of all the actions in the original p.c. plan. Thus, it works based on the fact that the alignment of actions in the original p.c. plan guarantees that causality and preserving constraints are satisfied. Specifically, all CSP variables are assigned values as follows:

Supporting Variables: For each variable S^p_A representing the action that is used to support precondition p of action A, we choose action A' such that:

1. $p \in E(A')$ and $et^p_{A'} < st^p_A$ in the p.c. plan P_{pc}.
2. There is no action B s.t: $\neg p \in E(B)$ and $et^p_{A'} < et^{\neg p}_B < st^p_A$ in P_{pc}.
3. There is no other action C that also satisfies two conditions above and $et^p_C < et^p_{A'}$.

Interference ordering variables: For each variable $\bigotimes^p_{AA'}$, we assign value:

1. $\bigotimes^p_{AA'} = \prec$ if $et^p_A < st^p_{A'}$ in P_{pc}.
2. $\bigotimes^p_{AA'} = \succ$ if $et^p_{A'} < st^p_A$ in P_{pc}.

Other ordering variables: For all the ordering variables $\bigcirc^p_{AA'}$ that are not enforced to have value $\prec$ by the assignments to supporting variables $S^p_{AA'}$, we assign values $\bigcirc^p_{AA'} = \perp$.

This strategy is backtrack-free due to the fact that the original p.c. plan is correct. Thus, all preconditions of all actions are satisfied and for all *supporting variables* we can always find an action A' that satisfies the three constraints listed above to support a precondition p of action A. More over, one of the temporal constraints that lead to the assignment of *interference ordering variables* $\bigotimes^p_{AA'}$ will always be satisfied because the p.c. plan is consistent and no pair of actions that have interference relations overlap each other. Finally, this strategy ensures that the orderings on P_{oc} are consistent with the original P_{pc}. Therefore, the search is backtrack-free and no constraint is violated because there is one legal dispatch of the final o.c. plan P_{oc}, which is the starting p.c. plan P_{pc}. Moreover, because the p.c plan P_{pc} is one among multiple p.c plans that are consistent with the o.c plan P_{oc}, the makespan of P_{oc} is guaranteed to be equal or better than P_{pc}.

Complexity: It is also easy to see that the complexity of the greedy algorithm is $O(S * A + I + O)$ where S is the number of supporting relations, A is the number of actions in the plan, I is the number of interference relations and O is the number of ordering variables. In turn $S \leq A * P$, $I \leq A^2$ and $O \leq P * A^2$ where P is the number of preconditions of an action. Thus, the complexity of the algorithm is $O(P * A^2)$.

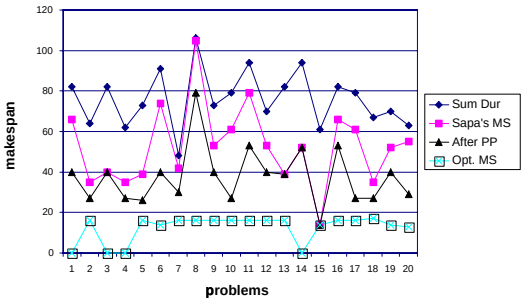


Figure 3: Temporal Logistics with *drive inter-city* actions.

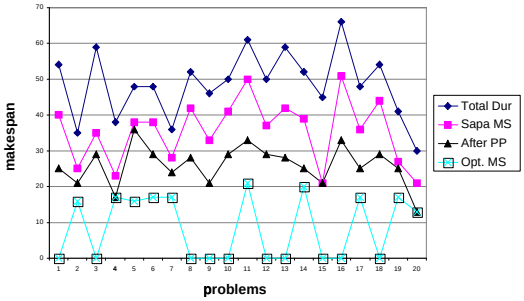


Figure 4: Temporal Logistics without *drive inter-city* actions.

5 Empirical results for Temporal Planning

We have implemented the variable and value ordering discussed in the last section (Section 4.1) and tested it with the Sapa planner. Sapa is a forward state space planner that outputs parallel p.c. plans. The results reported in [3] show that while Sapa is quite efficient, it often generates plans with inferior makespan values. Our aim is to see how much of an improvement our partialization algorithm provides for the plans produced by Sapa. The test suite is the 80 random temporal logistics provided with TP4 planner. In this planning domain trucks move packages between locations inside a city and airplanes move them between cities. Figure 3 and 4 show the comparison results for only the 20 largest problems, in terms of number of cities and packages, among 80 of that suite. In Figure 3, trucks are allowed to move packages between different locations in different cities, while in the Figure 4, trucks are not allowed to do so.

The graphs show the comparison between four different makespan values: (1) the optimal makespan (as returned by TGP [18]); (2) the makespan of the plan returned by Sapa; (3) the makespan of the o.c. resulting from the greedy algorithm for partialization discussed in the last section; and (4) the total duration of all actions, which would be the makespan value returned by several serial temporal planners such as GRT [17], or MIPS [5] if they produce the same solution as Sapa. Notice that the makespan value of *zero* for the optimal makespan indicates that the problem is not solvable by TGP.

For the first test which allows driving between cities action, compared to the optimal makespan, on the average, the makespan of the serial p.c. plans (i.e, cumulative action duration) is about 4.34 times larger, the makespan of the plans output by Sapa is about 3.23 times larger and the Sapa plans after post processing are about 2.61 times longer (over the set of 75 solvable problems; TGP failed to solve the other 5). For the second test, without the driving inter-city actions. The comparison results with regard to optimal solutions are: 2.39 times longer for serial plans, 1.75 times longer for the plans output by

Sapa, and 1.31 times longer after partialization. These results are averaged over the set of 69 out of the 80 problems that were solvable by TGP.[4]

Thus, the partialization algorithm improves the makespan values of the plans output by Sapa by an average of 20% in the first set and 25% in the second set. Notice also that the same technique can be used by GRT [17] or MIPS [5] and in this case, the improvement would be 40% and 45% respectively for the two problem sets.

The partialization and topological sort times are extremely short. Specifically, they are less than 0.1 seconds for all problems with the number of actions ranging from 16 to 37. Thus, using our partialization algorithm as a post-processing stage essentially preserves the significant efficiency advantages of planners such as Sapa, GRT and MIPS, that search in the space of p.c. plans, while improving the temporal flexibility of the plans generated by those planners.

Finally, it should be noted that partialization improves not only makespan but also other temporal flexibility measures. For example, the "scheduling flexibility" of a plan defined in [15], which measures the number of actions that do not have any ordering relations among them, is significantly higher for the partialized plans, compared even to the parallel p.c. plans generated by TGP. In fact, our partialization routine can be applied to the plans produced by TGP to improve their scheduling flexibility.

6 Temporal planning with continuous changes

An advantage of setting up an encoding for the partialization problem is that the encoding can be generalized to handle other types of constraints on the plan. In fact, we have extended the encoding to handle temporal problems in the presence of metric/resource consumption. In this section, we briefly summarize the extension, and present some preliminary empirical results.

Extending the Representation: Let V_t^r be a value of a metric resource r at a time point t, we assume that an action A that uses r has following constraints:

1. **Resource duration:** Like propositions, A uses r for a period between two time points st_A^r and et_A^r s.t: $st_A \leq st_A^r \leq et_A^r \leq et_A$.
2. **Resource preconditions:** At time point st_A^r, there may be some constraints on the value of r (such as the action should have enough resources to be executetable). We assume that the constraints are in the form of comparison such as: $V_t^r \diamond K$, where K is a constant, and $\diamond \in \{<, >, \leq, \geq, =\}$. To simplify the discussion, for the rest of this section, we will only discuss the case for $\diamond = >$. The other cases are very similar.
3. **Resource effects:** Actions may increase/decrease an amount U_A^r of r during the period from st_A^r to et_A^r.

Extending the encoding: To be able to output the o.c plan that is resource-consistent, which means that all resource related constraints $V_{st_A^r}^r > K$ are satisfied by the set of orderings in the o.c plan, we need to introduce a new set of variables and constraints to our general CSOP encoding discussed in previous sections. The details are as follows:

Variables: For each pair of actions A and A' that use the same resource r, we introduce one variable $\bigodot_{AA'}^r$ to represent the resource-enforced ordering between them (similar to the way $\bigcirc_{AA'}^r$ represents the precondition enforced orderings; see Section 3). If A and A' can not use the same resource at the same time, then $Dom(\bigodot_{AA'}^r) = \{\prec, \succ\}$, otherwise $Dom(\bigodot_{AA'}^r) = \{\prec, \succ, \perp\}$.

Constraints: There are two additional types of constraints:

1. Constraints representing the relations between the resource-related orderings and action start time variables:
   $\bigodot_{AA'}^r = \prec \Leftrightarrow st_A + (et_A^r - st_A) < st_{A'} + (st_{A'}^r - st_{A'})$
   $\bigodot_{AA'}^r = \succ \Leftrightarrow st_{A'} + (et_{A'}^r - st_{A'}) < st_A + (st_A^r - st_A)$
   Notice that the values $(st_A^r - st_A)$ and $(et_A^r - st_A)$ are constants for a given action A that uses resource r.
2. Constraints to guarantee the resource consistency for all actions: Specifically, for a given action A that uses resource r and has a resource constraint $V_{st_A^r}^r > K$, let $\{A_1, A_2, ...A_n\}$ be a set of actions that also use r and $Init_r$ be the value of r at the initial state. We set up one constraint that involves all variables $\bigodot_{A_i A}^r$ as follows:

$$Init_r + \sum_{A_i \prec_r A} U_{A_i}^r + \sum_{A_i \perp_r A, U_{A_i}^r < 0} U_{A_i}^r > K$$

   (where $A_i \prec_r A$ is shorthand for $\bigodot_{A_i A}^r = \prec$). The constraint above ensures that regardless of how the actions A_i that have no ordering relation with A ($\bigodot_{A_i A}^r = \perp$) are aligned temporally with A, the orderings between A and other actions guarantee that A has enough resource ($V_{st_A^r}^r > K$) to execute.

Greedy value ordering: Beside the default variable and value ordering used by any solver that we choose to solve our CSOP encoding, we can also use the value ordering similar to the strategy used to assign values to the causal and ordering variables in Section 4.1. Specifically, the variables $\bigodot_{AA'}^r$ can be assigned values based on their fixed starting times in the original p.c plan P_{pc} as follows:

- $\bigodot_{AA'}^r = \prec$ if $et_A^r < st_{A'}^r$ in P_{pc}.
- $\bigodot_{AA'}^r = \succ$ if $et_{A'}^r < st_A^r$ in P_{pc}.
- $\bigodot_{AA'}^r = \perp$ otherwise.

Due to the fact that the original p.c plan P_{pc} is correct, it is easy to see that the value ordering discussed above will lead to a backtrack-free search over the set of resource-related ordering variables (do not cause any temporal or resource inconsistency).

Preliminary Empirical Evaluation: We implemented this value ordering strategy and tested it with a set of logistics problems in which different trucks and airplanes consume fuel at different rates while moving packages. They also need to refuel when they do not have enough fuel in their tank to finish

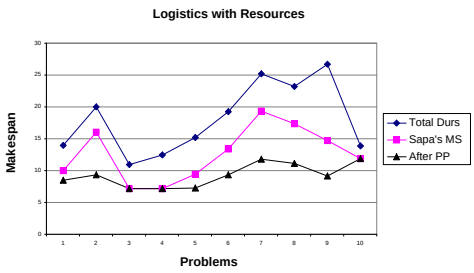


Figure 5: Results for temporal problems with resources

[4] While TGP could not solve several problems in this test suite, Sapa is able to solve all 80 of them.

the trip. We tested with 10 problems and the results are shown in Figure 5. Currently, there is no planner that can handle resources and output optimal makespan. Therefore, we compare only the total duration, the makespan of parallel plans output by Sapa, and the makespan values after partialization. The results show that on average, the backtrack-free value ordering strategy improves the makespan value by 22%.

7 Related Work

The complementary tradeoffs provided by the p.c. and o.c. plans have been recognized in classical planning. One of the earliest efforts to attempt to improve the temporal flexibility of plans was the work by Fade and Regnier [6] who discussed an approach for removing redundant orderings from the plans generated by STRIPS system. Later work by Mooney [13] and Kambhampati and Kedar [10]characterized this partialization process as one of explanation-based order generalization. Backstrom [2] categorized approaches for partialization into "de-ordering" approaches and "re-ordering" approaches. The order generalization algorithms fall under the de-ordering category. He was also the first to point out the NP-hardness of maximal partialization, and to characterize the previous algorithms as greedy approaches.

The work presented in this paper can be seen as a principled generalization of the partialization approaches to metric temporal planning. Our novel contributions include: (1) providing a CSP encoding for the partialization problem and (2) characterizing the greedy algorithms for partialization as specific value ordering strategies on this encoding. In terms of the former, our partialization encoding is general in that it encompasses both de-ordering and re-ordering partializations–based on whether or not we include the optional constraints to make the orderings on P_{oc} consistent with P_{pc}. In terms of the latter, the work in [21] and [10] can be seen as providing a greedy value ordering strategy over the partialization encoding for classical plans. However, unlike the strategies we presented in Sections 4.1 and 6, their value ordering strategies are not sensitive to any specific optimization metric.

It is interesting to note that our encoding for partialization is closely related to the so-called "causal encodings" [8]. Unlike casual encodings, which need to consider supporting a precondition or goal with every possible action in the action library, the partialization encodings only need to consider the actions that are present in P_{pc}. In this sense, they are similar to the encodings for replanning and plan reuse described in [12]. Also, unlike causal encodings, the encodings for partialization demand optimizing rather than satisficing solutions. Finally, in contrast to our encodings for partialization which specifically handle metric temporal plans, causal encodings in [8] are limited to classical domains.

8 Conclusion

In this paper we addressed the problem of post-processing position constrained metric temporal plans to improve their execution flexibility. We developed a general CSP encoding for partializing position-constrained temporal plans, that can be optimized under an objective function dealing with a variety of temporal flexibility criteria, such as makespan. We then presented greedy value ordering strategies that are designed to efficiently generate solutions with good makespan values for these encodings. We evaluated the effectiveness of our greedy partialization approach in the context of a recent metric temporal planner that produces p.c. plans. Our results demonstrate that the partialization approach is able to provide between 25-40% improvement in the makespan, with extremely little overhead. We also briefly discussed an extension of our partialization approach for temporal plans with resource constraints, and demonstrated empirically that partialization can lead to up to 22% improvement of the makespan. Currently, we are focusing on developing greedy value ordering strategies that are sensitive to other types of temporal flexibility measures besides makespan.

References

[1] Bacchus, F. and Ady, M. 2001. Planning with Resources and Concurrency: A Forward Chaining Approach. *Proc IJCAI-2001*.

[2] Backstrom, C. 1998. Computational Aspects of Reordering Plans *Journal of Artificial Intelligence Research 9, 99-137.*

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

[4] Dechter, R., Meiri, I., and Pearl, J. 1990. Temporal Constraint Network. *Artificial Intelligence Journal 49.*

[5] Edelkamp, S. 2001. First Solutions to PDDL+ Planning Problems *In PlanSIG Workshop.*

[6] Fade, B. and Regnier, P. 1990 Temporal Optimization of Linear Plans of Action: A Strategy Based on a Complete Method for the Determination of Parallelism *Technical Report*

[7] Haslum, P. and Geffner, H. 2001. Heuristic Planning with Time and Resources. *Proc ECP-2001*

[8] Kautz, H., McAllester, D. and Selman B. Encoding Plans in Propositional Logic *In Proc. KR-96.*

[9] Ihrig, L., Kambhampati, S. Design and Implementation of a Replay Framework based on a Partial order Planner. *Proc. AAAI-96.*

[10] Kambhampati, S. & Kedar, S. 1994. An unified framework for explanation-based generalization of partially ordered and partially instantiated plans. *Artificial Intelligence Journal 67, 29-70.*

[11] Laborie, P. and Ghallab, M. Planning with sharable resource constraints. *Proc IJCAI-95*.

[12] Mali, A. Plan Merging and Plan Reuse as Satisfiability *Proc ECP-99*.

[13] Mooney, R. J. Generalizing the Order of Operators in Macro-Operators *Proc. ICML-1988*

[14] Muscettola, N. 1994. Integrating planning and scheduling. *Intelligent Scheduling.*

[15] Nguyen, X., and Kambhampati, S. 2001. Reviving Partial Order Planning. *Proc IJCAI-01*.

[16] Penberthy, S. and Weld, D. 1994. Planning with Continous Changes. *Proc. AAAI-94*

[17] Refanidis, I. and Vlahavas, I. 2001. Multiobjective Heuristic State-Space Planning *Technical Report.*

[18] Smith, D. & Weld, D. Temporal Planning with Mutual Exclusion Reasoning. *Proc IJCAI-99*

[19] Srivastava, B., Kambhampati, S., and Do, M. 2001. Planning the Project Management Way: Efficient Planning by Effective Integration of Causal and Resource Reasoning in RealPlan. *Artificial Intelligence Journal 131*.

[20] Tsamardinos, I., Muscettola, N. and Morris, P. Fast Transformation of Temporal Plans for Efficient Execution. *Proc. AAAI-98.*

[21] Veloso, M., Perez, M, & Carbonell, J. 1990. Nonlinear planning with parallel resource allocation. *Workshop on Innovative Approaches to Planning, Scheduling and Control*.

[22] Wolfman, S. and Weld, D. 1999. The LPSAT system and its Application to Resource Planning. *In Proc. IJCAI-99.*

Mixed Propositional and Numerical Planning in the Model Checking Integrated Planning System

Stefan Edelkamp

Institut für Informatik,
Albert-Ludwigs-Universität,
Georges-Köhler-Allee, D-79110 Freiburg
eMail: edelkamp@informatik.uni-freiburg.de

Abstract

The paper considers the extensions to the domain-independent Model Checking Integrated Planning System (MIPS) to pre-processes and solve mixed propositional and numerical planning problems in PDDL+ syntax for the 3rd international planning competition.

The static analyzer grounds all predicates and functors, distinguishes constant from fluent atoms and numerical constants from variables. It further approximates the bounding intervals for the resource variables, and encodes their possible finite domain. Pre-compilation also establishes symmetries within the object set and dependencies among the set of operators.

Based on the inferred information, the directed search exploration algorithm applies critical path scheduling to parallelize sequential plans and to refine a relaxed plan graph heuristic, while different pruning approaches effectively reduce the branching factor.

Introduction

For the 2002's international planning competition new levels of the planning domain description language PDDL+ (Fox & Long 2001) have been designed to specify problems that include durative actions and resources. While Level 1 considers pure propositional planning, Level 2 also includes numerical resources, and Level 3 additionally includes actions with durations.

At the moment four Level 2-3 competition problems are published. *Desert-Rat* is a domain with an infinite branching factor which was manually discretized in (Edelkamp 2001b): n supply tanks are available as fuel resources for trucks to finally reach the goal distance d from the base. *Zeno-Travel* requires to fly passengers with aircrafts to their respective target airports. Boarding and debarking consumes a constant amount of time. Each plane has a determined capacity for fuel, while flying aircrafts changes the fuel level according to the distances between the cities and with respect to two different speeds. Fuel can be restored by refueling the aircraft. *Jugs-and-Water* problems model two jugs, namely Jug-1 and Jug-2. Initially, both are empty and have a predefined capacity. It is allowed to completely fill either Jug-1 or Jug-2 from a tap, to fill Jug-1 from Jug-2, or Jug-2 from Jug-1, and to empty either jug on the ground. The goal is to achieve content 1 in Jug-1. *Taxi* is a variant of a transportation domain with a representation of locations using numeric coordinates and distance as a calculated function of those values. In the example problem there are seven people located somewhere on the grid-structured map and four taxis serving them.

This paper presents extensions to the Model Checking Integrated Planning System MIPS (Edelkamp & Helmert 2001) to cope with this new expressiveness. It summarizes and extends precursory work as follows. In (Edelkamp & Helmert 1999) we showed how a static analyzer can cluster atoms into mutually exclusive fact groups to minimize the state description length, a technique especially important for symbolic planning strategies (Edelkamp & Reffel 1999). In (Edelkamp 2001b) first results on PDDL+ planning problems were presented. The preliminary treatment exemplifies the parsing process in *Zeno-Travel* and *Desert-Rats*. Moreover, propositional heuristics and manual branching cuts are applied to find sequential plans. In (Edelkamp 2002a) we proposed critical path scheduling for concurrent plans, an efficient method for detecting and using symmetry, and refinements to the relaxed planning heuristic. Explicit (Edelkamp 2001c) and symbolic pattern databases (Edelkamp 2002b) are off-line generated estimators referring to completely explored problem abstractions. In this unifying treatment we newly contribute two approximate exploration techniques to bound and to fix numerical domains, an any-time search wrapper to produce optimal plans and a numerical extension to the FF plan graph heuristic, yielding first plans in the *Taxi* domain.

The organization of the paper is as follows. To introduce the underlying problem structure we provide a formal characterization of *mixed propositional and numerical planning problems*. We then present a scheduling algorithm compacting a sequential plan into a concurrent plan with minimal critical path length. Next we introduce our static analyzer that infers problem descriptions according to the given framework and that

deduces all different kinds of information needed for exploration: dependencies, symmetry, bounds and encodings of domains for resource variables. In the experimental section we provide plans to challenging planning problems to all four benchmark domains. Finally, we reflect the current state of the MIPS planning system and draw conclusions.

Problem Structure

A PDDL+ domain specification contains *predicates* and *functions*, the basis to define parameterized actions with pre- and postconditions. Static analysis usually ground *predicates* and *functions*, by instantiating all parameters of the operators. Grounded predicates are called *atoms*, and grounded functions are called resource *variables*.

State Space

Let A be the set of propositional atoms and V be the set of variables in domain D, indexed by an isomorphism $\phi : B \to \{1, \ldots, |V|\}$ for another set of propositional atoms B, $A \cap B = \emptyset$. The domain D is the set of real numbers, but, as we will see later on, for each variable D can be individually refined to a smaller set. Probably the most important variable is *total-time*, which increases monotonically with each applied operator. The difference in *total-time* before and after an applied action is its *duration*. If the duration is zero an action is called *instantaneous*.

A *Mixed Numerical and Propositional Planning Problem* is a state space problem $\mathcal{P} =< \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$, with $\mathcal{S} \subseteq 2^A \times D^{|B|}$ being the set of states, 2^A being the power set notation of A, $\mathcal{I} \in \mathcal{S}$, $\mathcal{G} \subseteq \mathcal{S}$, and $\mathcal{O}$ being the set of operators that transform states into states. An operator $o = (\alpha, \beta, \gamma, \delta) \in \mathcal{O}$ has propositional preconditions α, propositional effects β, numerical preconditions γ, and numerical effects δ.

It is sufficient to assume that the propositional part satisfies the STRIPS setting, where $\alpha \subseteq A$ is the precondition and $\beta = (\beta_a, \beta_d)$ is the effect with add list $\beta_a \subseteq A$, and the delete list $\beta_d \subseteq A$. For the numerical part γ is a set of constraints of numerical variables and constants in the assumed calculus. Similarly, δ is a set of rules transforming constraints into others.

Numerical Preconditions and Effects

For γ (and δ) we assume variables to be conditioned on (or assigned to) the term evaluation of arithmetic formula trees in set T as follows.

A *numerical constraint / numerical precondition* $c \in \gamma$ is a triple $c = (h_c, \otimes, t_c)$ where $h_c \in B$, $\otimes \in \{\leq, <, =, >, \geq\}$, and $t_c \in T_c$. A *numerical modifier / numerical effect* $m \in \delta$, is a triple $m = (h_m, \oplus, t_m)$ where $h_m \in B$, $\oplus \in \{\leftarrow, \uparrow, \downarrow\}$, and $t_m \in T$.

There is no fundamental difference to more general preconditions and effects. Our current implementation generates one generic precondition tree, thereby including boolean and logical operators and arithmetic subtrees.

Semantics

An operator $o = (\alpha, \beta, \gamma, \delta) \in \mathcal{O}$ applied to a state $S = (S_p, S_n) \in 2^A \times D^{|B|}$, $S_p \in 2^A$ and $S_n \in D^{|B|}$, yields a successor state $S' = (S'_p, S'_n) \in 2^A \times D^{|B|}$ as follows. If $\alpha \subseteq S_p$ and S_n *satisfies* all $c \in \gamma$ then $S'_p = S_p \cup \beta_a \setminus \beta_d$ and for all $m \in \delta$ the vector S_n is *updated*.

A vector $S_n = (S_1, \ldots, S_{|B|})$ of numerical variables *satisfies* a numerical constraint $c = (h_c, \otimes, t_c) \in \gamma$ if $s_{\phi(h_c)} \otimes \text{eval}(S_n, t_c)$ is true, where $\text{eval}(S_n, t_c) \in D$ is obtained by substituting all $b \in B$ in t_c by $S_{\phi(h_c)}$ followed by a simplification of t_c. Similarly, the vector $S_n = (S_1, \ldots, S_{|B|})$ is *updated* to vector $S'_n = (S'_1, \ldots, S'_{|B|})$ by modifier $m = (h_m, \otimes, t_m) \in \delta$, if

- $S'_{\phi(h_m)} = \text{eval}(S_n, t_m)$ for $\oplus = \leftarrow$,
- $S'_{\phi(h_m)} = S_{\phi(h_m)} + \text{eval}(S_n, t_m)$ for $\oplus = \uparrow$, and
- $S'_{\phi(h_m)} = S_{\phi(h_m)} - \text{eval}(S_n, t_m)$ for $\oplus = \downarrow$.

Sequential and Concurrent Plans

A sequential plan $\pi_s = (o_1, \ldots, o_k)$ is an ordered sequence of operators $o_i \in \mathcal{O}$, $i \in \{1, \ldots, k\}$, that transform the initial state $\mathcal{I}$ into one of the goal states $G \in \mathcal{G}$, i.e. there exists a sequence of states $S_i \in \mathcal{S}$, $i \in \{0, \ldots, k\}$, with $S_0 = \mathcal{I}$, $S_k = G$ and S_i is the outcome of applying o_i to S_{i-1}, $i \in \{1, \ldots, k\}$.

Schedules order the operators along the time line, i.e. the value of *total-time* before applying o_i is required to start at t_i. In optimal schedules each event either starts or ends at the start or end time of another event for a possibly exponential but finite number of valid schedules. Therefore a *concurrent plan* $\pi_c = ((o_1, t_1), \ldots, (o_k, t_k))$ of π_s is an optimal *schedule* of a sequential plan. If $t_i = t_j$ for $i < j$ then o_i is executed before o_j. The definition is sound, since next section will show that optimal schedules with respect to sequential exist and can be computed efficiently.

Scheduling

An operator o is said to *precede* another operator o' in $\mathcal{O}$, $o \leq_o o'$ for short, if and only if o and o' are dependent and be the index of operator o not larger than the index of o'. Obviously, $\leq_o$ defines a partial order relation. Therefore, given a sequential plan $o_1, \ldots, o_k$ to the PDDL+ planning problem produces an acyclic set of precedence constraints $o_i \leq_o o_j$, $1 \leq i < j \leq k$, on the set of operators. It is also important to observe that the constraints are already topologically sorted according to $\leq_o$ by taking the ordering $\{1, \ldots, k\}$.

Critical Path Analysis

The *Project Evaluation and Review Technique* (PERT) is a critical path analysis algorithm usually applied to

Procedure *Critical-Path*
  for all $i \in \{1, \ldots, k\}$
    $e(o_i) = d(o_i)$
    for all $j \in \{1, \ldots, i-1\}$
      if $(o_j \leq_o o_i)$
        if $e(o_i) < e(o_j) + d(o_i)$
          $e(o_i) \leftarrow e(o_j) + d(o_i)$
  return $e(o_k)$

Table 1: Algorithm to Compute Critical Path Length.

project management problems. The critical path is established, when the total time for activities on this path is greater than any other path of operators. A delay in any tasks on the critical path leads to a delay in the project. The heart of PERT is a network of tasks needed to complete a project, showing the order in which the tasks need to be completed and their dependencies between them. Fortunately, as shown in Table 1, PERT scheduling reduces to a derivate of Dijkstra's single shortest path algorithm within acyclic graphs (Cormen, Leiserson, & Rivest 1990).

Usually, duration $d(o_i)$ is the difference of time stamps in the sequential plan. Since PDDL+ provides different objective function c, e.g. the sum of *total-time* and *total-fuel-used* in *Zeno-Travel*, $d(o_i)$ can be fixed as $c(s_i) - c(s_{i-1})$ for $i \in \{1, \ldots, k\}$.

The time and space complexities of the algorithm *Critical-Path* are $O(k^2)$, where k is the length of the sequential plan. Using an adjacency list representation these efforts can be reduced to time and space proportional to the number of vertices and edges in the dependence graph.

Static Analysis

Based on the number of counted objects, a unique index for each grounded predicate and function is devised. A relaxed, so-called *fact-space* exploration on the propositional part of the problem determines a superset of all reachable atoms and allows to distinguish constant from fluent atoms, since only the latter ones are reached by exploration (Edelkamp & Helmert 1999). Fact-space exploration also determines all grounded operators. Once all preconditions are satisfied and grounded, the symbolic effect-lists are instantiated.

Atoms are clustered into groups, so that each state in the planning space can be expressed as a conjunct of atoms selected from each group. In the *Zeno-Travel* domain, the unique position of the passengers and the unique position of each plane determine the partition.

According to the formal characterization of numerical modifiers and synchronous to fact space exploration of the propositional part of the problem, all heads of numerical formulae in the effect lists are grounded. This allows to early distinguish constant numerical quantities from variable ones.

In *ZenoTravel* only the current fuel level for each plane, the total amount of consumed fuel and the simulation time are variable. All other numerical predicates are constants to be are substituted in the formula-bodies. This simplifies the grounded operators, and the formula trees of most numerical conditions and assignments reduce to constants. However, some operators like *refueling* in *Zeno-Travel* depend on fluent state variables that have to be instantiated on the fly.

Dependent Operators

Two grounded operators $o = (\alpha, \beta, \gamma, \delta)$ and $o' = (\alpha', \beta', \gamma', \delta')$ in $\mathcal{O}$ are *dependent*, if one of the following three conditions holds:

1. $\alpha \cap (\beta'_a \cup \beta'_b) \neq \emptyset$, $(\beta_a \cup \beta_b) \cap \alpha' \neq \emptyset$,
2. For one $c = (h_c, \otimes, t_c) \in \gamma$ and one $m' = (h'_m, \oplus, t'_m) \in \delta'$ $h_c \in \mathit{LeafVariables}(t'_m)$ or $h'_m \in \mathit{LeafVariables}(t_c)$,
3. For one $c' = (h'_c, \otimes, t'_c) \in \gamma'$ and one $m = (h_m, \oplus, t_m) \in \delta$: $h_m \in \mathit{LeafVariables}(t'_c)$ or $h'_c \in \mathit{LeafVariables}(t_m)$,

where $\mathit{LeafVariables}(t)$ is defined as the union of all variable-leaves in the formula tree $t \in T$.

The coarse approximation of the exact dependence relation can be refined according to the PDDL+ guidelines for mutex operators, but for our purposes to define a preference relationship for improving sequential plans this approach is sufficient. In our implementation the dependence relation is computed beforehand and tabularized for a constant time access. It also allows to detect transpositions of two operators o_1 and o_2 for prune exploration in one case, which is called a transposition cut.

To detect domains for which any schedule leads to no improvement. a planning domain is said to be inherently sequential if all operators in any sequential plan are dependent or instantaneous. The static Analyzer checks by testing each operator pair. While *DesertRats* and *Jugs-and-Water* are inherently sequential, *Zeno-Travel* and *Taxi* are not.

Grounding Variables

Even plan existence for numerical planning is undecidable, since PDDL+ planning reduces to the halting problem for abacus programs (Helmert 2002). If the state space is finite then PDDL+ problems are trivially decidable, since planning reduces to graph search. Since $|2^A|$ is already finite, the crucial part is to show that $D^{|B|}$ is finite, which is true if both D and B are finite. Since $|B|$ is finite, the cardinalities of variable domains are good indicators for the hardness of the problems.

Static analysis can approximate variable domains by finding bounding intervals for the variables and by refining the actual contents of a finite domain intervals by another exploration scheme.

Procedure *Bounding*
$(\min, \max) \leftarrow (I_n, I_n)$
while $(\min, \max) \neq (\min', \max')$
for all $o \in \mathcal{O}$
if $o.test(\min, \max)$
$(\min'', \max'') \leftarrow o.restrict(\min, \max)$
$(\min', \max') \leftarrow o.update(\min'', \max'')$
$(\min', \max') \leftarrow$
$(\min, \max) \cup (\min', \max')$
$(\min, \max) \leftarrow (\min', \max')$
return $(\min, \max)$

Table 2: Algorithm to Compute Bounding Intervals for Variables.

We will use the domain information only for heuristic evaluation, so that lack of accurancy for this phase will only decrease the algorithms' performance, not its overall applicability.

Bounding Variables For finding bounding intervals for the variables we apply the strategy of Table 2.

At first, the minimal and maximal resource vectors $(\min, \max)$ are initialized to the value vector of the initial state. In the *while*-loop this vector is enlarged until a fixpoint is reached. In each iteration every operator is tested for applicability by checking all numerical preconditions with the current vector of intervals $(\min, \max)$. The corresponding variable bounds were propergated in the arithmetic precondition trees. If the preconditions are satisfied, the intervals in $(\min, \max)$ are restricted with constraint propergation within the set of preconditions expressions (Meriott & Stuckey 1998). Updating now takes the restricted vector of interval $(\min'', \max'')$ and applies the effect lists to it. Afterwards the resulting intervals $(\min', \max')$ are merged with the original vector pair $(\min, \max)$.

Unfortunately, the algorithm might not terminate for unbounded variable like `total-time`. The natural option we take is to allow each operator to apply only once. Since the above approximation scheme is used for a relaxed scheduling heurstic and not for the overall planning process this is not a severe restriction, since the relaxed plan graph construction also allows each operator only to be invoked at most once.

The *Taxi-Domain* has nine variables: `total-time`, `street taxi1`, `street taxi2`, `street taxi3`, `street taxi4`, `avenue taxi1`, `avenue taxi2`, `avenue taxi3`, and `avenue taxi4`. Since `total-time` is always assumed to be unbounded the process yields the interval $[24, 94]$ for the `street-*` variables and $[1, 97]$ for all `avenue-*`-variables.

Procedure *Instantiate*
for all $(r, v) \in I_n$
$Q.enqueue(r, v)$
$D_r \leftarrow \{v\}$
while $Q \neq \emptyset$
$(r, v) \leftarrow Q.dequeue()$
for all $\gamma \ \wedge \ m = (h_m, \oplus, t_m) \in \gamma$
if $r \in LeafVariables(t_m)$
for all $S \leftarrow generate(t_m, r, \otimes)$
if $S \in (\min, \max)$
$v \leftarrow eval(t_m, S)$
$Q.enqueue(h_m, v)$
$D_{h_m} \leftarrow D_{h_m} \ \cup \ \{v\}$

Table 3: Algorithm to Instantiate Variable.

Instantiating Variables as depicted in Table 3 neglects preconditions and computes a fixpoint for the variable domains by considering the numerical effects γ in the operator set only. Similar to fact space exploration we utilize a queue Q, containing possible variable-value pairs. First of all, the initial pairs are inserted into Q. As long as there is one pending element (r, v) in Q it is extracted and all effects t_m containing r as a leaf variables are selected. Now all combinations of domain values for the other variables are generated and evaluated. All new pairs that respect the established bounds $(\min, \max)$ are added to the queue. In D_r we maintain the current set of instantiation of variable r. Depending on the number of variable occuring in the evaluation tree and their corresponding domain sizes, *generate* is of exponential nature. However, in practice the number of occuring variables in the simplified expression tree are bounded by one or two leaf variables yielding a quick exploration scheme.

In the *Taxi*-Domain the instantiations for the `streets`-variables are: 50, 48, 54, 80, 34, 68, 73, 40, 94, 75, 78, 43, 66, 27, and 24, while the the instantiations for the `avenue`-variables are: 50, 54, 94, 46, 72, 1, 97, 36, 85, 49, 4, 47, 80, 39, and 3.

Symmetries

The core observation for symmetry reduction in a planning problem is that the symbolic definition of actions in the domain description language cannot distinguish between different objects in problem instances. This is due to the fact that predicates, functions and actions are parameterized with objects that only have to respect the specified type. Therefore, the main restriction to symmetry within a type class are the current state and the goal state.

Symmetry dedection exploits information on single-valued invariances that are used by our planner to build mutually exclusive fact groups. In (Edelkamp & Helmert 1999) fact groups are defined by balanc-

ing, merging and instantiating predicates. If we define $\#pred_i(p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n)$ as the number of objects p_i for which the fact (`pred` p_1 ... p_n) is true than we establish a single-valued invariances at i if $\#pred_i(p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n) = 1$. Object p_i is the representative of the invariance. More elaborated balance conditions require predicates mergings.

Comparing all instantiations of (`pred` p_1 ... p_n) for object p_i and p_i' now indicates symmetry in the planning domain. If the set of instantiations $pred_i(p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n)$ match for all predicates $pred$, a symmetry between p_i and p_i' is found. Based on this observation in (Edelkamp 2002a) we give an efficient algorithm for reducing symmetry of objects to symmetry of fact groups. All pre-compiled symmetries of groups were tested for the current and goal state. If the assumed isomorphism between the two group representatives is verified all operators that apply changes to one of objects are pruned.

In *Desert-Rats* all supply-tanks are found to be symmetric, such that in case their fuel level and position match, any operator will consider just one of them. In *Zeno-Travel* all passengers share symmetry. In *Taxi* at least all taxis are found to be isomorphic.

Heuristics

For the propositional part we have implemented the relaxed planning heuristic h_f (Hoffmann & Nebel 2001) and the pattern database heuristic h_p (Edelkamp 2001c).

Merging Propositional Estimates

One suitable combination of h_f and h_p heurstics is to comparing the retrieved result of the pattern database according to a problem abstraction with the set of operators in the plan graph that respect the pattern. The intuition is to slice the relaxed plan graph according to the given problem abstractions. If in the backward exploration an add-effect is selected the match will be assigned to its fact group. If the number of matches in an abstraction is smaller than the retrieved pattern database value it will be incresed by the lacking amount.

Coarse Numerical Estimate

If the goal state contains numerical information, with h_n we approximate the number of steps necessary to achieve the numerical goal independent of the propositional setting. For each variable r we take the difference d_r of the goal value g_r and the current variables instantiation c_r as an indicator. Since numbers can be arbitrary small and large, we normalize the difference values, deviding the d by their maximal absolute change a_r.

Once more we propose a relaxed fixpoint exploration to approximate the vector a of maximal change of the numerical quantities a_r by neglecting numerical preconditions to keep the exploration polynomial. For each operator and given a we instantiate the effect lists and determine if the a_r for a quantity r has to be increased or not. We terminate in case of no further change.

Scheduling Heuristic

Critical-path analysis can also guide the plan finding phase. We derive a heuristic estimate h_s that schedules relaxed plans. Reacall that for each state FF solves a relaxed planning problem explicitly, constructing the relaxed plan as a sequence of grounded operators. Different to the heuristic estimate that only considers the length of the greedily extracted plan, we also take the sequence of operators into account. As the success of the planner has shown, even though relaxed plans provide neither a lower nor an upper bounds they are very informative.

However, schedules are not additive. Adding the two PERT-schedules for the path $p_g(u)$ to a state and for the sequence of actions $p_h(u)$ in the relaxed plan is not as accurate as the PERT-schedule of the combined paths $p_g(u) \circ p_h(u)$. Therefore, the classical merit function of A*-like search engines $f = g + h$ for generating path length g and heuristic estimate h is not immediate for concurrent planning. Therefore, we define the schedule heurstic h_s as the critical path of $p_g \circ p_h$ minus the critical path of p_g.

Combined Relaxed Plan Heuristic

The refined combination of propositional and numerical information in a unified plan graph heursitic is performed in planning problems, in which at least some numerical quantities are grounded to finite domains.

If numerical variables are finite, the corresponding operators can be simplified by grounding the variables to their respective instantiations. However, to avoid the extensive blow-up in the number of operators, we decided to keep numerical values and explicitly excecute preconditioning and application numerical effect in the relaxed plan exploration.

In the forward phase only effects were applied to generate the layered structure of the relaxed plan graph, while in the backward phase we also apply preconditions for propergation. We restrict to simple variable-value equalities that fix values and fire further effects in actions.

In *Taxi* this combined relaxed plan graph heuristic h_c integrates driving actions to the backward phase that were not present in the relaxed plan of a purely propositional estimate.

Refining Relaxed Plan Estimate

The process of refining estimates (Edelkamp 2002a) criticizes the retrieved relaxed plan with complete solutions to problem abstractions. It generalizes the idea of mobile analysis in (Long & Fox 2001). From the set of operators in the relaxed plan a subset is extracted and all preconditions considering the selected fact groups are collected. The dependency graph for the operators

```
Procedure Any-Time
    G ← ∅
    α ← ∞
    Open ← S
    while Open ≠ ∅
        S ← Open.Extract()
        for all S' ∈ expand(S)
            if (S' ∈ G)
                cp ← Critical-Path()
                if cp < α
                    α ← cp
                    G ← S'
            else
                Open.Change(S')
    return G
```

Table 4: General Any-Time Search Algorithm.

fulfilling the preconditions based on the dependency relation given above will often contain cycles. An extended linear time topological sorting algorithm will include a new operator if a cycle is encountered.

Search Strategies

We have implemented A* (Pearl 1985) with the option of scaling the influence of the estimate, thus including the extremes of breadth-first and best-first search. In pure propositional planning we prefer a dial as the priority queue implementation, while in general numerical planning we chose weak-heaps (Edelkamp & Stiegeler 2002). For very large exploration problems we provide IDA* (Korf 1985) with and without bit-state hashing. Hoffmann's Enforced-Hill-Climbing algorithm (Hoffmann & Nebel 2001) has also been integrated. For symbolic exploration MIPS also provides the symbolic breadth-first and symbolic A* search (Edelkamp 2001a). Even if non-deterministic domain are not yet available in PDDL syntax, weak and strong planning algorithms (Cimatti, Roveri, & Traverso 1998) are also part of the portfolio.

Any-Time Search

Short sequential plans do not necessarily imply short concurrent plans and vice versa. Even scheduling extended sequential plans with relaxed plan graph approximations will not necessarily yield optimal plans. Nevertheless, as the experiments highlight the quality of the established schedules is considerably *good*. Table 4 indicates how to wrap a heuristic search planner for so-called *any-time* performance, gradually improving the plan quality.

For grounded PDDL+ problems with finite state-spaces the any-time extension for any heuristic search algorithm that changes the enumeration order in the tree expansion of the problem graph is complete and optimal. A more general result is given by Pearl (Pearl 1985): If the cost of every infinite path is unbounded, A* search fully enumerates state-space and preserves optimality. This indicates that any-time heuristic search algorithms eventually find optimal plans even in infinite state spaces.

Elimination of Duplicates

One subtle problem arises when eliminating duplicate states to avoid redundant work. Consider the two sequences `(zoom city-a city-c plane)`, `(board dan plane)`, `(refuel plane)`, `(zoom city-c city-a plane)`, `(board scott)`, `(debark dan)`, `(refuel plane)`, and `(board scott)`, `(zoom city-a city-c plane)`, `(board dan plane)`, `(refuel plane)`, `(zoom city-c city-a plane)`, `(debark dan)`, `(refuel plane)` in the *Zeno-Travel* domain. The set of operators is the same and so is the resulting state. However, the concurrent plan for the first sequence is shorter than the schedule for the second one, since in the previous case the time for boarding `scott` is compensated by the remaining two operators.

Therefore, to preserve completeness and optimality is to compute and store schedules instead of states.

Experiments

We apply Any-Time Weighted A* for h_f, h_p and the schedule heuristics h_s and h_c. To both sequential estimates the numerical offset h_n is added. The search depth of the plan, the number of expanded and the number of stored states are denoted by d, e and s, respectively. The sequential plan quality is depicted as s_{seq} and the corresponding concurrent plan length is abbreviated by s_{con}. CPU time is denoted by t and given in seconds on a Sun Ultra Workstation, 248 MHz.

Zeno-Travel

The results for Zeno-1 are as follows.

	s_{seq}	s_{con}	e	s	d	t
$h_p + h_n$	370	290	11	57	8	0.00s
$h_f + h_n$	400	380	8	41	6	0.00s
	376.667	330	24	105	7	0.00s
	340	290	62	270	7	0.01s
h_s	370	290	10	51	8	0.01s
$h \equiv 0$	400	380	429	1799	6	0.08s
	390	340	1975	8191	7	0.36s
	340	290	1982	8223	7	0.36s

Both $h_p + h_n$ and h_s find the optimum as the first established plan, while $h_f + h_n$ needs some efforts to consolidate. The optimal plan is:

```
0: (board scott plane city-a) [30]
30: (zoom plane city-a city-c) [100]
130: (board ernie plane city-c) [30]
130: (refuel plane city-c) [40]
170: (zoom plane city-c city-d) [100]
270: (debark scott plane city-d) [20]
270: (debark ernie plane city-d) [20]
```

Different to *Zeno-1 Zeno-2* the objective is to minimze fuel consumption.

	s_{seq}	s_{con}	e	s	d	t
$h_p + h_n$	666.66	666.66	8	41	6	0.00s
$h_f + h_n$	666.66	666.66	111	520	7	0.02s
h_s	-	-	-	-	-	-
$h \equiv 0$	666.66	666.66	429	1799	6	0.07s

The search with h_s fails, since zero-valued operators generate large plateaus, so that the search generates scheduled plans with almost arbitrary number of operators without eventually reaching the goal. The best concurrent plan is:

```
0: (board scott plane city-a) [0]
0: (fly plane city-a city-c) [333.33]
333.33: (board ernie plane city-c) [0]
333.33: (board dan plane city-c) [0]
333.33: (fly plane city-c city-d) [333.33]
666.66: (debark scott plane city-d) [0]
666.66: (debark ernie plane city-d) [0]
```

For *Zeno-3* the sum of time and fuel consumption has to be minimized with the following outcome.

	s_{seq}	s_{con}	e	s	d	t
$h_p + h_n$	1370	1290	11	57	8	0.00s
	1253.33	1173.33	30	145	8	0.01s
	1096.67	1046.66	150	727	7	0.04s
$h_f + h_n$	1066.66	1046.66	8	41	6	0.00s
h_s	1093.33	1046.66	7	45	7	0.01s
$h \equiv 0$	1066.66	1046.66	429	1799	6	0.08s

The optimal schedule is:

```
0: (board scott plane city-a) [30]
30: (fly plane city-a city-c) [483.33]
513.33: (board ernie plane city-c) [30]
513.33: (board dan plane city-c) [30]
543.33: (fly plane city-c city-d) [483.33]
1026.66: (debark scott plane city-d) [20]
1026.66: (debark ernie plane city-d) [20]
```

Since all problems are solved in less than a second, the efficiency is difficult to interpret. Therefore, we evaluate a more involved example of (Edelkamp 2001b), where the third passenger also has a pre-specified target location. The objective function is *total-time*. The following table depicts the improvement of plan quality in this extended *Zeno-Travel-1* problem.

	s_{seq}	s_{con}	e	s	d	t
h_p	803.33	733.33	234	1137	11	0.04s
$+$	780	713.33	842	3960	12	0.16s
h_n	743.33	673.33	876	4082	12	0.16s
	766.66	670	1172	5371	12	0.22s
	730	630	2549	11604	12	0.49s
	730	600	7712	36364	12	1.57s
	670	570	9423	44084	13	1.93s
	670	540	36894	167593	13	7.54s
h_f	780	683.33	987	5519	12	0.40s
$+$	766.66	670	1074	5878	12	0.42s
h_n	766.66	640	1179	6323	12	0.45s
	730	630	1345	7010	12	0.50s
	730	600	1450	7455	12	0.53s
	670	570	5971	29122	13	2.10s
	670	540	6367	31026	13	2.23s
h_s	710	540	1285	5596	14	4.31s

Both sequential heuristics lead to fast convergence in all cases, but the number of expansions grows considerably. The FF heuristic is more effective than the pattern database heuristic and consumes slightly more time for each considered state. The schedule heuristic still yields the optimal plan on the first shot and expands less states. This is counter-balanced in time consumption. Breadth-first search fails to encounter depth 11.

The best plan of the problem is:

```
0: (zoom plane city-a city-c) [100]
100: (board dan plane city-c) [30]
100: (refuel plane city-c) [40]
100: (board ernie plane city-c) [30]
140: (zoom plane city-c city-a) [100]
240: (debark dan plane city-a) [20]
240: (board scott plane city-a) [30]
240: (refuel plane city-a) [40]
280: (zoom plane city-a city-c) [100]
380: (refuel plane city-c) [40]
420: (zoom plane city-c city-d) [100]
520: (debark scott plane city-d) [20]
520: (debark ernie plane city-d) [20]
```

When comparing any-time performance of improving plans, the interpretation of the experimental outcome is not immediate. Even if not necessarily optimal and even if the first plan might be established later than with sequential plan improvements, the relaxed plan schedule is favorable, since, with respect to undecidablility result, stopping with the first plan found, is probably the best termination criterion we can get. On the other hand, zero-resource operators according to the objective function call for a cost function dependent on the path length.

Discretized Desert-Rat

In the discretized *Desert-Rat* domain sequential plans cannot be improved by critical-path analysis, since by our definition all operators in a sequential plan are dependent or *instantaneous*. Therefore, we evaluate the time and exploration efforts for finding the first plan

only. Since no propositional goal is specified the numerically extended FF and pattern database heuristics collapse to h_n.

We have summarized the results in *Desert-Rat* in the following table. The application of symmetry reduction is denoted by +.

d	s_{seq}	s_{con}	e	s	d	t
$(300)^-$	15	15	20	60	5	0.01s
$(300)^+$	15	15	20	60	5	0.01s
$(500)^-$	35	35	18,413	47,980	13	6.44s
$(500)^+$	35	35	358	549	13	0.47s
$(600)^-$	60	60	436,173	577,233	24	153.14s
$(600)^+$	60	60	26,723	43,782	24	8.46s

A sequential plan to the distance 600 problem is:

```
0: (load truck f5) [0]
0: (drive-out truck) [5]
5: (unload truck f5) [0]
5: (drive-back truck) [5]
10: (load truck f6) [0]
10: (refuel truck f2) [0]
10: (drive-out truck) [5]
15: (unload truck f6) [0]
15: (drive-back truck) [5]
20: (load truck f3) [0]
20: (refuel truck f1) [0]
20: (drive-out truck) [5]
25: (fill-up truck f5) [0]
25: (drive-out truck) [5]
30: (unload truck f3) [0]
30: (drive-back truck) [5]
35: (load truck f6) [0]
35: (refuel truck f5) [0]
35: (drive-out truck) [5]
40: (refuel truck f3) [0]
40: (drive-out truck) [10]
50: (unload truck f6) [0]
50: (refuel truck f6) [0]
50: (drive-out truck) [10]
```

Actually finding a such an involved plan for the challenging problem is a trademark for our efficient implementation and the advantage of accelerating sequential plan-finding first.

Jugs-and-Water

The *Jugs-and-Water* domain is a Level-2 problem. It contains no durative action and is inherently sequential. Unfortunately, the state spaces are very small, that solving even larger (m, n)-Jug problems is easy.

	s_{seq}	s_{con}	e	s	d	t
(5,3)	0	0	11	13	6	0.00s
(1237,1721)	0	0	216	218	108	0.05s

The established plan for the former case is

```
0: (fill jug2) [0]
0: (pour jug2 jug1) [0]
0: (fill jug2) [0]
0: (pour jug2 jug1) [0]
0: (empty jug1) [0]
0: (pour jug2 jug1) [0]
```

Taxi

The following plan with 35 operators was found with h_c, symmetry and transposition cuts in about about 10 seconds CPU time while expanding only 852 states. Since there is some space for improving the solution quality, we currently study further refinements to the heuristic estimate that can server better plans.

```
0: (schedule taxi1 arthur) [0]
0: (drive_to_fare taxi1 arthur up up) [79]
0: (schedule taxi2 ratburn) [0]
0: (drive_to_fare taxi2 ratburn down down) [73]
0: (schedule taxi3 grandma) [0]
0: (drive_to_fare taxi3 grandma up up) [46]
0: (schedule taxi4 prunella) [0]
0: (drive_to_fare taxi4 prunella down down) [34]
34: (load taxi4 prunella) [1]
35: (drive_to_dest taxi4 prunella up up) [104]
46: (load taxi3 grandma) [1]
47: (drive_to_dest taxi3 grandma up down) [81]
73: (load taxi2 ratburn) [1]
74: (drive_to_dest taxi2 ratburn up up) [49]
79: (load taxi1 arthur) [1]
80: (drive_to_dest taxi1 arthur down down) [77]
123: (unload taxi2 ratburn) [1]
124: (schedule taxi2 dw) [0]
124: (drive_to_fare taxi2 dw up down) [70]
128: (unload taxi3 grandma) [1]
129: (schedule taxi3 brain) [0]
129: (drive_to_fare taxi3 brain up up) [55]
139: (unload taxi4 prunella) [1]
157: (unload taxi1 arthur) [1]
158: (schedule taxi1 francine) [0]
158: (drive_to_fare taxi1 francine down down) [12]
170: (load taxi1 francine) [1]
171: (drive_to_dest taxi1 francine down up) [34]
184: (load taxi3 brain) [1]
185: (drive_to_dest taxi3 brain down up) [66]
194: (load taxi2 dw) [1]
195: (drive_to_dest taxi2 dw up up) [44]
205: (unload taxi1 francine) [1]
239: (unload taxi2 dw) [1]
251: (unload taxi3 brain) [1]
```

Conclusions

Essentially planning with numerical quantities and durative actions is planning with time and resources. The framework of grounded PDDL+ problems can be seen as a normal form for resource planning and allows to certify complexity results. We have proposed a planner for mixed propositional and numerical planning problems with finite branching, in which numerical pre- and postconditions are instantiated on the fly and which produces concurrent plans for a broad subclass of problems. The planner parses, pre-compiles, solves, and schedules PDDL+ problems with time and resources and different objective functions. Optimization is performed by an any-time extension to the underlying heuristic search engine.

Some other planners like TP4 (Haslum & Geffner 2001), SAPA (Do & Kambhampati 2001), and TL-Plan (Baccus & Ady 2001) can cope with different forms

of PDDL+ expressiveness. We expect the international planning competition to give more insights in current state-of-the-art in planning technology.

PERT scheduling and critical path analysis for timed precedence networks is one of the simpler cases for scheduling (Syslo, Deo, & Kowalik 1983). We have achieved a simplification by solving the sequential path problem first. The any-time search algorithm origins in *Localized A** (Edelkamp & Schrödl 2000) and shares similarities with depth-first branch-and-bound (Zhang & Korf 1995). Different forms of symmetry reduction based on the TIM inference module has also been shown to be effective (Fox & Long 1999; 2002).

The core objective for future research is to enlarge the problem class to conformant and infinite branching problems. We will try to suit planning to various application domains and to allow user interaction with a horizontal bar or line *Gantt* chart that visualizes schedules and includes the following features: actions are identified on the left hand side, time scale is depicted on the top of the chart, a horizontal open oblong is drawn against each activity indicating estimated duration.

References

Baccus, F., and Ady, M. 2001. Planning with resources and concurrency. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Cimatti, A.; Roveri, M.; and Traverso, P. 1998. Strong planning in non-deterministic domains via model checking. Technical Report 9801-07, IRST.

Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. 1990. *Introduction to Algorithms*. The MIT Press.

Do, M. B., and Kambhampati, S. 2001. Sapa: a domain-independent heuristic metric temporal planner. In *European Conference on Planning (ECP)*, 109–120.

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science, 135–147. Springer.

Edelkamp, S., and Helmert, M. 2001. The model checking integrated planning system MIPS. *AI-Magazine* 67–71.

Edelkamp, S., and Reffel, F. 1999. Deterministic state space planning with BDDs. In *European Conference on Planning (ECP), Preprint*, 381–382.

Edelkamp, S., and Schrödl, S. 2000. Localizing A*. In *National Conference on Artificial Intelligence (AAAI)*, 885–890.

Edelkamp, S., and Stiegeler, P. 2002. Implementing HEAPSORT with $n \log n - 0.9n$ and QUICKSORT with $n \log n + 0.2n$ comparisons. *ACM Journal of Experimental Algorithmics*.

Edelkamp, S. 2001a. Directed symbolic exploration and its application to AI-planning. In *AAAI-Spring Symposium on Model-based Validation of Intelligence*, 84–92.

Edelkamp, S. 2001b. First solutions to PDDL+ planning problems. In *Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG)*, 75–88.

Edelkamp, S. 2001c. Planning with pattern databases. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science. Springer. 13-24.

Edelkamp, S. 2002a. Strategies for mixed propositional and numerical planning. In *European Conference on Artificial Intelligence (ECAI)*. Submitted.

Edelkamp, S. 2002b. Symbolic pattern databases in heuristic search planning. In *Artificial Intelligence Planning and Scheduling (AIPS)*.

Fox, M., and Long, D. 1999. The detection and exploration of symmetry in planning problems. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 956–961.

Fox, M., and Long, D. 2001. PDDL2.1: An extension to PDDL for expressing temporal planning domains. Technical report, University of Durham, UK.

Fox, M., and Long, D. 2002. Extending the exploitation of symmetries in planning. In *Artificial Intelligence Planning and Scheduling (AIPS)*.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In *European Conference on Planning (ECP)*, 121–132.

Helmert, M. 2002. Decidability and undecidability results for planning with numerical state variables. In *Artificial Intelligence Planning and Scheduling (AIPS)*.

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

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

Long, D., and Fox, M. 2001. Hybrid stan: Identifying and managing combinatorial optimisation subproblems in planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Meriott, K., and Stuckey, P. 1998. *Programming with Constraints*. MIT Press.

Pearl, J. 1985. *Heuristics*. Addison-Wesley.

Syslo, M. M.; Deo, N.; and Kowalik, J. S. 1983. *Discrete Optimization Algorithms with Pascal Programs*. Prentice-Hall.

Zhang, W., and Korf, R. E. 1995. Performance of linear-space search algorithms. *Artificial Intelligence* 79:241–292.

A Temporal Planning System for Level 3 Durative Actions of PDDL2.1

Antonio Garrido
Dpto. Sistemas Informaticos y Computacion
Universidad Politecnica de Valencia
Camino de Vera s/n, Valencia (Spain)
{**agarridot@dsic.upv.es**}

Abstract

Many planning domains have temporal features that can be expressed as durations associated with actions. Unfortunately, the conservative model of actions of most temporal planners is not appropriate for some domains which require richer models. Level 3 of PDDL2.1 introduces a model of durative actions which includes local conditions and effects to be satisfied at different times during the execution of the actions, thereby giving the planner freedom to plan concurrent actions. This paper presents a temporal planning system (TPSYS), which combines the ideas of Graphplan and TGP, to plan with such durative actions. The approach necessitates the modification of some aspects of the basic planning algorithm: the mutex reasoning, the generation of the temporal graph and the search for an optimal plan. Although the algorithm becomes more complex, the experimental results demonstrate it remains feasible as a way to deal with durative actions.

Introduction

Typically, classical planning systems simplify real problems by imposing unreal constraints on the problems. Particularly, planners rely on a model of actions in which all actions have the same duration. Although this assumption may be adequate for some planning problems, it becomes inadequate when dealing with temporal planning problems. For instance, this assumption is not true in real temporal environments, where different actions take different times of execution and concurrent actions are required to minimise the duration of the plan. Consequently, in temporal environments the optimisation criterion must be changed because the interest lies in obtaining a plan of minimal duration rather than a plan of minimal number of actions.

Most temporal planners appeared in the recent literature, such as parcPLAN, TGP or TP4 (El-Kholy & Richards 1996; Smith & Weld 1999; Haslum & Geffner 2001) have yielded some success when dealing with temporality on actions. Nevertheless, these temporal planners have adopted the same conservative model of actions of non-temporal planners. This means that two actions cannot overlap in *any way* if an effect or precondition of one is the negation of an effect or precondition of the other. Although this makes it possible to produce reasonable plans in most benchmark planning domains, there exist some domains which require a richer model of actions, and in which better quality plans can be found if a richer model of actions is used.

PDDL2.1 (Fox & Long 2001) is the new version of the standard language (PDDL) for the encoding of the planning domains which has been proposed for the the AIPS-2002 Planning Competition. PDDL2.1 provides five levels to define planning problems. Concretely, the level 3 introduces a new model of actions, called durative actions, which makes it possible to allow actions to overlap even when their preconditions or effects refer to the same propositions. This is possible because traditional preconditions and effects are now annotated with time points.

This paper presents a Temporal Planning SYStem (from now on TPSYS) in order to manage the model of durative actions proposed in level 3 of PDDL2.1. TPSYS is based on a three-stage process, which combines the ideas of Graphplan (Blum & Furst 1997) and TGP (Smith & Weld 1999). Hence, the main contributions of this paper are:

- An analysis of how durative actions can be managed in a Graphplan-based approach.
- An explanation of how a compact temporal graph can be generated.
- The extension of the mutual exclusion reasoning to manage PDDL2.1 durative actions, based on the work of TGP.
- A description of the plan extraction stage and the way it obtains the plan of optimal duration (in terms of the duration of the actions) as an acyclic flow of actions through the temporal graph.
- Some experimental results showing the importance of the mutual exclusion reasoning in richer models of actions, as indicated in (Smith & Weld 1999).

This paper is organized as follows. In the second section, we briefly review the motivations for introducing the model of durative actions of level 3 of PDDL2.1. The third section introduces the action model, the components of a durative action and the terminology used through the paper. The TPSYS algorithm and its three stages are described in the fourth section. This section provides the modifications the planning algorithm necessitates to deal with durative actions. Some experimental results are shown in the fifth section, demonstrating the feasibility of the system proposed. The sixth section discusses two approximations for dealing with durative actions in traditional planners. Finally, the conclusions are presented in the seventh section.

Motivation

PDDL does not allow the definition of actions with duration, which imposes an important limitation in real temporal problems. In developing PDDL2.1 to allow the modelling of temporal planning domains it was considered critical to allow a fuller exploitation of concurrency than can be captured using the strong mutex relation of the conservative model of actions, as the used in TGP (Smith & Weld 1999). This entails a more precise modelling of the state transitions undergone by different propositions within the durative interval of the action. In particular, the preconditions of the starting point of the action do not necessarily need to be maintained throughout the interval. There may be *pre*conditions of the final effect of the action that can be achieved concurrently rather than maintained throughout the interval. Hence, it becomes necessary to distinguish invariant from non-invariant conditions because there might be invariant conditions that cannot be affected during the interval of execution. Moreover, there might be initial effects of the starting point that can be exploited by concurrent actions. All these distinctions give rise to quite sophisticated opportunities for concurrent actions in a PDDL2.1 plan.

We motivate the modelling of the state transitions with the following example of the classical logistics domain in the conservative model of actions. Let us consider the action *fly(plane,origin,destination)*. This action requires the proposition *at(plane,origin)* to be true before executing the action, and asserts the propositions ¬*at(plane,origin)* and *at(plane,destination)* at the end of the action. This implies that the location of the *plane* is inaccessible until the end of the action, preventing concurrent actions (for instance, those that require the *plane* not to be in the origin) from being executed in parallel with *fly(plane,origin,destination)*. However, as presented in (Fox & Long 2001), this may exclude many valid plans. In PDDL2.1 this can be easily avoided by asserting ¬*at(plane,origin)* as an initial effect.

In addition, if we want to know the fact of being *flying* during the action *fly*, it would be enough by asserting the proposition *(flying-plane)* as an initial effect of the starting point and ¬*(flying-plane)* as a final effect of the end point. But, in a conservative model of actions, the equivalent action for this *fly* durative action would not represent the fact of being *flying* due to the impossibility of including the proposition *(flying-plane)* and ¬*(flying-plane)* as initial and final effects, respectively. Therefore, it is impossible to work with actions which require this proposition, such as the possible action *refuel-during-flight*.

Although in real problems instantaneous actions are never really *instantaneous*, there are some cases in which these actions could be useful for modelling purposes. Level 3 of PDDL2.1 also allows the definition of these actions, i.e. traditional actions with no duration. Since PDDL2.1 intends to provide *physics* instead of *advice* of the planning problem, instantaneous actions could be useful in order to obtain a valid plan for different executive agents when the duration of the action is very small (or even unknown) to be considered by the planning agent. More generally, the domain engineer might choose to model the domain at a level of abstraction at which it is not interesting to capture the durations of practically instantaneous actions. That is, the engineer might choose to emphasise the durations of some actions but not of others.

These modelling choices do not lead to conflict with the semantics presented in (Fox & Long 2001) because it is possible, at level 3 of PDDL2.1, to express an instantaneous action as an action with barely measurable duration. This duration is epsilon, an amount so small that it makes no sense to split it. This means that non-interfering actions that take epsilon time can happen in parallel but they cannot be interleaved. This epsilon is so small that it never changes the sequence of actions in the plan. Epsilon has to be chosen appropriately for a given domain and problem, because it represents a discretization of the time-line into indivisible units, the end points of which mark the points at which actions can be initiated or terminated.

Action Model and Terminology

Unlike traditional actions of PDDL, durative actions present more conditions to be guaranteed for the success of the action. Moreover, durative actions do not only have effects that hold at the end of the actions but also effects to be asserted immediately after the actions start.

Definition 1 ***Components of a durative action*** *(see Figure 1). Let a be a durative action which starts at time s and ends at time e, being executed through the interval $[s..e]$. The components of a are the following:*

- *Conditions. The three types of local conditions of a durative action are: i) $SCond_a$, the set of conditions to be guaranteed at the start of the action; ii) Inv_a, the set of invariant conditions to be guaranteed over the execution of the action; and iii) $ECond_a$, the set of conditions to be guaranteed at the end of the action.*

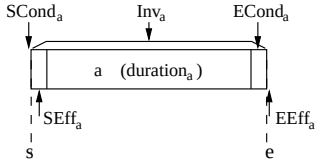


Figure 1: Components of a durative action a.

- *Duration. The duration of the action is a positive value represented by $D_a \in \mathbb{R}^+$.*
- *Effects. The two types of effects of a durative action are: i) $SEff_a = \{SAdd_a \cup SDel_a\}$, with the positive and negative effects respectively to be asserted at the start of the action; and ii) $EEff_a = \{EAdd_a \cup EDel_a\}$, with the positive and negative effects respectively to be asserted at the end of the action.*

Although level 3 allows the modelling of numeric conditions and effects as well as logical transitions, this version of TPSYS does not manage them yet.

Durative actions entail an important difficulty: there exist some effects ($SEff_a$) which can be obtained before the action ends. Hence, it might be possible that an initiated action could not end because its end conditions ($ECond_a$) are not satisfied in the future. In that case, all the start effects (and the actions which are dependent on them) should be invalidated. We call these kind of actions *conditional actions* because they are provisional until their end conditions are guaranteed, and we define them as:

Definition 2 ***Conditional action***. *One action a with $D_a > 0$ is a conditional action if $(SEff_a \neq \emptyset) \wedge (ECond_a \neq \emptyset)$ holds. This way, the set of propositions $SEff_a$ of a conditional action a only becomes valid when all propositions in $ECond_a$ are satisfied.*

Conditional actions are motivated by observing that there are domains in which durative actions are required precisely for some effect achieved through the duration of execution of an action (it is bounded by that duration). Such initial effects cannot be exploited as end effects because they do not persist beyond the end of the action. For example, in a logistics domain the plane is *flying* only during the action *fly*, so the initial effect *(flying-plane)* cannot be exploited beyond the end of the *fly* action. Further, when plans are validated, the successful termination of a durative action must be confirmed even if a goal is achieved before the end of its durative interval. This is because durative actions *promise* to terminate initiated actions in a stable state. If anything in the plan prevents this stable termination then the plan must be considered invalid. Richer goal specifications might allow one to consider goals that must persist only over finitely bounded intervals (Do & Kambhampati 2001), but PDDL2.1 does not yet support this.

Definition 3 ***Conditional proposition***. *One proposition p is conditional if all the actions $\{a_i\}$ which achieve p are conditional and they have not ended their execution yet.*

Intuitively, if p is only achieved by conditional actions $\{a_i\}$, p will be conditional until at least one action a_i ends successfully, which implies both $SCond_{a_i}$ and $ECond_{a_i}$ are satisfied. Once this happens, p is valid (stopping being conditional).

As we have seen in the previous section, instantaneous actions are allowed in level 3 of PDDL2.1. This does not represent a serious inconvenience because the correspondence rule below can transform an instantaneous action into a durative action. This way, all the instantaneous actions present in the planning domain can be managed in the same way as durative actions.

Definition 4 ***Correspondence rule*** $\mathcal{R}_{a_i \mapsto a_d}$. *The correspondence rule maps an instantaneous action a_i, with Pre_{a_i}, $Effs_{a_i} = Add_{a_i} \cup Del_{a_i}$ into a durative action a_d in the following way:*

$$SCond_{a_d} = ECond_{a_d} = Inv_{a_d} = Pre_{a_i}$$
$$SAdd_{a_d} = EAdd_{a_d} = Add_{a_i}$$
$$SDel_{a_d} = EDel_{a_d} = Del_{a_i}$$
$$duration_{a_d} = 0$$

Figure 2 shows the definition of the simple logistics domain *zeno-travel* for durative actions of level 3 of PDDL2.1. The three actions are *board*, *fly* and *debark*, which have duration, conditions and effects. According to Definition 1, the actions have *at start* and *over all* conditions with the conditions to be satisfied just at the beginning of the action and during all its execution, respectively. Analogously, the *at start* and *at end* effects have the effects to be asserted at the beginning and the end of the execution of the action.

At first blush the extension of a Graphplan-based planner to deal with durative actions of level 3 would seem quite easy. However, durative actions imply important changes in the way the temporal graph is generated and in the way the search for a plan is performed. These modifications are presented in the next section.

The Temporal Planning SYStem

In TPSYS, a temporal planning problem is specified as the 4-tuple $\{\mathcal{I}_s, \mathcal{A}, \mathcal{F}_s, \mathcal{D}_{max}\}$, where $\mathcal{I}_s$ and $\mathcal{F}_s$ represent the initial and final situation, respectively. $\mathcal{A}$ represents the set of durative actions in the planning domain. Time is modelled by $\mathbb{R}^+$ and their chronological order. $\mathcal{D}_{max}$ stands for the maximum duration allowed by the user. Although this bound is not defined in PDDL2.1 and it could be difficult to be decided, it allows the user a good way to constrain the goals deadline and the makespan of the plan as in (Do & Kambhampati 2001).

TPSYS is executed in three consecutive stages (see Figure 3). After the first stage, the second and the third stage

```
(:durative-action board
 :parameters (?p - person ?a - aircraft
              ?c - city)
 :duration (= ?duration (boarding-time ?c))
 :condition (and (at start (at ?p ?c))
                 (at start (free ?a))
                 (over all (at ?a ?c)))
 :effect (and (at start (not (at ?p ?c)))
              (at start (not (free ?a)))
              (at end (in ?p ?a))))

(:durative-action fly
  :parameters (?a - aircraft ?c1 ?c2 - city)
  :duration (= ?duration (flight-time ?c1 ?c2))
  :condition (and (at start (at ?a ?c1)))
  :effect (and (at start (not (at ?a ?c1)))
               (at end (at ?a ?c2))))

(:durative-action debark
 :parameters (?p - person ?a - aircraft
              ?c - city)
 :duration (= ?duration (debarking-time ?c))
 :condition (and (at start (in ?p ?a))
                 (over all (at ?a ?c)))
 :effect (and (at start (not (in ?p ?a)))
              (at end (free ?a))
              (at end (at ?p ?c))))
```

Figure 2: Definition of a simple domain in level 3 of PDDL2.1.

are executed in an interleaved way until a plan is found or the duration exceeds $\mathcal{D}_{max}$.

First stage: Preprocessing and Mutex Reasoning

Graphplan approaches define binary mutual exclusion relations between actions and between propositions. As TGP, TPSYS needs to calculate action-action mutex relationships, action-proposition mutex and proposition-proposition mutex. Since proposition-proposition mutex appears as a consequence of action-action mutex (Blum & Furst 1997), this stage only calculates the action-action and action-proposition *static* mutex relationships. These mutex relationships are static because they only depend on the definition of the actions and they always hold. Therefore, there is no reason to postpone their calculus to the next stages, speeding up the second and third stages. The process of calculating the mutex relationships is complicated by the semantics of PDDL2.1, which embodies a more permissive mutual exclusion relation than the languages of other temporal planners. The components of durative actions in PDDL2.1, presented in Definition 1, have some important implications for mutex reasoning. In particular, the strong mutex used by traditional temporal planners, such as TGP, must be modified to allow durative actions to be applied in parallel even in cases in which they refer to the same propositions. In traditional approaches, if two actions have

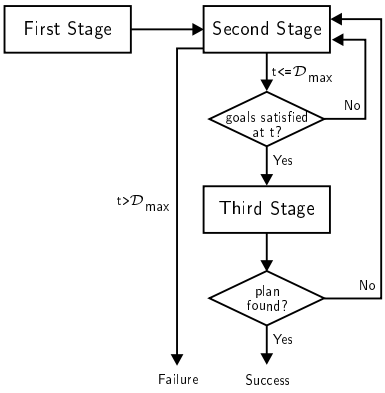


Figure 3: The three stages of TPSYS.

interfering propositions they cannot be executed in parallel, but when dealing with PDDL2.1 durative actions it may be possible for such actions to co-occur.

There exist four action-action mutex situations, presented in Table 1. Case 1 (*at start*) represents the mutex in which actions cannot start at the same time because start effects are contradictory or start effects and start conditions are conflicting. Case 2 (*at end*) represents the mutex in which actions cannot end at the same time because end effects are contradictory or end effects and end conditions are conflicting. Case 3 (*at end-start*) represents the mutex in which two actions cannot end and start at the same time, i.e. the actions cannot meet, because the end effects of one action are conflicting with the start conditions or effects of the other action. This mutex (which does not appear at Graphplan) might seem a stronger requirement than is really required, but it takes account of the fact that simultaneity can never be relied upon in the real world —it cannot be guaranteed that the action requiring the *at start* condition will definitely happen after the achievement of that condition at execution time. Furthermore, the computationally efficient testing of validity of a plan relies on not having to consider all possible orderings of so-called simultaneous happenings. This issue is discussed in depth in the PDDL2.1 semantics. Moreover, Graphplan is tailored to work with simple propositional formulae and it cannot be assumed that the positive assertion of a proposition will not interact harmfully with more complex precondition formulae. However, TPSYS takes the correctness-preserving assumption of including an epsilon ($\epsilon > 0$) between the action which ends and the action which starts to avoid this mutex and to make easier the implementation of the algorithm. Finally, case 4 (*during*) represents the mutex in which one action cannot start or end during the execution of the other because the start or end effects of the former are conflicting with the invariant conditions of the latter.

In addition to the action-action static mutex, the

Case	Condition for the mutex	Type of mutex	Relation
1	$(SAdd_a \cap SDel_b \neq \emptyset) \vee (SAdd_b \cap SDel_a \neq \emptyset)$ $((SAdd_a \cup SDel_a) \cap (SCond_b \cup Inv_b) \neq \emptyset)$ $((SAdd_b \cup SDel_b) \cap (SCond_a \cup Inv_a) \neq \emptyset)$	$AA_{start-start}$	
2	$(EAdd_a \cap EDel_b \neq \emptyset) \vee (EAdd_b \cap EDel_a \neq \emptyset)$ $((EAdd_a \cup EDel_a) \cap (ECond_b \cup Inv_b) \neq \emptyset)$ $((EAdd_b \cup EDel_b) \cap (ECond_a \cup Inv_a) \neq \emptyset)$	$AA_{end-end}$	
3	$((EAdd_a \cup EDel_a) \cap (SCond_b \cup Inv_b) \neq \emptyset)$ $((EAdd_b \cup EDel_b) \cap (SCond_a \cup Inv_a) \neq \emptyset)$ $(EAdd_a \cap SDel_b \neq \emptyset) \vee (EDel_a \cap SAdd_b \neq \emptyset)$ $(EAdd_b \cap SDel_a \neq \emptyset) \vee (EDel_b \cap SAdd_a \neq \emptyset)$	$AA_{end-start}$	
4	$(Inv_a \cap SDel_b \neq \emptyset) \vee (Inv_b \cap SDel_a \neq \emptyset)$ $(Inv_a \cap EDel_b \neq \emptyset) \vee (Inv_b \cap EDel_a \neq \emptyset)$	$AA_{during-during}$	

Table 1: Conditions for the static action-action mutex relationships between two durative actions a and b.

proposition-action mutex relationships are also calculated in the first stage. As demonstrated in (Smith & Weld 1999), when actions have different duration in a Graphplan-based approach, mutex between propositions and actions help deduce more inconsistencies because they better connect mutex between actions to mutex between propositions when actions are executed in parallel.

Definition 5 ***Static pa-mutex (proposition/action mutex)***. *One proposition p is statically mutex with action a iff $p \in \{SDel_a \cup EDel_a\}$.*

Second stage: Extension of the Temporal Graph

The second stage performs the extension of the temporal graph. The temporal graph consists of a directed, layered graph which alternates temporal levels of propositions and temporal levels of actions, represented by $P_{[t]}$ and $A_{[t]}$ respectively (Garrido, Onaindía, & Barber 2001). The levels are chronologically ordered by their instant of time, by means of a label t which represents the instant of time in which propositions are present and actions can start, or end, their execution. The way of extending the temporal graph is performed in a similar way to Graphplan. Particularly, the process consists of generating all the actions a_i in action level $A_{[t]}$ of the graph as soon as their start conditions are non pairwise mutex in the proposition level $P_{[t]}$, generating their start and end effects in the proposition levels $P_{[t]}$ and $P_{[t+D_{a_i}]}$, respectively. This process finishes once all the propositions in the final situation are present, non pairwise mutex in a proposition level $P_{[t]}$, and the actions which achieved them have already ended.

Modifications in the Extension of the Temporal Graph Although the idea of extending the temporal graph is conceptually simple, it contains some subtle details due to the local conditions and effects of durative actions. In each temporal level it is necessary to study first the effects achieved by the actions which end (whose *at end* conditions hold), and then the effects achieved by the actions which start. In consequence, each temporal level t is divided into two parts, *end*-part and *start*-part, in which the following action-action ($AA_{[t]}$ mutex), proposition-action ($PA_{[t]}$ mutex) and proposition-proposition ($PP_{[t]}$) mutex relationships must be calculated. We use the notation $AA_{[t]}$, $PA_{[t]}$ and $PP_{[t]}$ to represent the mutex relationships that hold at time t. These mutex relationships are temporary and can disappear in time, in contrast with the notation AA and PA that represent the static mutex relationships which always hold. The actions which end at action level $A_{[t]}$ are stored in $A_{[t]end}$, whereas the actions which start at action level $A_{[t]}$ are stored in $A_{[t]start}$. Analogously, the propositions achieved at the *end*-part are stored in $P_{[t]end}$, and the propositions achieved at the *start*-part are stored in $P_{[t]start}$.

On one hand, the mutex relationships to be calculated in the *end*-part are: $AA_{[t]end-end}$ with the actions which are mutex ending at t; $PA_{[t]end-end}$ with the propositions which are mutex with the actions which end at t; and $PP_{[t]end-end}$ with the propositions which are mutex at t after ending all the actions. On the other hand, the mutex relationships to be calculated in the *start*-part are: $AA_{[t]start-start}$ with the actions which are mutex starting at t; $AA_{[t]end-start}$ with the mutex between the actions which end and start at t; $PA_{[t]start-start}$ with the propositions which are mutex with the actions which start at t; $PP_{[t]end-start}$ with the propositions which are mutex at t and have been achieved by actions which end at t and actions which start at t, respectively; and $PP_{[t]start-start}$ with the propositions which are mutex at t after starting all the actions. The main reason for breaking down these mutex relationships into *end*-part and *start*-part lies in making their calculus simpler, as can be seen in the following definitions:

Definition 6 $AA_{[t]end-end}$. *Two actions a, b are end-end mutex at time t if one of the following holds: i) a, b are $AA_{end-end}$, ii) $ECond_a, ECond_b$ are $PP_{[t]end-end}$, or iii) a, b are $AA_{[t-\min(D_a,D_b)]start-start}$.*

Definition 7 $PA_{[t]end-end}$. *Let p be a proposition and a be an action. For each action b_i which achieves p at t, let*

$\Upsilon_{i[t]}$ *be the condition under* b_i *is mutex with the persistence of* p *at time* t*, i.e.* $\Upsilon_{i[t]} = [(p, b_i \text{ are } PA) \vee (p, ECond_{b_i} \text{ are } PP_{[t]end-end})]$*. Proposition* p *and action* a *are end-end mutex at time* t *if the following condition holds:* $\bigwedge_i[\Upsilon_{i[t]} \wedge (a, b_i \text{ are } AA_{[t]end-end})]$.

Definition 8 $PP_{[t]end-end}$. *Let* p, q *be two propositions and* $\{a_i\}, \{b_j\}$ *be the sets of actions which achieve* p *and* q *at time* t*, respectively. Propositions* p, q *are end-end mutex at time* t *if both of the following conditions hold: i)* $\forall b_j : p, b_j$ *are* $PA_{[t]end-end}$*, and ii)* $\forall a_i : q, a_i$ *are* $PA_{[t]end-end}$.

Definition 9 $AA_{[t]start-start}$. *Two actions* a, b *are start-start mutex at time* t *if one of the following holds: i)* a, b *are* $AA_{start-start}$*, or ii)* $SCond_a, SCond_b$ *are* $PP_{[t]start-start}$.

Definition 10 $AA_{[t]end-start}$. *Two actions* a *(ending at* t*) and* b *(starting at* t*) are end-start mutex at time* t *if one of the following holds: i)* a, b *are* $AA_{end-start}$*, or ii)* $ECond_a, SCond_b$ *are* $PP_{[t]end-end}$.

Definition 11 $PA_{[t]start-start}$. *Let* p *be a proposition and* a *be an action. For each action* b_i *which achieves* p *at* t*, let* $\Psi_{i[t]}$ *be the condition under* b_i *is mutex with the persistence of* p *at time* t*, i.e.* $\Psi_{i[t]} = [(p, b_i \text{ are } PA) \vee (p, SCond_{b_i} \text{ are } PP_{[t]start-start})]$*. Proposition* p *and action* a *are start-start mutex at time* t *if the following condition holds:* $\bigwedge_i[\Psi_{i[t]} \wedge (a, b_i \text{ are } AA_{[t]start-start})]$.

Definition 12 $PP_{[t]end-start}$. *Let* p *be a proposition first achieved at time* t *by the set of actions* $\{a_i\}$ *which end at* t*. Analogously, let* q *be another proposition first achieved at* t *by the set of actions* $\{b_j\}$ *which start at* t*. Propositions* p, q *are end-start mutex at time* t *if the following condition holds:* $\forall a_i, b_j : a_i, b_j$ *are* $AA_{[t]end-start}$.

Definition 13 $PP_{[t]start-start}$. *Let* p, q *be two propositions and* $\{a_i\}, \{b_j\}$ *be the sets of actions which achieve* p *and* q *at time* t*, respectively. Propositions* p, q *are start-start mutex at time* t *if both of the following conditions hold: i)* $\forall b_j : p, b_j$ *are* $PA_{[t]start-start}$*, and ii)* $\forall a_i : q, a_i$ *are* $PA_{[t]start-start}$.

Intuitively, $AA_{[t]}$ mutex relationships represent the impossibility of two actions ending, starting or abutting together at the same time t. $PA_{[t]}$ mutex represents the impossibility of having a proposition and one action starting or ending at time t. $PP_{[t]}$ mutex represents the impossibility of having two propositions together at time t. These calculus of the mutex relationships obtains the same mutex as Graphplan and, thereafter, they provide very useful information to improve the process of search by avoiding combination of actions, propositions and propositions/actions which cannot be satisfied simultaneously, thus reducing the space search (Blum & Furst 1997).

As can be seen in the previous definitions, the calculus of the mutex relationships in the *end*-part and *start*-part are nearly identical, with the only difference of recovering and storing the information in different structures. Thus, in some cases the structures could be the same improving the efficiency. Concretely, the implementation of the second stage only keeps one structure $PP_{[t]}$ for $PP_{[t]end-end}$, $PP_{[t]end-start}$ and $PP_{[t]start-start}$.

An important point to take into account when dealing with durative actions in a Graphplan-based approach and which forced us to modify the algorithm is the condition to finish the extension of the temporal graph. In Graphplan or TGP, this condition holds once all the propositions of the final situation are non pairwise mutex. However, conditional actions assert *at start* effects which might be included in the final situation *before* these actions end. This implies that the temporal graph extension might end in a level in which it is impossible to find a feasible plan because any of the propositions in the final situation is still conditional (it has not been validated yet), losing the benefits of the Graphplan-based graph extension. Loosely speaking, it means that the action which achieves that effect has not ended yet and the effects could be invalid (unavailable) if the *at end* conditions of the action fail. In order to tackle this drawback, it becomes necessary to propagate some additional *heuristic* information about the validity of the propositions achieved in the temporal graph. In this case, the same disjunctive reasoning on propositions of Graphplan can be applied on the instants of time at which the propositions stop being conditional. This propagation mechanism is quite straightforward, according to the following definition:

Definition 14 ***End time of a conditional proposition****. Let* p *be a conditional proposition and* $\{a_i\}$ *the set of conditional actions which achieve* p*. In the proposition level* $P_{[t]}$ *(at time* t*), the end time in which* p *stops being conditional,* $\max_{etc}$ *(the maximum end time conditional) is calculated as* $\min(\alpha_i)$*, where* α_i *is defined as:*

- $\max(\max_{etc}(SCond_{a_i}) + D_{a_i}, \max_{etc}(ECond_{a_i}), t)$*, if* p *is achieved in an end-part of the graph.*
- $\max(\max_{etc}(SCond_{a_i}) + D_{a_i}, t)$*, if* p *is achieved in a start-part of the graph.*

Algorithm for the Extension of the Temporal Graph. After introducing the modifications which are necessary to extend the temporal graph, we present the algorithm (see Figure 4) for extending the temporal graph. Starting at time $t = 0$ (with all the mutex structures empty), the algorithm generates new proposition and action levels (*end*-part and *start*-part), calculating all the mutex relationships. First, the *end*-part of the temporal graph is generated with the actions which can end (their end conditions are satisfied). Hence, the algorithm updates $A_{[t]end}$ with the actions which end at

Algorithm *Temporal Graph Extension*
$t = 0$
while $(t \leq \mathcal{D}_{\max}) \wedge (\mathcal{F}_s$ is not satisfied in $P_{[t]}) \wedge$
$(\mathcal{F}_s$ has not conditional propositions do
forall $< a_i, s_i, t >$ which can end at $A_{[t]end}$ do
$A_{[t]end} = A_{[t]end} \cup a_i$
$P_{[t]end} = P_{[t]end} \cup EAdd_{a_i}$
Generate *start*-part mutex
endforall
forall $< b_j, t, e_j >$ which can start at $A_{[t]start}$ do
$A_{[t]start} = A_{[t]start} \cup b_j$
$P_{[t]start} = P_{[t]start} \cup SAdd_{b_j}$
Generate *end*-part mutex
endforall
$t =$ next level in the *Temporal Graph*
endwhile

Figure 4: Algorithm for the temporal graph extension performed in the second stage.

time t, $P_{[t]end}$ with their end effects, and calculates all the mutex relationships presented above. Then, the algorithm generates the *start*-part of the graph with the actions which can start (their start conditions are satisfied). The algorithm updates $A_{[t]start}$ and $P_{[t]start}$ with the actions which start at time t and their start effects, respectively, calculating all the mutex relationships. Here, new temporal levels are generated according to the duration of the actions generated. This way, for each b_j generated in $A_{[t]start}$, the temporal levels $P_{[e_j]}$ and $A_{[e_j]}$ are created, where obviously $e_j = t + D_{b_j}$. $No-op$ actions and delete-edges (which represent the negative effects) are not stored in the temporal graph during its extension. This extension continues until the propositions in the final situation are achieved and they are not conditional, i.e. the actions which achieve them have ended and those propositions are valid. Moreover, the extension also finishes if the maximum time allowed by the user $\mathcal{D}_{max}$ is exhausted, returning '*Failure*' (see Figure 3).

Lemma 1 ***The extension of the temporal graph is complete****. If the temporal graph extension ends at time t, the algorithm generates all the necessary temporal levels (at which actions can end or start) between time 0 and t.*

Proof 1 *The proof is direct by definition of the algorithm. The algorithm generates all the actions $\{b_j\}$ whose $SCond_{b_j}$ hold in each temporal level. Each action level contains all the actions present in the previous action levels —analogously for the proposition levels. This way, once one action b_j appears in an action level, this action will appear in the next levels, and all the temporal levels in which b_j could end and start are calculated and created.*

Third stage: Extraction of a Plan

The third stage performs the extraction of an optimal plan, as an acyclic flow of actions, through the temporal graph extended in the second stage. In a Graphplan-based approach the plan is obtained by moving through the graph in a backward way. The process consists of obtaining the actions which achieve the propositions to be satisfied. Now, durative actions allow different ways to achieve these propositions, not only by their *at end* effects but also by their *at start* effects. Moreover, in order to plan an action all its conditions must be satisfied, which with durative actions entails to satisfy the start, end and invariant conditions. This breaks the traditional right to left *directionality* of Graphplan or TGP as shown in the following example.

Let us suppose an instant of time t during the extraction of a plan at which a proposition p must be satisfied. Let us suppose that action a achieves p at t as a start effect ($p \in SAdd_a$). If a has end conditions ($ECond_a$), they will have to be satisfied at time $t' = t + D_a$, forcing the algorithm to move again to an already visited instant of time $t' > t$. For this reason, the algorithm first selects the set of actions $\{a_i\}$ which achieve each proposition as end effects in order to keep the traditional directionality of the search.

Moreover, before planning an action a it is necessary to study whether a is compatible with the actions already planned, i.e. that the new action a does not modify the invariant conditions of the other actions ($AA_{during-during}$ mutex relationships of Table 1), discarding a if it is not compatible.

The algorithm for the extraction of an optimal plan is shown in Figure 5. It uses two structures, one queue $GoalsToSatisfy$ formed by pairs $< p, t >$ with the goal proposition p to be satisfied at time t, and one list $Plan$ formed by $< a_i, s_i, e_i >$ 3-tuples with the planned action a_i starting at s_i and ending at e_i. $GoalsToSatisfy$ is initialized with the propositions of the final situation to be satisfied at the instant of time at which the temporal graph extension has finished. $Plan$ is initially empty. The algorithm proceeds in the following way. While there are (sub)goal propositions in $GoalsToSatisfy$, the algorithm dequeues a pair $< p, t >$ to be satisfied. Note that now, p could be already satisfied at time t because actions are planned in different points of time and not always in a right to left order. If p is not already satisfied at time t in $Plan$, actions that satisfy p at time t are selected in a backtracking point. Although all the set of actions $\{a_i\}$ which are compatible with actions in $Plan$ must be considered for completeness, the actions which achieve p as end effects are firstly selected to keep the traditional right to left directionality. If action a_i is not mutex with the actions in $Plan$, then a_i is planned updating the structures $Plan$ and $GoalsToSatisfy$ with a_i and the start, invariant and end conditions of a_i, respectively.

Since the temporal graph extension finishes as soon as all the propositions in the final situation are present, non pairwise mutex, and the plan extraction is complete, the algorithm obtains the optimal plan in terms of the duration of

Algorithm *Plan Extraction*
$GoalsToSatisfy = \mathcal{F}_s$ at the end time of second stage
$Plan = \emptyset$
while $(GoalsToSatisfy \neq \emptyset)$ do
  Dequeue $< p, t >$ from $GoalsToSatisfy$
  if $< p, t >$ is not already satisfied in $Plan$
    Select $< a_i, s_i, e_i >$ which satisfies p at t and compatible with $Plan$
    $Plan = Plan \cup < a_i, s_i, e_i >$
    $GoalsToSatisfy = GoalsToSatisfy \cup SCond_{a_i} \cup Inv_{a_i} \cup ECond_{a_i}$
  endif
endwhile

Figure 5: Algorithm for the plan extraction performed in the third stage.

the actions (Garrido, Onaindía, & Barber 2001).

Lemma 2 ***The extraction of a plan is a complete process.***

Proof 2 *The proof is trivial due to the fact that the algorithm considers all the possible actions (backtracking point) which satisfy each proposition p from $GoalsToSatisfy$.*

Theorem 1 ***Optimality of the algorithm.*** *The first plan the algorithm extracts is the plan of optimal duration.*

Proof 3 *By contradiction, let $\mathcal{P}_t$ be the first plan (of duration t) the algorithm extracts. We assume this plan is not optimal, so we deduce that there exists a plan $\mathcal{P'}_{t'}$ (of duration $t' < t$) which has not been found by the algorithm and is optimal. This implies one of the following cases: i) the temporal level t' has not been generated during the extension of the temporal graph, or ii) the temporal level t' has been generated but the extraction stage has not considered the plan $\mathcal{P'}_{t'}$ from that level t'. The first case is false by Lemma 1 which claims the completeness of the temporal graph extension, and the second case is also false by Lemma 2 which claims the completeness of the plan extraction stage. In consequence, this contradicts the initial choice of the existence of $\mathcal{P'}_{t'}$. Hence, $\mathcal{P}_t$ is the plan of optimal duration.*

Application Example

We present a simple application example, based on the logistics domain *zeno-travel* presented in Figure 2. This example allows us to illustrate the extension of the temporal graph. In order to keep the temporal graph simple enough, the example to be solved consists of transporting one person, $ernie$, from $city - a$ to $city - b$ by using a $plane$ which is initially in $city - a$. The duration of the actions is 5 for $board$ and $debark$, and 10 for fly. Table 2 shows the proposition levels and the action levels. For each proposition level $P_{[t]}$, only the $P_{[t]end}$ part of the graph is shown because the actions of the domain have no positive *at start* effects —negative effects are not stored in the temporal graph. For each action level $A_{[t]}$, both the $A_{[t]end}$ and $A_{[t]start}$ are shown with the actions which end, and start, respectively at each instant of time. In time $t = 0$, actions *board(ernie,plane,city-a)* and *fly(plane,city-a,city-b)* are generated, but because they are $AA_{start-start}$ mutex the propositions *in(ernie,plane)* and *at(plane,city-b)* are mutex until time $t = 15$, in which the action *debark(ernie,plane,city-b)* is generated, thus obtaining the goal *at(ernie,city-b)* in time $t = 20$. As can be seen, although the actions have differing duration, the extension of the temporal graph is equivalent to Graphplan. The process of extraction of a plan selects the instances of actions which obtain the goals, then the start and end conditions of these actions, and so on. The plan obtained consists of the following sequence of actions:

$$\begin{array}{rll} 0+\epsilon: & \textit{board(ernie,plane,city-a)} & [5] \\ 5+2\epsilon: & \textit{fly(plane,city-a,city-b)} & [10] \\ 15+3\epsilon: & \textit{debark(ernie,plane,city-b)} & [5] \end{array}$$

The offset ϵ in the instant of time at which the actions are executed is a necessary feature for a valid plan of PDDL2.1 (Fox & Long 2001). This ϵ is included to avoid the simultaneity of the actions when they meet, as presented in the case 3 of the mutex relationships of Table 1.

Experimental Results

Currently, there does not exist an extensive collection of benchmarks for durative actions of PDDL2.1. Consequently, we have adapted some of the traditional domains of PDDL, such as *logistics*, *travel-bulldozer*, *ferry*, *gripper*, *monkey*, *blocksworld* and *zeno-travel* to the model of durative actions of PDDL2.1. Direct comparison between TPSYS and recent temporal planner such as Sapa (Do & Kambhampati 2001) or TP4 (Haslum & Geffner 2001) is difficult because they handle resources and even non-admissible heuristics which cannot guarantee the optimal solution. Nevertheless, we want to do direct comparison in the immediate future. Consequently, we compare TPSYS with TGP to demonstrate that the algorithm presented here remains feasible in dealing with traditional temporal planning problems. We use two versions of TGP: TGP, which consists of the original version of (Smith & Weld 1999), and TGP-ng, which extends TGP to keep minimal nogoods, doing backjumping during the backward search in the way proposed in (Kambhampati 2000). The tests were censored after 60 seconds. The results of the tests obtained in a 64 Mb. memory Celeron 400 MHz. can be seen in Table 3.

The results show that TPSYS behaves well enough in all the problems. Unlike TGP, TPSYS calculates more mutex relationships under the model or durative actions,

Level t	$P_{[t]}$ $P_{[t]end}$	$A_{[t]}$ $A_{[t]end}$	$A_{[t]start}$
0	*at(plane,city-a), at(ernie,city-a)*	-	*board(ernie,plane,city-a), fly(plane,city-a,city-b)*
5	*at(plane,city-a), at(ernie,city-a), in(ernie,plane)*	*board(ernie,plane,city-a)*	*board(ernie,plane,city-a), debark(ernie,plane,city-a), fly(plane,city-a,city-b)*
10	*at(plane,city-a), at(ernie,city-a), in(ernie,plane), at(plane,city-b)*	*board(ernie,plane,city-a), debark(ernie,plane,city-a), fly(plane,city-a,city-b),*	*board(ernie,plane,city-a), debark(ernie,plane,city-a), fly(plane,city-a,city-b), fly(plane,city-b,city-a)*
15	*at(plane,city-a), at(ernie,city-a), in(ernie,plane), at(plane,city-b)*	*board(ernie,plane,city-a), debark(ernie,plane,city-a), fly(plane,city-a,city-b)*	*board(ernie,plane,city-a), debark(ernie,plane,city-a), debark(ernie,plane,city-b), fly(plane,city-a,city-b), fly(plane,city-b,city-a)*
20	*at(plane,city-a), at(ernie,city-a), in(ernie,plane), at(plane,city-b), at(ernie,city-b)*	*board(ernie,plane,city-a), debark(ernie,plane,city-a), debark(ernie,plane,city-b), fly(plane,city-a,city-b), fly(plane,city-b,city-a)*	-

Table 2: Outline of the temporal graph extension for the application example.

which allows to reduce the search space in the plan extraction. This allows the complexity of TPSYS to follow the same order of magnitud of TGP —and even TGP-ng. The most important differences appear in the problems *att-log3* and *big-bull2*, in which TGP is clearly better than TPSYS. Although the differences between TGP and TGP-ng are not very significant in these tests, the benefits which can be obtained by exploiting the CSP techniques presented in (Kambhampati 2000) are very promising to dramatically improve the behaviour of the plan extraction stage.

Discussion

The temporal planning system described in this paper represents an approximation for dealing with durative actions of PDDL2.1 in a Graphplan-based approach. Therefore, most of the extensions used in Graphplan-based planners could be used here, such as memoization (Blum & Furst 1997) and regression (Kambhampati 2000) to improve the third stage, propositions in the initial (final) situation being placed (required) at any time during the execution of the plan, and exogenous events as presented in (Smith & Weld 1999).

Now, we discuss two alternative methods to tackle with durative actions with *at start* effects and *at end* conditions in a temporal planner with ability to manage instantaneous actions. Both of them consist of splitting each durative action into a collection of simple actions.

The first alternative splits each durative action into two instantaneous actions (which represent the start and end points of the durative action) and one action with duration (which represents the process of the action). All these three new actions will have neither *at start* effects nor *at end* con-

Problem	TPSYS	TGP (TGP-ng)
att-log0	0.42	0.02 (0.01)
att-log1	0.44	0.05 (0.01)
att-log2	0.47	0.06 (0.05)
att-log3	14.10	2.65 (2.50)
bulldozer-prob	0.88	0.55 (0.45)
big-bull1	0.58	0.80 (0.75)
big-bull2	14.31	2.15 (2.10)
ferry1	0.01	0.01 (0.01)
ferry2	0.03	0.02 (0.02)
ferry3	0.30	0.03 (0.02)
gripper2	0.03	0.03 (0.02)
gripper4	0.17	0.13 (0.16)
gripper6	6.88	4.53 (13.50)
monkey1-test	0.20	0.17 (0.15)
monkey2-test	0.63	0.75 (0.70)
tower2	0.02	0.03 (0.02)
tower4	0.28	0.45 (0.50)
tower6	2.52	3.60 (3.25)
zeno-travel1	0.01	0.01 (0.01)
zeno-travel2	0.02	0.01 (0.01)
zeno-travel3	0.02	0.01 (0.01)

Table 3: Comparison of TPSYS and TGP (results are in seconds).

ditions. Thus, a durative action a is divided into:

- $a1$, with no duration. $Pre_{a1} = \{SCond_a \cup Inv_a\}$ and $Eff_{a1} = \{SEff_a \cup ef_{a1}\}$.
- $a2$, with the duration of a (D_a). $Pre_{a2} = \{Inv_a \cup ef_{a1}\}$ and $Eff_{a2} = ef_{a2}$.
- $a3$, with no duration. $Pre_{a3} = \{ECond_a \cup Inv_a \cup ef_{a2}\}$ and $Eff_{a3} = EEff_a$.

The inclusion of the *artificial* effects ef_{a1} and ef_{a2} of actions $a1$ and $a2$ respectively, allows to generate the action $a2$ after $a1$, and $a3$ after $a2$, simulating the behaviour of the original action a. This way, during the plan extraction, action $a3$ only can be planned if $a2$ has been previously planned, and analogously, $a2$ only can be planned after planning $a1$. The main drawback of this method is the increment in the number of actions (in a factor of three per each durative action) and in the number of propositions (in a factor of two per each durative action) in the domain, which by itself may be prohibitive. Moreover, if one goal of the problem is satisfied by Eff_{a1}, i.e. the original $SEff_a$, only the action $a1$ would be planned (without needing to plan $a2$ nor $a3$), which would imply an unreal situation in which only a part of the indivisible action a is executed.

The second alternative is based on the semantic mapping described in (Fox & Long 2001), and consists of splitting each durative action into a collection of simple actions. The collection includes two instantaneous actions (which represent the start and end points of the durative action) and a number of identical monitoring actions responsible for confirming the maintenance of invariants. The monitoring actions can be achieved by requiring the $no-ops$ corresponding to the invariants of an action to be active in the interval between the start and end points of that action. Therefore, they do not need to be built explicitly and only two actions have to be constructed per durative action. Doubling up the number of actions need not present a blow-up at instantiation time, because the durative actions can be instantiated first and then split, rather than vice versa. During plan extraction it is necessary to maintain the link between the actions representing the start and end points of a durative action because neither one can be exploited without the other. In addition, it is necessary to manage the temporal constraints implied by the durations of the actions. A planner based on this approach has been constructed and appears to perform well in initial experiments (Long & Fox 2001). The approach still suffers from the problem caused when the start of a durative action is added to the plan for its effect (the initial effect of the durative action) necessitating the addition of the end action to the plan if it has not already been chosen. This in turn can introduce new preconditions, so there is an iterative structure to the plan extraction algorithm. This is highly reminiscent of the DP-Plan approach (Baioletti, Marcugini, & Milani 2000) in which the directionality of Graphplan is exchanged for a Davis-Puttnam search process.

Conclusions through Related Work

Last years have seen many attempts of dealing with temporal planning. The parcPLAN approach (El-Kholy & Richards 1996) handles a rich set of temporal constraints, instantiating time points in a similar way to TPSYS. TGP (Smith & Weld 1999) introduces a complex mutual exclusion reasoning which is very valuable in temporal environments. The critical difference between TGP and TPSYS is based on several points. First, TPSYS calculates the static mutex relationships in a preprocessing stage which allows to speed up the rest of stages. Second, TGP uses a more compact temporal graph in which actions and propositions are only annotated with the first level at which they appear. This reduces vastly the space costs but it increases the complexity of the search process, which may traverse cycles in the planning graph. In opposition, TPSYS uses a much more informed temporal graph which reduces the overhead during the search. Third, the mutex reasoning is managed in TGP by means of inequalities and sophisticated formulae, whereas TPSYS calculates the mutex relationships level by level in a more similar way to Graphplan. Finally, TPSYS uses a richer model of actions which implies: i) fewer constraints on the execution of the actions, ii) some modifications in the planning algorithm, and iii) a significantly larger space of search. More recent temporal planners, such as Sapa (Do & Kambhampati 2001) or TP4 (Haslum & Geffner 2001) handle concurrent actions and use heuristic metrics to deal with resources in planning. Sapa uses a model of actions similar to PDDL2.1, but it does not perform mutex propagation as our system. Sapa scales up quite well, but it uses non-admissible heuristics which cannot guarantee the optimal plan. On the other hand, TP4 uses admissible heuristic search to handle actions with time and resources, but it assumes a conservative model of actions.

This paper has presented a temporal planning system which handles durative actions provided by level 3 of PDDL2.1. Instead of using a conservative model of action, TPSYS manages actions with local conditions and effects. Although durative actions make the calculus of the mutex relationships, the temporal graph extension and the plan extraction stages more complex, they allow modelling of richer planning domains. Briefly, the main contributions of the paper have been the description of:

- The new components of level 3 durative actions based on (Fox & Long 2001) and the mutual exclusion relationships they entail.
- The modifications needed during the temporal graph ex-

tension. In the temporal graph extension, each temporal level has been divided into two parts to make easier the calculus of the mutex relationships.

- The modifications needed during the plan extraction. We have presented how the plan is found through the temporal graph without following the traditional right to left directionality.

The algorithm still has some limitations. According to our experiments, the performance of the algorithm degrades when there are many actions and propositions in the planning domain, due to the calculus of the mutual exclusion relationships. Moreover, the performance of the second stage degrades when the duration of the actions is wildly different. Particularly, the worst performance happens when the greatest common divisor of the durations of the actions is 1, which forces the algorithm to consider the maximum number of temporal levels, thus increasing the complexity of the third stage. For this reason, the areas of future work are focused on the inclusion of memoization techniques similar to the memoization performed in Graphplan and the inclusion of some of the CSP techniques presented in (Kambhampati 2000), which have been already tested on TGP. We also want to extend TPSYS to handle additional features of level 3 of PDDL2.1, such as numeric conditions and effects and inequality relations on conditions.

Acknowledgements

This work has been partially supported by the Spanish MCyT under project DPI2001-2094-C03-03, and by the Universidad Politecnica de Valencia under projects 20010017 and 20010980.

References

Baioletti, M.; Marcugini, S.; and Milani, A. 2000. DP-PLAN: An algorithm for fast solutions extraction from a planning graph. In *Proc. of AIPS*, 13–21.

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90:281–300.

Do, M., and Kambhampati, S. 2001. Sapa: a domain-independent heuristic metric temporal planner. In Cesta, A., and Borrajo, D., eds., *Proc. European Conf. on Planning (ECP-01)*, 109–120.

El-Kholy, A., and Richards, B. 1996. Temporal and resource reasoning in planning: the parcPLAN approach. In *Proc. 12th European Conference on Artificial Intelligence (ECAI-96)*, 614–618.

Fox, M., and Long, D. 2001. PDDL2.1: an extension to PDDL for expressing temporal planning domains. Technical report, University of Durham, UK.

Garrido, A.; Onaindía, E.; and Barber, F. 2001. Time-optimal planning in temporal problems. In Cesta, A., and Borrajo, D., eds., *Proc. European Conf. on Planning (ECP-01)*, 397–402.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In Cesta, A., and Borrajo, D., eds., *Proc. European Conf. on Planning (ECP-01)*, 121–132.

Kambhampati, S. 2000. Planning graph as (dynamic) CSP: Exploiting EBL, DDB and other CSP techniques in graphplan. *Journal of Artificial Intelligence Research* 12:1–34.

Long, D., and Fox, M. 2001. Fast temporal planning in a graphplan framework. Technical report, Dept. of Computer Science, University of Durham, UK.

Smith, D., and Weld, D. 1999. Temporal planning with mutual exclusion reasoning. In *Proc. 16th Int. Joint Conf. on AI (IJCAI-99)*, 326–337.

Timed Automata as an Underlying Model for Planning and Scheduling (a Position Paper)

Oded Maler
VERIMAG
2, av. de Vignate
38610 Gières, France
Oded.Maler@imag.fr

Introduction and Motivation

In this position paper we propose the model of timed automata, originating from the verification of real-time systems, as a model for posing and solving time-dependent planning and scheduling problems. We believe that in the same sense as automata are used as the major vehicle for verification of systems where the model of time is *qualitative*, timed automata can be the center of a a unifying mathematical modeling framework for *quantitative* time, having the following attractive features:

1. It is sufficiently expressive to describe the essential aspects of time-dependent real-life problems in a variety of application domains.
2. It provides for models with well-defined and clear dynamic semantics.
3. These models are amenable to computer-aided design methods such as simulation, testing, verification and automatic synthesis of (optimal) schedules and plans.
4. These methods are currently supported by tools of various levels of maturity, that treat the specific computational problems of time-related reasoning.

The problems of time-dependent behavior in general, and dynamic resource allocation in particular, pervade many aspects of modern life. A computer-aided timing technology can contribute to domains ranging from the reliability and efficient use of communication resources in a telecommunication network to the allocation of tracks in a continental railway network, from scheduling for the computational resources on a chip for durations of nano-seconds to the weekly, monthly or longer-range reactive planning in a factory or a supply chain.

Timed automata provide a key modeling technology for the controlled design and analysis of all sorts of embedded systems. In particular, the state-of-the-art of application of tools for timed automata is very promising, with notable applications to verification (and debugging) of industrial real-time communication protocols and control programs, and applications to sequencing and resource allocation problems. Timed automata also seems to provide an interesting middle-ground between purely finite-state systems and general hybrid systems: there are at present no tools or techniques available for the analysis of systems with general continuous dynamics of more than toy size, whereas there are such tools available if the dynamics are abstracted to timed automata.

Innovative Aspects

We sketch below some of the innovation that timed automata bring to various approaches to time-dependent system analysis.

Time-dependent Behaviors: Toward State-Space Models A lot of the success in discrete verification and in control theory is due to *state-space* based models of their underlying dynamical systems. Verification is based on transition systems models such as automata while control theory is based on continuous dynamical systems where state-variables evolve according to differential equation. Such system models where the values of state variables determine the possible future evolutions have a tremendous, positive effect on the understanding of system dynamics. However, the phenomena that we want to treat cannot benefit from these two classes of models as they are: purely-discrete models are too poor and continuous models are too detailed (for the purpose of solving a scheduling problem there is no use in modeling the process of executing a production step using differential equations). The timed models that we want to use are the ideal candidate for filling this modeling gap. They enrich discrete models with additional state-variables, the clocks, which encode into a state exactly the information necessary to determine the future: each clock represents the time that has elapsed since the occurrence of a certain past event upon which the future depends.

In contrast, many approaches to timing problems, such as those used in operation research or performance modeling, are not always based on such a rich dynamical model, but rather on formulating and solving static optimization problems whose relation with the underlying dynamics is sometimes obscured. Such an approach is sometimes very successful in solving particular problems efficiently, but their ad-hoc nature can prevent their reusability. We strongly believe that if computer-aided timing analysis and design is to become a mature discipline, its approach to problem solving should be based on modeling problems faithfully by a clean

semantic model, and not in terms of the specific technique used to solve them.Such an approach makes the computational difficulty of the original problem explicit and allows much more freedom later in choosing the solution method that gives the best trade-off between its computational complexity and the quality of the solution it provides.

Another advantage of the automaton-based approach is that it enables the user to formulate, in a very natural fashion, distributed systems comprising of small interacting subsystems. In other approaches one does not have such an intuitive notion of communicating sub-systems to solve such problems, but rather a very large number of equations and inequalities in which the dynamical and compositional aspects are not made explicit.

In the context of AI, our feeling is that the underlying dynamic models are sometimes hidden by too much emphasis on logical and syntactic considerations. Syntax is important, especially for the efficient treatment of large interacting sub-systems, but these issues should be considered only after the problems are well understood from a semantical point of view.

Verification: From Untimed to Timed and from Safety to Performance Verification methodology had a lot of success during the last decade due to verification tools that can predict the behaviors of complex discrete systems such as digital circuits and communication protocols. Many models used in this methodology are purely discrete and their treatment of time is purely qualitative, that is, behaviors are just sequences of events appearing one after the other but without any quantitative timing information about the duration of actions and the time between events. Timed models provide for a more detailed level of modeling and incur, because of this, a considerable computational overhead associated with the treatment of clocks. Another dimension of innovation with respect to standard verification is the evaluation of behaviors in terms of quantitative properties of (timed) behavior, such as total elapsed delay time, i.e. a judgment in terms of *performance* rather than the traditional classification into "good" and "bad" behaviors.

Scheduling: Certainty vs. Uncertainty Classical models for scheduling in manufacturing such as the job-shop problem, are somewhat detached from industrial practices. They assume that the duration of every step as well as the arrival times are fixed and known with certainty. In practice, it is rarely the case that a schedule is executed as planned. The problem of coping with uncertainty is identified (by providers of scheduling tools and by their clients) as one of the major problems in the domain. There have been various attempts to model and solve such problems, but no unified approach has emerged. Using non-deterministic timed automata with controlled and uncontrolled transitions (for representing the uncertainty coming from the plant) we can model a large class of such problems, and provide efficient offline algorithms for synthesizing reactive schedulers. Such algorithms can plan for the best, worst or average case, but the scheduling strategies they produce are adaptive and can take advantage, for example, of the fact that a task has terminated before it was expected, and use the empty time slot.

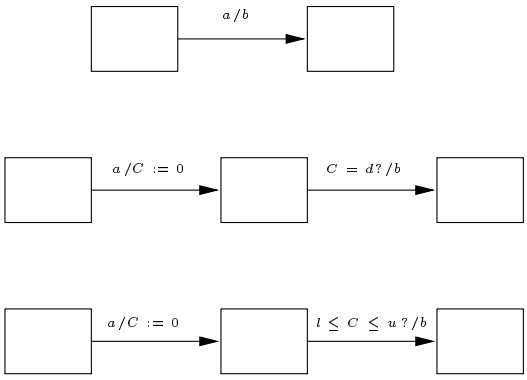


Figure 1: Refining an untimed system description to a timed one. In the untimed automaton a is followed by b, while in the timed automaton the distance between the two events is d. Uncertainty in the duration is modeled using a non-deterministic automaton in which a transition can be taken anywhere in the interval $[l, u]$.

Uncertainty: Quantitative vs. Qualitative Due to historical reasons, most uncertain phenomena in system behavior are treated probabilistically. For example in the theory of queuing systems, it is often assumed that the inter-arrival times and service times of clients are random variables. In this setting, an optimal scheduler is one which optimizes the *expected value* of the performance over all possible behaviors. Under certain assumptions on the nature of the probabilistic processes analytical solutions can be found for such optimization problems. Unfortunately, these assumptions are sometimes very restrictive, and unrealistic for many modern applications. Without them numerical solutions can be devised, but their complexity can be very high, making them often also unsuitable for the treatment of large systems. Timed automata suggest an alternative formulation for temporal uncertainty: instead of specifying a probability distribution on durations only upper- and lower-bounds are given (see Figure 1 for a small example). From these models, policies can be derived which are optimal with respect to optimistic or pessimistic or average estimates, but which are nevertheless guaranteed to function for all cases. Because of the less involved model, we hope that the computational difficulty of deriving such policies can be much smaller than in the probabilistic framework.

Example: Job-Shop Scheduling

Instead of giving formal definition let us illustrate how our approach is used to model and solve the classical job-shop scheduling problem, a generic resource allocation problem in which common resources ("machines") are required at various time points (and for given durations) by different tasks. The goal is to find a way to allocate the resources such that all the tasks terminate as soon as possible while respecting resource constraints.

Consider two machines $\{m_1, m_2\}$ and two jobs $J^1 = (m_1, 4), (m_2, 5)$ and $J^2 = (m_1, 3)$ meaning that the the first

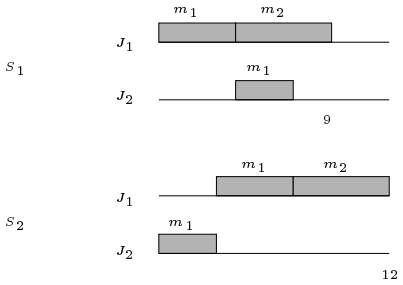


Figure 2: Two schedule S_1 and S_2.

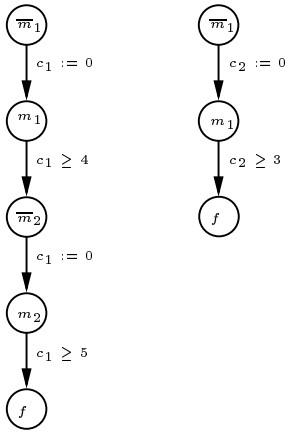


Figure 3: The automata corresponding to the jobs $J^1 = (m_1, 4), (m_2, 5)$ and $J^2 = (m_1, 3)$.

job needs to use m_1 for 4 time units and then machine m_2 for 5 units, while the second job uses machines m_1 for 3 units. A machine cannot be used simultaneously by two jobs. Two possible schedules are depicted in Figure 2. The length of S_1 is 9 and it is the optimal schedule.

Timed automata are automata augmented with continuous clock variables whose values grow uniformly at every state. Clocks are reset to zero at certain transitions and tests on their values are used as pre-conditions for transitions. A run of a timed automaton is an alternating sequence of discrete transition and time periods in which the automaton stays in a state. We model the job descriptions using the two timed automata of Figure 3. Each automaton represents the precedence and duration constraints of a job in isolation. The resource constraints are capture by a composition of the automata that does not allow global states that violate them — in our example state (m_1, m_1) is forbidden. The automaton of Figure 4 represents the whole system and all its runs correspond to feasible schedules.

The two schedules appearing in Figure 2 correspond to the following two runs of the automaton (the 4-tuples correspond to discrete state and clock values and we use notation $\perp$ to indicate inactive clocks):

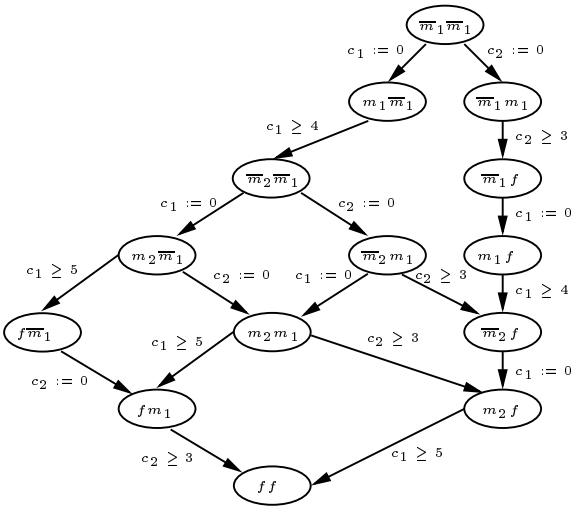


Figure 4: The global timed automaton for the two jobs.

S_1 :
$(\overline{m}_1, \overline{m}_1, \perp, \perp) \xrightarrow{0} (m_1, \overline{m}_1, 0, \perp) \xrightarrow{4} (m_1, \overline{m}_1, 4, \perp)$
$\xrightarrow{0} (\overline{m}_2, \overline{m}_1, \perp, \perp) \xrightarrow{0} (m_2, \overline{m}_1, 0, \perp) \xrightarrow{0} (m_2, m_1, 0, 0)$
$\xrightarrow{3} (m_2, m_1, 3, 3) \xrightarrow{0} (m_2, f, 3, \perp) \xrightarrow{2} (m_2, f, 5, \perp)$
$\xrightarrow{0} (f, f, \perp, \perp)$

S_2 :
$(\overline{m}_1, \overline{m}_1, \perp, \perp) \xrightarrow{0} (\overline{m}_1, m_1, \perp, 0) \xrightarrow{3} (\overline{m}_1, m_1, \perp, 3)$
$\xrightarrow{0} (\overline{m}_1, f, \perp, \perp) \xrightarrow{0} (m_1, f, 0, \perp) \xrightarrow{4} (m_1, f, 4, \perp)$
$\xrightarrow{0} (\overline{m}_2, f, \perp, \perp) \xrightarrow{0} (m_2, f, 0, \perp) \xrightarrow{5} (m_2, f, 5, \perp)$
$\xrightarrow{0} (f, f, \perp, \perp)$

The problem of optimal finding optimal schedules can then be reduced to the problem of finding the shortest (in terms of elapsed time) path in a timed automaton. In (AM01) an efficient implementation has been reported.

There is much more to be said about the computational techniques used for analyzing timed automata, on the existing tools, on abstraction and approximation techniques, etc. Instead we will give a a list of related publications by members of our group.

References

Y. Abdeddaim and O. Maler, Job-Shop Scheduling using Timed Automata in G. Berry, H. Comon and A. Finkel (Eds.), *Proc. CAV'01*, 478-492, LNCS 2102, Springer 2001.

Y. Abdeddaim and O. Maler, Preemptive Job-Shop Scheduling using Stopwatch Automata, *Proc. TACAS'02*, to appear.

K. Altisen, G. Goessler, A. Pnueli, J. Sifakis, S. Tripakis and S. Yovine, A Framework for Scheduler Synthesis. *Proc. RTSS'99*, 154-163, IEEE, 1999.

E. Asarin and O. Maler, As Soon as Possible: Time Optimal Control for Timed Automata, *Proc. HSCC'99*, 19-30, LNCS 1569, Springer, 1999.

E. Asarin, O. Maler and A. Pnueli, Symbolic Controller Synthesis for Discrete and Timed Systems, *Hybrid Systems II*, LNCS 999, Springer, 1995.

E. Asarin, O. Maler, A. Pnueli and J. Sifakis, Controller Synthesis for Timed Automata, *Proc. IFAC Symposium on System Structure and Control*, 469-474, Elsevier, 1998.

M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine, Kronos: a Model-Checking Tool for Real-Time Systems, *Proc. CAV'98*, LNCS 1427, Springer, 1998.

T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, Symbolic Model-checking for Real-time Systems, *Information and Computation* 111, 193–244, 1994.

O. Maler, A. Pnueli and J. Sifakis. On the Synthesis of Discrete Controllers for Timed Systems, *Proc. STACS'95*, 229-242, LNCS 900, Springer, 1995.

P. Niebert, S. Tripakis S. Yovine, Minimum-Time Reachability for Timed Automata, *IEEE Mediteranean Control Conference*, 2000.

P. Niebert and S. Yovine, Computing Optimal Operation Schemes for Chemical Plants in Multi-batch Mode, *Proc. HSCC'2000*, 338-351, LNCS 1790, Springer, 2000.

S. Yovine, Kronos: A Verification Tool for Real-time Systems, *Int. J. of Software Tools for Technology Transfer* 1, 1997.

Temporal Constraints and the Physics v/s Advice Issue from a Practical Perspective

Biplav Srivastava
Email: sbiplav@in.ibm.com
IBM India Research Laboratory
Block 1, IIT Delhi, Hauz Khas, New Delhi 110016, India.

Abstract

A beneficial aspect of the planning competitions has been the fact that it has forced the planning community to question and begin standardization of the input and output semantics of the very problem. A theme that often gets debated during the formalization of any new extension to the PDDL representation is that of *"physics, not advice"*, i.e., the modeling of the domain should be independent of the intention of any agent in it. Though this philosophy is perfectly reasonable from a competition standpoint because it provides no advantage to any specific type of planner, its adherance should be balanced with the reality that planners have to solve real applications where temporal constraints and advice are both present. Moreover, what constitutes physics of a domain may not be always clear and when applied alone, it can lead to very simplistic domain descriptions that are devoid of practical interest.

In recent extensions to PDDL (PDDL2.1), the semantics of durative actions and temporal constraints is quite similar to how advice is used in Hierarchical Task Network (HTN) planning (i.e., schema reductions). Moreover, planning has increasingly been solved as multi-stage sub-problems, where each stage refines the problem at a lower level of abstraction. These characteristics are essentially that of HTN planners which can incorporate both physics and any available advice from the domain. The bias against advice in the planning competitions has had the unintended effect that although HTN planners are widely used in the industry, their techniques have not been evaluated in any of the planning competitions. Consequently, when one wants to use a planner from the competition to solve problems in real applications, it is not clear which planner would be able to use advice better. We argue that the distinction between physics and advice is not as important as the need to conciliate physics with advice in the interest of competition in challenging problems in emerging and interesting domains. We investigate temporal constraints from a practical perspective and make suggestions on how to incoporate temporal advice.

Introduction

The International Planning Competitions (IPCs)(McDermott 2000; Bacchus 2000; Fox & Long 2002) have become a widely anticipated and significant event in the AI horizon. Besides the competitive (performance) element, each IPC has uniquely contributed to better understanding of the planning problem and solution methods - AIPS-98 saw the emergence of graphplan-based algorithms on the forefront while AIPS-00 saw the dominance of heuristic-based algorithms. Another beneficial aspect of the planning competitions has been the fact that it has forced the planning community to question and begin standardization of the input and output semantics of the very problem. A theme that often gets debated during the formalization of any new extension to the PDDL representation is that of *"physics, not advice"*, i.e., the modeling of the domain should be independent of the intention of any agent in it. Though this philosophy is perfectly reasonable from a competition standpoint because it provides no advantage to any specific type of planner, it has had the unintended effect that though Hierarchical Task Network (HTN) planners have been widely used in the industry, their techniques have not been evaluated in any of the planning competitions[1]. HTN planners can incorporate both physics and any available advice from the domain. Consequently, when one wants to use a planner from the competition to solve problems in real applications, it is not clear which planner would be able to use advice better.

The planning competitions should have domains that demonstrate the growing reach of the area. However, what constitutes physics of a domain is not always clear and can lead to a very simplistic domain description that is devoid of practical interest. For example, planning has been considered in data integration to determine the best way to integrate data from different sources(Knoblock1995; Knoblock & Ambite1998; Kwok & Weld1996), and monitor the actual execution of source requests. Traditionally, sources have only been repositories of data but in new domains like bioinformatics(Srivastava 2002), the sources can be applications as well. Planning for query decomposition seems to have lost support in favour of cheaper methods (Levy 1998) like rule inversion(Duschka1996) and view unfolding(Qian1996) when sources are data stores. The main reason is that due to the physics of the domain, the search space is made up of *information* states and there is no subgoal interaction among actions as they can always

[1] SHOP(Nau et al 2000) participated in the hand-coded track of AIPS-00 competition but it was not evaluated for its ability to use advice.

be executed on the sources to gain the information needed. Hence, the conclusion drawn was that using planning for just sequencing source-call actions is an over kill. But when sources can also be applications, they may encode advice (or *policy*) of their interactions which may prevent an action from being always applicable. For example, if a user's authentication request is denied (by a trusted third-party), her previously available credentials (information like password or certificate) to access a source may become invalid. A different kind of advice may come from practicality considerations already known by the users in the domain (domain "wisdom"). For example, there is such a wide disparity in the size of data that is accessed in bioinformatics that some classes of query plans are practically inexecutable. One can encode them into the physics of the domain without upseting any actor in the domain.

Another domain where interaction among applications is key to problem modeling is that of web services composition(McDermott 2002). The scenario here is that an agent or a service needs to determine if another service can fulfill its requirements based on the operations that it exposes to the outside world. There is a tremedous potential for applying planning techniques to handle interactions in these increasingly important domains characterized by their online demand for good timely results.

In the recent extensions to PDDL, the semantics of durative actions and temporal constraints (in PDDL2.1) is quite similar to how advice is used in HTN planning (i.e., schema reductions)(Fox & Long 2002). Moreover, planning has increasingly been solved as multi-stage problems, where each stage is solved at a lower level of abstraction. These trends suggest that the distinction between physics and advice is not as important as the need to conciliate physics with advice for effective problem solving. We are specifically interested in temporal constraints from a practical perspective.

Here is the outline of the paper: we start with a discussion of how temporal constraints get specified through the two main approaches to provide advice in planning - search control knowledge and HTN schemas. Next, we consider the role of advice for feedback in multi-module planning which has addressed resources and temporal constraints. We then argue for adoption of a formal mechanisms in the competition to express temporal advice/semantics to planners independent of the physical dynamics of time used for solving the problem.

Advice in Planning

There are two main mechanisms to incorporate advice in a planning algorithm, namely, as control knowledge or through HTN schemas.

Control Knowledge

This approach requires declarative rules to be specified which are used by the search algorithm to prune newly created search nodes based on some matching criteria. Control knowledge is closely related with the solution search strategy and can either be over syntactic/topographical conditions(Penberthy & Weld 1992; Huang et al 1999) during search or as temporal formula(Bacchus & Kabanza 2000). Temporal formulas can be checked explicitly when the search strategy directly tracks the progress of world during search, e.g., forward chaining, or implicitly through model checking techniques(McMillan 1992).

Although advice is usually expected from the user, there exist automatic pre-processing techniques to detect and incorporate some types of advice, like invariants, e.g., TIM(Fox & Long 1999), DISCOPLAN(Gerevini & Schubert 1998)). These techniques use the domain description and the particular problem being solved to deduce necessary conditions. Planners should be credited for taking less advice from the user.

HTN Schemas

A HTN planning problem(Kambhampati et al 1998) can be seen as a planning problem where in addition to the primitive actions, the domain contains schemas which represent non-primitive (abstract and non-executable) actions and acceptable rules to reduce non-primitive actions to primitive and other non-primitive actions (hence an hierarchy of actions). All non-primitive actions are eventually reduced to primitive actions so that the resultant plan is executable. The acceptable solutions to a HTN problem not only achieve the top-level goals but can also be parsed in terms of the non-primitive actions that are provided for the top-level goals(Barrett & Weld 1994).

Temporal properties like duration of an action have not been explicitly modeled in HTN. However, a schema reduction indirectly encodes temporal ordering/constraints among actions. For example, in Figure 1, an outline of schema reduction is shown for travelling from a source to a destination using either the bus or the train, while hitchhiking is not allowed. The duration of Travel is bounded by:

$$|Travel(source, dest)| \geq min[\ \begin{matrix}|GobyBus(source, dest)|,\\ |GobyTrain(source, dest)|\end{matrix}\ ]$$

$$|Travel(source, dest)| \leq max[\ \begin{matrix}|GobyBus(source, dest)|,\\ |GobyTrain(source, dest)|\end{matrix}\ ]$$

Since GobyBus and GobyTrain are also non-primitive actions, the duration of Travel can be refined to[2]:

$$|Travel(source, dest)| \geq$$
$$min[\ \begin{matrix}|GetinBus| + |Buyticket| + |GetoutBus|,\\ |GetinTrain| + |Buyticket| + |GetoutTrain|\end{matrix}\ ]$$

$$|Travel(source, dest)| \leq$$
$$max[\ \begin{matrix}|GetinBus| + |Buyticket| + |GetoutBus|,\\ |GetinTrain| + |Buyticket| + |GetoutTrain|\end{matrix}\ ]$$

The semantics of a schema is derived from the set of primitive actions that result from the application of reductions. The relative ordering of actions in a schema can be

[2]Parameters of actions may be omitted for clarity.

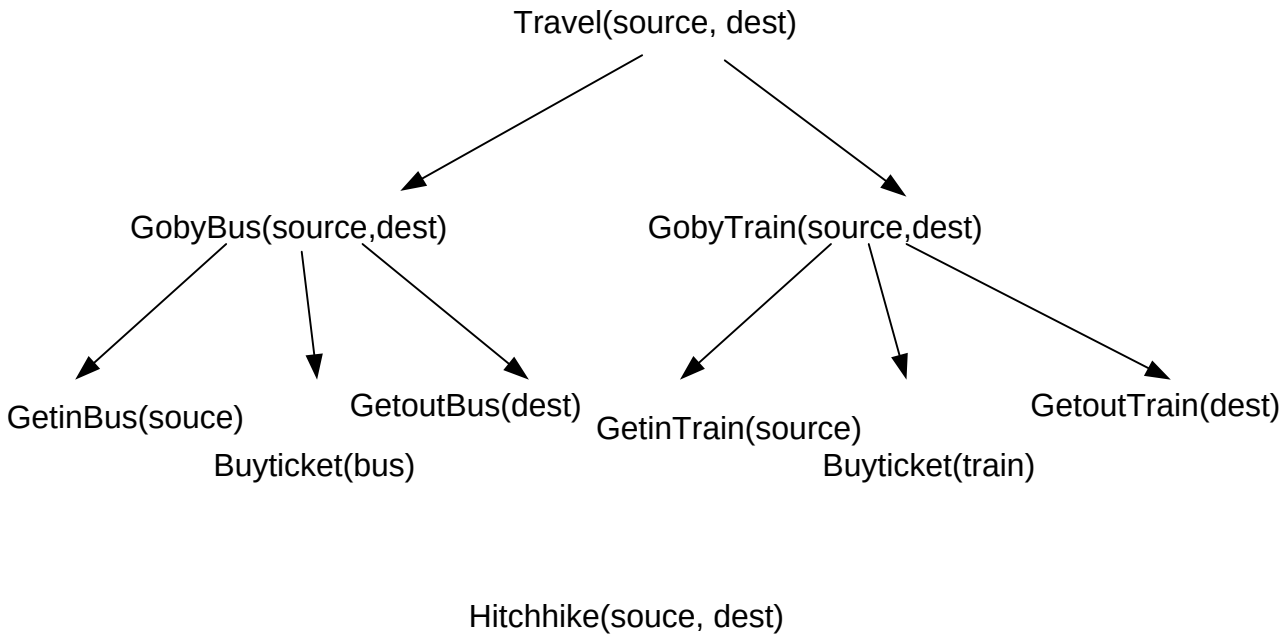


Figure 1: *Schema reduction in a Travel domain (from (Kambhampati et al 1998)). Travel, GobyBus and GobyTrain are non-primitive actions.*

interpreted as temporal advice. If the ordered monotonicity property holds during refinement(Erol 1995), the partial plan and any derived information is sequentially communicated to subsequent refinements until the plan is concretized.

Multi-Module Planning and Advice

A growing trend in planning is to solve it as multi-stage sub-problems, where each stage refines the problem at a lower level of abstraction. With the increasingly complex problems being posed in planning, usually a problem P can be divided into multiple sub-problems p_i that may differ in their characterisitics – causal reasoning or different forms of resource reasoning. For example, the planning problem of having courier packages delivered can be divided into a routing problem of finding the path to be taken and a driver allocation problem. If one were to solve P as a whole, the choice of technique M for it may not be the most appropriate technique for $\forall p_i \in P$. Instead, with a multi-stage approach, the most appropriate module m_i for each p_i is used and they collaborate to solve P.

For example, in STAN4(Fox & Long 2001), an enhanced version of the automatic TIM utility is used to detect if a problem is of a generic transportation problem. In that case, an abstract planning problem is solved where operators and preconditions of the identified sub-problem are removed, and solution is combined with the result of specialized sub-problem solver. In Realplan(Srivastava et al 2001), planning is considered to comprise of causal reasoning and resource reasoning and the latter is handled in a separate scheduling phase. The details about resources is omitted during causal reasoning and an abstract plan is obtained. The scheduling phase concretizes the plan by solving the resource allocation sub-problem with discrete resources as a Constraint Satisfaction Problem (CSP) problem. Moreover, the planner and scheduler can interact either in a master-slave or peer-peer manner repeatedly. The LPSAT planner (Wolfman & Weld 1999) solves planning problems with goals of achievement and metric resources (P). It transforms P into an assignment problem for discrete state variables (p_1) and continuous state variables (p_2) and solve it by satisfiability (m_1) and a simplex-based linear program (LP) solver (m_2).

Multi-module planners mainly need advice on how to detect the sub-problems and how to effectively invoke the necessary modules. In some cases like metric quantities, it is quite clear from the domain description syntax but identification of a generic class of sub-problem or resources is difficult. Durative actions can give early feedback about whether a subsequent module (processing at a lower level of detail) needs to be invoked.

Temporal Advice in Competition

While extending PDDL to handle temporal constraints in PDDL2.1, a fear has been raised that the semantics of detailed modeling of time may serve as advice to the planner(Fox & Long 2002). We saw that the concern is not unfounded because usage of HTN schema reductions, an

advice mechanism, can be interpreted to provide duration bounds for actions. But modeling time is also important because it provides the much needed expressivity to tackle richer and more complex problems in planning.

Going forward, we suggest that planning competition should have:

- Planning domains where only essential temporal advice is defined as part of the domain specification in a standard formalism (e.g., temporal logic). This would allow for richer planning scenarios to be considered in the competition which is more important than the strict abhorance of advice.
- The standard formalism for advice be independent of the nature of the planner. Hence, it could be translated into control knowledge or reduction schemas based on the working of the planner. The requirement is in line with PDDL where the planner decides how it will use the standard domain description.
- Participants stay free to incorporate any automated domain analysis (i.e., automatically generated advice) as before.
- Continuation of the current theme(McDermott 2000) that pre-canned programs cannot be embedded to compete in a plan synthesis competition.

Conclusion

In this paper, we argued that the distinction between physics and advice in practical planning is not as important as the need to conciliate physics with advice in the interest of competition in challenging problems in emerging and interesting domains. We investigated temporal constraints from a practical perspective and outlined how temporal sematics/advice could be incorporated in future planning competitions.

References

Bacchus, F. 2000. AIPS-00 Planning Competition. *At http://www.cs.toronto.edu/aips2000/*

Bacchus, F., and Kabanza, F. 2000. Using Temporal Logics to Express Search Control Knowledge for Planning. *Artificial Intelligence volume 16, pages 123–191.*

Barrett, A., and Weld, D. 1994. Task Decomposition via Plan Parsing. *Proc. AAAI.*

Duschka, O. 1997. Query Optimization Using Local Completeness. *Proc. AAAI.*

Erol, K. 1995. *Hierarchical task network planning: Formalization, Analysis, and Implementation.* Ph.D. thesis, Dept. of Computer Science, Univ. of Maryland, College Park, USA.

Gerevini, A., and Schubert, L. 1998. Inferring State Constraints for Domain-Independent Planning. *Proc. AAAI.*

Fox, M., and Long, D. 2002. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Available at http://www.dur.ac.uk/d.p.long/competition.html.*

Fox, M., and Long, D. 2002. AIPS-02 International Planning Competition. *At http://www.dur.ac.uk/d.p.long/competition.html.*

Fox, M., and Long, D. 2001. AIPS-02 Hybrid STAN: Identifying and Managing Combinatorial Optimisation Subproblems in Planning. *Proc. IJCAI.*

Fox, M. and Long, D. 1999. The Automatic Inference of State Invariants in TIM. *Journal of AI Research, Volume 9, pages 367-421.*

Huang, Y., Selman, B., and Kautz, H. 1999. Control Knowledge in Planning: Benefits and Tradeoffs. *Proc. AAAI.*

Kambhampati, S., Mali, A., and Srivastava, B. 1998. Hybrid planning for partially hierarchical domains. *Proc. AAAI.*

Kambhampati, S., and Srivastava, B. 1995. Universal Classical Planner: An algorithm for unifying state space and plan space approaches. *In New Trend in AI Planning: EWSP 95, IOS Press.*

Knoblock, C. 1995. Planning, Executing, Sensing and Replanning for Information Integration. *Proc. IJCAI-95.*

Knoblock, C., and Ambite, J. 1997. Agents for Information Gathering. *Software Agents, J. Bradshaw (ed), AAAI/MIT Press, Menlo Park, CA; Also at http://citeseer.nj.nec.com/169819.html.*

Kwok, C., and Weld, D. 1996. Planning to Gather Information. *13th AAAI National Conf. on Artificial Intelligence, AAAI/ MIT Press, Portland, Oregon, 32–39.*

Levy, A. 1998. Combining Artificial Intelligence and Databases for Data Integration. *At http://citeseer.nj.nec.com*

McDermott, D. 2000. The 1998 AI Planning Competition. *AI Magazine, 21(2).*

McDermott, D. 2002. Estimated-Regression Planning for Interactions with Web Services. *Proc. AIPS.*

McMillan, M. 1992. Symbolic Model Checking. *Ph.D. Dissertation, Computer Sc. Dept., CMU.*

Nau, D., Cao, Y., Lotem, A., and Muoz-Avila, H. 1995. SHOP and M-SHOP: Planning with Ordered Task Decomposition. *Tech. Report CS TR 4157, University of Maryland, College Park, MD, June, 2000.*

Penberthy, J. S. and Weld, D. 1992. UCPOP: A Sound, Complete, Partial-Order Planner for ADL. *Third International Conference on Knowledge Representation and Reasoning (KR-92), Cambridge, MA.*

Qian, X. 1996. Query Folding. *12th Int. Conference on Data Engineering, New Orleans, Louisiana, 48–55.*

Srivastava, B. 2002. Using Planning for Query Decomposition in Bioinformatics *Sixth Intl. Conf. on AI Planning & Scheduling (AIPS-02) Workshop on "Is There Life Beyond Operator Sequencing? – Exploring Real World Planning".*

Srivastava, B, Kambhampati. S. and Do, M. 2001. Planning the Project Management Way: Efficient Planning by Effective Integration of Causal and Resource Reasoning in RealPlan. *Artificial Intelligence, Vol. 131 (1-2) (2001) pp. 73-134.*

Wolfman, S., and Weld, D. *The* LPSAT *Engine and its Application to Resource Planning.* Proc. IJCAI-99. 1999.

TimeLine: An HTN planner that can reason about time

Fusun Yaman **Dana S. Nau**

Department of Computer Science
University of Maryland
College Park, Maryland 20742
{fusun, nau}@cs.umd.edu

Abstract

In this paper we present a formalism for explicitly representing time in HTN planning. Actions can have durations and intermediate effects in this formalism. Methods can specify qualitative and quantitative temporal constraints on decompositions. Based on this formalism we defined a planning algorithm TimeLine that can produce concurrently executable plans in the presence of numeric state variables. We state and prove the soundness of the algorithm. We also present the experimental results of the TimeLine implementation that shows the feasibility of our approach.

Introduction

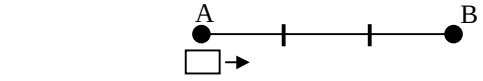
Actions with different durations, simultaneous action execution and reasoning with metric quantities are three characteristic of real-world planning problems. Recently studies on artificial intelligence planning concentrated on developing formalisms for representing time and creating temporal plans. The planning domain definition language (PDDL 2.1) for AIPS 2002 planning competition can define actions with durations, and address the concurrency issues in the presence of numeric state variables.

The difficulty aroused with concurrency is to control the overlapping action executions. The problem gets more complicated when there are limited number of shared resources. When resources are identified and resource needs for every action are explicitly defined, then two actions with conflicting resource requirements can be defined as mutually exclusive. In this approach the search space can be pruned effectively if it's accompanied by good resource management techniques. The more general case is when there are numeric state variables that can be updated concurrently. Numeric state variables can be used to represent resources but not every numeric variable can be seen as a resource.

Numeric computations and time can be handled easily by HTN planners. For this reason HTN planners are conveniently used for practical applications. In this paper we present a formalism for explicitly representing time in HTN planning. Actions can have durations and intermediate effects in this formalism. Methods can specify qualitative and quantitative temporal constraints on decompositions. Based on this formalism we defined a planning algorithm that can produce concurrently executable plans in the presence of numeric state variables. We state and prove the soundness of the algorithm. We also present the experimental results of the implementation that shows the feasibility of our approach.

Formalism

Performing numerical computations is an important issue for real-world problems. Some HTN planners like SHOP have already incorporated this functionality. Resources generally represent some features in the domain that are limited in number, like space available in a truck. Even though numeric state variables can be used to represent these resources, the opposite need not to be true. For example, let's say the distance between two cities A and B is 6 units and there is a truck T that has a speed of 2 units per unit time. If T is at A and will travel to B then as T moves, the distance between A and the current location of T increases (see dist(A,T) in Figure 1). Similarly travel time left to B is a numeric variable (see timeTo (T,B) in Figure 1). We believe these two numeric variables do not represent any resources. Therefore, instead of identifying the resources and defining operations on these resource, we will go with the more general way and define concurrent update rules for numeric state variables.



Time	0	1	2	3
Dist(A,T)	0	2	4	6
TimeTo(T,B)	3	2	1	0

Figure 1 Dist(A,T) is distance between A and current location of truck T, TimeTo(T,B) is time left to reach B

The value of a numeric variable can be assigned to a constant, decreased or increased by constant amount. We

define assignment operations on the same variable at the same time, as mutually exclusive updates. Therefore we don't allow two assignment operations on the same variable at the same time, even though the assigned values are same. Concurrent increase and decrease operations on the same numeric variable can be permitted as long as the value of the variable stays in the defined range in all intermediate states produced by any permutation of these operations. Since addition and subtraction are commutative operations, the result of the any permutation will be the same. To ensure that the value always stays in the range, it is enough to check pessimistic cases in which all increase or all decrease operations are performed first.

State, Agenda, Operator

Definition 1: State is a collection of positive ground atoms of the form *(p t_1 t_2 ..t_n)* where *p* is the predicate name and t_1 to t_n are argument terms. Value of a numeric state variable is represented by an atom of the form *(= variable value)* where *variable* is the numeric state variable and *value* is the value of the *variable* in this state. A valid state can not contain both (= *variable value1*) and (= *variable value2*).

Main elements of HTN planning are simple tasks and composite tasks. Operators define a set of changes in the current state in order to achieve simple tasks. Composite tasks can be achieved by decomposing them into subtasks and then achieving these subtasks. Methods define decompositions for composite tasks.

Classical HTN operators have a precondition to hold in the state just before the operator is applied. Operators have effects which will be true in the next state of the world. We extend this definition to represent operators that may have a duration of more than one unit time. We do not require the precondition of an operator to hold through out the execution. Effects of an operator can not change the state in which it's precondition is evaluated. We eliminated instant effects because they make it hard to trace the deleted precondition interactions. This way we always evaluate the preconditions in a stable state. We let the operators have effects at intermediate time points, not only at the end so the operators may represent gradual changes in the successor states. *Effects* are the promises that will be true in a successor state. Effects may assign a value to a numeric state variable, increase or decrease the value of a numeric state variable, add or delete an atom in or from the state.

Definition 2: An operator has the following form

(:operator head precondition effect-list)

where head is a simple task, precondition is a conjunctive expression and effect-list is a list of timed effects. Timed effects can be in one of the following forms:

([$time_1$] (e_1….e_n)) or *([$time_2$,$time_3$] (e_1….e_n))*

where e_i's are effects and the intended meaning of first form is e_i's will be true at (*start time of operator* + $time_1$). The intended meaning of the second form is e_i's will be true in the states associated with inclusive time interval [*start time* + $time_2$, *start time* + $time_1$]. In this notation $time_1$, $time_2$ and $time_3$ should be positive integers or numeric expressions. If the result of the numeric expression is not an integer we take the ceiling of the result. More over $time_3$ should be greater than or equal to $time_2$.

```
(:operator (!drive ?truck ?loc-from ?loc-to)
;;PRECONDITIONS
 ( (not (moving ?truck ?dest))
   (= (truck-user ?truck) ?user)
   (call = ?user 0)
   (truck-at ?truck ?loc-from)
   (distance ?loc-from ?loc-to ?dist)
   (assign ?duration (call ceil (call / ?dist 2))
  ))
 ;;EFFECTS
 (([1]
   ((=(truck-user ?truck) 1)
    (=(truck-arrives ?truck ?loc-to)
                          (call - ?duration 1))
    (moving ?truck ?loc-to)
    (not (truck-at ?truck ?loc-from))))
 ([2,?duration]
    ((-= (truck-arrives ?truck ?loc-to) 1)))
 ([?duration ]
    ((= (truck-user ?truck) 0)
     (not (moving ?truck ?loc-to))
     (truck-at ?truck ?loc-to))))
)
```

Figure 2 Drive operator for extended logistic domain

Figure 2 shows *drive* operator we defined for logistic domain in which we added some numeric state variables. The precondition of the operator states that number of users that are working on this truck should be zero and truck should not be in motion. *Assign* statement in the precondition simply binds the value of its second term to its first term. In this case ?duration is assigned to travel time between ?loc-from and ?loc-to when the truck speed is 2. One unit time later the state and current location of the truck is updated also the number of truck users for this truck is set to 1 and a counter that shows the time left to arrive ?loc-to is initialized. After that at every clock tick this counter is decreased by one. Finally the state and the location of the truck is updated . We also decrease the number of users for this truck. One thing to notice is we assign the value of (truck-user ?truck) to one at the beginning instead of increasing it by one. That is because we want two overlapping drive operations on same truck to be mutually exclusive.

Definition 3: Two effects *e1* and *e2* are mutually exclusive if any of the following holds:

- if e_1 and e_2 are logical negations of each other
- if e_1 assigns a value to a numeric state variable *v* and e_2 assigns or increases or decreases the value of *v*
- if e_1 decreases or increases the value of a numeric state variable *v* and e_2 assigns a value to *v*.

Since we have delayed effects that may appear sometime in the future we need a structure that remembers all of the promises toward future states.

Definition 4: Agenda is a collection of pairs *(t, e)* where *e* is an effect and *t* is the time when *e* will be true in the state.

If in an agenda A every t is grater than T then A is an agenda *after T*.

An effect e_1 that is promised to be true at time T is *consistent* with an agenda A iff A does not contain an effect e_2 that is promised to be true at time T and e_2 is mutually exclusive with e_1.

Figure 3 shows the load operator we defined for our extended logistic domain. There may be a state that satisfies the preconditions of both *load* and *drive* operators. It is obvious that for a truck these two operators should not overlap on the timeline. To handle this case we should extend the definition of applicability for an operator, to include consistency with the current agenda. Therefore if a *drive* operator on a truck is scheduled at time *T*, *load* operator on the same truck should not be applicable. We should allow concurrent *load* operators unless the cumulative numerical effects of these load operators lead to an out of range value for a numeric state variable in the future states. It is not always possible to detect these inconsistencies by looking at the agenda, since there may be additions to agenda and that can fix what seems to be a problem. However it is easy to check for the state just after the current one. Let's say in the current state truck B has 5 units of space available. If we schedule 6 concurrent loading into B now, we can immediately see that (given that all packages have positive sizes) one unit time later we will ran out of space. No unload operation can fix this problem because the value should always stay in the range no matter in what order these effects are carried on.

```
(:operator
 (!load ?obj ?truck ?loc)
  ( (not (moving ?truck ?dest))
    (obj-at ?obj ?loc)
    (truck-at ?truck ?loc)
    (= (truck-space ?truck) ?space)
    (volume ?obj ?vol)
    (call >= ?space ?vol) )
  (([1]((+=(truck-user ?truck) 1)
        (-=(truck-space ?truck) ?vol)
        (not (obj-at ?obj ?loc))))
  ([1,2]((truck-at ?truck ?loc)))
  ([2]((-= (truck-user ?truck) 1)
       (in-truck ?obj ?truck)))))
```

Figure 3 Load operator for extended logistic domain

Definition 5: Let S be the state for time T, A be an agenda after T, t_1 be a simple task and O be an operator. Let *mgu* be the most general unifier that unifies with the head of O and t_1. Then O^{mgu} is an *applicable operator instance* for t_1 at time T in state S with agenda A iff the following holds

- There is a satisfier α for the precondition of O^{mgu} in S.
- None of effects in effect-list of $(O^{mgu})^{\alpha}$ with same time is mutually exclusive with each other.
- All of the effects of $(O^{mgu})^{\alpha}$ are consistent with A
- All the numeric variables stay in the range at time $T+1$.

Let S be the state at time T and A be the agenda in which all of the effects are after T. A simple task t_1 is *T-executable* if there exists an applicable operator instance for t_1 in S with A.

The purpose of agenda is to keep track of changes that will be made to future states. Given the current time, state and the current agenda successor states can be generated by performing the effects in the temporal order.

Definition 6: Let S be the state at time T and A be an agenda after T. Let $e_1..e_n$ be the effects in A that are promised for time T' where T' is T+1 then *Exec(A,S,T')* creates a new state S' in and a new agenda A' with the following properties:

- Let p be *(= numeric-variable ex-value)* and S contains p. If there is an effect e_i that assigns value *new-value* to *numeric-variable* then S' does not contain p and S' contains (= *numeric-variable new-value*)
- Let p be *(= numeric-variable ex-value)* and S contains p. Let E^+*(numeric-variable)* be subset of e_1 to e_n such that every $e_i \in E^+$*(numeric-variable)* increases the value of numeric-variable. E^-*(numeric-variable)* is defined similarly for the effects that decrease the value of *numeric variable*. *Total increase* is sum of the increase amounts of effects in E^+. *Total decrease* is sum of the decrease amounts of the effects in E^-. Then S' does not contain p and S' contains (= numeric-variable new-value) where *new-value* is equal to *ex-value + total increase – total decrease*. If *ex-value + total increase* or *ex-value– total decrease* is not in the range defined for *numeric-variable* then S' is an invalid state.
- Let p be an atom of the form *(p-name* arg_1 arg_2 .. arg_n *)* and *p-name* is not +=, -= or =. If p is in S and there is an e_i such that e_i is *(not p)* then S' does not contain p, if there is no such e_i S' contains p. If there is an ei such that ei is p then S' contains p.
- A' is same as A except A' does not contain effects e_1 .. e_n.

Time Constraint, Task network, Method

End points of a task t are the start and end times of t which we represent as *(start t)* and *(end t)* respectively. End time of a simple task is the time of last effect in the operator instance that is chosen to achieve that task. Therefore once the operator is chosen, end time of the simple task is known. End time of a composite task is maximum of the end times of the subtasks in the decomposition of the method chosen to achieve this composite task. End time of a composite task becomes known when all of its subtasks end times are known. We use the end points of tasks to define temporal constraints. We concentrated on constraining the start time of tasks explicitly. We can define both metric and qualitative constraints. For example if t1 and t2 are two tasks the following are the time constraints on start time of t1; (start t1) ≥ ((end t2) + 5) or (start t1) = ((start t2) + 4) or (start t1) ≥ (start t2) While we are handling the general cases, there are some combinations that we don not allow in our time constraint definition. For example if t1 and t2 are two tasks we do not allow the following (start t1) < (end t2) or (start t1) ≥ ((start t2) - 3) or (start t1) = (end t2 - 5). All of these constraints require t1 to start before some time that we do not know in advance and by the time these points become

known these constraints are either satisfied or not and there is no time point after that that can satisfy these constraints. We find these constraints hard to trace therefore do not make use of them. For similar reasons we do not define constraints on end time of tasks for example (end t1) ≥ (end t2).

Definition 7: Given a task t_1 and another task t_2 , a time constraint on start time of t_1 is one of the following expressions:

- *(= (start t_1) bound)* where *bound* can be either *(start t_2)* or (end t_2) or "*now*", which means current time, or a nonnegative integer *c* or (+ base delay) in which *delay* is a nonnegative integer and *base* is either *(start t_2)* or *(end t_2)* or "*now*".
- *(>= (start t_1) bound)* where bound is as defined above
- *(>= (start t_1) (max $bound_1$ $bound_2$ .. $boundk$)* where $bound_i$ is as defined above. This is a short hand notation for a list of time constraints of (>= (start t_1) $bound_i$) where *i* is in [1,k]

Time constraints of the first type are called equality constraints where as second and third types are called greater than constraints. The following time constraints are satisfied at time T

- (= (start t_1) bound) where *bound* is either "*now*" and current time is *T* or a nonnegative integer *c* that is equal to *T*.
- (>= (start t_1) bound) where *bound* is either "*now*" and current time is *T* or a nonnegative integer *c* that is less than or equal to *T*.
- (>= (start t_1) (max $bound_1$ $bound_2$ ..$bound_k$) where for every *i*, $bound_i$ is nonnegative integer that is less than or equal to *T*.

Given a set of time constraints *U* on start time of task t_1, *U* is satisfied at time *T* if all of the following holds;

- *U* contains at most one equality constraint *C*, and C is satisfied at T.
- All of the greater than constraints in U are satisfied at T.

Definition 8: Task Network is a list of tasks *(t_1..t_n)* and a list of time constraints on the start time of these tasks such that end points of a task referenced in a time constraint is in the list of tasks t_1..t_n.

Definition 9: A method has the following form

(:method head precondition subtasks)

where *head* is a composite task, *precondition* is a conjunctive expression and *subtasks* is a task network.

Figure 4 includes two, methods we defined for our extended logistic domain. First method decomposes *air-deliver* task into two subtasks which are labeled as t_1 and t_2. According to the time constraints t_1 should start immediately and t_2 can start after t_1 ends. The second method decomposes the task unload-airplane-at into two subtasks. There is no time constraint for t_3 but t_4 should start when t_3 ends.

```
(:method (air-deliver ?obj ?airport-from ?city)
 ;;PRECONDITION
 ((obj-at ?obj ?airport-from)
  (in-city ?dest ?city)
  (airport ?dest)
  (= (airplane-space ?plane) ?space)
  (volume ?obj ?vol)
  (call >= ?space ?vol))
;; SUBTASKS
 ((:t1 (load-airplane ?obj ?plane ?airport-from)
   :t2 (unload-airplane-at ?obj ?plane ?dest))
  ((= (start t1) now) (>= (start t2) (end t1))))
 )

(:method load-airplane ?obj ?plane ?airport)
;;PRECONDITION
((not (moving ?plane ?dest))
 (airplane-at ?plane ?somewhere)
 (different ?somewhere ?airport)
 (= (airplane-user ?plane) ?user)
 (call < ?user 1)
 )
;; SUBTASKS
 ((:t3 (!fly-airplane ?plane ?somewhere ?airport)
   :t4 (!load-airplane ?obj ?plane ?airport ))
  ((= (start t4) (end t3) ) ) )
)
```

Figure 4 Method for air-deliver task in extended logistic domain

Definition 10: Let *S* be the state at time T, *t* be a composite task and *M* be a method. Let *mgu* be the most general unifier that unifies the head of *M* and *t*. Then M^{mgu} is an *applicable* method instance for *t* in state *S* at time *T* if the precondition of M^{mgu} is satisfied in *S*. If α is a list of bindings for the free variables in precondition of M^{mgu} such that precondition of M^{mgu} is satisfied then $(subtasks^{mgu})^{\alpha}$ is a *reduction* of *t* at time *T*.

The idea behind reducing a task network is to replace a task in the network with one of its reductions and update all the time constraints that refer to the old task to include references to new tasks. Figure 5 gives an example task network *R* which is reduced to *R'* using the method for *load-airplane* task defined in Figure 4.

```
R =((:t1 (load-airplane package plane airport1)
    :t2 (unload-airplane package plane airport2))
    ((= (start t1) now)(>= (start t2) (end t1))))
    )

r =((:t3 (!fly-airplane plane airport3 airport1)
     :t4 (!load-airplane package plane airport1))
    ((= (start t4) (end t3) ) ) )

R'=((:t2(unload-airplane package plane airport2))
     :t3 (!fly-airplane plane airport3 airport1)
     :t4 (!load-airplane package plane airport1))
    ((>= (start t2) (max (end t3) (end t4)))
     (= (start t4) (end t3) ) ) )
```

Figure 5 Example for reducing a task network

Definition: Let *R* be a task network and *t* be a task in *R*. Let *r* be a task network with tasks t_1..t_n then *reduce(R, t, r, $time_{start}$, $time_{end}$)* is a new task network *R'* satisfying the following:

- *R'* contains all tasks in *r* and all tasks in *R* except *t*
- *R'* contains all time constraints in *r* and in *R* except the constraints that are on start time of *t* and those refer to end points of *t*.

- If *C* is a time constraint in *R* and *C* refers to *(start t)*, then *R'* contains a constraint *C'* such that *C'* is same as *C* except *(start t)* is replaced with $time_{start}$.
- If *C* is a time constraint in R and C refers to *(end t1)* then *R'* contains a constraint *C'* such that *C'* is same as *C* except *(end t1)* is replaced by $time_{end}$.

When we are talking about a simple task we can easily point the start and end time of it. Basically the time when matching operator is applied is its starting time and the time for the last effect of that operator is the end time. However this can not be directly applied to composite tasks. What happens if at time *T* a composite task *t* is decomposed into n subtasks using an applicable method and none of its subtasks start at time *T*. In such a case it is does not make sense to say that starting time for *t* is *T*. On the other hand if at least one of its subtasks can start at *T* then we can safely say that starting time of *t* is *T*. This leads to the definition of T-executable reduction.

Definition : Let S be the state at time T and A be an agenda after T. Let *t* be a composite task, *r* be a reduction of *t* at time *T* and t_i be a task in *r* such that time constraints on start time of t_i are satisfied at time *T*, then T-executable reduction *R* for *t* is defined as:

- If t_i is a simple task and it is T-executable in S with A then R is equal to r and R has an additional time constraint *C* (= (start t_i) T) if r does not contain *C*.
- If t_i is a composite task and it is R' is a T-executable reduction of t_i that with tasks $t_{i1}..t_{in}$ then R is equal to reduce(r ,t_i, R', $time_{start}$, $time_{end}$) where $time_{start}$ is equal to T and $time_{end}$ is equal to (max (end t_{i1}) (end t_{i2}) .. (end t_{in})).

Let T be the time and S be the state at T and A be an agenda after T. A composite task *t* is T-executable at time T in state S with A if there exists a t-executable reduction for *t*.

Plan and Planning Problem

We now define what is a plan, a planning problem and what is a solution to the planning problem.

Definition : Let T be the time, S be the state at T and A is an agenda after T. The effect of achieving a *progress task* at time T is defined as Exec(A,S,T+1)

Definition : A plan is a list of ($task_1$ [$time_{start}$, $time_{end}$]) where $task_1$ is a ground simple tasks and $time_{start}$ and $time_{end}$ are the start and end times of $task_1$.

Definition : A planning problem P is a tuple (N,A,S,T) where N is a task network, A is an agenda after T, S is a state and T is the current time.

Definition: Let P be a planning problem (N,A,S,T) then a solution Π of problem P is defined as follows:

- *Case 1*: If both N and A are empty then Π is an empty list.
- *Case 2*: If there exists a task t_i in N such that the time constraints on start time of t_i are satisfied at T and there exists an equality constraint for start time of t_i and
  - *Case 2.1*: ti is a simple task. Let O be an applicable operator instance of ti and $time_{end}$ be the time of latest effect in *O*. Let A' and N' be defined as:
    A' = A ∪ effects of *O*
    N' = reduce(N ,ti, empty task network, T, $time_{end}$)
    Let Π' be the solution to the problem (N',A', S, T) then Π = (ti [T, $time_{end}$]) + Π'
  - *Case 2.2*: ti is a composite task. Let R be an T-executable reduction of ti and ti1 .. tin are the tasks in R. Let $time_{end}$ be(max (end t_{i1}) (end t_{i2}) .. (end t_{in})) Let N' be reduce(N ,ti, R, T, $time_{end}$) then Π is the solution to the problem (N',A, S, T).
- *Case 3*: If A is empty and there is no task t in N such that the time constraints on start time of ti are satisfied at T'>T. Then let ti be a task in N such that t_i's time constraints are satisfied at T.
  - *Case 3.1*: ti is a simple task. Then Π is defined as in *Case 2.1*
  - *Case 3.2*: ti is a composite task. Then Π is defined as in *Case 2.2*
- *Case 4*: Let ti be the progress task or a task in N such that time constraints on its start time are satisfied. Then
  - *Case 4.1*: ti is progress task. Then let S' and A' be defined as
    (S',A') = Exec(S,A,T+1)
    If S' is a valid state then Π is a solution to problem (N,A',S',T+1);
- *Case 4.2*: ti is a simple task. Then Π is defined as in *Case 2.1*
- *Case 4.3*: ti is a composite task. Then Π is defined as in *Case 2.2*

Algorithm

We now define the TimeLine algorithm that finds a solution to the planning problem (N,A,S,T) as defined in pervious section. The pseudo-code for the algorithm is presented in Figure 6. TimeLine is non-deterministic straight forward implementation of the solution defined for a planning problem in the previous section.

```
TimeLine (N, A, S, T)

1 If N and A are empty
2   return  empty  plan;
3 else
4 if there is a task t such that time constraints
  on start time of t is satisfied at T
5    If t is a simple task
6     Choose applicable operator instance o for t
7     end_time = time of the latest effect in o's
                 effects
8     A'=A ∪ effects of o
9     N'=reduce(N,t,empty tasknetwork,T,end_time)
10    Π' = TimeLine(N',A',S,T)
11    return  (t [T, end_time])+ Π'
12   else if ti is a composite task
13    Choose a T-executable reduction R for t,
      R contains subtasks t_1 to t_n
14    end_time=(max(end t_1)(end t_2)..(end t_n))
15     N' = reduce(N, t, R, T, end-time)
16    return TimeLine(N',A,S,T)
```

```
17   end if
18 else
19 if A is empty and there is not a task t
   such that time constraints on start time of t
   is satisfied at T' > T
19    Chose a task t such that time constraint on
       starting time of t is satisfied at T.
20     Go to line 5
21 else
22    Choose ti be the progress task or a task in
      N such that time constraints on its start
      time are satisfied.
23    if ti is the progress task
24       (S',A') = Exec(S,A,T+1)
25       if S' is a valid state
25          return TimeLine(N,A',S',T+1)
26       end if
27    else
28       Go to line 5
29    end if
30 end if
31  End TimeLine
```

Figure 6 Pseudo code of TimeLine algorithm

Theorem 1: If one of the non-deterministic traces of TimeLine(N,A,S,T) returns a solution Π, then Π is a solution to the planning problem (N,A,S,T).

Proof : TimeLine returns a solution at

- *Line 2*: This line is executed when both N and A are empty and returned solution is an empty plan. This is Case 1 of the definition for solution planning problem.
- *Line 11*: If line 5 is executed after line 4 this case corresponds to Case 2.1 of solution definition. If line 5 is executed after line 20 this case corresponds to Case 3.1 of solution definition. If line 5 is executed after line 28 this case corresponds to Case 4.2 of solution definition);
- *Line 16*: If line 12 is executed after line 4 this case corresponds to Case 2.2 of solution definition. If line 12 is executed after line 20 this case corresponds to Case 3.2 of solution definition. If line 12 is executed after line 28 this case corresponds to Case 4.3 of solution definition;
- *Line 25*: The solution returned in this line corresponds to Case 4.1 of solution definition. □

Implementation and experiments

We have implemented deterministic version of TimeLine algorithm. We have tested our implementation on the Logistic domain which is naturally concurrent and easily extendible to include numeric state variables.

The problems we used in our experiments were based on 30 problems used in AIPS98 planning competition. Basically we tried to create same set up that is defined in (Kvarnstrom, J. and Doherty, P. 2001). We set the space capacity for trucks to 5 and for airplanes to 25. We randomly generated the package sizes between 1 and 3. We also randomly defined distance between two locations in the range 1 to 25. We have create 20 random instanced for each of the 30 problems. We ran TimeLine on 20 problems instances then take the average and we did this for 30 problems.

We ran the experiments on a Pentium III-600 machine with 128 memory and Windows 98 operating system running on it. We compared our results with the published results of TAL planner and verified the feasibility of our approach. In fact most of the cases TimeLine performed better than TAL planner. Considering the configuration differences between the machines (TAL planner experiments performed on Pentium II-333) and the problems, this performance difference may not illustrate a great deal. As we can see from the preliminary results our approach is feasible and worth for future study.

No	TaL Planner	TimeLine		No	TaL Planner	TimeLine
1	270	591		16	10004	1455
2	811	644		17	2895	1248
3	2063	1165		18	21080	5047
4	5889	1412		19	18466	4649
5	541	601		20	37815	6317
6	6729	1877		21	39436	3224
7	1061	643		22	71402	20107
8	5658	1163		23	2434	3354
9	9594	2303		24	39096	1455
10	5738	3731		25	146921	10559
11	911	646		26	83960	22369
12	14871	1084		27	72814	9388
13	16524	2016		28	670284	24974
14	6800	2218		29	34550	21261
15	1512	3850		30	312099	9023

Table 1 Planning time in milliseconds for 30 extended logistic problems**.**

Related Work

Allen's interval algebra defines the relations on two tasks using their end points. Our approach does not define explicitly any constraints on end time of a task, so it can not express all of the relations in Allen's algebra. On the other hand we believe our approach is easier to track and expressive enough for many practical problems.

Dechter and Meiri can reason about metric time constraints and propose an algorithm that can solve simple temporal constraint satisfaction problems in polynomial time.

Bacchus suggested a simplistic approach to generate plans that include actions with same time stamp but in fact the plans are sequential because the ordering of the actions with same time stamp is important. The actions in this approach can have instant effects that are used to control concurrency and delayed effects to represent the actions that have a duration greater than one.

TaL planner is extended to reason about time, concurrency and resources. It prunes the search space using control rules written in temporal logic. They define

two actions as mutually exclusive if the effects of them conflict at some point that these two actions overlap. Tal planner can perform concurrent numeric computation only on resources.

Smith and Weld extends the definition of mutual exclusion for actions that can have durations. TGP uses a more generalized planning graph that can handle actions with durations and employs the extended mutual exclusion reasoning when searching for a plan. TGP actions have preconditions that hold through out the execution and effects that are guarantied to be true at the end of action. TGP does not allow intermediate effects. One can argue that preconditions holding during the actions may be too restricted.

Conclusion

In this paper we have presented a formalism to explicitly represent time in HTN planning. Based on this formalism we defined TimeLine a sound and complete HTN planner that can reason about time. Our experiments concluded that our approach is feasible and worth future study.

The formalism we present is expressive enough to represent most of the practical problems and yet still not complex. We do not require the specification of any resource usage in any level of the task abstraction. Instead we define concurrent update rules for numeric state variables that can represent these recourses.

A future study may concentrate on reducing the backtracking points in the implementation. Number of backtracking becomes a real problem as the problem size and concurrency level increases. A better implementation may be backtracking to representative time points instead of backtracking all time points.

Acknowledgements

This work was supported in part by the following grants, contracts, and awards: Air Force Research Laboratory F306029910013 and F30602-00-2-0505, Army Research Laboratory DAAL0197K0135, and the University of Maryland General Research Board. Opinions expressed in this paper are those of authors and do not necessarily reflect opinion of the funders.

References

Allen,J. 1983. Maintaining knowledge about temporal intervals. Communication of the ACM 26(11):832-843

Simith D.,Weld, D. 1999. Temporal Planning with Mutual exclusion reasoning. (IJCAI-99)

Kohler,J. 1998. Planning under resource constraints.(Proc. ECAI-98)

Karlsson, L..Gustavfsson, J., Doherty, P. 1998. Delayed effects of Actions .(ECAI-98)

Fox, M.,Long, D. 2001. PDDL2.1: An Extension to PDDL for expressing Temporal Planning Domains.

Kvarnstrom, J., Doherty, P. 2001. TAL planner: A Temporal Logic-based Forward Chaining Planner. Annals of Mathematics and Artificial Intelligence.

Erol, K., Hendler, J., Nau, D. 1994. Semantics for Hierarchical Task-Network Planning. Tech. Report CS TR-3239, UMIACS TR-94-31, ISR-TR-95-9, University of Maryland, March, 1994a.

Erol, K., Hendler, J., Nau, D. 1994. UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning. In *Proc. Second International Conf. on AI Planning Systems (AIPS-94*), June, 1994b, pages 249-254.

Munoz-Avila, H., Aha, D., Breslow L., Nau, D. HICAP: an interactive case-based planning architecture and its application to noncombatant evacuation operations. In IAAI-99,1999, pages 870-875.

Dechter, R,;Meiri, I.; and Pearl, J. 1991.Temporal constraint networks. Artificial Intelligence 49:61-95

Koehler, J. 1998. Planning under resource constraints. (ECAI -98)

Rintanen, J., Jungholt, H. Numeric state variables in constraint-based planning.

One Step on the Left, One Step on the Right and Back to the Middle : Exploring Temporal Domains in a POP Fashion

Romain Trinquart, Solange Lemai and Stephane Cambon
LAAS-CNRS
7 Av. du Colonel Roche
Toulouse 31000 France
{lastname}@laas.fr

Abstract

In this paper we present recent advances in the context of a framework for planning in temporal domains, namely the IxTeT system. This system is a lifted partial order planner relying on a functional and CSP-based representation. The paper is intended to constitute both an introduction to the specificity of the system and a more detailed presentation of recent results. Two contributions are more specially stressed out. The first one is related to resource handling : we propose algorithms to deal with allocated resources and the usage of variable quantities of resources. The second consists in a heuristic estimator to control the exploration of the plan space through a reachability analysis.

INTRODUCTION

Long confined to "toy" domains, planning systems now exhibit impressive performance and can deal with very large problems. Such capabilities have given a new impulse to the development of planners which aim at reasoning on more realistic descriptions. Several systems have recently been proposed which handle numeric features such as time and resources. They take advantage of the ideas that turned out to be successful in the classical planning paradigm, such as search in the state space guided by heuristics. However these systems put restrictive assumptions on the domains they consider and the plans they produce are somehow over-constrained. These criticisms are motivated by the actual goal pursued when extending the scope of planning problems : real world applications such as robot control or autonomous spacecraft, where both flexibility and concurrency are required. For instance, let us consider a problem involving a 2-arms robot which has to grasp an object. Unless strictly required, a flexible plan could leave up to the execution system the choice of which arm to use and when next actions should start.

In this paper, we expose a framework for planning in temporal domains which aims at getting rid of such limiting assumptions. Some principles of this framework are common to several existing systems - see for instance (Muscettola 1994), (Rabideau *et al.* 1999), (El-Kholy & Richards 1996). However this paper focuses on the IxTeT system (Ghallab & Laruelle 1994) that we are extending to match the issues hereby discussed.

First we discuss the range of temporal domains that we address and the formalism on which we rely to capture them. The main feature of this representation is a detailed description of the distribution of the effects of actions over time, which allows complex actions interleaving. Then we analyze the algorithms involved in the production of plans in such domains. The choice of these methods, namely lifted partial order planning relying on CSPs, was guided by the need for flexibility. However it raises specific difficulties when considering resources which are discusses in a dedicated section. We conclude with a look at the crucial problem of controlling the plan search and we propose a new heuristic estimator based on an original structure : the Plan Space Planning Graph.

REPRESENTATION IN THE IXTET FORMALISM

Classical planning has abstracted away time and resource in state transition systems. Such an approach has severe drawbacks in expressiveness which limit its use in real applications. In the real world, actions have different durations, need some resource, their start points do not necessary meet, some actions do not only change the environment but might also enforce some of its characteristics to remain constant and, last but not least, effects might depend on time.

From these remarks, we draw the conclusion that temporal planning involves moving ahead of global, still pictures of the entire world, i.e. states, and their associated transitions. Actions should be considered as partial specifications of evolutions which spread over time and which can be combined. The considered description should keep track of the way these composed occurrences affect the world, representing both local changes and persistence in order to provide a consistent base to decide what can and cannot be done.

The temporal representation we are interested in here focuses on local individual evolutions of state and resource variables. A state variable is a function of time which describes the evolution of a characteristic of the domain over time. A resource represents a substance or a set of objects whose availability induces constraints on the actions which use it. The following paragraphs give details about the description of domains and the underlying logic.

State variables : attributes of the domain

As said above, a state variable, or *attribute* is a partially specified function of time whose evolution over time is specified through a set of temporal propositions. The range of

values for these functions are either sets of symbolic values or disjunctions of numeric intervals. Attributes are parameterized by a vector of variables valued in finite sets. For instance, in a domain where several robots are moving around, we might declare the following state variable :

$$attribute\ Position(?Rb)\{$$
$$?Rb\ in\ ROBOT;$$
$$value\ in\ LOCATION;\}$$

In order to describe the dynamic of the world and the way actions are going to affect it, we need to specify evolutions of attributes but also persistence. This is achieved through a temporal propositional logic whose two core propositions are $event$, which tips up instantaneously at a given time-point the value of an attribute, and $hold$, which maintains the value of an attribute between two given time-points (see Fig.1).

Considering only persistence and instantaneous change, one is limited to describe only piece-wise constant functions and thus is not able to catch much of the real world dynamic. The will to extend the range of domains which might be handled within IxTeT led us to introduce a third proposition : $change$. It specifies a linear evolution of an attribute over a temporal interval. For instance, the opening of an automatic gate which opens vertically could be represented by the following proposition : $change(Opening(OurGate) : (0.0, 3.0), (t1, t2))$. The numeric values are the initial and final heights of the gate opening. Such a representation offers the possibility to infer intermediate positions of the door. Consequently, a robot could go through the gate although it is not completely open. So far, we considered only linear changes but in order to be exhaustive, we should extend the formalism to represent all type of functions. For instance, if the gate is opening with an acceleration, the change must follow a quadratic function. One might argue that many domains, including some which are closely related to real world applications, can be described without such a representation. But it is at the price of a higher complexity of the search space and there are some problems which cannot be coded without the notion of continuous change.

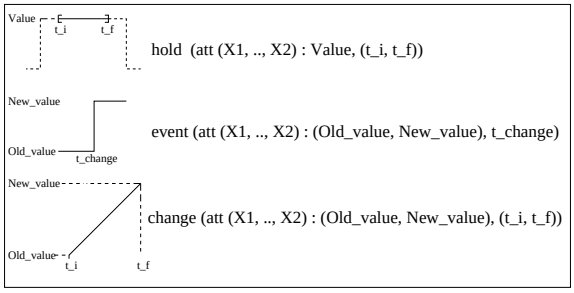


FIG. 1 – *The three types of temporal propositions*

Resources

Contrary to state variables, the evolution of the level of a resource is not explicitly described. Instead the representation focuses on the relative changes that the occurrence of an action will cause on a resource that it uses. Resource usage can be described thanks to three types of temporally qualified propositions : use (borrowing over a temporal interval), $produce$ and $consume$ (production and consumption at a given time-point).

First, we focused on proposing efficient handling of parameterized resources. These are resources which are defined as instances of a same resource type which might be used indifferently by actions. For instance a satellite might store a camera picture either on its first or second disk as long as there is a disk with enough available memory. A resource type $R(?X)$ is defined by its initial capacity and a domain for its parameter :

$$resource\ Arm(?Rb)\{$$
$$?Rb\ in\ ROBOT;$$
$$capacity = 2;\}$$

Another important new feature of the formalism is the possibility to specify the use of a variable quantity of resource by an action : one could express that storing a camera picture might use between 200 and 300 ko of memory depending on the selected target. This variable quantity can then be constrained, either by other variables or action's duration. Such resource usages are specified through propositions of the following forms :

$use(R(?X) : ?q, [t_s, t_e])$ (borrowing),
$produce(R(?X) : ?q, t_p)$ (production),
$consume(R(?X) : ?q, t_c)$ (consumption)

where $?q$ is a variable quantity whose domain is an interval $[q_{min}, q_{max}]$, t_s, t_e, t_p and t_c are time-points.

Linking partial descriptions through constraints

All these temporal propositions (on resources and on attributes) can be linked with each others through constraints on time-points, variable state parameters or values. The descriptions of both actions and the world consist in conjunctions of such temporal propositions and constraints.
Applying an action - a task in IxTeT's vocabulary - means integrating its propositions and constraints into the world's conjunction in a consistent way. Consistency checking takes into account the explicit constraints network but also the more implicit constraints linked to the semantic of state variables, which are functions of time.

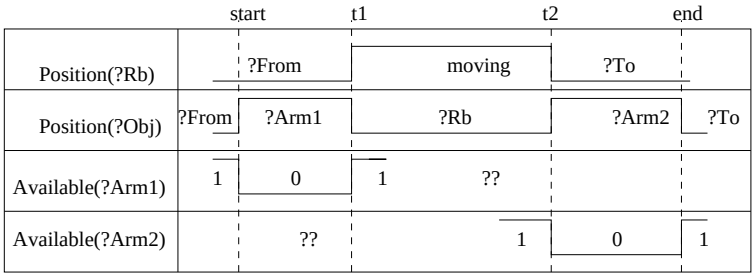
This notion of actions as conjunctions of local evolutions offers a sufficient granularity to express complex actions concurrency (see Fig. 2).

The extensive use of constraints to link temporal propositions also provides support to one other specificity of temporal planning : influence of actions' durations over their effects. For instance, the distance covered by a robot will be proportional to the duration of its "Move" action. This is achieved by expressing constraints that mix the values of state variables and durations.

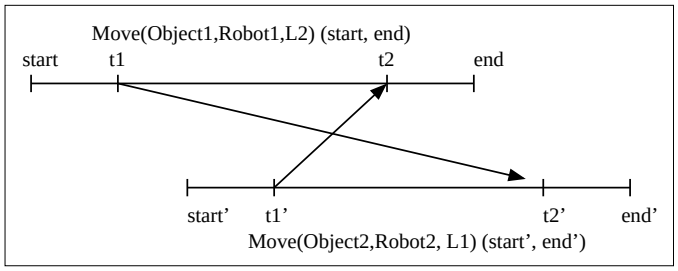
This last feature, among others, has been avoided by several systems because a straightforward coding might lead to infinite branching factors. On the contrary it fits quite naturally into our representation since using ungrounded actions prevent us from considering individually all possible durations.

The next subsection insists on other specific aspects of the representation and provides ground to efficient encoding of domains.

```
task MOVE(?Obj,?Rb,?From,?To)(start,end) {

    event(Position(?Obj):(?From,?Arm1),start);
    hold(Position(?Obj):?Arm1,(start,t1));
    event(Position(?Obj):(?Arm1,?Rb),t1
    use(Available(?Arm1):1,(start,t1));

    hold(Position(?Rb):?From,(start,t1));
    event(Position(?Rb):(?From,moving),t1);
    hold(Position(?Rb):moving,(t1,t2));
    event(Position(?Rb):(moving,?To),t2);
    hold(Position(?Rb):?To,(t2,end));

    event(Position(?Obj):(?From,?Arm2),t2);
    hold(Position(?Obj):?Arm2,(t2,end));
    event(Position(?Obj):(?Arm2,?To),end);
    use(Available(?Arm2):1,(t2,end));

    ?Obj in Objects;
    ?From in PLACES; ?To in PLACES;
    ?From != ?To;
    ?From in {L1} => ?Arm1 in {A1};
    ?To in {L2} => ?Arm2 in {A2}
     ?distance = dist(?From,?To);

    (t1 − start) in [00:03:00,00:03:30];
    ?distance = speed*(t2 − t1);
    (end − t2) in [00:03:00,00:03:30];

}
```



State Variables Table



Intertwining Actions For 2 Robots

FIG. 2 – *A Task Example in the IxTeT Formalism*

Efficient use of constraints and attributes

The formalism we are dealing with offers interesting features, such as an explicit representation of some mutex information through the use of functions of time instead of predicates. However, as is true of other representations, a domain can be described in many ways, some of which being by far harder to solve by the planner. The context of the 2002 International Planning Competition lead us to study more precisely domain encoding, with the goal of realizing an automatic translation tool from PDDL 2.1 specifications (Fox & Long 2002) to IxTeT's formalism.
Through this work we identified two important principles to obtain descriptions which can be efficiently used by our planner. They could be summarized as follows :

– do not write rigid attributes (i.e. constant predicates),

– translate predicates into functions as much as possible.

The first rule is closely related to the fact that our planner is a lifted partial order planner. It constitutes a shift toward a more active handling of the constraints induced by constant predicates. A constant predicate with one argument is replaced by a domain constraint over a variable. A constant predicate with two arguments is translated into a set of dependency constraints between the two arguments.

The second rule is simply an advice to hand-code evident mutual exclusion information. For instance, a predicate $Position(?Rb, ?loc)$ should be translated into an attribute $Position$ with one parameter $?Rb$ and valued into the set of locations for which the predicate is interpreted as true. Once again it is of special interest in the context of ungrounded operators since it reduces the domains of variables.

Applying as much as possible the second rule might involve more than simply translating one predicate into one attribute: there are cases where this is not possible but further domain analysis detects that merging two predicates into one state variable is valid. Examples of such predicates can be found in (Edelkamp 1999). Based on the propositions of this article, we developed a domain analyzer which automatically performs such translations and merging.

HANDLING TIME IN A POP FRAMEWORK

As was previously pointed out, real applications require to keep flexibility in plans so as to minimize the cost of adapting to the possible gap between the real world and the available model. Search in the plan space, with only necessary ordering constraints and partially grounded operators, provides an elegant solution to the need for flexible planning. In this section we expose an adaptation of the classical plan space schema to the formalism exposed in the first section.

POP in the Classical Setting

The purpose of planning is to find a sequence of actions between an initial state and a final state given by the user.

The classic Partial Order Planning algorithm searches the solution plan into the partial plan space. A partial plan is a tuple $(\mathcal{A}, \mathcal{O}, \mathcal{B}, \mathcal{L}, \mathcal{OC}, \mathcal{UL})$ where $\mathcal{A}$ is a set of potentially ungrounded actions, $\mathcal{O}$ a set of constraints which order $\mathcal{A}$, $\mathcal{B}$ a set of constraints over the variables figuring in elements of $\mathcal{A}$ and $\mathcal{L}$ is a set of causal links (i.e. a link between an action which establishes one precondition of another action). The description of a partial plan is completed by two sets of flaws : $\mathcal{OC}$, a set of open conditions, and $\mathcal{UL}$, a set of unsafe links (threats). A partial plan stands for a family of plans. It is considered to be a valid solution if all of these plans are consistent. This implies that the sets $\mathcal{OC}$ and $\mathcal{UL}$ are empty. An open condition figures a non-explained precondition of one of the actions belonging to $\mathcal{A}$. A threat appears when an action a_1 explains a precondition p of an action a_2 while a third action a_3 might cause the cancellation of this precondition, i.e $not(p)$ is an effect of a_3 and a_3 might occur between a_1 and a_2 without violating any of the constraints in $\mathcal{O}$.

The algorithm starts its search from a partial plan which contains only two virtual actions. The first action has only effects: it establishes the initial state. The second one has only preconditions: it initializes the open conditions with the desired final state, thus requiring the planner to refine this initial node.

Step by step, the POP algorithm is going to try to resolve all the flaws of the partial plan. A step consists in selecting a flaw and choosing a resolvent. In the case of an open condition we have the choice to explain it either by inserting a new action whose effects contain this open condition or by establishing a new causal link with an action which is already part of the plan. To resolve a threat, it is necessary to overconstrain actions so as to ensure non-overlapping between the threatening action and the threatened link. This is achieved by posting temporal (promotion/demotion) or atemporal (inequalities) constraints.

If the search reaches a node which contains one flaw with no possible resolvent, it has gone into a dead-end branch : it has to backtrack on one of its previous choices.

Using this short summary as a background, we now turn to the full context of IxTeT, considering lifted actions with explicit representation of time and resource, and explain how the notions of flaws and resolvents are extended to this formalism.

Extension to IxTeT's Logic

The first step to ensure the soundness of the search consists in transposing the notion of flaws, including both threats (or unsafe links) and open conditions, to the considered temporal propositions.

Flaws are split into two clearly disjoint sets, with their associated detection and resolution modules : those concerning state variables and those concerning resources. This part focuses on state variables, whereas the next one is dedicated to the "Resource Conflicts Detection and Resolution" module.

Within the IxTeT's frame, the notion of threats and open-conditions on state variables encodes the semantic of attributes as functions of time. The set of propositions on an attribute should describe a consistent sequence of transitions and ensure that at no time two different values are associated with the same attribute.

In order to build consistent sequences of transitions on an attribute, two sources of open-conditions are considered : assertions of persistence and event propositions. A $hold$ proposition requires the state variable to reach the maintained value before it comes into effect. In the same way, an $event$ proposition requires the state variable to be at its initial value before it happens.

Establishers are event propositions whose final value is going to match the one required by the open condition. They are either looked for within the current partial plan or within new tasks to insert.

The other kind of flaw on attributes, namely threats are defined so as to check that mapping an attribute to its value at any time is deterministic. Threats are defined that involve two $hold$ propositions or one $event$ and one $hold$ propositions. Two $hold$ should not overlap unless they maintain an attribute to the same value or the attributes they constrain can be differentiated by separation constraints on their parameters. An $event$ should not occur during a $hold$ proposition unless separation constraints can be posted to differentiate their respective attributes; if an $event$ is bound to meet a $hold$ at start or end, their value should be unified to obtain a consistent evolution of the attribute, or once again the attributes should be separated by constraints on parameters.

Through the definition of flaws in IxTeT, two specifities of the considered temporal context arise. The first one is the ambivalence of the $event$ propositions, which can be considered as effects (establishers) or preconditions (open condition) at the same time. The second one is that it relies heavily on CSPs, most of the resolvents consisting in either ordering, binding or inequalities constraints. The next subsection is dedicated to the module in charge of handling variables and enforcing an important part of the consistency of a partial plan.

A Framework relying on CSPs : Expressiveness and Flexibility

CSPs managers play a central role in Partial Order Planning. It is true even in the classical planning since a partial plan can be seen as a set of constraints defining a family of candidate solutions. In our temporal framework, efficient propagation of constraints to compute accurate minimal domains for variables is essential. First, as explained above, the notions of flaws and resolvents have been transposed in a quite straightforward manner to temporal propositions but at the price of an increased complexity pushed on the requests to the CSP handlers. Furthermore, several of the features discussed in the first section are handled through dedicated constraints. The most representative example is the expression of actions with effects depending on durations. This involves constraints such as $?d = a * (t_f - t_i)$ which mix time-points (t_f and t_i) and variables ($?d$) - see (Trinquart & Ghallab 2001) for details.

Initially disregarding such coupling constraints, the system was developed with two disjoint CSP managers : the Variable-Map on one hand, which combines 2-consistency and forward checking to handle atemporal variables, and the Time-Map on the other hand, which implements a Floyd-Warshall-like propagation schema (Dechter, Meiri, & Pearl 1989). The link between these two modules is achieved by a

supervisor which transfers information from one CSP to the other when required. While this class of CSP is known to be NP-complete, preliminary tests indicate that actual planning problems lead to tractable instances.

RESOURCE CONFLICTS DETECTION AND RESOLUTION

The problem of ensuring consistency of the set of resource propositions at each step of the search consists in ensuring that there is no over-consumption of a resource. The difficulty lies in the impossibility to compute the actual level of a resource because the order on propositions is only partial and some variables might be non-instantiated. The complexity is even worsen by the fact that variables might appear in temporal propositions on resources as parameters and as borrowed quantities.

This section details the module which is responsible for this aspect of consistency. It is based on an extension of an efficient clique-search algorithm on a possible intersection graph. Resolvents are chosen among resource production (task insertion) or separation (temporal orderings or variables inequalities). There again the desired expressiveness entails a high number of resolvents and consequently a high branching factor. We finish this section with the proposition of a criterium to discriminate resolvents leading to dead-ends, thus pruning out useless branches as early as possible.

Detection of potential resource contentions

The resource analysis is divided into two phases : first find for each resource attribute over-consuming sets of potentially overlapping propositions and then propose a disjunction of resolvents for each conflict.

To detect conflicts, any *potential* intersection should be considered, whereas to be sure to solve a conflict, a production should *necessarily* precedes the set of involved propositions.

In (Ghallab & Laborie 1995), the authors reason on borrowing propositions and then suggest to translate any production $produce(AttR : q, t)$, where $AttR$ is a non-parameterized type and q a fixed quantity, into an increment of the capacity of $AttR$ by q and a borrowing proposition $use(AttR : q,]-\infty, 0])$. Thus considering *potential* intersections only is still valid. Extending this "trick" to parameterized resources with variable quantities raises difficulties. Let us consider a proposition $produce(AttR(?X) : ?q, t)$. The first point is that if $?X$ is not instantiated, it is not possible to decide which resource's capacity should be increased. Second, $?q$ might also be non-instantiated, thus it is not possible to decide how to increment directly the capacity of a resource. In order to solve these two problems and to keep detecting resource contention by searching through possible intersection sets, we propose the following rewriting rule for a *produce* proposition. $produce(AttR(?X) : ?q, t)$ $(q \in [q_{min}, q_{max}])$ is transformed into two propositions and a capacity increment:

$$u_{bef} = use(AttR(?X) : q_{max}, [0, t]),$$
$$u_{aft} = use(AttR(?X) : q_{max} - ?q, [t, +\infty[)$$
$$\text{and } C_{Incr} = q_{max}.$$

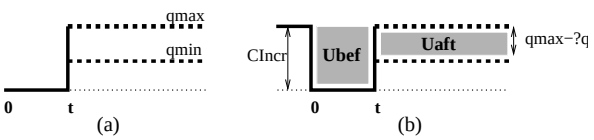


FIG. 3 – *Equivalent resource profile for a* produce *proposition (a), and for its translation* $< u_{bef}, u_{aft}, C_{Incr} >$ *(b).*

C_{Incr} will be taken into account in the detection of any potential overconsumption involving u_{bef}. Figure 3 presents this rule in a graphical way. $consume(AttR(?X) : ?q, t)$ is simply transformed into $use(AttR(?X) : ?q, [t, +\infty[)$.

Once all *produce* and *consume* propositions have been translated into their equivalent borrowing propositions and capacity increments, conflict detection comes down to search for *Minimal Critical Sets*. Actually, as no subset of a *MCS* can lead to a conflict, a *MCS* can be completely solved by the insertion of one resolvent. These *MCS* are detected as minimal over-consuming cliques in a *Possible Intersection Graph*.

A *PIG* is associated to each resource type $R(?r)$ (whose initial capacity is denoted by $Capa_R$ below). The vertices of the graph are propositions on R in the partial plan, and its edges are the pairs of propositions which may intersect in some linearization of the plan. The resource parameter $?r$ is taken into account in the computation of cliques (i) and in the criterion to detect over-consumption (ii) :

– (i) the parameters of all propositions of a clique $\mathcal{C}$ can be unified in a variable $?r_C$.

– (ii) the capacity corresponding to each clique $\mathcal{C}$ is calculated by :

$$Capa(\mathcal{C}(?r_C)) = Capa_R + \sum_{u_{bef} \in \mathcal{C}} CIncr(u_{bef}(?r))$$
$$\text{where } ?r \text{ and } ?r_C \text{ can be unified.}$$

Consider a unified clique $\mathcal{C} = \{u_1(?r_1), \ldots, u_k(?r_k)\}$ with u_i a borrowing of a quantity $?q_i$ in $[q_{min}(u_i), q_{max}(u_i)]$. $\mathcal{C}$ is a *MCS* if :

$$\sum_{u \in \mathcal{C}} q_{max}(u) > Capa(\mathcal{C})$$
$$\text{and } \sum_{u \in \mathcal{C}} q_{max}(u) - \min_{u \in \mathcal{C}}\{q_{max}(u)\} \leq Capa(\mathcal{C})$$

Resolvents computation

The second phase of the analysis is the computation of all possible resolvents. A *MCS* can be solved by posting one of the following usual resolvents : *promotion/demotion* and *parameters separation*. But as we are considering variable resource usage we need to propose an additional resolvent to guarantee the soundness and completeness of the search process : *reduction of the resource usage* so as to nullify the overconsumption. This resolvent is applicable in case $\sum_{u \in \mathcal{C}} q_{min}(u) \leq Capa(\mathcal{C})$ holds. It consists in posting a constraint : $\sum_{u_i \in \mathcal{C}} ?q(u_i) \leq Capa(\mathcal{C})$. Such a constraint respects the least commitment strategy.

The last possible resolvent is the *insertion of an action* containing a proposition $u_p(?r_p)$ producing a quantity $?q(u_p)$ such that : $?q(u_p) \geq \sum_{u_i \in \mathcal{C}} ?q(u_i) - Capa(\mathcal{C})$.

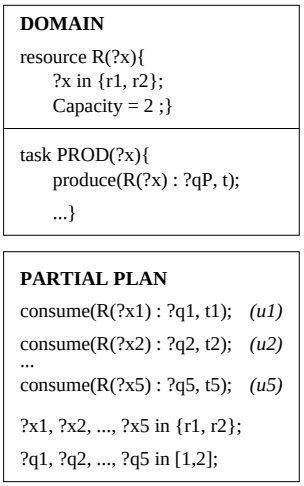


FIG. 4 – *Resource contention example.*

The result of such a resource analysis is the set of the smallest *MCS* and their associated disjunction of resolvents. This can lead to a quite important branching factor. Hopefully it is possible to detect and discard early during this analysis phase resolvents which will lead to dead-ends branches.

Consider for instance the simple partial plan described in figure 4 and containing five *consume* propositions on the resource attribute $R(?x)$. The resource analysis process will detect and propose resolvents for all conflicts of the minimal size (i.e. 2 propositions). The *MCS* u_1, u_2 for instance can be solved by $(x_1 \neq x_2) \vee (?q_1 + ?q_2 \leq 2) \vee (PROD(?x))$. But the two first resolvents appear to be useless as the total resource capacity is 4 whereas the minimal consumption in the plan is 5. Therefore, *separation* and *domain reduction* can be discarded from the disjunction of resolvents in certain cases.

We implemented such a strategy by doing a pre-analysis for each resource attribute. Using the same algorithm as for *MCS* detection, but without considering resource parameters, we search for *type-conflicts* corresponding to cliques such that no precedence constraint is possible and:

$$\sum_{u \in \mathcal{C}} q_{min}(u) > Capa_R * nb_{rsce} + \sum_{u_{bef} \in Plan} CIncr(u_{bef})$$

Then during the conflict detection process, we test if the *MCS* is included in at least one *type-conflict*. In that case, the set of resolvents is reduced to a deterministic one : *action insertion*.

This notion of *type-conflicts* is a way to decrease the branching factor by discarding resolvents. However it is not sufficient to prevent the search from getting lost in the search space. In the next section, we propose a new heuristic control to choose from resolvents and enforce the efficiency of the search.

SEARCH CONTROL: INTRODUCING THE PLAN SPACE PLANNING GRAPH

In the previous sections, we stressed out the advantages of IxTeT, mainly the expressiveness offered by the formalism it can handle and the flexibility of the plan it produces. However one important drawback of the system has also been raised at some places : the important branching factor it encounters at each step of the search process.

To choose from the different ways of refinement which are determined by the "Flaws and Resolvents Analysis" module, IxTeT relies on different heuristic functions which mix means-ends analysis and least commitment evaluation. The cost of inserting an action into a partial plan is computed through the limited development of a regression graph to estimate the cost of the new subgoals. In addition, the cost of any ordering or binding constraint is estimated in term of the portion of the plan space which is pruned out of the candidates for refinement. This is a measurement of least commitment which is supposed to maximize chances to find a plan-solution as long as the distribution of solutions is uniformed over the plan space.

This set of cost functions has proven to be useful in a large set of domains (Laborie 1995). However, additional hand control is needed to balance these different heuristic estimators in order to scale-up to larger problems such as the benchmarks from IPC 2002.

In this section we discuss recent advances we made in defining an appropriate domain-independent heuristic for guiding the search in the plan space. First we will present the inherent difficulties of estimating costs in a partially ordered / bounded context, basing our discussion on the ideas explained in (Nguyen & Kambhampati 2001). From this analysis, we will then propose a new structure, derived from the notion of the planning graph, to estimate distance within the Plan Space.

RePOP: Bringing back POP to light

RePOP is a partial order planner which explores the plan space using grounded actions to refine partial plans. It integrates several ideas to reduce the search, but the one which is most often referred to is the use of a heuristic obtained thanks to a planning graph : it has proven that, when efficiently guided, a partial order planner could deal with large problems and produce plans of good quality. Given this statement, we naturally turned to this system to see how such ideas could be adapted to a temporal context. And we met difficulties which prevented us from directly transposing this solution to out framework, but which also gave us the ground to further investigations.

To understand why RePOP's heuristic cannot be used in our framework, one should first remember that the use of an informative heuristic is only one of the propositions in (Nguyen & Kambhampati 2001). One other important feature of this system is the way it handles ordering constraints over actions in a disjunctive way. When considering a threat between a causal link and an action, RePOP does not choose between resolving the threat by demotion or promotion : instead it refines the plan with a disjunctive ordering constraints ensuring the non-overlapping of the causal link and the involved action, without committing to one of the alternative. This has an important consequence with respect to the heuristic choice: the system does no longer have to choose between threat resolvents or open-condition establishers. It only has to estimate the cost of establishing open-conditions by

causal links with new actions or elements of the current partial plan.

Is it possible to use such disjunctive constraints to reduce the branching factor in the temporal framework of IxTeT? To answer this question, let us detail the constraints considered in RePOP. In the classical setting, with grounded actions, a threat involves a causal link of the form $A \rightarrow^{p} B$, which means that the action A establishes the precondition p of the action B, and a third action C whose effects contain $not(p)$ and with such ordering constraints that $A < C < B$ is consistent. RePOP solves such a threat by posting the disjunctive constraint $C < A \; or \; B > C$. If we transpose this for instance to a threat involving two propositions $event(Att(?X) : (v_i, v_f), t_e)$ and $hold(Att(?X') : v_h, (t_i, t_f))$, the associated disjunctive constraints is of the form $t_e < t_i \; or \; t_f > t_e$. Such a constraint cannot be translated into a distance between two time-points with a disjunctive range. Thus encoding it into the Time-Map mentionned in a previous section actually boils down to handling two matrices of distances. Furthermore, whereas RePOP makes the assumption to use only grounded operators, IxTeT also has to consider a third alternative to Promotion and Demotion : parameters separation. Integrating these three alternatives into one disjunctive constraint to separate two propositions has a huge cost in terms of CSP consistency checking. So far we preferred to keep these alternatives into separated branches of the search tree and elaborate a satisfactory heuristic estimator to choose from these refinements.

The PS-PG : estimating distances within the Plan Space

Our goal is to keep least commitment as a principle to build plans, for instance through the use of ungrounded actions, but not as part of the choice criteria. Thus we have to provide a heuristic control to choose from a set of various refinements including task insertion, temporal constraints and constraints over variables. This raises conflicting issues : the computation of the heuristic should take into account the way ordering constraints might affect further refinements but it should also be computed quickly. Using a planning graph to estimate the cost of the remaining refinements to obtain a solution has been used successfully in different planning schemas, but the previous subsection exposes our reasons not to use it directly. Moreover, a more subjective reason not to reuse RePOP's heuristic is that it does not seem so natural to explore the plan space and estimate distances in the state space.

In this last section, we propose to build a structure which is related to the planning graph structure but accounts for reachability within the plan space. We then propose a heuristic estimator derived from this structure to guide IxTeT within its search tree.

A Plan Space Planning Graph structure First introduced in (Blum & Furst 1995), the Planning Graph is a compact representation of the state space portion which can be reached from a given initial node. It is built as an alternation of propositions and actions layers, which are both disjunctive structures. Several heuristic functions have been proposed which rely on the computation of such a graph in a relaxed domain to estimate distances to reachable solutions.

Leaving aside some specificities of construction proposed by Blum and Furst and considering a planning graph simply as a compact representation of the part of an oriented graph accessible from a given node, it seems appropriate to adapt it to a special type of oriented graph : Plan Space. Heuristic estimators derived from the planning graph are based on some estimations of the number of transitions in the graph needed to reach a solution node. In state space, this corresponds to the number of actions to add to the plan to reach a solution state. In the plan space it will correspond to the number of resolvents that will lead to a solution plan.

A key feature of the Planning Graph structure is disjunctivity : both propositions and actions layers are disjunctive, they collapse into one node potentially mutually exclusive descriptions of the world. Compacting the plan space in the same way involves handling disjunctive layers of resolvents and layers with disjunctive partial plans. This is of particular concern since the cost of maintaining consistency in a partial plan was already considered as a drawback of partial order planning.

The specific difficulty of handling disjunctive partial plans with ungrounded actions is to avoid abusive propagations of constraints (both temporal and atemporal) across disjoint components. Instead of explicitly putting disjunctions into the CSPs, we propose a "rewriting" rule to transform an open condition into as many established propositions as considered resolvents. For instance, if an assertion $H = hold(Att(?X) : v_h, (t_i, t_f))$ can be explained either by $event(Att(?Y) : (\_, v_h), t_e)$ or by $event(Att(a) : (\_, v_h), t'_e)$, we will replace H by 2 propositions $hold(Att(?Y) : v_h, (t_i, t_f))$ and $hold(Att(a) : v_h, (t_i, t_f))$ and add the corresponding causal links to the partial plan.

The remaining open issue to transpose a planning graph structure into a plan space is the way layers are expanded. When building a planning graph in the state space, each actions layer is developed by adding applicable actions with respect to the previous propositions layer. Neglecting mutual exclusions, an action is considered to be applicable if all of its preconditions are contained by the propositions layer. Thus an actions layer contains all the valid transitions from the set of states figured by the previous propositions layer. In the plan space, nodes are partial plans and transitions are refinements. Contrary to what happens in state space, where the graph is somehow built in a blind forward manner, it is not necessary to keep all possible refinements of a plan: it is possible to restrict the graph expansion to refinements which are resolvents of flaws in the considered partial plans and to limit the size of the portion of the plan space captured by the planning graph. Moreover, in the same way mutexes can be ignored to decide which actions should be included in an actions layer, we only consider resolvents for open-conditions during the initial development of the graph.

Before moving on to the discussion of the heuristic function we propose to derive from this structure, we will turn back to one of the initial requirements we put on a heuristic estimator in a lifted POP context : the ability to choose from various resolvents such as causal link, task insertion, temporal constraints or inequalities over variables. This implies that the Plan Space Planning Graph (PS-PG) enables us to catch the difference between the accessible parts of the plan

space induced by posting, for instance, either promotion or demotion constraints. It is achieved by revising along the search which refinements are actually applicable to a given partial plan.

More precisely, there are two different phases in the management of the Plan Space Planning Graph during the plan search process. The search starts by developing a complete PS-PG which is built from the initial partial plan. Then at each step, resolvents are inserted in the partial plan and the constraints that they bring into the plan are also propagated in the PS-PG. Having chosen a resolvent which establishes an open-condition discards alternative resolvents and thus reduces uncertainty due to disjunctions. Posting constraints over time-points (Promotion/Demotion) or over variables (inequalities or upper boundaries to solve resource conflicts) might over-constrain the partial plan in such a way than some resolvents are now impossible and thus also leads to reduce the reachable part of the Plan Space.

Thanks to this propagation mechanism, it is possible to adapt the PS-PG accordingly to the information carried by each resolvent. Thus it can be used, at each step of the search, as ground for the evaluation of the possible different refinements of the partial plan. In the next section we propose a heuristic which uses this structure to estimate the cost of a refinement in terms of the number of resolvents which should be further inserted to reach a solution-plan.

Extracting a heuristic estimation Our main motivation to use such a structure as the PS-PG is to enhance the search process in IxTeT through a better control. We aim at keeping the methods of plan analysis and resolvents computation unchanged and at using a unique estimator to rank all the possible resolvents of the flaws in the current partial plan at each step of the search.

In the context of a $\mathcal{A}_\epsilon$ search algorithm, we want to rank resolvents by measuring the minimal distance between a solution plan and the partial plan that would result from the insertion of a resolver in the current partial plan. In the plan space, distances are numbers of refinement steps. When refining a partial plan into a solution plan, two types of refinements can be done : insertion of the establisher of an open-condition and insertion of a conflict resolvent. Using the PS-PG, we will only count establishers to approximate the distance between a partial plan and a solution plan.

The formulas presented below, which define the way the costs of partial plans (i.e distances to solution) are computed, are induced by the interpretation of the PS-PG as an AND/OR structure : all the open-conditions in a partial plan should be established to reach a solution and each open-condition can be established in different ways. In the following equations, $\mathcal{P}$ denotes a partial plan, $\mathcal{OC}(\mathcal{P})$ the set of open-conditions in $\mathcal{P}$, p a temporal proposition, $Est(p)$ the set of possible establishers of p which appear in the PS-PG, E a task and finally $\mathcal{OC}(E)$ is the set of open-conditions E would introduce in the plan.

$$(1)\ cost(\mathcal{P}) = \sum_{p \in \mathcal{OC}(\mathcal{P})} cost(p)$$

$$(2)\ cost(p) = \min_{E \in Est(p)} cost(E) \text{ if } Est(p) \neq \emptyset, \quad \infty \text{ otherwise}$$

$$(3)\ cost(E) = 1 + \sum_{p \in \mathcal{OC}(E)} cost(p)$$

Although these equations might seem quite common, a few comments should be stated with respect to the PS-PG structure. Two related issues should be payed attention to : the costs associated to establishers by (2) and the measurement of actions reuse (positive interactions).

In the PS-PG, two kinds of establishers can occur : simple causal links with elements of the partial plan and causal links with elements of new tasks inserted in the plan. The first point that should be stressed out is that causal link establishments are explicitly measured: they are transitions in the plan space which induce non-null distance between partial plans. The second point is that we choose to associate the same cost to the two considered types of establishers. Once again this should be linked to the interpretation of establishers as transitions in the plan space.
More over, this second choice is also supported by some considerations on positive interactions measurement. Since establishment through a causal link with an element of the current plan is associated with a non-null cost, the reuse of a task is not costless. However it should be taken into account. Let us consider an example where two open-conditions $oc1$ and $oc2$ appear at the same layer of the PS-PG and can be explained by the same action A. The cost of establishing $oc1$ and $oc2$ should be equal to the cost of inserting A in the plan and linking it to $oc1$ plus the cost of linking A to $oc2$. The difficulty comes from the way the PS-PG is expanded : both $oc1$ and $oc2$ will be explained in the same layer and the resulting cost will be two times that of inserting A and linking it to one of $oc1$ and $oc2$. This also leads us to associate the same cost to the two kinds of establishment.

Finally, equation (2) indirectly raises the question of when to stop the development of the planning graph. Contrary to what happens when developing a classical planning graph, the size of the set open-conditions does not decrease along the construction of the PS-PG. However equation (2) allows us to stop developing the PS-PG as soon as a layer is reached where each open-condition has at least one establisher with a non infinite cost. In addition, we limit the development of the PS-PG to a maximum depth, considering that an open-condition which has not been established so far has an infinite cost.

Looking at a simple example The Fig. 5 illustrates a PS-PG built for a simple problem. We consider to actions Eat and $Bake$ defined as follows :

$$\begin{array}{l} task\ Eat(?c)(Start, End)\ \{ \\ \quad event(Eaten(?c) : (\_, true), End); \\ \quad event(Have(?c) : (true, false), End);\ \} \\ task\ Bake(?c)(Start, End)\ \{ \\ \quad event(Have(?c) : (false, true), End);\ \} \end{array}$$

The initial situation states that one cookie is available and the plan should lead to eat one cookie and have another one. From this initial situation, the PS-PG is developed until it reaches a layer that supports a non-null cost for each open-condition in the initial layer. Let us describe through this example how it is used to choose which resolvent to insert in the partial plan.

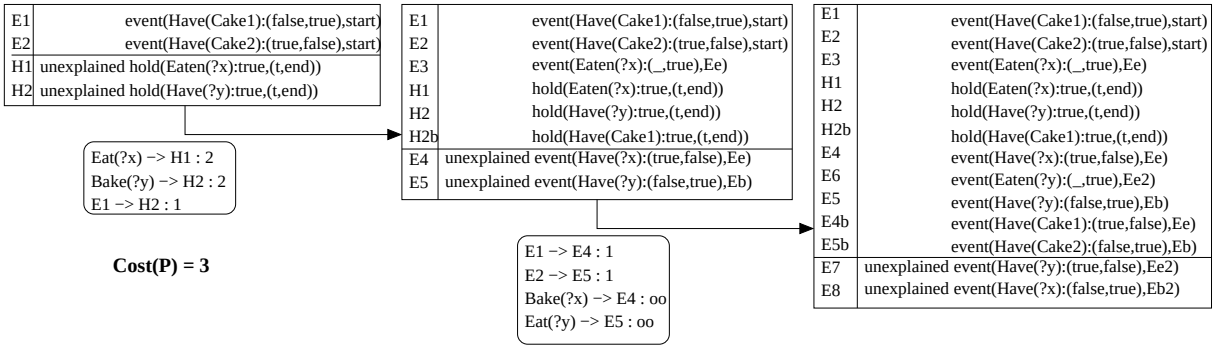


FIG. 5 – *A PS-PG in a simple domain*

On the first step of the search, the only flaws in the plan are open-conditions and the resolvents are the three establishers on the first layer of the PS-PG.

If the causal link $E1 \rightarrow H2$ is inserted in the plan, then the PS-PG would be modified : the establisher $Bake(?y)$ on the first layer is no longer considered, H2 and E5 are discarded from the second layer, the insertion of $Eat(?y)$ and the causal link $E2 \rightarrow E5$ are also removed from the second layer, as well as the causal link $E1 \rightarrow E4$ since $E1$ already contributes to $H2$. So the only establisher for E4 is the insertion of Bake(?y) which is associated with an infinite cost in the PS-PG.

On the contrary, if the task $Bake(?y)$ is inserted in the plan, then the causal link $E1 \rightarrow H2$ is discarded from the first layer and H2b is removed from the PS-PG. The rest of the structure is left unchanged and the resulting estimation is 4.

CONCLUSION AND FUTURE WORK

In this paper we presented a formalism for temporal domains and a related planning framework. We pointed out the great expressiveness it provides and the adequacy of plan space search to solve temporal problems. Beyond its functional and CSP-based representation, the system we describe distinguishes itself by a strong interleaving between planning and scheduling and by great possibilities in terms of resource handling.

Such features entail a high complexity in solving problems. This is especially a critical issue since Partial Order Planning in classical domains was already criticized for loose results with respect to search control. Being aware of this weakness of most of the partial order planners proposed so far but convinced of the possibility to guide efficiently a search process in the plan space, we focused on the heuristic control and proposed a new method to compute estimations of the distances between partial plans and solution plans.

This new heuristic function relies on an original structure : the Plan Space Planning Graph, which is a transposition to Plan Space of the planning graph initially proposed by Blum and Furst and which has been successfully used for heuristic purposes ever since. Implementation is still an on-going process and we hope to validate our proposition in the context of the 2002 International Planning Competition.

References

Blum, A., and Furst, M. L. 1995. Fast planning through planning graph analysis. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Dechter, R.; Meiri, I.; and Pearl, J. 1989. Temporal constraint networks. In *KR'89: Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann.

Edelkamp, S. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In *Proceedings of the European Conference on Planning (ECP)*.

El-Kholy, A., and Richards, B. 1996. Temporal and resource reasoning in planning: the parcplan approach. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*.

Fox, M., and Long, D. 2002. Pddl 2.1 : An extension to pddl for expressing temporal planning domains. *Technical Report, University of Durham, UK*.

Ghallab, M., and Laborie, P. 1995. Planning with sharable resource constraints. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Ghallab, M., and Laruelle, H. 1994. Representation and Control in Ixtet, a Temporal Planner. In *Proceedings of the International Conference on AI Planning Systems*, 61–67.

Laborie, P. 1995. Ixtet: une approche integrée pour la gestion de ressources et la synthèse de plans. *Thesis Report, LAAS-CNRS*.

Muscettola, N. 1994. Hsts: Integrated planning and scheduling. In *Fox, M., and Zweben, M., eds, Intelligent Scheduling, Morgan Kaufman*.

Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

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

Trinquart, R., and Ghallab, M. 2001. An extended functional representation in temporal planning : towards continuous change. In *Proceedings of the European Conference on Planning (ECP)*.

Planning Under Continuous Time and Resource Uncertainty: A Challenge for AI

John Bresina, Richard Dearden[1], Nicolas Meuleau[2], David Smith and Rich Washington[1]

NASA Ames Research Center
Mail stop 269–2
Moffett Field, CA 94035
{bresina, dearden, nmeuleau, de2smith, richw}@email.arc.nasa.gov

Keywords: Challenge Paper, Planning, Uncertainty, Markov decision processes

Abstract

There has been considerable work in AI on decision-theoretic planning and planning under uncertainty. Unfortunately, all of this work suffers from one or more of the following limitations: 1) it relies on very simple models of actions and time, 2) it assumes that uncertainty is manifested in discrete action outcomes, and 3) it is only practical for very small problems. For many real world problems, these assumptions fail to hold. A case in point is planning the activities for a Mars rover. For this domain none of the above assumptions are valid: 1) actions can be concurrent and have differing durations, 2) there is uncertainty concerning action durations and consumption of continuous resources like power, and 3) typical daily plans involve on the order of a hundred actions. We describe the rover problem, discuss previous work on planning under uncertainty, and present a detailed, but very small, example illustrating some of the difficulties of finding good plans.

The Problem

Consider a rover operating on the surface of Mars. On a given day, there are a number of different scientific observations or experiments that the rover could perform, and these are prioritized in some fashion (each observation or experiment is assigned a scientific value). Different observations and experiments take differing amounts of time and consume differing amounts of power and data storage. There are, in general, a number of constraints that govern the rover's activities:

- There are time, power, data storage, and positioning constraints for performing different activities. Time constraints often result from illumination requirements – that is, experiments may require that a target rock or sample be illuminated with a certain intensity, or from a certain angle.
- Experiments have setup conditions (preconditions) that must hold before they can be performed. For example, the rover will usually need to be at a particular location and orientation for each experiment and will need instruments turned on, initialized, and calibrated. In general, there may be multiple ways of achieving some of these setup conditions (*e.g.* different travel routes, different choice of cameras).
- The amount of power available varies according to the time of day, since solar flux is a function of the angle of the sun.

Given these constraints, the objective is to maximize scientific return for the rover – that is, find the plan with maximal utility. Unfortunately, for many rover activities, there is inherent uncertainty about the duration of tasks, the power required, the data storage necessary, the position and orientation of the rover, and environmental factors that influence operations, *e.g.*, soil characteristics, dust on the solar panels, ambient temperature, etc.

For example, in driving from one location to another, the amount of time required depends on wheel slippage and sinkage, which varies depending on slope, terrain roughness, and soil characteristics. All of these factors also influence the amount of power that is consumed. The amount of energy collected by the solar panels during this traverse depends on the length of the traverse, but also on the angle of the solar panels. This is dictated by the slope and roughness of the terrain.

Similarly, for certain types of instruments, temperature affects the signal to noise ratio and, hence, affects the amount of time required to collect useful data. Since the temperature varies depending on the time of day and the weather conditions, this duration is uncertain. The amount of power used depends upon the duration of the data collection. The amount of data storage required depends on the effectiveness of the data compression techniques, which ultimately depends on the nature of the data collected.

In short, this domain is rife with uncertainty. Plans that do not take this uncertainty into account usually fail miserably. In fact, it has been estimated that the 1997 Mars Pathfinder rover spent between 40% and 75% of its time doing nothing because of plan failure.

1. Research Institute for Advanced Computer Science (RIACS).

2. QSS Inc.

One way to attack this problem is to rely on real-time or *reactive* replanning when failures occur. While this capability is certainly desirable, there are several difficulties with exclusive reliance on this approach:

- Spacecraft and rovers have severely limited computational resources due to power limitations and radiation hardening requirements. As a result, it is not always feasible to do timely onboard replanning.
- Many actions are potentially risky and require pre-approval by mission operations personnel. Because of the cost and difficulty of communication, the rover receives infrequent command uplinks (typically one per day). As a result, each daily plan must be constructed and checked for safety well in advance.
- Some contingencies require anticipation; *e.g.*, switching to a backup system may require that the backup system be warmed up in advance. For time critical operations such as orbit insertions or landing operations there is insufficient time to perform these setup operations once the contingency has occurred, no matter how fast the planning can be done.

For these reasons, it is sometimes necessary to plan in advance for potential contingencies – that is, anticipate unexpected outcomes and events and plan for them in advance.

The problem that we have just described is essentially a decision-theoretic planning problem. More precisely, the problem is to produce a (concurrent) plan with maximal expected utility, given the following domain information:

- A set of possible goals that may be achievable, each of which has a value or reward associated with it.
- A set of initial conditions, which may involve uncertainty about continuous quantities like temperature, energy available, solar flux, and position. This uncertainty is characterized by probability distributions over the possible values.
- A set of possible actions, each of which is characterized by:
  - a set of conditions that must be true before the action can be performed. (These may include metric temporal constraints as well as constraints on resource availability.)
  - an uncertain duration characterized by a probability distribution.
  - a set of certain and uncertain effects that describe the world following the action. Uncertain effects on continuous variables are characterized by probability distributions.

Decision-theoretic planning is already known to be quite hard both in theory [18] and in practice. However, there are some characteristics of this domain, which, when taken together, make this planning problem both difficult and different from the kinds of problems that have been studied in the past:

- **Time** – actions take differing amounts of time and concurrency is often necessary.
- **Continuous outcomes** – most of the uncertainty is associated with continuous quantities like time and power. In other words, actions do not have a small number of discrete outcomes.
- **Problem size** – a typical daily plan for a rover will involve on the order of a hundred actions.

While we have described this scenario for a rover, this kind of problem is not limited to robotics or even space applications. For example, in a logistics problem, travel durations are influenced by both traffic and weather considerations. Fuel use is likewise influenced by these "environmental" factors. There are temporal constraints on the availability and delivery of cargo, as well as on the availability of both facilities and crew. There are also constraints on fuel loading and availability, and on maintenance operations.

Previous Work

There has been considerable work in AI on planning under uncertainty. Table 1 classifies much of this work along the following two dimensions:

- **Representation of uncertainty** – whether uncertainty is modeled strictly logically, using disjunctions, or is modeled numerically, with probabilities.
- **Observability assumptions** – whether the uncertain outcomes of actions are not observable, partially observable, or fully observable.

	Disjunction	**Probability**
Non-Observable	CGP [31] CMBP [9, 1] C-PLAN [13, 8] Fragplan [16]	Buridan [17] UDTPOP [23]
Partially-Observable	SENSp [12] Cassandra [25] PUCCINI [14] SGP [34] QBF-Plan [27] GPT [6] MBP [2]	C-Buridan [10] DTPOP [23] C-MAXPLAN [19] ZANDER [19] Mahinur [22] POMDP [7]
Fully-Observable	WARPLAN-C [33] CNLP [24]	JIC [11] Plinth [15] Weaver [4] PGP [3] MDP [7]

Table 1: A classification of planners that deal with uncertainty. Planers in the top row are often referred to as *conformant* planners, while those in the other two rows are often referred to as *contingency* planners.

We do not discuss this work in detail here. A survey of some of this work can be found in Blythe [5]. A more detailed survey of work on MDPs and POMDPs can be found in Boutilier, Dean and Hanks [7]. Instead we will focus on why this

work is generally not applicable to the rover problem and what can be done about this.

There are a number of difficulties in attempting to apply existing work on planning under uncertainty to spacecraft or rovers. First of all, the work listed in Table 1 assumes a very simple model of action in which concurrent actions are not permitted, explicit time constraints are not allowed, and actions are considered to be instantaneous. As we said above, none of these assumptions hold for typical spacecraft or rover operations. These characteristics are not as much of an obstacle for Partial-Order Planning frameworks such as SENSp [12], PUCCINI [14], WARPLAN-C [33], CNLP [24], Buridan [17], UDTPOP [23], C-Buridan [10], DTPOP [23], Mahinur [22] and Weaver [4]. In theory, these systems could represent plans with concurrent actions and complex temporal constraints. The requirements for a rich model of time and action are more problematic for planning techniques that are based on the MDP or POMDP representations, satisfiability encodings, the graphplan representation, or state-space encodings. These techniques rely heavily on a discrete model of time and action. (See [30] for a more detailed discussion of this issue.) Although semi-Markov decision processes (SMDPs) [26] can be used to represent actions with uncertain durations, they cannot model concurrent actions with complex temporal dependencies. The factorial MDP model has recently been developed to allow concurrent actions in an MDP framework. However, this model is limited to discrete time and state representations. Moreover, existing solution techniques are either too general to be efficient on real-world problems (*e.g.* Singh and Cohn [28]), or too domain-specific to be applicable to the rover problem (*e.g.* Meuleau *et al.* [20]).

A second, and equally serious, problem with existing contingency planning techniques is that they all assume that uncertain actions have a small number of discrete outcomes. For example, in the representation popularized by Buridan and C-Buridan, a rover movement action might be characterized as shown in Figure 1. In this representation, each arrow

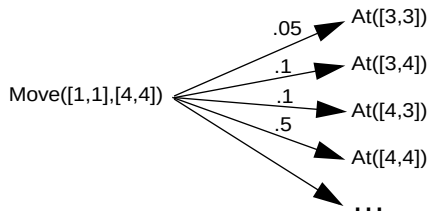


Figure 1: A C-Buridan action for movement.

to a propositions on the right indicates a possible outcome of the action, along with the associated probability of that transition.[3] To characterize where a rover could end up after a move operation, we have to list all the different possible discrete locations. We would need to do something similar to characterize power usage. For most spacecraft and rover activities this kind of discrete representation is impractical – most of the uncertainty involves continuous quantities, such as the amount of time and power an activity requires. Action outcomes are distributions over these continuous quantities. There is some recent work using models with continuous action outcomes in both the MDP [29, 21] and POMDP [32] literature, but this has not yet been applied to SMDPs and has primarily been applied to reinforcement learning rather than planning problems.

Ultimately, the state that results from performing an action determines the future actions that will be taken, so in this sense an action's outcomes are discretized. However, this discretization is not a static property of the actions– instead, it depends on what goals or subgoals the planner is trying to accomplish. For example, suppose that the rover is trying to move to a certain location. If the objective is to place an instrument on a particular rock feature, then the tolerance in position is quite small. In contrast, if the objective is to take a picture from a different vantage point, then the tolerance can be significantly larger.

A third problem with conventional contingency planning technology is that it does not scale to larger problems. Part of the problem is that most of the algorithms attempt to account for all possible contingencies. In effect, they try to produce *policies*. For spacecraft and rover operations, this is not realistic or tractable – a daily plan can involve on the order of a hundred operations, many of which have uncertain outcomes that can impact downstream actions. The resulting plans must also be simple enough that they can be understood by mission operators, and it must be feasible to do detailed simulation and validation on them in a limited time period. This means that a planner can only afford to plan in advance for the "important" contingencies and must leave the rest to run-time replanning. Of the planning systems discussed above, only *Just-In-Case* (JIC) contingency scheduling [11] and Mahinur [22] exhibit a principled approach to choosing what contingencies to focus on. We will discuss this approach in more detail later.

A Detailed Example

In order to illustrate the problem further, in this section we give a detailed example of a very small rover problem. Figure 2 shows a "primary" plan and two potential branches. The primary plan consists of approaching a target point (VisualServo), digging the soil (Dig), backing up (Drive), and taking spectral images of the area (NIR). One potential alternate branch consists of replacing the spectral image with a high-resolution camera image of the target (Hi res). A second potential branch consists of taking a low-resolution panorama of the area (Lo res), performing on-board image analysis to find rocks in the panorama (Rock finder), and then taking spectral images of the rocks found (NIR). For this example, we assume that energy is only being depleted. (More generally, a rover would also be receiving energy input from charging.

3. We have omitted some details here. For each transition, there is a condition that the rover must be at location [1,1] to start with, and that the rover is no longer at [1,1] for each outcome.

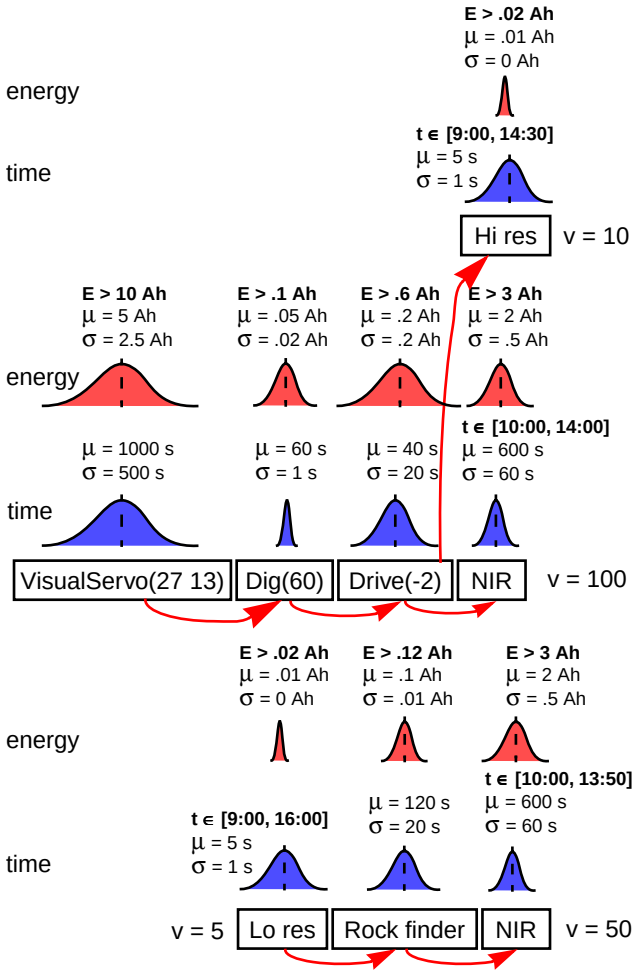


Figure 2: A detailed rover problem. A "main" plan, and two possible alternative branch plans are shown. Probability distributions for time and energy usage are shown for each action. Time and energy constraints for actions are shown in bold.

Precedence constraints are denoted by arrows in the figure; for example, since HiRes can only be performed after Drive, there is an arrow from Drive to HiRes. For each action, there may be preconditions, expectations, and a local utility; in the figure, these appear above the plan actions. The preconditions specify under what conditions execution of the action may start. The expectations describe the expected resource consumption of the actions (in terms of mean and standard deviation); the relative width of distributions is illustrated graphically as well. The local utility is the reward received when the action terminates successfully: in this example, this will be when the preconditions are met and when the energy resource is non-negative at the end of execution.

In the example, consider the HiRes action. It has an energy precondition E > 0.02 Ah and a time precondition of 9:00 ≤ t ≤ 16:00. The expected energy usage is 0.01 Amp-hours (Ah) with a standard deviation of 0 Ah (so in this case there is no uncertainty in the model). The expected duration is 5 seconds with a standard deviation of 1 second. The local utility of the action is v=10.

Approaches

There are several possible ways of attacking this problem of planning with continuous uncertain variables. In this section, we briefly discuss some of these, and the issues that arise.

Computing the Optimal Value Function

Figure 3 shows the optimal value function for the problem in Figure 2. The figure was computed by working backwards

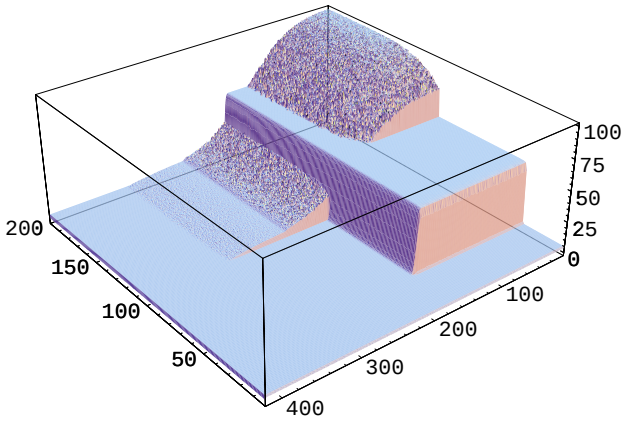


Figure 3: Optimal value function for the example in Figure 2.The left axis is increasing energy from 0 to 20. The right axis is start time from 14:30 down to 13:20. Vertical axis is expected utility.

from all possible activities that have positive reward and using dynamic programming to construct the optimal plan. The curved hump where there is lots of power and time available corresponds to the primary plan, while the rectangular block corresponds to branching to the Rock finder plan and completing the NIR. The tail of the curved hump is a branch after the drive action to the HiRes plan. The flat surface with value 5 is again an immediate branch to the RockFinder plan, but in this area there is not enough power or time to complete the plan, and only the LoRes reward is received. Figure 5 shows a cross-section through this surface for power equal to 11, showing how the various branches contribute to the overall plan. Note that the utility of the overall plan is higher in some places than the value of any original branch. This is because future branch points allow us to wait and see whether a particular plan will succeed, and if it is unlikely to succeed, we can take an alternative branch.

Given a detailed contingent plan and the distributions for time and resource usage, it is relatively straightforward to evaluate the expected utility of the plan. If the distributions are very simple, it may be possible to compute this quantity exactly; more generally, this will have to be done with stochastic simulation. Thus, if we could generate all possible contingent plans for a problem, we could evaluate each of them and choose the one with highest utility. Of course this is completely impractical for problems of any size, partly because it is impossible to enumerate the conditions for con-

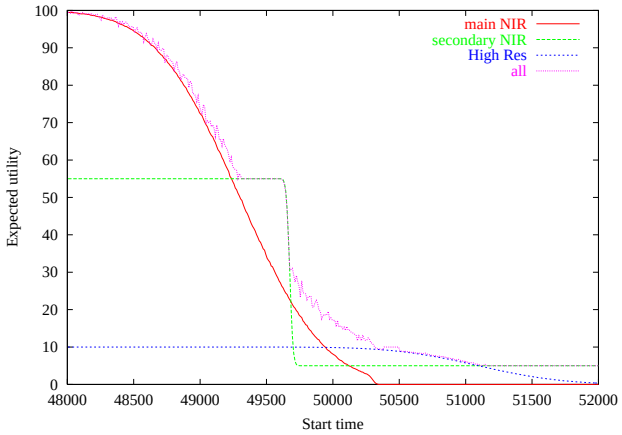


Figure 4: Slice of the optimal value function for energy = 11 Ah, along with the component curves that contribute to the overall utility.

ditional branches. The dynamic programming approach we took above is an alternative, but it too is computationally expensive, and it fails completely when resource availability is not monotonically decreasing (because optimization can no longer be performed through a single backward pass).

Heuristic Approaches

One possibility is to try to plan for the worst case scenario. Thus, in the example from the last section, we could assume that the drive operation requires time and power that is one or perhaps even two standard deviations above the mean. The trouble is, this approach is overly conservative and leads to plans with less science gain than is typically possible. In the example from the previous section, if plan execution was expected to begin at 13:45, this approach would lead us to build a "safe" primary plan that replaces NIR with the HiRes action, with expected utility of 10 in all cases, instead of the more ambitious current primary plan, with expected utility of 0 in the worst case, but 32 in the average case and 100 in the best case.

A more ambitious approach to the problem would be to build an initial plan based on the expected behavior of various activities and then attempt to improve that plan by augmenting it with contingent branches. This is the approach taken by Drummond, Bresina and Swanson with their Just-in-Case (JIC) telescope scheduling [11]. This approach is intuitively simple and appealing, but extending it to problems like the one we have outlined is non-trivial. The primary difficulty is to decide where contingent branches should be added to a plan. In JIC scheduling, branches were added at the points with the greatest probability of plan failure. Given the distributions for time and resource usage this is relatively easy to calculate by statistical simulation of the plan. Unfortunately, the points most likely to fail are not necessarily the points where useful alternatives are available. The points of maximal failure probability may be too late in the plan to have enough time or power left for any useful alternative.

Unfortunately, the problem of finding "high utility" branch points is non-trivial. Figure 5 shows the expected utility over time of the possible plans with a single branch, for a fixed starting energy of 11. Note that at earlier start

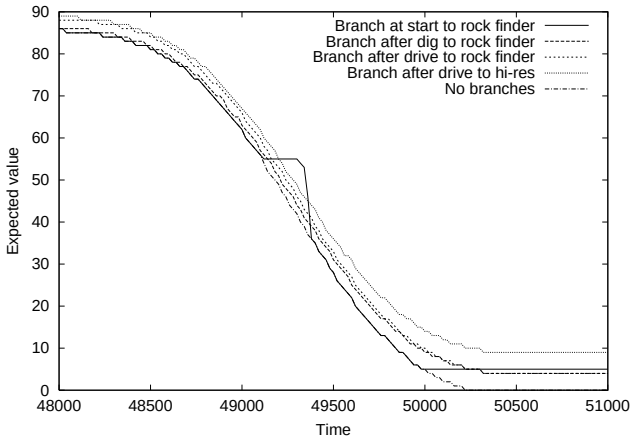


Figure 5: Utility for a single branch at different possible branch points with energy = 11.

times, the plans with the highest expected utility are those that postpone the decision to later in the primary plan, where the possibility of receiving the 100 reward for the NIR action can be more accurately assessed. In a small region, the expected utility of the full RockFinder plan makes that plan more valuable. As time advances, the probability of succeeding in either the primary plan or the full RockFinder plan diminishes, and the HiRes branch becomes the dominant plan. Without the HiRes branch, the early branch to the RockFinder plan (slightly) dominates the other branches late in the time window, since delaying that branch may, with small probability, cause a failure due to energy, resulting in no utility.

Finding the Branch Conditions

Once we've decided to add a branch to a plan, there is still a problem of deciding under what conditions to take the branch. Once again, we could use dynamic programming to compute the optimal conditions, but this suffers from the problems we described above. In addition, as Figure 3 illustrates, the optimal conditions can be extremely complex and hard to represent. The flat surfaces of utility 5 and 55 correspond to branching to the RockFinder plan before the first step of the primary plan. The primary plan (along with the later possible branch to the HiRes plan) is of higher expected utility where the surface is curved. The conditions for the branch point at the beginning of the primary plan are thus the boundaries between the curved surfaces and the flat surfaces. The boundaries are in this case discontinuous, corresponding to a disjunctive condition

It is important to bear in mind that the boundaries are generally places where the values of two different branches are equal, which means that approximate solutions will usually be acceptable here. One possibility is to treat the continuous dimensions of the problem as independent, which results in rectangular regions. This works well in most cases, but the boundaries must be chosen with care where there are

abrupt edges in the value function. This approximation may also fail if there are dependencies between the dimensions, for example when the energy used for driving is dependent on the actual time spent, rather than being treated independently as in our example.

Conclusions

For a Mars rover, uncertainty is absolutely pervasive in the domain. There is uncertainty in the duration of many activities, in the amount of power that will be used, in the amount of data storage that will be required, and in the location and orientation of the rover. Unfortunately, current techniques for planning under uncertainty are limited to simple models of time, and actions with discrete outcomes. In the rover domain there is concurrent action, actions of differing duration, and most of the uncertainty is associated with continuous quantities like time, power, position and orientation.

For any non-trivial problem, it seems unlikely that exact or optimal solutions will be possible. Nor do we have good heuristic techniques for generating effective contingent plans. It seems that new and dramatically different approaches are needed to deal with this kind of problem.

Acknowledgments

Thanks to Tania Bedrax-Weiss, Jeremy Frank, Keith Golden and Sailesh Ramakrishnan for discussions on this subject and comments on drafts of the paper. This research was supported by NASA Ames Research Center and the NASA Intelligent Systems program.

References

1 Bertoli, P.; Cimatti, A.; and Roveri, M. 2001. Heuristic Search + Symbolic Model Checking = Efficient Conformant Planning. *Proc. 17th Int. Joint Conf. on AI*.

2 Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2001. Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking. *Proc. 17th Int. Joint Conf. on AI*.

3 Blum, A, and Langford, J. 1999. Probabilistic Planning in the Graphplan Framework. *Proc. 5th European Conf. on Planning*.

4 Blythe, J. 1998. Planning Under Uncertainty in Dynamic Domains. Ph.D. Dissertation. Carnegie Mellon University.

5 Blythe, J. 1999. Decision-theoretic planning. *AI Magazine* 20(2), 37–54.

6 Bonet, B., and Geffner, H. 2000. Planning with Incomplete Information as Heuristic Se arch in Belief Space. *Proc. 5th Int. Conf. on Artificial Intelligence Planning and Scheduling*, 52–61.

7 Boutilier, C, Dean, T, and Hanks, S. 1999. Decision theoretic planning: structural assumptions and computational leverage. *JAIR* 11, 1–94.

8 Castellini, C., Giunchiglia, E, and Tacchella, A. 2001. Improvements to sat-based conformant planning. *Proc. 6th European Conf. on Planning.*

9 Cimatti, A., and Roveri, M. 2000. Conformant Planning via Symbolic Model Checking. *JAIR* 13, 305–338.

10 Draper, D. Hanks, S., and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. *Proc. 2nd Int. Conf. on AI Planning Systems*, 31–36.

11 Drummond, M, Bresina, J, and Swanson, K. 1994. Just-In-Case scheduling. *Proc. 12th National Conf. on AI*, 1098–1104.

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

13 Ferraris, E. and Giunchiglia, E. 2000. Planning as Satisfiability in Nondeterministic Domains. *Proc. 17th National Conf. on Artificial Intelligence.*

14 Golden, K. 1998. Leap before you look: information gathering in the PUCCINI planner. *Proc. 4th Int. Conf. on AI Planning Systems*, 70–77.

15 Goldman, R. and Boddy, M. 1994. Conditional linear planning. *Proc. 2nd Int. Conf. on AI Planning Systems*, 80–85.

16 Kurien, J., Nayak, P., and Smith, D. 2002. Fragment-based conformant planning, To appear in *Proc. 6th Int. Conf. on AI Planning & Scheduling.*

17 Kushmerick, N., Hanks, S., and Weld, D. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76(1–2), 239–286.

18 Littman, M., Goldsmith, J. and Mundhenk, M. 1998. The computational complexity of probabilistic planning. *JAIR* 9, 1–36.

19 Majercik, S. and Littman, M. 1999. Contingent Planning under Uncertainty via Stochastic Satisfiability. *Proc. 16th National Conf. on AI*.

20 Meuleau, N., Hauskrecht, M, Kim K., Peshkin, L, Kaelbling, L., Dean, T. and Boutilier, C. 1998. Solving very large weakly coupled Markov decision processes. *Proc. 15th Nat. Conf. on AI*. 165–172.

21 Munos, R. 2000. A study of reinforcement learning in the continuous case by the means of viscosity solutions. *Machine Learning Journal* 40, 265–299, 2000.

22 Onder, N. and Pollack, M. 1999. Conditional, Probabilistic Planning: A Unifying Algorithm and Effective Search Control Mechanisms. *Proc. of the 16th National Conf. on AI*, 577–584.

23 Peot, M. 1998. *Decision-Theoretic Planning*. Ph.D. dissertation, Dept. of Engineering-Economic Systems, Stanford U.

24 Peot, M, and Smith, D. 1992. Conditional nonlinear planning, *Proc. 1st Int. Conf. on AI Planning Systems*, 189-197.

25 Pryor, L. and Collins, G. 1996. Planning for contingencies: a decision-based approach. *JAIR* 4, 287–339.

26 Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley.

27 Rintanen, J. 1999. Constructing Conditional Plans by a Theorem Prover, *Journal of Artificial Intelligence Research* 10, 323–352

28 Singh, S. and Cohn, D. 1998. How to dynamically merge Markov decision processes, *Advances in Neural Information Processing Systems* 11.

29 Smart, W. and Kaelbling, L. 2000. Practical reinforcement learning in continuous spaces. *Proc. 17th Int. Conf. on Machine Learning.*

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

31 Smith, D, and Weld, D. 1998. Conformant Graphplan. *Proc. 15th National Conf. on AI*, 889–896.

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

33 Warren, D. 1976. Generating Conditional Plans and Programs. *Proc. Summer Conf. on AI and Simulation of Behavior.*

34 Weld, D, Anderson, C. and Smith, D. 1998. Extending Graphplan to handle uncertainty & sensing actions. *Proc. 15th National Conf. on AI*, 897–904.