

Temporal Planning with Preferences and Time-Dependent Continuous Costs

J. Benton

Dept. of Computer Science and Eng.
Arizona State University
Tempe, AZ 85287 USA
email: j.benton@asu.edu

Amanda Coles and Andrew Coles

Department of Informatics,
King's College London,
London, WC2R 2LS, UK
email: {amanda,andrew}.coles@kcl.ac.uk

Abstract

Temporal planning methods usually focus on the objective of minimizing makespan. Unfortunately, this misses a large class of planning problems where it is important to consider a wider variety of temporal and non-temporal preferences, making makespan a lower-order concern. In this paper we consider modeling and reasoning with plan quality metrics that are not directly correlated with plan makespan, building on the planner POPF. We begin with the preferences defined in PDDL3, and present a mixed integer programming encoding to manage the interaction between the hard temporal constraints for plan steps, and soft temporal constraints for preferences. To widen the support of metrics that can be expressed directly in PDDL, we then discuss an extension to soft-deadlines with continuous cost functions, avoiding the need to approximate these with several PDDL3 discrete-cost preferences. We demonstrate the success of our new planner on the benchmark temporal planning problems with preferences, showing that it is the state-of-the-art for such problems. We then analyze the benefits of reasoning with continuous (versus discretized) models of domains with continuous cost functions, showing the improvement in solution quality afforded through making the continuous cost function directly available to the planner.

1 Introduction

For years, much of the research in temporal planning has worked toward finding plans with the shortest makespan, making the assumption that the utility of a plan corresponds with the time at which it ends. In many problems, however, this does not align well with the true objective. Though it is often critical that goals are achieved in a timely manner, it does not always follow that the shortest plan will be the best in terms of achievement time for *individual* goals. These objectives can occur, for example, when planning for crew activity, elevator operations, consignment delivery, or manufacturing. A few temporal planners (c.f., Gerevini et al. 2006; Coles et al. 2010) are capable of reasoning over similar problems by, for instance, defining hard deadlines. But ranking plans in terms of temporal preferences on plan trajectory or soft deadlines (i.e., those deadlines that can be exceeded, but at a cost) has been less widely explored (Edelkamp, Jabbar, and Nazih 2006).

Copyright © 2012, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

The first challenge we face in considering these problems is how best to define them. PDDL3, introduced during the 5th International Planning Competition (IPC-2006), provides an attractive solution to this. This language introduced a method for modeling soft deadlines and other temporal preferences, where if a deadline or preference is broken, an associated discrete penalty cost is incurred. Discrete models like this have their downsides, however. With deadlines, for instance, when goal achievement occurs after the deadline point, even by a small amount, the full cost must be paid. This fits some situations—for example, arriving at a ferry terminal after the ferry has left. But it mismatches others, such as being one second late in delivering retail goods. In those cases, once the ideal time for an activity has passed, it is still desirable to achieve the goal at some point, though preferably sooner. The cost is continuous and *time-dependent*: zero for a certain amount of time, then progressively increasing. Since both discrete and continuous models of cost have their place, we look toward handling both temporal preferences definable in PDDL3 and time-dependent, monotonically increasing cost functions.

In dealing with these types of problems, we present techniques that build on POPF (Coles et al. 2010), a planner particularly well-suited to handle temporal constraints such as soft deadlines due to its rich temporal reasoning engine. First, we consider PDDL3 preferences. Linear-time scheduling (used by existing approaches) cannot always find a preference-cost-optimal schedule for a given plan, so we use a mixed integer program (MIP) for this. Second, we present an encoding of time-dependent cost in PDDL+ (Fox and Long 2006), and show how the planner can be adapted to support it. In the evaluation we show that the resulting planner, OPTIC (Optimizing Preferences and Time-dependent Costs), has state-of-the-art performance on temporal PDDL3 benchmark domains; and show the direct specification of a continuous cost function is not just elegant, but also offers better performance (with search pruning) than if simply compiled to a single sequence of discrete-cost deadlines.

2 Related Work and Background

While temporal planning has long held the interest of the planning community (c.f., Zeno (Penberthy and Weld 1994), TGP (Smith and Weld 1999), TLPlan (Bacchus and Kabanaz 2000), Sapa (Do and Kambhampati 2003), LPG (Gerevini

et al. 2006), CRIKEY (Coles et al. 2008), TFD (Eyerich et al.)), strong interest in preference-based and partial satisfaction planning (e.g., *net-benefit* planning) is relatively recent (c.f., orienteering planner (Smith 2004), Sapa^{PS} (Benton et al. 2009), GAMER (Edelkamp and Kissmann 2008), soft goal compilation (Keyder and Geffner 2009)). These two areas may appear somewhat disparate in the context of contemporary research, but this is far from the case. Indeed, as more complex temporal problems come within reach of solvability, it becomes less likely that plan duration (i.e., makespan), the usual quality measure for a temporal plan, will continue to be judged an adequate measure of quality.

A few examples of cross-over between the areas have emerged over the years. To our knowledge, the earliest work in this direction is by Haddawy & Hanks (1992), in their planner PYRRHUS, which allows a decision-theoretic notion of *deadline goals*, such that late goal achievement grants diminishing rewards. For several years after, the topic of handling costs and preferences in temporal planning received little attention. As mentioned earlier, in 2006, PDDL3 (Gerevini et al. 2009) introduced a subset of linear temporal logic (LTL) constraints and preferences into a temporal planning framework. PDDL3 provides a quantitative preference language that allowed the definition of temporal preferences within the already temporally expressive language of PDDL 2.1 (Fox and Long 2003). However, few temporal planners have been built to support the temporal preferences available (c.f., MIPS-XXL (Edelkamp et al. 2006), SGPLAN5 (Hsu et al. 2006)), and none that are suitable for temporally expressive domains (Cushing et al. 2007). Other recent work uses the notion of time-dependent costs/rewards in continual planning frameworks (c.f., (Lemons et al. 2010; Burns et al. 2012)).

In helping to remedy this situation, we support both the discrete models of PDDL3 and continuous models in the unifying framework of POPF, a planner capable of solving temporally expressive domains. In the rest of this section, we summarize PDDL3, the continuous cost functions we use, and the planner POPF.

2.1 Preferences in PDDL3

In PDDL2.1 (Fox and Long 2003), a plan quality metric can be specified, with terms comprising the values of task variables (at the end of the plan), and the variable `total-time` denoting makespan. The ability to characterize quality was extended in PDDL3 (Gerevini et al. 2009) with the introduction of preferences. These can be split into two broad categories. First, ‘simple’ preferences correspond to soft goals or soft preconditions on actions. Second, ‘complex’ preferences, written using operators such as `(sometime-after f g)`, that each correspond to a limited fragment of LTL. In both cases, the extent to which violating the preference affects plan cost is captured in the metric, through the use of a variable `(is-violated p)`, which for a precondition preferences counts the number of violations, and for other preferences, takes the value 1 if it has been violated, or 0 otherwise.

A range of planners support PDDL3 preferences, to a lesser or greater extent (Hsu et al. 2006; Edelkamp, Jabbar, and Nazih 2006; Coles and Coles 2011; Baier, Bac-

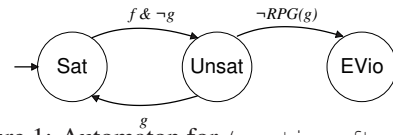


Figure 1: Automaton for `(sometime-after f g)`

chus, and McIlraith 2007; Benton, Do, and Kambhampati 2009). Key to many of these is the representation of preferences as automata, with the position of each preference automaton stored in the state alongside the facts and numeric variable values, a methodology that we also adopt. The update of these is synchronized with the application of actions to states: if a new state meets the condition on a transition out of an automaton’s current position, the transition fires, i.e., its position in the state is updated. As an example, an automaton for `(sometime-after f g)` is shown in Figure 1. If the preference is Sat, and a state S is reached where $S \models (f \wedge \neg g)$, then it moves to being Unsat, i.e., it has been violated. Subsequently, meeting g returns it to Sat; or, if g is provably unreachable, the preference can be marked as eternally violated, denoted E-Vio.

2.2 Time-Dependent Goal Achievement Cost

While PDDL3 preferences can capture the metrics in many temporal planning problems, they cannot cleanly represent continuous increases on cost, despite the ubiquity of real-world problems with this property. As an example, consider a simple logistics problem where blueberries, oranges and apples must be delivered to locations, B , O and A respectively. Each fruit has a different shelf-life. From the time they are harvested, apples last 20 days, oranges 15 days and blueberries 10 days. The truck has a long way to travel, driving with the perishable goods from an origin point P . Let us assume equal profit for the length of time each item is on a shelf. The time to drive between P and B is 6 days, between P and A is 7 days, between B and O is 3 days, and between A and B is 5 days. To make all deliveries, the shortest plan has a duration of 15 days; that is, drive to points A , B , then O in that order. If we were to deliver the goods in this order, the blueberries and oranges will rot before they reach their destinations, and the total time-on-shelf for the apples would be 13 days. Instead, we need a plan gets the best overall value. A plan that drives to point B , O , then A achieves this, though it does so in 17 days. In this case, summing the total time-on-shelf across all fruits gives us 15 days.

To handle these cases, we support two types of time-dependent goal achievement costs: those with deadlines that must be exactly on time, as defined with PDDL3; and those with costs that increase gradually over time. Specifically, in our test problems, we use linearly increasing costs, with a soft deadline after which cost begins to increase, and a second deadline where the full cost is paid. If a goal is achieved between these time points, the cost is determined pro rata.

2.3 Partial Order Planning Forwards

In this work we build on the planner POPF (Coles et al. 2010). The key distinction between POPF and other forward-chaining temporal planners is that rather than enforcing a strict total-order on all steps added to the plan, it builds a

partial-order based on the facts and variables referred to by each step. To support this, each fact p and variable v is annotated with information relating it to the plan steps. Briefly:

- $F^+(p)$ ($F^-(p)$) is the index of the plan step that most recently added (deleted) p ;
- $FP^+(p)$ is a set of pairs, each $\langle i, d \rangle$, used to record steps with a precondition p . i denotes the index of a plan step, and $d \in \{0, \epsilon\}$. If $d=0$, then p can be deleted at or after step i : this corresponds to the end of a PDDL over all condition. If $d=\epsilon$, then p can only be deleted ϵ after i .
- $FP^-(p)$, similarly, records negative preconditions on p .
- $V^{eff}(v)$ gives the index of the step in the plan that most recently had an effect upon variable v ;
- $VP(v)$ is a set containing the indices of steps in the plan that have referred to the variable v since the last effect on v . A step depends on v if it either has a precondition on v ; an effect needing an input value of v ; or is the start of an action with a duration depending on v .

The application of actions to states then updates these annotations and, based on their values, produces ordering constraints. Steps adding p are ordered after $F^-(p)$; those deleting p , after $F^+(p)$. Hence, there is a total-ordering on the effects applied to each fact. Preconditions are fixed within this ordering: applying a step with a precondition p orders it after $F^+(p)$; and recording it in $FP^+(p)$ ensures the next deleter of p will, ultimately, be ordered after it. Similarly, negative preconditions are ordered after some $F^-(p)$ and before the next $F^+(p)$. Finally, steps modifying v are totally ordered, and steps referring to v are fixed within this order (due to effects on v being ordered after the pre-existing $VP(v)$).

An important difference between partially and totally ordered approaches is that the preconditions to support an action are only forced to be true simultaneously if it is added to the plan. Consider a precondition formula F that refers to multiple facts/variables. We say that $S \models F$ if the facts/variable values in S support F . If we apply an action with precondition F we add ordering constraints as discussed above, as otherwise, we could not guarantee the requisite fact/variable values for F are met simultaneously.

For example, consider $F=(a \wedge \neg b)$. In a state where $S \models F$ it is possible that another action, B^+ , adding b can be applied after its last deleter, $F^-(b)$. Since the last adder of a , $F^+(a)$, is not necessarily ordered w.r.t. either $F^-(b)$ or B^+ the plan may be scheduled such B^+ is before $F^+(a)$, and thus $a \wedge \neg b$ is not necessarily true at any point. The key point here is that visiting a state S_i where $S_i \models F$ is not sufficient to guarantee F will be satisfied during the plan. We will see the importance of this when we discuss preferences.

3 Encoding Trajectory Preferences in a Forward-Chaining Partial Order

The objective function in temporal planning is often assumed to be makespan. In the presence of trajectory preferences, however, this is not enough; we must schedule actions based on the costs incurred due to violating these. To do this, we take inspiration from previous work on handling PDDL3

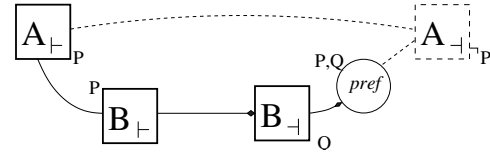


Figure 2: A Dummy Preference Step leading to an Invalid STP

preferences capturing their semantics by building finite-state automata. Using these, we build a mixed integer program (MIP) model to aid in scheduling the plan to minimize cost.

Preferences can be embedded within planning search by augmenting the states with a record of the current position of the corresponding automata. Each time an action is applied, the positions of the automata are updated to reflect the state reached. For instance, if an automaton p has a transition $i \rightarrow j$ conditioned on some logical formula c , and a state is reached that satisfies c and in which $p=i$, the automaton's position is moved to j . This is somewhat similar to adding a dummy step to the plan, with precondition c and its effect being to update the automaton's position, an approach used directly by MIPS-XXL (Edelkamp et. al. 2006).

In the sequential non-temporal case, this encoding works well; all the separation constraints between the steps in the plan (and the dummy preference steps) are of the form $(t(i) < t(j), i < j)$. Thus, the temporal constraints implied by the need to support preferences are trivial, and can never render a plan invalid. In the temporal case, however, actions' duration constraints impose maximum separation constraints: for an action A , its start A_+ cannot precede its end A_- by any more than the maximum duration of A . In Figure 2, for instance, we have applied A_+ which, in adding P , allowed us to apply B_-, B_+ to obtain Q . This allows us to satisfy the condition of a preference (sometime (and (P) (Q))), shown as the dummy-step node $pref$. This node is ordered after the achievers of P and Q , and a record made of this. In the terminology of POPF:

$$\begin{aligned} F^+(P) &= A_+, & FP^+(P) &= \{\langle B_-, \epsilon \rangle, \langle pref, \epsilon \rangle\} \\ F^+(Q) &= B_+, & FP^+(Q) &= \{\langle pref, \epsilon \rangle\} \end{aligned}$$

To complete the plan, we must apply A_- , and without the preference, this would not be an issue: A_- deletes P , so would be ordered ϵ after B_- . Even if B was longer than A , we do not insist that its execution is wholly within A — only its start. With the preference, however, in forcing there to be a point at which both P and Q are true, we would order A_- after $pref$. If the duration of B exceeds that of A , the resulting plan would be temporally invalid: the Simple Temporal Problem (STP) has a negative cycle from A_- back to itself, via $pref$ and B .

The situation of Figure 2 could be resolved by backtracking, applying A_- before B_- , and thus never trying to satisfy the preference. However, to do so would be undesirable: we would rather unsatisfiable preferences did not impact search in this way. Thus, we embed the STP within a MIP, allowing temporal constraints on dummy steps to be relaxed in some (but not all) circumstances, meaning we do not always have to backtrack. We will now explain how we encode soft constraints in an MIP framework, and each constraint's relationship to dummy-step preconditions. Later we explore

how to use these constraints to determine the status of preferences given the ordering constraints maintained by POPF.

3.1 Encoding Soft Constraints for Preconditions

The ordering of a dummy step i with precondition referencing fact f , with respect to actions with an effect on f is fixed: it is ordered between a and d , an adder and deleter of f . To indicate when this ordering has been broken, we create a binary variable f_i^a that takes the value 1 *iff* i precedes a ; and similarly, a variable f_i^d that is 1 *iff* d precedes i . A variable f_i is then defined to be 1 *iff* either of these two variables hold the value 1, i.e., if one of the orderings due to f was broken. This collection of variables and constraints can be stated as follows, where N is some large constant:

$$\begin{aligned} step_i - \epsilon &\geq step_a - N \cdot f_i^a \\ N \cdot f_i^a &\geq step_i - \epsilon - step_a \\ step_i + \epsilon &\leq step_d + N \cdot f_i^d \\ N \cdot f_i^d &\geq step_d - \epsilon - step_i \\ f_i &\geq f_i^a, f_i \geq f_i^d, f_i - f_i^a - f_i^d = 0 \end{aligned}$$

To define $\neg f_i$, a variable that is 1 *iff* the constraints to support $\neg f$ are broken, step a is the deleter before i , and step d the adder after i . For v_i , that is 1 *iff* the constraints to support the value of some variable v are broken, a is the step with an effect on v prior to i , and d the step with an effect on v after i . From here on we refer to such variables as f_i , $\neg f_i$ or v_i , and assume constraints of this form have been included in the MIP if necessary. These variables denote whether orderings to support facts/variable values were satisfied.

We must also include the precondition of each dummy step, in the MIP, as a function of these variables. Any PDDL precondition can be written in Negation Normal Form (NNF), through grounding existential quantification and applying De Morgan's laws. For a precondition formula F , we introduce a variable F_i that holds the value 1 *iff* the satisfaction of temporal constraints at i was not sufficient to guarantee the truth value of F . Constraints on F_i are derived from recursively instantiating MIP encodings of \vee and \wedge , as appropriate, with reference to the necessary f_i , $\neg f_i$ and v_i .

One final remark here is that though $F_i=0$ guarantees that F is true, $F_i=1$ does not guarantee that F is false: it may be that the step i has been scheduled at some point where there are appropriate supporters for its preconditions, just not those chosen during plan expansion.

3.2 Dummy Steps and Encoding Preferences

Having defined F_i for the condition on each dummy step i , we can define when dummy steps are added to the plan, during plan construction, and how these are used to encode preference information in the MIP. Within the MIP we create a binary variable P for each preference P that is 1 *iff* the preference is violated. The dummy plan steps and MIP constraints added are determined on a per-preference basis. First, consider preferences defined by a single desirable formula F . We say a plan step affects formula F if it has at least one effect on facts/variables in F .

1) If P is (always F), a dummy step with condition F is added after each plan step that affects F . $P=1$ *iff* $F_i=1$ for some dummy step i added due to P .

2) If P is (sometime F), a dummy step with condition F is added after each plan step that affects F . $P=0$ *iff* $F_i=0$ for some dummy step i added due to P .

3) If P is (within d F), a similar encoding is used, with an additional auxiliary variable D_i that is 0 *iff* $step_i \leq d$. Then, $P=0$ if $F_i=0 \wedge D_i=0$ for a dummy step i due to P .

4) If step i' of the plan has a precondition preference P with formula F , then a dummy step i with condition F is added prior to i' , constrained such that $step_i = step_{i'}$. P is the sum of all such F_i variables in the plan.

In all four cases, we are being pessimistic: as noted earlier, $F_i=1$ does not mean F is necessarily false at time $step_i$. Thus, search may have to backtrack to circumvent this limitation. Next, consider preferences comprising a desirable formula F , but also an undesirable formula F' :

5) If P is (sometime-after $F' F$), we add a dummy step i with condition F after all steps affecting F . Then, each F' -affecting-step in the plan (leading to state S_n) is followed by a dummy step j . If $S_n \models F'$, j has condition F' and we enforce $F'_j = 0$ in the MIP (i.e. enforce support for F'); otherwise it has condition $\neg F'$ and we enforce $\neg F'_j = 0$. In the MIP we now require:

$$(\exists F'_j \text{ s.t. } \exists i \text{ s.t. } F_i=0 \wedge step_j \leq step_i) \Leftrightarrow P=1$$

6) If P is (sometime-before $F' F$), we have an issue. Suppose, as in the previous case, we added dummy steps i and j for F and F' . Ordering an i before j does not guarantee that F is actually true before F' : if time passes between the actions supporting preconditions of j , and j itself, F' may actually have been true much sooner, and indeed before i . Thus, we limit the *sometime-before* preferences we support: F' can only contain a single term (one fact, one negated fact, or one precondition on a single numeric variable). Then, if F' becomes true after step j , we know step j was responsible for this: it has added/deleted the relevant fact, or appropriately changed the relevant variable. Adding dummy steps, each i , for F , as before, we then constrain the MIP as:

$$(P=1) \Leftrightarrow (\exists j \text{ s.t. } \exists i \text{ s.t. } F_i=0 \wedge step_j \geq step_i + \epsilon)$$

7) If P is (always-within d $F' F$), we add a dummy step i with condition F after all steps affecting F , and restrict F' to a single term. For each step j that made F' true:

$$(P=1) \Leftrightarrow (\exists j \text{ s.t. } \exists i \text{ s.t. } F_i=0 \wedge 0 \leq step_i - step_j \leq d)$$

8) Encoding a preference P (at-most-once $F' F$), in a MIP is a challenge. This is the one preference for which we entirely compromise our ideal of having soft constraints. Each F' -affecting-step in the plan (leading to state S_n) is followed by a dummy step j . If $S_n \models F'$, j has condition F' and we enforce $F'_j = 0$ in the MIP (i.e. enforce support for F'); otherwise it has condition $\neg F'$ and we enforce $\neg F'_j = 0$. Further, each j is ordered to be no earlier than the last dummy step due to P . Insisting on this could make the STP portion of the MIP be unsolvable, as in Figure 2. In the benchmark domains, however, this is not an issue: the ordering constraints due to the dummy steps are subsumed by those introduced due to the relationships between the other steps in the plan.

For (hold-during a b F) and (hold-after d F), we use the timed-initial literal compilation (Gerevini et al. 2009). This reduces them to preferences handled above: respectively, (always (imply (AB) F)) (AB is true in the interval $[a, b]$), and (sometime (and D F)) (D becomes true at time d).

3.3 Calculating Plan Cost

Given the MIP contains variables denoting whether preferences are satisfied, we can set its objective to minimize the cost of the current plan. The MIP will seek to optimise the assignments of timestamps to steps, given the costs of preferences and other terms in the metric, subject to the ordering constraints. The objective needed to do this comprises a constant offset C and a weighted sum of several terms:

1. For each at-end F preference P , if F is false in the current state, increment C by its metric cost $cost(P)$;
2. For each at-most-once F preference P , if the preference is eternally violated (E-Vio) in the current state (i.e., F has become true, false, then true), increment C by $cost(P)$;
3. For each other class of preference P , with cost $cost(P)$:
 - a) if P has a MIP variable, include the term $cost(P).P$;
 - b) otherwise, if P is unsatisfied by default (sometime, within), increment C by $cost(P)$;
4. If the metric contains $(* w (total-time))$, create a dummy MIP variable M , constrained such that each $step_i$ in the plan cannot exceed M . Then include the term $w.M$.
5. For any other metric term $m.v$, increment C by m multiplied by the value of v in the current state.

If plan cost is monotonically worsening, we can also use the MIP to calculate reachable cost, with two small modifications to the objective: exclude from point 3(a) terms added due to sometime-after and always-within preferences; and do not increment C at step 3(b). The cost found reflects that reachable assuming all sometime, within, sometime-after and always-within preferences become satisfied in the future.

3.4 The Remaining Modifications to POPF

So far, we have described how a combination of dummy plan steps and a MIP can be used to monitor and optimize preferences as a plan is expanded forwards in POPF. To guide search effectively, we need a preference-aware heuristic. To this end, we apply the RPG modifications from LPRPG-P (Coles and Coles 2011) to the temporal RPG heuristic of POPF¹. A summary of these modifications follows.

RPG heuristics comprise alternate time-stamped fact and action layers. In the non-temporal setting, time stamps are natural numbers. In POPF’s TRPG, they are related to the durations of actions: if an action A cannot start until action layer t , its end cannot appear until action layer $t + dur_{min}(A)$, where $dur_{min}(A)$ is an admissible estimate of the duration of A . In both cases the fundamental structure and the process of graph expansion is the same. Thus, the five key changes to the RPG described by Coles and Coles for LPRPG-P (2011) can also be applied to a TRPG:

- 1) Each fact layer records the ‘best’ reachable position of each automaton by that point, transitions to better positions fire automatically as and when their preconditions are met. For instance, referring to Figure 1, if fact layer t satisfies g and the preference is Unsat, the preference becomes Sat at the following layer. For the temporal case we add that the

¹Those familiar with LPRPG-P should note that this does not include the LP component of this heuristic; just the general-purpose RPG modifications for the propositional and numeric case.

condition F in (within $d F$) can only be true at RPG layers before d .

- 2) Each action/fact is associated with a preference violation set, recording the preferences violated in applying/achieving it. An action violates a preference if it is a ‘certain trigger’ of a transition from the preference’s current optimistic position, to a position in which it is violated.

- 3) If the automaton position of a preference P moves to a better position, due to the condition g on a transition being met, actions that were previously violated then re-appear with the extra precondition g . If chosen during solution extraction, this then ensures the condition that allowed them to be applied without violating P is met.

- 4) Graph expansion continues while there are still preferences that are not satisfied (or until two identical layers are produced). Any preference that could not be satisfied is marked as E-Vio, i.e. unreachable, in the state being evaluated: it cannot possibly be satisfied in any future state.

- 5) As well as choosing actions to satisfy goals, solution extraction chooses actions to satisfy (reachable) preferences that are unsatisfied in the state being evaluated.

4 Planning with Continuous Cost Functions

PDDL3 preferences cannot directly model continuously changing time-dependent costs on goals. Further, representing such costs using automata may be insufficient and inefficient. In considering problems with continuously changing cost on goals, we therefore face two key challenges:

1. How to best represent planning problems where the value of a plan rests with the time individual goals are achieved.
2. Given a representation, how to solve these problems.

In addressing the first point, we explore using PDDL3 to represent discretizations of the cost function, and representing cost functions directly using a combination of PDDL+ and cost evaluation actions. The semantics of PDDL3 offer an all-or-nothing approach to cost, requiring the generation of a set of deadlines (and internally automata) for the same goal, giving a piece-wise representation of the original cost function. This may be sufficient (or even accurate) for many problems. For example, the London Underground system operates on a fixed schedule, where making a stop 5 minutes late may be no worse than being 3 minutes late: either way the train will depart at the same time. But in other problems, it leaves open questions on the granularity of cost deadlines. For the fruit shelf-life example in Section 2, given a fruit type f and a self-life, sl_f (in days), we can create a set of deadlines such that the cost increases by $1/sl_f$ each day.

An unfortunate disadvantage of this approach is that it may improperly represent costs; for the example, missing the deadline by only a few moments would immediately place the cost in the next day “bracket”, an overly strict requirement for this problem. In this case, a more direct approach to representing cost is desirable. Therefore, we also consider cost represented by a continuous, monotonically increasing function, comprising arbitrary piecewise monotones expressible in PDDL. In our examples, cost is zero until time point t_d , then increases continuously until it reaches a cost c at a time point $t_{d+\delta}$. Our approach removes issues

```

(:action collect-goal-g1 :parameters (?p1 ?p2 - obj)
:precondition (and (goal-g1 ?p1 ?p2) (g1 ?p1 ?p2))
:effect (and (collected-g1 ?p1 ?p2)
            (not (goal-g1 ?p1 ?p2))
            (not (g1 ?p1 ?p2)) *)
(when (> (current-time) (final-deadline-g1 ?p1 ?p2))
  (increase (total-cost) (full-penalty ?p1 ?p2)))
(when (and (> (current-time) (deadline-one-g1 ?p1 ?p2))
          (<= (current-time) (final-deadline-g1 ?p1 ?p2))))
(increase (total-cost)
  (* (full-penalty ?p1 ?p2)
    (/ (- (current-time) (deadline-one ?p1 ?p2))
      (- (final-deadline ?p2 ?p2) (deadline-one-g1 ?p2 ?p2))
    ))))

```

Figure 3: Structure of a Cost-Collection Action for Time-Dependent Cost. (* See note in text)

of granularity for the domain modeler when they are not required.

4.1 Continuous Cost Functions in PDDL+

We can model continuous cost functions using PDDL+ (Fox and Long 2006) without reference to the preference language introduced in PDDL3. First, in order to track the time elapsed throughout the plan we introduce a variable (`current-time`), assigned the value 0 in the initial state. This is updated continuously by a process with no conditions and the effect (`increase (current-time) (* #t 1)`), increasing the value of `current-time` by one per time-unit. As processes execute whenever their conditions are met, and in this case the condition is tautologous, we can now write actions whose effects are dependent on the time at which they are executed.

For each goal fact g_i upon which we want to enforce a time-dependent cost, we add a fact `goal- g_i` to the initial state, and replace the goal with a fact `collected- g_i` . Then we create an action following the template in Figure 3; the action can have arbitrary parameters, as required by the goal, and the cost function can differ for different goals. The line marked with `*` is optional, depending on the semantics required: for goals that we wish to persist after the cost has been collected, the line is present; otherwise, it is not. The conditional effects of our example increase the variable `total-cost` by a linear formula if `current-time` is after `deadline-one- g_i` (i.e., t_d) but before `final-deadline- g_i` and by a fixed amount of `current-time` is after `final-deadline- g_i` (i.e., $t_{d+\delta}$). With additional conditional effects (i.e., intermediate deadlines), the cost function can consist of an arbitrary number of stages, each taking the form of any mathematical function expressible in PDDL. If we restrict our attention to cost functions that monotonically increase (i.e., problems where doing things earlier is always better) we can see that any reasonable planner using this model will apply such actions sooner rather than later to achieve minimal cost.

Note that the actions of Figure 3 can also be modeled as PDDL+ events. Then, as soon as g_i occurred, they would happen. By ensuring there is no gap allowed between g_i and collecting its cost, an event-based encoding would also be a useful part of a mechanism for capturing costs that are not monotonically worsening (as well as eliminating the search step of adding a collect action to the plan).

4.2 Comparison to PDDL 3

The cost functions above (omitting the asterisked effect) have a PDDL3 analog. We can, in theory, obtain the same expressive power by creating a sequence of several `within` preferences, with the spacing between them equal to the greatest common divisor (GCD) of action durations, and each with an appropriate fraction of the cost. In other words, we can define a step function approximation of the cost function using the GCD to define cost intervals. In many problems this could give a substantial blow-up in the size of the problem. A more coarse discretization with `within` preferences that are spaced further apart than the GCD may be more practical. However, a planner using such a model may also fail to reach optimal solutions: it may be possible to achieve a goal earlier but not sufficiently early to achieve the earlier `within` preference, so the planner will not recognize this as an improved plan.

4.3 Solving the Problem

OPTIC handles these problems by extending the POPF scheduler, heuristic and the search strategy. In addition we make a small extension to handle the very basic type of PDDL+ process needed to support the `current-time` ticker. Specifically, processes with static preconditions and linear effects on a variable defined in the initial state (but not subsequently changed by the effect of any other actions). Supporting these requires very little reasoning in the planner.

Scheduling: The compilation (in the absence of support for events) requires that all cost functions be monotonically increasing. Given this (and the absence of preferences and continuous numeric change, other than the ticker) an STN scheduler suffices; we know that the lowest cost for a given plan can be achieved by scheduling all actions at their earliest possible time. The earliest time for each action can be found by performing a single-source shortest path (SSSP) algorithm on the temporal constraints of a plan (this is already done in POPF to check that plans are temporally consistent). When a `collect- g_i` action is first added to the plan we increase the recorded plan cost according to its cost function evaluated at its allotted timestamp. Subsequently, if the schedule of a plan moves `collect- g_i` to a later timestamp, the cost of the plan is increased to reflect any consequential increase in the cost function of the action.

Admissible Heuristic: Now that we can compute the cost of solutions, we need a heuristic to guide search toward finding high-quality solutions; and ideally, an admissible heuristic that we can use for pruning. In satisficing planning, relaxed plan length has been a very effective heuristic, and OPTIC uses this to guide search. We continue to use this for successor selection, but use a second, admissible, heuristic for pruning. Each reachable `collect-cost` action yet to be applied will appear in the TRPG. In OPTIC’s TRPG we can obtain an admissible estimate of each `collect- g_i` ’s achievement time by using its cost at the action layer in which it appears. Since costs are monotonically worsening, this cost is an admissible estimate of the cost of collecting the associated goal. Since `collect- g_i` actions achieve a goal which is never present as a precondition of an action, and they have

	Problem	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Pipes-world	OPTIC	36.4	142.5	92.8	40.3	48	71.2	121.7	75	34.2	94.6	-	70.3	-	41.5	0					
	MIPSXXL	0	78.7	22.2	15.2	23.8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	SGPLAN	36.4	105.1	86	35	48	63.8	95.8	45.9	34.2	-	75.9	30.5	92	88.7	20.3					
	SGPLAN-W	36.4	105.1	86	28.2	34.4	63.8	171.7	67	21.4	69.8	53.8	33.3	92	71.2	12.1					
Trucks	OPTIC	0	0	0	0	0	0	0	6	4	15	13	0	6	21	0	36	54	45	19	73
	MIPSXXL	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	SGPLAN	4	0	2	2	2	1	6	4	4	24	8	6	12	11	8	23	50	29	32	46
	SGPLAN-W	0	1	2	13	127	208	397	477	737	914	1334	1472	2063	2366	3206	3383	-	-	-	-
Storage	OPTIC	0	1	2	13	127	208	397	477	737	914	1334	1472	2063	2366	3206	3383	-	-	-	-
	MIPSXXL	3	4	38	81	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	SGPLAN	18	37	86	113	227	322	505	591	887	1082	1532	1670	2248	2622	3290	3571	4556	4813	6332	6378
	SGPLAN-W	13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
TPP	OPTIC	0	0	0	0	5	0	14	22	-	-	-	26	-	-	-	-	-	-	-	-
	SGPLAN	1	1	0	2	1	8	16	7	29	42	41	40	50	48	36	55	70	65	87	79
Pathways-debugged	OPTIC	3.0	13.6	10	18	23.442	20	27.3	33	29	33.9	31	39.1	40.1	40.3	36.3	52.9	39.7	46.1	54.3	48
	MIPSXXL	2	8.2	11	16.6	25	21	28.3	34	30	34.9	33	-	42.1	42.3	37.3	-	-	-	-	-
	SGPLAN	6.0	12.9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	SGPLAN-W	6.0	12.9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Pathways 20+	Problem	21	22	23	24	25	26	27	28	29	30										
	OPTIC	59.2	50.9	58.4	62.8	67.4	67.1	67.1	73.8	65.6	58.1										

Table 1: Quality of plans produced on IPC-2006 Benchmarks (limited to 30 minutes and 4GB RAM). Smaller is better in all domains except Pipesworld. Absence of a planner in a given domain indicates that it solved no problems.

numeric effects only on cost, they fit the model of *direct-achievement costs* used in the heuristic of POPF (Coles et al. 2011). Thus, the sum of the costs of the outstanding collect actions, at their earliest respective layers, is an admissible estimate of the cost of reaching the remaining goals.

Tiered Search: While searching for a solution, we can use our admissible estimate h_a for pruning. In general, we can prune a state s , reached by incurring cost $g(s)$ (as computed by the scheduler), with admissible heuristic cost $h_a(s)$, if $g(s) + h_a(s) \geq c$, where c is an upper-bound on cost (e.g., the cost of the best solution so far). If the granularity of cost is N , then states are kept if $g(s) + h_a(s) \leq c - N$. In the case of PDDL3, where exceeding deadlines incurs a discrete cost, N is the cost of the cheapest preference. When searching with continuous time-dependent costs, though, N is arbitrarily small, so the number of such states is large. Hence, compared to the discrete-cost case, we are at greater risk of exhausting the available memory. If we inflated N we would prune more states, though forfeit optimality, effectively returning to the discretized case.

As a compromise, we use a tiered search strategy. Specifically, we invoke WA* a number of times in sequence, starting with a larger value of N and finishing with $N=\epsilon$ (some small number). The principle is similar to IDA* (Korf 1985), and a reminiscent of iterative refinement in IPP (Koehler 1998), but applied to pruning on plan quality. That is, introduce an aggressive bound on cost, i.e., assume there exists a considerably better solution than that already found; if this does not appear to be the case, gradually relax the bound. The difference from IDA* comes in the heuristic value used for search: we still use relaxed plan length to guide search, using the admissible cost-based heuristic and cut-off value only for pruning.

5 Results

In our evaluation we first focus on the performance of OPTIC on temporal problems with preferences. We then go on to analyze the difference between the continuous and discretized modeling of continuous time-dependent cost. We focus in our evaluation only on temporal domains. In non-

temporal domains, neither our MIP nor our changes to the heuristic beyond LPRPG-P are required. We refer the reader to Coles and Coles (2011) for an evaluation in this setting.

We consider all temporal domains with preferences from IPC-2006², and compare to the only other two planners supporting such domains: MIPS-XXL and SGPLAN. We include two versions of SGPLAN: SGPLAN 5.21; and SGPLAN-W, the same planner, inside a wrapper that adds a dummy action that can never be grounded, but marginally alters the textual properties of domain (Coles and Coles 2011).

Table 1 shows the best quality metric found within 30 minutes (4GB memory limit) by each of the planners, according to the metric given in the problem files. OPTIC produces the best solution on most problems, bettering even standard SGPLAN, and shows good scalability. Coverage is worse in TPP due to difficulties in finding the first solution plan with OPTIC, regardless of quality. This happens in the larger problems of Pipesworld, too; though if found, the first plan has a good makespan, and hence reasonable quality. In Storage, in the larger storage, grounding preferences exhausted available memory. MIPS-XXL performs admirably on many domains but does not scale as well or produce solutions of as high quality. We note that the performance of SGPLAN-W is significantly worse than that of SGPLAN, solving no problems at all in TPP or Trucks. The only domain in which the performance was equal is Pathways. The debugged domain contains an extra action, and though inapplicable until problem 7, its presence disrupted SGPLAN on all problems, compared to the original domain.

No benchmarks with continuous cost functions exist so we created some based on existing problems; namely, Elevators, Crew Planning and Openstacks, from IPC-2008. In Elevators, deadlines were generated based on greedy, independent solutions for each passenger, thereby generating a “reasonable” wait time for the soft deadline and a partially randomized “priority” time for when full cost is incurred (with the idea that some people are either more important

²Pathways was debugged: self-association reactions no longer have two effects on the same variable, which is illegal in PDDL.

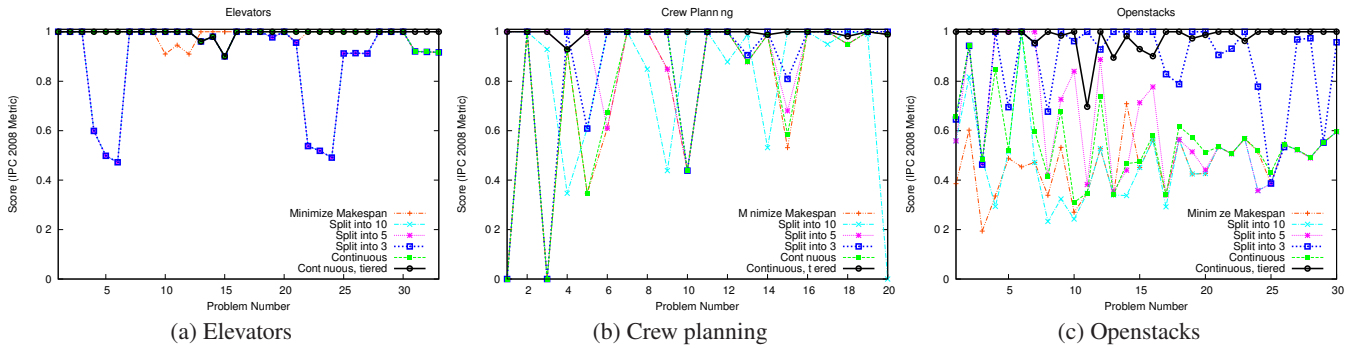


Figure 4: IPC-2008 scores per problem, validated against the continuous cost domain

or more impatient than others.) For each of problems 4–14 from the original problem set (solvable by POPF), we generated three problems. In Crew Planning, for each problem solvable by POPF (1–20) we generated soft deadlines on each crew member finishing sleep, and random deadlines for payloads each day. In Openstacks, each original problem is augmented by soft deadlines based on production durations.

The critical question that we must answer is whether supporting continuous costs is better than using a discretization comprising a series of incremental PDDL3 within preferences. Thus, for each continuous model, we made discretized problems, with each continuous cost function approximated by either 3, 5 or 10 preferences (10 being the closest approximation). With these we use OPTIC, as the best performing planner above to solve the discrete-cost problems. This is compared to OPTIC with the continuous model, and either normal search (only pruning states that cannot improve on the best solution found), or the tiered search described in Section 4.3. In the latter, the value of N was based on the cost Q of the first solution found. The tiers used were $[Q/2, Q/4, Q/8, Q/16, \epsilon]$. Each tier had at most a fifth of the 30 minutes allocated. The results are shown in Figure 4, the graphs show scores calculated as in IPC-2008; i.e. the score on a given problem for a given configuration is the cost of the best solution found (by any configuration) on that problem, divided by the cost of its solution.

First we observe that the solid line, denoting tiered search, has consistently good performance. Compare this to continuous-cost search without tiers; it is worse sometimes in Elevators, often in Crew Planning, and most noticeably in Openstacks. These domains, in left-to-right order, have a progressively greater tendency for search to reach states that could potentially be *marginally* better than the incumbent solution; risking exhausting memory before reaching a state that is *much* better. This is consistent with the performance of the most aggressive split configuration: ‘Split into 3’. In Elevators, and some Crew Planning problems, its aggressive pruning makes it impossible for it (or the other split configurations) to find the best solutions. But, as we move from left-to-right, the memory-saving benefits of this pruning become increasingly important, and by Openstacks, it is finding better plans. Here, too, the split configurations with weaker pruning (5 and 10) suffer the same fate as non-tiered continuous search, memory use limits performance.

From these data, it is clear that the benefit of tiered-

search is that it is effectively performing dynamic discretization. Because we have modeled continuous-costs in the domain, rather than compiling them away, the ‘improvement requirement’ between successive solutions becomes a search-control decision, rather than an artifact of the approximation used. In earlier tiers, search prunes heavily, and makes big steps in solution quality. In later tiers, pruning is less zealous, allowing smaller steps in solution quality, overcoming the barrier caused by coarse pruning. This is vital to close the gap between a solution that is optimal according to some granularity, but not globally optimal. A fixed granularity due to a compilation fundamentally prevents search from finding the good solutions it can find with a tiered approach.

We finally revisit our initial observations—that plan makespan is not always a good analog for plan cost. In Elevators, it appears to be reasonable (likewise in the PDDL3 encoding of the Pipesworld domain earlier in the evaluation). In Crew Planning and Openstacks, though, we see that minimizing makespan produces poor quality solutions; indeed in Openstacks, low makespan solutions are specifically bad.

6 Conclusion

In this paper we have considered temporal planning problems where the cost function is not directly linked to plan makespan. We have introduced new methods for handling PDDL3 preferences in temporal domains and shown that a planner using these can outperform the state-of-the-art in temporal planning with preferences. Further, we have explored temporal problems with continuous cost functions that more appropriately model certain classes of real-world problems and gone on to show the advantages of reasoning with a continuous model of such problems versus a compilation to PDDL3 via discretization. Our final system, OPTIC is capable of handling both of these problems. In future, we intend explore ways of integrating PDDL3 and continuous cost models, and supporting other continuous-cost measures, such as a continuous-cost analog to *always-within*.

Acknowledgments Many thanks go to Patrick Eyerich and Robert Mattmüller for excellent discussions on handling time-dependent costs. This research is supported in part by the Office of Naval Research grants N00014-09-1-0017 and N00014-07-1-1049, the National Science Foundation grant IIS-0905672, DARPA and the U.S. Army Research Laboratory under contract W911NF-11-C-0037 and by EPSRC fellowship EP/H029001/1.

References

- Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 16:123–191.
- Baier, J.; Bacchus, F.; and McIlraith, S. 2007. A heuristic search approach to planning with temporally extended preferences. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*.
- Benton, J.; Do, M. B.; and Kambhampati, S. 2009. Anytime heuristic search for partial satisfaction planning. *Artificial Intelligence* 173:562–592.
- Burns, E.; Benton, J.; Ruml, W.; Do, M.; and Yoon, S. 2012. Anticipatory on-line planning. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*.
- Coles, A. J., and Coles, A. I. 2011. LPRPG-P: Relaxed Plan Heuristics for Planning with Preferences. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*.
- Coles, A. I.; Fox, M.; Long, D.; and Smith, A. J. 2008. Planning with problems requiring temporal coordination. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI)*.
- Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*.
- Coles, A. J.; Coles, A. I.; Clark, A.; and Gilmore, S. T. 2011. Cost-sensitive concurrent planning under duration uncertainty for service level agreements. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*.
- Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. 2007. When is temporal planning *really* temporal planning? In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*.
- Do, M. B., and Kambhampati, S. 2003. Sapa: Multi-objective Heuristic Metric Temporal Planner. *Journal of Artificial Intelligence Research* 20:155–194.
- Edelkamp, S., and Kissmann, P. 2008. GAMER: Bridging Planning and General Game Playing with Symbolic Search. In *IPC6 booklet, ICAPS*.
- Edelkamp, S.; Jabbar, S.; and Nazih, M. 2006. Large-Scale Optimal PDDL3 Planning with MIPS-XXL. In *IPC5 booklet, ICAPS*.
- Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proceedings of 19th International Conference on Automated Planning and Scheduling (ICAPS)*.
- Fox, M., and Long, D. 2003. PDDL2.1: An Extension of PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research* 20:61–124.
- Fox, M., and Long, D. 2006. Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research* 27:235–297.
- Gerevini, A. E.; Long, D.; Haslum, P.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic Planning in the Fifth International Planning Competition: PDDL3 and Experimental Evaluation of the Planners. *Artificial Intelligence* 173:619–668.
- Gerevini, A.; Saetti, A.; and Serina, I. 2006. An Approach to Temporal Planning and Scheduling in Domains with Predictable Exogenous Events. *Journal of Artificial Intelligence Research* 25:187–231.
- Haddawy, P., and Hanks, S. 1992. Representations for decision-theoretic planning: Utility functions for deadline goals. In *Proceedings of the 3rd International Conference of Principles of Knowledge Representation and Reasoning (KR)*.
- Hsu, C.-W.; Wah, B.; Huang, R.; and Chen, Y. 2006. New Features in SGPlan for Handling Preferences and Constraints in PDDL3.0. In *IPC5 booklet, ICAPS*.
- Keyder, E., and Geffner, H. 2009. Soft goals can be compiled away. *Journal of Artificial Intelligence Research* 36:547–556.
- Koehler, J. 1998. Planning under resource constraints. In *Proceedings of the 13th European Conference on Artificial Intelligence*.
- Korf, R. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27:97–109.
- Lemons, S.; Benton, J.; Ruml, W.; Do, M.; and Yoon, S. 2010. Continual on-line planning as decision-theoretic incremental search. In *AAAI Spring Symposium on Embedded Reasoning: Intelligence in Embedded Systems*.
- Penberthy, S., and Weld, D. 1994. Temporal Planning with Continuous Change. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI)*.
- Smith, D. E., and Weld, D. S. 1999. Temporal Planning with Mutual Exclusion Reasoning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*.
- Smith, D. E. 2004. Choosing objectives in over-subscription planning. In *Proceedings of the 14th International Conference on Automated Planning & Scheduling (ICAPS)*.