A Heuristic Search Approach to Planning with Temporally Extended Preferences

Abstract

In this paper we describe a method for planning with TEPs. Our approach involves compiling domains with TEPs into simpler domains containing only simple preferences and metric functions, thus opening the door to a wide range of techniques and systems that have already been developed for planning with simple preferences. We conduct an investigation into how best to find high-quality plans in these reduced simple-preference domains. In particular, we define and evaluate a number of different heuristics for guiding the search for a high-quality plan. We also investigate the use of bounds for pruning the search space and incrementally generating better plans. Our techniques have been implemented in the HPLAN-P planner, which competed in IPC-2006 and achieved distinguished preformance in the TEP track. HPLAN-P was implemented as an extension of the TLPLAN planner. However, these techniques we propose could be easily exploited by most simple-preference planners.

1 Introduction

Standard goals only distinguish between plans that satisfy the goal and those that do not. Such goals provide no way of discriminating between two successful plans. Preferences, on the other hand, express information about how "good" a plan is, thus enabling a planner to distinguish between successful plans that are more, or less desirable. Planning with temporally extended preferences (TEPs), i.e., preferences that refer to the whole execution of the plan, has been the subject of recent research, e.g., [Delgrande *et al.*, 2004; Son and Pontelli, 2004; Bienvenu *et al.*, 2006]. It was also a theme of the 2006 International Planning Competition (IPC-2006). In contrast to TEPs, simple preferences involve only conditions on the plan's final state, and are thus like classical planning goals, except that they are optional.

In this paper we propose techniques for planning with TEPs such as those specified in PDDL3 [Gerevini and Long, 2005]. PDDL3, designed specifically for IPC-2006, is an extension of previous plan specification languages that includes facilities for expressing TEPs, described by a subset of linear temporal logic (LTL). A metric function over simple and temporally extended preferences is then used to quantify the plan's value. The aim in solving a PDDL3 planning instance is to generate a plan with maximum metric value, while also satisfying any hard goals and constraints.

A key insight developed early in our investigation was that, to be effective, a preference-based planner must *actively* guide search towards the achievement of preferences. To this end, we propose a compilation method that reduces a planning problem with TEPs into a new problem containing only simple preferences and some metric functions. The new problem contains some additional domain predicates that emulate the satisfaction of the TEPs in the original problem. Roughly, this means that for any TEP φ, we have a new predicate P_φ that is true in the final state of a plan iff φ was satisfied during the execution of the plan. The advantage of having such a compilation is that the possibly complex process of satisfying a TEP is reduced to the simpler requirement of satisfying an extra classical goal condition. Hence, if we can find ways of adapting existing heuristic search techniques to achieve these extra goal conditions, we would obtain a method for solving planning problems containing TEPs.

Unfortunately, heuristics for hard goals do not apply directly to planning with preferences since preferences are defeasible. Nevertheless, the ideas embedded in existing heuristics for classical goals can be exploited, and a further contribution of this paper is the development of a number of new search heuristics, inspired by existing ideas, that can be used in planning problems with TEPs encoded as simple preferences. These heuristics also work quite well with ordinary simple preferences that are not encoding TEPs.

Using these heuristics, we propose a planning algorithm that incrementally finds better plans. Once a plan is found, its metric value can be used as a bound for future plans: any plan that exceeds this metric value can be pruned from the search space. As a further contribution, we propose heuristic functions that estimate the best value a partial plan can achieve, thus allowing us to prune partial plans from the search space.

We have implemented a planner, HPLAN-P, which uses these techniques to find good quality plans. The planner is built as an extension of the TLPLAN system [Bacchus and Kabanza, 1998],[1] and we have used it to evaluate the performance of a number of different heuristics on problem from the IPC-2006 Qualitative Preferences track.

In what follows, we briefly describe PDDL3. Then we outline our compilation method, the proposed heuristics, the algorithm we used to realize them and the experiments we performed to evaluate them. We conclude with a discussion of system performance and related work.

2 Brief Description of PDDL3

PDDL3 extends PDDL2.1 by allowing the specification of preferences and hard constraints. It also provides a way of

[1]The basic TLPLAN system can use LTL formulae to prune the search space, but it has no mechanism for predicting the future satisfaction or falsification of the LTL formula. Hence it provides no heuristic guidance to the search.

defining a *metric function* that defines the quality of a plan. The rest of this section briefly describes these new elements.

Temporally extended preferences/constraints PDDL3 specifies TEPs and temporally extended hard constraints in a subset of LTL. Both are declared using the `:constraints` construct. Preferences are given names in their declaration, which are used elsewhere to refer to the preference. The following PDDL3 code defines two preferences and one hard constraint.

```
(:constraints
 (and
  (preference cautious
    (forall (?o - heavy-object)
      (sometime-after (holding ?o)
                      (at recharging-station-1))))
  (forall (?l - light)
    (preference p-light (sometime (turn-off ?l))))
  (always (forall ?x - explosive) (not (holding ?x)))))
```

The `cautious` preference suggests that the agent be at a recharging station sometime after it has held a heavy object, whereas `p-light` suggests to eventually turn all the lights off. Finally, the (unnamed) hard constraint establishes that an explosive object cannot be held by the agent at point in a valid plan.

When a preference is *externally* universally quantified, it defines a family of preferences, containing an individual preference for each binding of the variables in the quantifier. Therefore, preference `p-light` defines an individual preference for each object of type `light` in the domain. Preferences that are not quantified externally, like `cautious`, can be seen as defining a family containing a single preference.

Temporal operators, such as `sometime-after` in the example above, cannot be nested in PDDL3. However, our approach can handle the more general case of nested TEPs.

Precondition Preferences
Precondition preferences are atemporal formulae expressing conditions that should ideally hold in the state the action is preformed. They are defined as part of the action's precondition. For example, the `econ` preference below specifies a preference for picking up objects that are not heavy.

```
(:action pickup :parameters (?b - block)
 (:precondition (and (clear ?b)
                     (preference econ (not (heavy ?b)))))
 (:effect (holding ?b)))
```

Precondition preferences behave something like conditional action costs. They are violated each time the action is executed in a state where the condition does not hold. In the above example, `econ` will be violated every time a heavy block is picked up in the plan. Therefore these preferences can be violated a number of times.

Simple Preferences
Simple preferences are atemporal formulae that express a preference for certain conditions to hold in the final state of the plan. They are declared as part of the goal. For example, the following PDDL3 code:

```
(:goal (and (delivered pck1 depot1)
            (preference truck (at truck depot1))))
```

specifies both a hard goal (`pck1` must be delivered at `depot1`) and a simple preference (that `truck` is at `depot1`). Simple preferences can also be externally quantified, in which case they again represent a family of individual preferences.

Metric Function
The metric function defines the quality of a plan, generally depending on the preferences that have been achieved by the plan. To this end, the PDDL3 expression `(is-violated name)`, returns the number of individual preferences in the family of preferences `name` that are violated by the plan. When `name` refers to a precondition preference, the expression returns the *number of times* this precondition preference was violated during the execution of the plan.

The quality metric can also depend on the function `total-time`, which returns the plan length. Finally, it is also possible to define whether we want to maximize or minimize the metric, and how we want to weigh its different components. For example, the PDDL3 metric function:

```
(:metric minimize (+ (total-time)
                     (* 40 (is-violated econ))
                     (* 20 (is-violated truck))))
```

specifies that it is twice as important to satisfy preference `econ` as to satisfy preference `truck`, and that it is less important, but still useful, to find a short plan.

3 Preprocessing PDDL3

The preprocessing phase compiles away many of the more complex elements of PDDL3, yielding an simpler planning problem containing only simple preferences, a metric function that must be *minimized*, and possibly some hard atemporal constraints. In the new problem, the TEPs have been encoded in new domain predicates.

This phase is key in adapting existing heuristic techniques to planning with TEPs. One reason for this is that now the achievement of a TEP is reduced the the simple satisfaction of a domain predicate, i.e., a new optional goal condition. Generating a compact compiled problem is also key for good performance; our compilation achieves this in part by avoiding grounding the planning problem. The rest of this section describes how we do this for each of the PDDL3 elements described in the previous section.

Temporally extended preferences/constraints
We use techniques presented by Baier and McIlraith [2006] to represent the achievement of first-order temporally extended formulae within the planning domain, ending up with a new augmented problem. The new problem contains, for each temporally extended preference or hard constraint φ, a *new* domain predicate that is true in the final state of a plan if and only if the plan satisfied φ during its execution.

The advantage of using such a compilation, is that first-order LTL formulae are directly compiled without having to convert the formula into a possibly very large set of ground instances. As a result, the compiled domain is much more compact, avoiding the exponential blowup that can arise when grounding. As we see later in Section 5, this is key to our planner's performance.

The compilation process first constructs a parameterized nondeterministic finite state automata (PNFA) A_φ for each

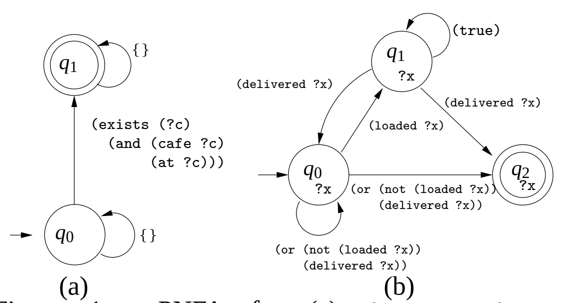

Figure 1: PNFA for (a) `(sometime (exists (?c) (and (cafe ?c) (at ?c))))`, and (b) `(forall (?x) (sometime-after (loaded ?x) (delivered ?x)))`

temporally extended preference or hard constraint expressed as an LTL formula φ.[2] The PNFA represents a family of non-deterministic finite state automata. Its transitions are labeled by sets of first-order formulae. Its states intuitively "monitor" the progress towards satisfying the original temporal formula. An PNFA A_φ accepts a sequence of domain states iff such sequence satisfies φ. Figure 1 shows some examples of PNFA for first-order LTL formulae.

Parameters in the automata appear when the LTL formula is externally quantified (e.g. Figure 1(b)). The intuition is that different *objects* (or tuples of objects) can be in different states of the automata. As an example, consider a transportation domain with two packages, *A* and *B*, which are initially not loaded in any vehicle. Focusing on the formula of Figure 1(b), both objects start off in states q_0 and q_2 of the automata because they are not *loaded* in the initial state. This means that initially both objects satisfy the temporal formula, since both are in the automaton's accepting state q_2. That is, the null plan satisfies the formula (b) of Figure 1. Now, assume we perform the action $load(A, Truck)$. In the resulting state, *B* stays in q_0 and q_2 while *A* now moves to q_1. Hence, *A* no longer satisfies the formula; it will satisfy it only if the plan reaches a state where $delivered(A)$ is true.

To represent the automata within the domain, for each automaton, we define a predicate specifying the automaton's current set of states. When the automaton is parameterized, the predicate has arguments, representing the current set of automaton state for a particular *tuple of objects*. In our example, the fact `(aut-state q0 A)` represents that object `A` is in `q0`. Moreover, for each automaton we define an *accepting predicate*. The accepting predicate is true of a tuple of objects if the plan has satisfied the temporal formula for such a tuple.

As actions are executed, the automata states (as well as properties of the world) need to be updated. To accomplish this we define an *automata update* for each automata. Our planner performs this update automatically after performing any domain action. For the automata of Figure 1(b), the update would include rules such as:

```
(forall (?x) (implies (and (aut-state q0 ?x) (loaded ?x))
                      (add (aut-state q1 ?x))))
```

[2]The construction works for only a subset of LTL [Baier and McIlraith, 2006]. However, this subset includes all of PDDL3's TEPs. It also includes TEPs in which the temporal operators are nested.

That is, an object `?x` moves from state `q0` to `q1` whenever `(loaded ?x)` is true.

In an analogous way, we define an update for the accepting predicate, which is performed immediately after the automata update—if the automata reaches an accepting state then we add the accepting predicate to the world state.

In addition to specifying how the automata states are updated, we need also to specify what objects are in what automata states initially. This means we must augment the problem's initial state by adding a collection of initial automata facts. Given the original initial state and an automaton, the planner computes all the states in which a tuple of objects can be, and then adds the corresponding facts to the new problem. In our example, the initial state of the new compiled problem contains facts stating that both *A* and *B* are in states q_0 and q_2.

If the temporal formula originally described a hard constraint, the accepting condition of the automaton can be treated as additional mandatory goal. During search we also use TLPLAN's ability to incrementally check temporal constraints to prune from the search space those plans that have already violated the constraint.

Precondition Preferences

Precondition preferences are very different from TEPs: they are atemporal, and are associated to the execution of actions. If a precondition preference `p` is violated *n* times during the plan, then the PDDL3 function `(is-violated p)` returns *n*.

Therefore, the compiled problem contains a *new* domain function `is-violated-counter-p`, for each precondition preference `p`. This function keeps track of how many times the preference has been violated. It is (conditionally) incremented whenever its associated action is performed in a state that violates the atemporal preference formula. In the case where the preference is quantified, the function is parameterized, which allows to compute the number of times different objects have violated the preference.

For example, consider the PDDL3 `pickup` action given above. In the compiled domain, the original declaration is replaced by:

```
(:action pickup :parameters (?b - block)
 (:precondition (clear ?b))
 (:effect (and (when (heavy ?b)
                  (increase (is-violated-counter-econ) 1)))
               (holding ?b))) ;; add (holding ?b)
```

Simple preferences

As with TEPs, we add new *accepting predicates* to the compiled domain, one for each simple preference. These predicates become true iff the preference is satisfied. Moreover, if the preference is quantified, they are parameterized.

Metric Function

For each preference, we define a new function `is-violated`. Its value is defined in terms of the accepting predicates (for temporally extended and simple preferences) and in terms of the violation counters (for precondition preferences). If preference `p` is quantified, then the is-violated function counts the number of object tuples that fail to satisfy the preference.

The metric function is then defined just as in the PDDL3 definition but making reference to these new functions. If the objective was to maximize the function we invert the sign of the function body. Therefore, we henceforth assume that the metric is always to be minimizing.

4 Planning with Heuristic Search

With the new compiled problem in hand, we propose several heuristics for planning with preferences using forward search. The rest of this section describes these heuristics, and proposes a planning algorithm for planning with preferences.

4.1 Heuristics for Planning with Preferences

The new augmented planning domain no longer has TEPs. Instead, the domain is much like a standard planning domain. Thus, to compute heuristics for planning with preferences, we use a well-known artifact, the *relaxed planning graph* [Hoffmann and Nebel, 2001]. We can view this graph as composed by *relaxed states*. A relaxed state at depth $n+1$ is generated by applying all the positive effects of actions that can be performed in the relaxed state of depth n.

To compute heuristics for a state s, we expand the relaxed graph starting from state s until all *goal facts* and all *preference facts* appear in the relaxed state. The set of goal facts contains all facts corresponding to the hard goals. The set of preference facts contains the instantiations of the accepting predicates.

Intuitively, any heuristics for planning with preferences should guide the search towards (1) satisfying the goal, and (2) satisfying high-value preferences. Nevertheless, high-value preferences might be very hard to achieve, and therefore this fact should be considered by the heuristics. Below we describe heuristics that intend to incorporate some of these intuitions. Each function addresses part of these intuitions. By combining them, we obtain heuristics that can guide the planner to satisfying both the goal and the preferences.

Goal distance (G) This function is a measure of how hard it is to reach the goal. It is based on a heuristics proposed by Zhu and Givan [2005]. Formally, let $\mathcal{G}$ be the set of goal facts that appear in the relaxed graph. Furthermore, if f is a fact, let $d(f)$ as the depth at which f first appears during the construction of the graph. If all the problem's goal facts are in $\mathcal{G}$, then $G = \sum_{f \in \mathcal{G}} d(f)^k$, where k is a positive, real parameter. Otherwise $G = \infty$.

Preference distance (P) A measure of how hard it is to reach the all preference facts. It is analogous to G but for preferences. Let $\mathcal{P}$ be the set of preference facts the relaxed graph. Then $P = \sum_{f \in \mathcal{P}} d(f)^k$, for a parameter k. Notice the P is not penalized when there are unreachable preference facts.

Optimistic metric (O) An estimation for the metric function of any successor of a partial plan. We compute it assuming no precondition preference will be violated in the future, and that all temporal formulae that are not currently violated by the partial plan will be true in the completed plan. Finally, we assume simple preferences will be satisfied. This function is a variation of the optimistic metric by Bienvenu *et al.* [2006].

Best relaxed metric (B) In the same spirit of the optimistic metric, the best relaxed metric is an estimation of the best metric value that a partial plan can achieve when completed to satisfy the goal. However, this time we use the relaxed planning graph to obtain a better estimate. We compute this value as the best (lowest) metric value of the relaxed worlds. Intuitively, this estimate regards unreachable preference facts as violated by all possible extensions of the partial plan.

Discounted metric ($D(r)$) A weighting of the metric function evaluated in the relaxed states. Assume $s_0, s_1, \ldots, s_n$ are the relaxed states of the graph, where s_i is at depth i. The discounted metric, $D(r)$, is:

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

where $M(s_i)$ is the metric function evaluated in the relaxed state s_i, and r is a discount factor ($0 \le r \le 1$).

Observe that $M(s_{i+1}) - M(s_i)$ is the amount of metric *gained* when passing from state s_i to state s_{i+1}. The D function applies a higher discount to gains occurring deeper in the graph. Therefore, gains due to "hard" preference facts are considered less likely to happen by D.

Final heuristic function Since in planning with TEPs it is mandatory to achieve the goal, the final heuristic function must always guide the search towards the goal. It is natural then to use the G function as the first *priority* of the heuristics, and other heuristics for preferences with a lower priority.

As a generalization of this, the final heuristic can be any lexicographically prioritized sequence of the functions defined above, having G as the first priority. For example, consider prioritization $GD(0.3)O$. When comparing two states of the frontier, the planner first looks at the G function. The best state is the one with lower G value (i.e. lower distance to the goal). However, if there is a tie, then it uses $D(0.3)$ function (the best state being the one with a smaller value). Finally, if there is still a tie, we use the O function. In Section 5, we investigate the effectiveness of several heuristics.

4.2 The Planning Algorithm

The planning algorithm (Figure 2) performs a best-first search in the space of states. Furthermore, it incrementally generates plans with better quality. Before finding the first plan,

```
Input : init: initial state, goal: goal formula, hardConstraints: a
        formula for hard constraints, METRICFN: Metric function,
        USERHEURISTIC: a heuristic function, METRICBOUNDFN:
        function estimating max. metric for a partial plan
begin
  frontier ← INITFRONTIER(init)
  bestMetric ← $\infty$; HEURISTICFN ← $G$
  while frontier $\neq \emptyset$ do
    current ← REMOVEBEST(frontier)
    $f$ ←Progress hardConstraints over to last state of current
    if $f$ is not false then
      if current is a plan and its metric is $<$ bestMetric then
        Output the current plan
        if this is first plan found then
          HEURISTICFN ← USERHEURISTIC
          hardConstraints ← hardConstraints$\wedge$
                always(METRICBOUNDFN $<$ bestMetric)
        bestMetric ← METRICFN(current)
      succ ← EXPAND(current)
      frontier ← MERGE(succ, frontier, HEURISTICFN)
end
```

Figure 2: HPLAN-P's search algorithm.

it uses the goal distance (G) as a heuristic. After finding the first plan, it uses the heuristic provided by the user (USERHEURISTIC), which should be a prioritization of the functions described above.

Additionally, the algorithm prunes a state from the search space if either (1) it has violated a hard constraint, or (2) it will not reach a metric that is better than the one that has been found so far. This happens when f *progresses* to false. Initially, f contains only the hard constraints of the problem. After a plan is found, f additionally requires that METRICBOUNDFN, evaluated in the current state, is strictly better than the best metric found so far. The METRICBOUNDFN function should be an estimation of the metric that could be achieved by a successor of the current partial plan. Therefore, functions such as O or B can be used here. Henceforth, we refer to METRICBOUNDFN as the *pruning strategy*.

The planner returns a plan only if the plan's metric value is better that the best one found so far. The result, is an incremental heuristic planner for TEPs.

4.3 Properties of the Heuristics

We have investigated two relevant properties of the proposed heuristics: whether they are admissible, and whether they are sound for pruning. An admissible heuristic always guides the planner towards an optimal solution (in terms of metric). A sound pruning strategy prunes a plan p from the search space if no successor of p will achieve a metric value that is better than the one that has been found so far.

The admissibility and soundness-for-pruning of the proposed heuristics depend on the structure of the metric function. For example, in the problems of IPC-2006, the metric function is of the form $\sum_{i=0}^{n} c_i \times \texttt{is-violated}(p_i)$, with $c_i > 0$. In this case, we can prove that O, B, and D are all admissible and sound for pruning. On the other hand, the G function is not an admissible heuristic. We can therefore prove that each prioritization of heuristics that uses G is not admissible. Proofs and more details about properties of the proposed heuristics (and combinations of them) under different metric function scenarios are given in the longer version.

5 Implementation and Evaluation

We have implemented the proof-of-concept planner HPLAN-P. The planner consists of two modules. The first is a preprocessor that reads PDDL3 problems (allowing nesting of temporal operators), and generates a TLPLAN output domain. The second module is a modified version of TLPLAN which is able to compute the heuristic functions of Section 4.

As a testbed, we used the problems of the qualitative preferences track of IPC-2006, all of which contain TEPs. The IPC-2006 domains are: *TPP* and *trucks*, which are essentially transportation domains; *openstacks*, which is a production domain; *storage*, which involves moving objects using machines under several restrictions; and finally, *rovers*, which models a rover that must move and collect experiments. Each domain consists of 20 problems. The problems in the trucks, openstacks, and rovers domains have hard goals and preferences. The remaining problems have only preferences. Preferences in these domains impose interesting restrictions on plans, and usually there is no plan that can achieve them all.

To evaluate the effectiveness of pruning (cf. Section 4.2), we compared the performance of three pruning strategies: the optimistic metric, the best relaxed metric, was essential to finding good plans, and in general pruning with the B metric function yielded better results than O. Notably, in most of the openstacks problems it was impossible to improve upon the first plan found without the use of B-function pruning. For example, when using the best-performing heuristic along with B for pruning, the planner was able to improve the first plan found in 18 of 20 problems. However, the same heuristic used with *no* pruning was only able to improve plans in 2 of 20 problems. A similar result was observed in the trucks domain. On the other hand, in the rovers, storage, and the TPP domains, B-function pruning yielded only minimal improvements over no pruning.

A side effect of pruning is that it sometimes can prove that an optimal solution has been found. Indeed, the algorithm stops on most of the simplest problems across all domains (therefore, proving it has found an optimal plan). If no pruning is used, though, the algorithm would generally never stop.

To determine the effectiveness of various final heuristic functions (cf. Section 4.1) we compared 44 heuristics functions using B as a pruning function, allowing the planner to run for 15 minutes over each of the 80 IPC-2006 problem instances. All the heuristics had G as the highest priority (therefore, we omit G from their names). Specifically, we experimented with O, B, OP, PO, BP, PB, OM, MO, and $BD(r)$, $D(r)B$, $OD(r)$, $D(r)O$ for $r \in \{0, 0.01, 0.05, 0.1, 0.3, 0.5, 0.7, 0.9, 1\}$. The M in OM and MO denotes the PDDL3 metric function. Note that the O, OM, and MO heuristics are the only ones that do not use the relaxed planning graph to guide the search towards the satisfaction of the preferences.

Space restrictions preclude us from showing the results on each problem. However, Table 5 shows a summary of the performance of the heuristics. When comparing two heuristics, we consider that one is better than another if it obtains plans of better quality. If tied under that criterion, the best is the one that expands fewer nodes.

In many domains, there are several heuristics tied, therefore there is no overall best-performing heuristic. Moreover, we conclude that the heuristic functions that use the relaxed graph are key to good performance. In all problems, save TPP, the heuristics that used the relaxed graph had the best performance. The case of TPP is pathological in the qualitative preference track: the almost blind heuristic O produced the best performance.

To evaluate the overall performance of our system, we entered our proof-of-concept implementation, HPLAN-P, into the IPC-2006 *Qualitative Preferences* track, achieving 2nd place behind SGPlan$_5$ [Hsu *et al.*, 2006]. Despite HPLAN-P's distinguished standing, SGPlan$_5$'s performance was superior to HPLAN-P's, sometimes finding better quality plans, but generally solving more problems faster. Nevertheless, superior performance was not unique to the preferences tracks. SGPlan$_5$ dominated all 6 tracks of the IPC-2006 *satisficing planner* competition. As such, we conjecture that their superior performance may be substantially attributed to the partitioning techniques they use, which are not specific to plan-

Problem	Found 1 Plan	Found 1+ Plans	Evaluation
openstacks	18	18	**Good:** D-, -D, BP **Bad:** O, OM, MO
trucks	3	3	**Good:** DO, OD, BP **Bad:** OM, MO
storage	16	9	**Good:**BD(r) for $r \leq 0.1$ **Bad:** *O*
rovers	11	10	**Good:** DB, DO for $r \leq 0.5$
TPP	20	20	**Very Good:** O, **Bad:** all the rest

Table 1: Summary of performance of different heuristics in the problems of IPC-2006. The second column shows the number of problems where at least one plan was found. The third, shows how many of these plans were subsequently improved upon by the planner.

ning with preferences. HPLAN-P consistently performed better than MIPS-BDD [Edelkamp *et al.*, 2006] and MIPS-XXL [Edelkamp, 2006]; HPLAN-P can usually find plans of better quality and solve many more problems. Although MIPS-BDD and MIPS-XXL use techniques based on propositional Büchi automata to treat LTL preferences, we think that part of our superior performance can be explained because our compilation does not ground LTL formulae, avoiding blowups.

While we did not enter the *Simple Preferences* track, experiments performed after the competition indicate that HPLAN-P would have also received 2nd place in this track, again behind SGPlan$_5$. In the problems of this track, the best-performing heuristics are also those that use the *D* function.

6 Summary and Related Work

In this paper we presented a suite of techniques for planning with TEPs and hard constraints. Our first contribution was to propose a compilation method that reduces planning problems with PDDL3 LTL preferences into problems containing only final-state preferences. A unique feature of our compiled representation is that it is parameterized, preserving the quantification inherent in PDDL3. With the planning problem conveniently transformed, we proposed a number of heuristics for planning with simple preferences, including a heuristic function that enables sound pruning of the search space during incremental planning. These heuristics were experimentally vetted and integrated into a planning algorithm implemented in the TLPLAN planning system. A significant merit of both the compilation technique and plan heuristics is that we believe they can be easily exploited by any classical or simple-preference planning system.[3]

There is a variety of related work on planning with preferences. Systems by Bienvenu *et al.* [2006] and Son and Pontelli [2004] can plan with TEPs; however, they are far less efficient predominantly because they do not use heuristics to guide search towards achievement of preferences. Brafman and Chernyavsky [2005] proposed a CSP approach to planning with final-state qualitative preferences specified using TCP-nets. *Yochan*$^{\mathcal{PS}}$ [Benton *et al.*, 2006] is a heuristic planner for simple preferences. Our approach is similar to theirs in the sense that both use a relaxed graph to obtain a heuristic estimate. However *Yochan*$^{\mathcal{PS}}$ explicitly selects a subset of preferences to achieve, which can be very costly in the presence of many preferences, it is not incremental, and does not use any pruning. MIPS-XXL [Edelkamp *et al.*, 2006] and MIPS-BDD [Edelkamp, 2006] both use Büchi automata to plan with temporally extended preferences. However, since the LTL formulae need to be grounded it is prone to exponential blow-up. Further, the search techniques used in both of these planners are quite different from those we exploit. SGPlan$_5$ [Hsu *et al.*, 2006] uses a completely different approach. It partitions the planning problem into several subproblems. It then solves them and integrates their solutions. We elaborate further on related work in the final version.

[3]Links to code will be made available in the fi nal version.

References

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

[Baier and McIlraith, 2006] J. A. Baier and S. A. McIlraith. Planning with first-order temporally extended goals using heuristic search. In *AAAI-06*, 2006. To appear.

[Benton *et al.*, 2006] J. Benton, S. Kambhampati, and M. B. Do. YochanPS: PDDL3 simple preferences and partial satisfaction planning. In *IPC-2006*, pp. 54–57, 2006.

[Bienvenu *et al.*, 2006] M. Bienvenu, C. Fritz, and S. McIlraith. Planning with qualitative temporal preferences. In *KR-06*, pp. 134–144, 2006.

[Brafman and Chernyavsky, 2005] R. Brafman and Y. Chernyavsky. Planning with goal preferences and constraints. In *ICAPS-05*, pp. 182–191, 2005.

[Delgrande *et al.*, 2004] J. P. Delgrande, T. Schaub, and H. Tompits. Domain-specific preferences for causal reasoning and planning. In *ICAPS-04*, pp. 63–72, 2004.

[Edelkamp *et al.*, 2006] S. Edelkamp, S. Jabbar, and M. Naizih. Large-scale optimal PDDL3 planning with MIPS-XXL. In *IPC-2006*, pp. 28–30, 2006.

[Edelkamp, 2006] S. Edelkamp. Optimal symbolic PDDL3 planning with MIPS-BDD. In *IPC-2006*, pp. 31–33, 2006.

[Gerevini and Long, 2005] A. Gerevini and D. Long. Plan constraints and preferences for PDDL3. Tech. Rep. 2005-08-07, University of Brescia, 2005.

[Hoffmann and Nebel, 2001] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.

[Hsu *et al.*, 2006] C.-W. Hsu, B. Wha, R. Huang, and Y. Chen. Handling soft constraints and goals preferences in SGPlan. In *IPC-2006*, pp. 39–41, 2006.

[Son and Pontelli, 2004] T. C. Son and E. Pontelli. Planning with preferences using logic programming. In *LPNMR-04*, number 2923 in LNCS, pp. 247–260. Springer, 2004.

[Zhu and Givan, 2005] L. Zhu and R. Givan. Simultaneous heuristic search for conjunctive subgoals. In *AAAI-05*, pp. 1235–1241, 2005.