A Heuristic Search Approach to Planning with Temporally Extended Preferences

Jorge A. Baier and **Fahiem Bacchus** and **Sheila A. McIlraith**
Department of Computer Science
University of Toronto
Toronto, Canada

Abstract

In this paper we propose a suite of techniques for planning with temporally extended preferences (TEPs). To this end, we propose a method for compiling TEP planning problems into simpler domains containing only final-state (simple) preferences and metric functions. With this simplified problem in hand, we propose a variety of heuristic functions for planning with final-state preferences, together with an incremental best-first planning algorithm. A key feature of the planning algorithm is its ability to prune the search space. We identify conditions under which our planning algorithm will generate optimal plans. We implemented our algorithm as an extension to the TLPLAN planning system and report on extensive testing performed to evaluate the effectiveness of our heuristics and algorithm. Our planner, HPLAN-P, competed in the 5th International Planning Competition, achieving distinguished performance in the *qualitative preferences* track.

1 Introduction

Standard goals enable a planner to distinguish between plans that satisfy goals and those that do not. They provide no further means of discrimination between successful plans. Preferences, on the other hand, convey information about how "good" a plan is, thus enabling a planner to distinguish between successful plans of differing quality. *Simple preferences* express preferences over properties of the final state of a plan, while *temporally extended preferences* (TEPs) refer to properties of the whole plan. Planning with TEPs 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 5th International Planning Competition (IPC-5).

In this paper we propose techniques for planning with TEPs such as those specified in PDDL3 [Gerevini and Long, 2005]. PDDL3, is a planning domain definition language designed specifically for IPC-5. It extends PDDL2.2 to include, among other things, facilities for expressing TEPs, described by a subset of linear temporal logic (LTL). A metric function over simple and TEPs is then used to quantify the plan's quality. The aim in solving a PDDL3 planning instance is to generate a plan that satisfies the hard goals and constraints while optimizing the metric function.

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 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 problem of satisfying a simple preference, i.e., a final-state preference. Hence, we observed that if we could find ways of adapting existing heuristic search techniques to achieve simple preferences, we would obtain a method for solving planning problems containing TEPs.

Unfortunately, heuristics for classical planning goals are not directly applicable to planning with simple preferences. A contribution of this paper is the development of a number of new search heuristics for planning with simple preferences, that we exploit for planning problems with TEPs encoded as simple preferences.

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. We also prove that under certain fairly natural conditions our algorithm can generate optimal plans.

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]. We used our implementation to evaluate the performance of our algorithm and the relative performance of different heuristics on problems from both the IPC-5 Simple and Qualitative Preferences tracks.

In what follows, we briefly describe PDDL3. Then we outline our compilation method, the proposed heuristics, the al-

[1] The basic TLPLAN system uses LTL formulae to express *domain control knowledge*; thus, LTL formulae serve to prune the search space. Nevertheless, it has no mechanism for predicting the future satisfaction or falsification of the LTL formula, providing no heuristic guidance to the search.

gorithm 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 PDDL2.2 by enabling the specification of preferences and hard constraints. It also provides a way of 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, to allow for later reference. By way of illustration, 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 that the agent eventually turn all the lights off. Finally, the (unnamed) hard constraint establishes that an explosive object cannot be held by the agent at any 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 in which the action is preformed. They are defined as part of the action's precondition. For example, the preference labeled `econ` 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 `name` family of preferences 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 a simpler planning problem containing only simple preferences, a new metric function that must be *minimized* that only refers to those simple preferences, 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 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

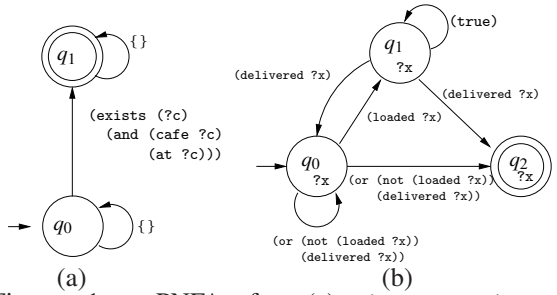

Figure 1: PNFA for (a) `(sometime (exists (?c) (and (cafe ?c) (at ?c))))`, and (b) `(forall (?x) (sometime-after (loaded ?x) (delivered ?x)))`

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 temporally extended preference or hard constraint expressed as an LTL formula φ.[2] The PNFA represents a family of nondeterministic 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. A PNFA A_φ accepts a sequence of domain states iff such a 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))))
```

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

Analogously, 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 with 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`.

[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.

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 minimized.

4 Planning with Heuristic Search

After applying the preprocessing phase described above we are left with a planning problem containing only simple preferences. We propose to solve this problem with a novel combination of heuristic search techniques.

Heuristic search has been very successful in solving classical planing problems where the conjunction of all goals must be achieved. In our case, however, it is generally impossible to satisfy all preferences. Instead the planner must try to achieve a "good" subset of the preferences. In particular, this subset of preferences must be jointly achievable and must yield a preferred value. Some planners have used techniques for selecting, during search, a subset of preferences and then solving that subset as a classical goal using standard planning heuristics (e.g., *Yochan*$^{\mathcal{PS}}$ [Benton *et al.*, 2006]). However, this introduces the non-trivial problem of first selecting such a subset.

Our approach is to utilize a unified heuristic search technique that attempts to tradeoff preference desirability and ease of achieving during the search for a plan. Another important factor is that in addition to the preferences the planning problem generally contains a classical goal, that *must* be achieved. Hence, the search must give priority to achieving the hard goal.

To solve this problem of tradeoffs we have developed an iterative planning technique that uses a sequence of heuristically guided planning episodes. Instead of selecting the preferences we want to satisfy, we simply ask the planner to find a better plan in each planning episode.

Turning to the details we first present the suite of heuristics we have developed to use within each planning episode, and then we explain how we control the sequence of planning episodes.

4.1 Heuristics for Planning with Preferences

Many of our heuristics are based on the well-known technique of computing a *relaxed planning graph* [Hoffmann and Nebel, 2001]. We can view this graph as composed of *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 (i.e., by ignoring delete lists and by applying all of the actions in one step).

Most of the heuristics given below are computed for a state s by constructing the relaxed graph starting at s and growing this graph until all *goal* facts and all *preference* facts appear in the relaxed state, or we reach an empirically determined bound on the depth of the relaxed states. The goal facts correspond to the hard goals, and the preference facts correspond to instantiations of the accepting predicates used to convert TEPs to simple preferences (as described above).

Goal distance function (G) This function is a measure of how hard it is to reach the goal. It is based on a heuristic 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 $d(f)$ be 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 function (P) This function is a measure of how hard it is to reach the various preference facts. It is analogous to G but for preferences. Let $\mathcal{P}$ be the set of preference facts that appear in the relaxed graph. Then $P = \sum_{f \in \mathcal{P}} d(f)^k$, for a parameter k. Notice that P is not penalized when there are unreachable preference facts since in general the plan will not achieve all preferences.

Optimistic metric function (O) This is an estimate of the value achievable by any plan extending the partial plan reaching s. O does not require constructing the relaxed planning graph. Rather it is computed by evaluating the metric function in s assuming no precondition preference will be violated in the future, and that all unviolated preferences will be achieved in the future. Under the condition that the metric function is non-increasing in the number of achieved preferences, O will be a lower bound on the best plan extending s. O is a variant of *"optimistic weight"* Bienvenu *et al.* [2006].

Best relaxed metric function (B) B is another estimate of the value achievable by extending s. B utilizes the relaxed planning graph to obtain its estimate. In particular, we evaluate the metric function in each of the relaxed worlds of the planning graph and take B to be the minimum among these values. B will generally yield a better (higher) estimate of the optimal value achievable by extending s since it will regard preferences that do not appear in the relaxed states as being unsatisfiable. Under a slightly more complex method for building the relaxed planning graph B can be guaranteed to be a lower bound under the same condition as O (i.e., that the preference metric is non-increasing in the number of achieved preferences). However, this technique was not used in the empirical results reported. Hence in our experiments B was not a guaranteed lower bound.

Discounted metric function ($D(r)$) 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, r is a discount factor ($0 \leq r \leq 1$).

The D function is optimistic with respect to preferences that seem easy and pessimistic with respect to preferences that look hard. Intuitively, the D function estimates the metric of a plan that is a successor of the planning state by "believing" more in the satisfaction of preferences that appear to be easier. Observe that $M(s_{i+1}) - M(s_i)$ is the amount of metric value *gained* when passing from state s_i to state s_{i+1}.

This amount is then multiplied by r^i, which decreases as i increases. Observe also that, although the metric gains are discounted, preferences that are weighted higher in the PDDL3 metric will also have a higher impact on the value of D. That is, D achieves the desired tradeoff between the ease of achieving a preference and the value of achieving it.

A computational advantage of the D function is that it is very easy to compute. As opposed to other approaches, this heuristic never needs to make an explicit selection of the preferences to be pursued by the planner.

Sequence of Planning Episodes

When search is started (i.e., no plan has been found), the algorithm uses the goal distance function (G) as its heuristic in a standard *best first search*. The other heuristics are ignored in this first planning episode. This is motivated by the fact that the goal is a hard condition that must be satisfied. In some problems the other heuristics (that guide the planner towards achieving a preferred plan) can conflict with achieving the goal, or might cause the search to become too difficult.

After finding the first plan, the algorithm restarts the search from scratch, but this time it uses some combination of the above heuristics to guide the planner towards a preferred plan. Let USERHEURISTIC() denote this combination. USERHEURISTIC() could be any combination of the above heuristic functions. Nevertheless, in this paper we consider only a small subset of all possible combinations. In particular, we consider only *prioritized* sequences of heuristics, where the lower priority heuristics are used only to break ties in the higher priority heuristics.

Since achieving the goal remains mandatory, USERHEURISTIC() always uses G as the first priority, and with some of the other heuristics at a lower priority. For example, consider the prioritization sequence $GD(0.3)O$. 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 $D(0.3)$ (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 such prioritized heuristics sequences.

One other component of the planning algorithm is that we utilize a scheme for caching relaxed states (and the heuristic computed at these states) so that we can short-circuit the relaxed planning graph construction if the same relaxed state is encountered again. Since constructing the relaxed states can be expensive this caching scheme yields a useful speedup.

Increasing Plan Quality Once we have completed the first planning episode (using G) we want to ensure that each subsequent planning episode yields a better plan.

This is achieved by using increasingly restrictive pruning in each planning episode. In particular, in each planning episode the algorithm prunes from the search space any state s that we estimate cannot reach a better plan than the best plan found so far. This estimate is provided by the function METRICBOUNDFN() which is given as an argument to the search algorithm. METRICBOUNDFN(s) must compute or estimate a lowerbound on the metric of any plan extending s. We have used two of the above heuristics, B and O, for this bounding function.

```
Input : init: initial state, goal, hardConstraints: a set of hard
        constraints, USERHEURISTIC(): a heuristic function,
        METRICBOUNDFN(): function estimating metric for a partial
        plan
begin
  bestMetric ← worst case upper bound; HEURISTICFN ← G
  frontier ← INITFRONTIER(init)
  while frontier ≠ ∅ do
    current ← REMOVEBEST(frontier)
    f ←Evaluate hardConstraints in 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()
          frontier ← INITFRONTIER(init) [search restarted]
          hardConstraints ← hardConstraints∪
                {always(METRICBOUNDFN() < bestMetric)}
        bestMetric ← METRICFN(current)
      succ ← EXPAND(current)
      frontier ← MERGE(succ,frontier,HEURISTICFN)
end
```

Algorithm 1: HPLAN-P's search algorithm.

PDDL3 domains may also contain hard constraints. Hence, in addition to pruning by bounding, the algorithm prunes from its search space any state that violates a hard constraint.

Putting the above together we obtain the Algorithm 1.

4.2 Properties of the Algorithm

We can show that under certain conditions our search algorithm is guaranteed to return an *optimal* or a *k-optimal* solution. It is important to note that our conditions impose no restriction on the USERHEURISTIC() function. In particular, we can still ensure optimality even if this function is inadmissible. In planning this is important, as inadmissible heuristics are typically required for adequate search performance.

We require in our proofs that METRICBOUNDFN(s) be a lower bound on the metric value of the optimal plan extending s. In this case we say that the pruning is *sound*. When sound pruning is used, optimal plans are never pruned from the search space. Therefore, we can be sure no state that leads to an optimal plan will be discarded by the algorithm. Moreover, optimality can be guaranteed when the algorithm stops.

Lemma 1 *If Algorithm 1 terminates and a sound (or no) pruning has been used, then the last plan returned, if any, is optimal.*

Proof: Each planning episode has returned a better plan, and the algorithm stops only when the final planning episode has rejected all possible plans. Since the bounding function never prunes an optimal plan this means that no plan better than the last one returned exists. □

As described above, if the metric function is non-increasing in the number of achieved preferences, O will be a lower bound. As a matter of fact, all metric functions used in IPC-5 are non-increasing in the number of achieved preferences.

Lemma 1 still does not guarantee that an optimal solution will be found because the algorithm might never terminate. To guarantee this, we impose further conditions to sound

pruning. First, we require that the initial value of *bestMetric* (worst case upper bound) be finite. Second, we require of the plan metric function that for every value r less than the initial value of *bestMetric* the number of plans with metric value less then r is finite.

Theorem 1 *If the metric function satisfies the conditions above, the initial value of bestMetric is finite, and a sound (or no) pruning is used, then Algorithm 1 is guaranteed to find an optimal plan, if any exists.*

Proof: Each planning episode only examines plans with metric value less than *bestMetric*. By assumption this is a finite set of plans, so each episode must complete and the algorithm must eventually terminate. Now the result follows from Lemma 1. □

***k*-Optimality** Another condition that our algorithm can achieve is k-optimality. We say that a planning algorithm is k-optimal if it is always able to find an plan that is optimal with respect to the set of plans of length $i \leq k$.

Theorem 2 *If a sound (or no) pruning is used, then Algorithm 1 is k-optimal when search is restricted to plans of length bounded by k.*

Proof: Follows from a similar argument as Lemma 1. □

5 Implementation and Evaluation

We have implemented our ideas in the planner HPLAN-P. HPLAN-P consists of two modules. The first is a preprocessor that reads PDDL3 problems and generates a planning problem with only simple preferences expressed as a TLPLAN domain. The second module is a modified version of TLPLAN that is able to compute the heuristic functions and implements the algorithm of Section 4.

Recall that two of the key elements in our algorithm are the iterative pruning strategy and the heuristics used for planning. In the following subsections we evaluate the effectiveness in obtaining good quality plans using several combinations of the heuristics. As a testbed, we use the problems of the qualitative preferences track of IPC-5, all of which contain TEPs. The IPC-5 domains are composed by two transportation domains: TPP and trucks, a production domain: openstacks, a domain which involves moving objects by using machines under several restrictions: storage, 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.

At the end of the section, we compare our planner against the other planners that participated in IPC-5. The results are based on the data available from IPC-5 [Gerevini *et al.*, 2006] and our own experiments.

5.1 The Effect of Iterative Pruning

To evaluate the effectiveness of iterative pruning we compared the performance of three pruning functions: the optimistic metric, the best relaxed metric, and no pruning at all. From our experiments, we conclude that most of the time pruning produces better results than no pruning, and that, overall, pruning with B usually produces better results than pruning with O.

The impact of pruning varies across different domains. In the TPP domain pruning has a minor effect. Pruning with O produces very slight improvements in the plans' quality (around 1%), whereas pruning with B produces no clear improvement. In the openstacks domain, pruning with O has no effect, whereas pruning with B improves the quality of plans by over 5%. A similar phenomenon occurs in the rovers domain, where O has no effect, but B improves the quality of plans by over 6%. In storage, pruning with O improves the plan's metric by about 1%, whereas B *worsens* the quality by about 1%. Finally, in trucks, pruning with O improves quality by over 10%, and pruning with B by over 15%.

An interesting observation is that, although pruning with B can sometimes remove optimal solutions, in some domains its use is critical to obtaining good performance. Notably, in most of the openstacks problems it was impossible to improve upon the first plan found without the use of B for 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 or with O pruning was able to improve plans in only 2 of 20 problems.

A side effect of pruning is that it can sometimes prove (when the conditions of Lemma 1 are met) that an optimal solution has been found. Indeed, the algorithm stops on most of the simplest problems across all domains. If no pruning was used the search would generally never terminate.

5.2 Performance of Heuristics

To determine the effectiveness of various prioritized heuristic sequences (Section 4.1) we compared 44 heuristic sequences using B as a pruning function, allowing the planner to run for 15 minutes over each of the 80 IPC-5 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.

In general, we say that a heuristic is better than another if it produces plans with better quality, where quality is measured by the metric of the plans. To evaluate how good a heuristic is, we measure the percent improvement of the metric of the last plan found with respect to the metric of the first plan found. Thus, if the first plan found has metric 100, and the last has metric 20, the percent improvement is 80%. Since first plan is always found using G its metric value is always the same, regardless of the heuristic we choose. Hence this measure can be used to objectively compare performance.

Table 1 shows the best and worst performing heuristics in each of the domains tested. In many domains, several heuristics yield very similar performance. Moreover, we conclude that the heuristic functions that use the relaxed graph are key

Domain	1 Plan	>1 Plan	Best heuristics	Worst heuristics
openstacks	18	18	for all r's: DO(r) [9.17], OD(r)[9.17], BP[8.73], MO[8.05], OM[8.05]	PO[7.06], OP[7.06], PB[7.06], O[7.14], B[7.15]
trucks	3	3	PB[96.76], BP[96.76], for $r \geq 3$: DO(r)[96.76], OD(r)[96.76], BD(r)[96.76], DB(r)[96.76]	all remaining[89.75]
storage	16	9	for $r < 0.05$: BD(r)[66.49], OD(0.01)[66.49], BD(0.1)[65.58], OM[65.58]	B[46.96], O[46.96], PO[51.44], PB[51.44], MO[53.00]
rovers	11	10	DO(0.5)[21.91], DB(0.5)[21.41], for $r \in \{.01, .05, .1\}$ DB(r), DO(r)[21.35]	BP[11.35], OP[12.02], PB[15.63],PO[15.63]
TPP	20	20	O[58.69], B[46.92], DO(0.3)[26.93]	PO[12.81], PB [12.81] BD(1)[13.95], BD(0.9)[14.09], BD(0.7)[14.09]

Table 1: Performance of different heuristics in the problems of the *qualitative preferences track* of IPC-5. 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. The average percent metric improvement wrt. the first plan found is shown in square brackets.

to good performance. In all problems, save TPP, the heuristics that used the relaxed graph had the best performance.

5.3 Comparison to Other Approaches

We entered HPLAN-P in the IPC-5 *Qualitative Preferences* track [Gerevini *et al.*, 2006], achieving 2nd place behind SGPlan_5 [Hsu *et al.*, 2007]. 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 and solving them faster. SGPlan_5's superior performance was not unique to the preferences tracks. SGPlan_5 dominated all 6 tracks of the IPC-5 *satisficing planner* competition. As such, we conjecture that their superior performance can be attributed to the partitioning techniques they use, which are not specific to planning with preferences, and that these techniques could be combined with those of HPLAN-P.

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. MIPS-BDD and MIPS-XXL use related techniques, based on propositional Büchi automata, to handle LTL preferences. We think that part of our superior performance can be explained because our compilation does not ground LTL formulae, avoiding blowups, and also because the heuristics are easy to compute. For example, MIPS-XXL and MIPS-BDD were only able to solve the first two problems (the smallest) of the openstacks domain, whereas HPLAN-P could quickly find plans for almost all of them. In this domain the number of preferences was typically high (the third instance already contains around 120 preferences). On the other hand, something similar occurs in the storage domains. In this domain, though, there are many fewer preferences, but these are quantified. More details can be found on the results of IPC-5 [Gerevini *et al.*, 2006].

While we did not enter the *Simple Preferences* track, experiments performed after the competition indicate that HPLAN-P would have done well in this track. To perform a comparison, we ran our planner for 15 minutes[3] on the first 20 instances[4] of each domain. In Table 2, we show the performance of HPLAN-P's best heuristics compared to all other participants, in those domains on which all four planners solved at least one problem. In the table, #S is the number of problems solved by each approach, and *Ratio* is the average ratio between the metric value obtained by the particular planner and the metric obtained by our planner. Thus, values over 1 indicate that our planner is finding better plans, whether values under 1 indicate the opposite.

Domain	HPLAN-P		SGPlan_5		*Yochan*$^{\mathcal{PS}}$		MIPS-BDD		MIPS-XXL	
	#S	Ratio	#S	Ratio	#S	Ratio	#S	Ratio	#S	Ratio
TPP	20	1	20	.82–.86	12	1.1–1.2	9	1–1.1	9	1.99–2.23
trucks	3	1	20	1	13	1	4	1	3	1
storage	20	1	20	.75–.82	4	3.6–4.3	4	0.88–1	4	12.8–15.4
pathways	20	1	20	.81–.82	4	1.13	10	0.88	15	1.33–1.34

Table 2: Relative performance of HPLAN-P's best heuristics for simple preferences, compared to other IPC-5 participants. *Ratio* compares the performance of the particular planner and HPLAN-P's. Ratio > 1 means HPLAN-P is superior, and Ratio < 1 means otherwise. #S is the number of problems solved.

We conclude that HPLAN-P is typically outperformed by SGPlan_5. Although not in the table, in the most simple instances usually HPLAN-P does equally well or better than SGPlan_5. HPLAN-P can solve more instances than those solved by *Yochan*$^{\mathcal{PS}}$, MIPS-XXL and MIPS-BDD. Furthermore, it outperforms *Yochan*$^{\mathcal{PS}}$ and MIPS-XXL in terms of achieved plan quality. HPLAN-P's performance is comparable to that of MIPS-BDD in those problems that can be solved by both planners. Finally, we again observed that the best-performing heuristics in domains other than TPP are those that use the relaxed graph, and, in particular, the D heuristic.

6 Summary and Related Work

In this paper we presented a suite of techniques for planning with TEPs and hard constraints. The techniques are amenable to integration with a variety of classical and simple-preference planners.

Our first contribution was to propose a compilation method that reduces planning problems with PDDL3 LTL preferences into problems containing only simple (i.e., 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

[3] In IPC-5, planners where given 30 min. on a similar machine.

[4] Only the pathways domain has more than 20 problems.

transformed, we proposed a number of heuristics for planning with simple preferences. We also proposed an incremental best-first search planning algorithm, guided by a prioritized sequence of these heuristics. A key feature of our algorithm is that it can use heuristic functions to prune the search space during incremental planning. We proved that under some fairly natural conditions our algorithm can generate optimal plans.

We implemented our algorithm and heuristics as an extension to the TLPLAN planning system and performed extensive experiments on IPC-5 problems to evaluate the effectiveness of our heuristic functions and algorithm. While no heuristic dominated all test cases, several clearly provided superior guidance towards solutions. In particular, those that use the relaxed graph in some way proved to be the most effective in almost all domains. Experiments also confirmed the essential role of pruning in solving large problems. HPLAN-P scales better than many other approaches to planning with preferences. We attribute much of this superior performance to the fact that we do not ground our planning problems.

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. These planners also use a different, qualitative language to describe the quality of a plan that can refer explicitly to numbers, as opposed to the PDDL3 metric function.

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*$^{\mathcal{PS}}$ is not an incremental planner and does not use pruning. To compute its heuristic, it explicitly selects a subset of preferences to achieve. This can be very costly in the presence of many preferences.

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. MIPS-XXL iteratively invokes a modified Metric-FF [Hoffmann, 2003] forcing plans to have decreasing metric values. MIPS-BDD, on the other hand, performs a cost-optimal breath-first search that does not use heuristics.

SGPlan$_5$ [Hsu *et al.*, 2007] uses a completely different approach. It partitions the planning problem into several subproblems. It then solves this problems using heuristics, and then integrates their solutions.

Finally, less related is the approach by Brafman and Chernyavsky [2005] proposed a CSP approach to planning with final-state qualitative preferences specified using TCP-nets. The preferences cannot be temporal.

References

[Bacchus and Kabanza, 1998] F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence*, 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 *Proc. of the 21st National Conference on Artificial Intelligence (AAAI-06)*, pp. 788–795, Boston, MA, 2006.

[Benton *et al.*, 2006] J. Benton, S. Kambhampati, and M. B. Do. YochanPS: PDDL3 simple preferences and partial satisfaction planning. In *5th International Planning Competition Booklet (IPC-2006)*, pp. 54–57, Lake District, England, July 2006.

[Bienvenu *et al.*, 2006] M. Bienvenu, C. Fritz, and S. McIlraith. Planning with qualitative temporal preferences. In *Proc. of the 10th Int'l Conference on Knowledge Representation and Reasoning (KR-06)*, pp. 134–144, Lake District, England, 2006.

[Brafman and Chernyavsky, 2005] R. Brafman and Y. Chernyavsky. Planning with goal preferences and constraints. In *Proc. of the 15th Int'l Conference on Automated Planning and Scheduling (ICAPS-05)*, pp. 182–191, June 2005.

[Delgrande *et al.*, 2004] J. P. Delgrande, T. Schaub, and H. Tompits. Domain-specific preferences for causal reasoning and planning. In *Proc. of the 14th Int'l Conference on Automated Planning and Scheduling (ICAPS-04)*, pp. 63–72, Whistler, Canada, June 2004.

[Edelkamp *et al.*, 2006] S. Edelkamp, S. Jabbar, and M. Naizih. Large-scale optimal PDDL3 planning with MIPS-XXL. In *5th International Planning Competition Booklet (IPC-2006)*, pp. 28–30, Lake District, England, July 2006.

[Edelkamp, 2006] S. Edelkamp. Optimal symbolic PDDL3 planning with MIPS-BDD. In *5th International Planning Competition Booklet (IPC-2006)*, pp. 31–33, Lake District, England, July 2006.

[Gerevini and Long, 2005] A. Gerevini and D. Long. Plan constraints and preferences for PDDL3. Technical Report 2005-08-07, Department of Electronics for Automation, University of Brescia, Brescia, Italy, 2005.

[Gerevini *et al.*, 2006] A. Gerevini, Y. Dimopoulos, P. Haslum, and A. Saetti. 5th International Planning Competition, July 2006. `http://zeus.ing.unibs.it/ipc-5/`.

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

[Hoffmann, 2003] J. Hoffmann. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003.

[Hsu *et al.*, 2007] C.-W. Hsu, B. Wah, R. Huang, and Y. Chen. Constraint partitioning for solving planning problems with trajectory constraints and goal preferences. In *Proc. of the 20th Int'l Joint Conference on Artificial Intelligence (IJCAI-07)*, Hyderabad, India, January 2007. To appear.

[Son and Pontelli, 2004] T. C. Son and E. Pontelli. Planning with preferences using logic programming. In *Proc. of the 7th Int'l Conference on Logic Programming and Nonmonotonic Reasoning (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 *Proc. of the 20th National Conference on Artificial Intelligence (AAAI-05)*, pp. 1235–1241, Pittsburgh, Pennsylvania, USA, July 9-13 2005.