Artificial Intelligence 173 (2009) 593–618

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

A heuristic search approach to planning with temporally extended preferences

Jorge A. Baier [a,b,*], Fahiem Bacchus [a], Sheila A. McIlraith [a]

[a] Department of Computer Science, University of Toronto, Canada
[b] Department of Computer Science, Pontificia Universidad Católica de Chile, Chile

ARTICLE INFO

Article history:
Received 27 October 2007
Received in revised form 28 November 2008
Accepted 28 November 2008
Available online 6 December 2008

Keywords:
Planning with preferences
Temporally extended preferences
PDDL3

ABSTRACT

Planning with preferences involves not only finding a plan that achieves the goal, it requires finding a *preferred* plan that achieves the goal, where preferences over plans are specified as part of the planner's input. In this paper we provide a technique for accomplishing this objective. Our technique can deal with a rich class of preferences, including so-called *temporally extended preferences* (TEPs). Unlike simple preferences which express desired properties of the final state achieved by a plan, TEPs can express desired properties of the entire sequence of states traversed by a plan, allowing the user to express a much richer set of preferences. Our technique involves converting a planning problem with TEPs into an equivalent planning problem containing only simple preferences. This conversion is accomplished by augmenting the inputed planning domain with a new set of predicates and actions for updating these predicates. We then provide a collection of new heuristics and a specialized search algorithm that can guide the planner towards preferred plans. Under some fairly general conditions our method is able to find a most preferred plan—i.e., an optimal plan. It can accomplish this without having to resort to admissible heuristics, which often perform poorly in practice. Nor does our technique require an assumption of restricted plan length or make-span. We have implemented our approach in the HPLAN-P planning system and used it to compete in the 5th International Planning Competition, where it achieved distinguished performance in the *Qualitative Preferences* track.

1. Introduction

Classical planning requires a planner to find a plan that achieves a specified goal. In practice, however, not every plan that achieves the goal is equally desirable. Preferences allow the user to provide the planner with information that it can use to discriminate between successful plans; this information allows the planner to distinguish successful plans based on plan quality.

Planning with preferences involves not just finding a plan that achieves the goal, it requires finding one that achieves the goal while also optimizing the user's preferences. Unfortunately, finding an optimal plan can be computationally expensive. In such cases, we would at least like the planner to direct its search towards a reasonably preferred plan.

In this paper we provide a technique for accomplishing this objective. Our technique is able to deal with a rich class of preferences. Most notably this class includes *temporally extended preferences* (TEPs). The difference between a TEP and a so-called *simple* preference is that a simple preference expresses some desired property of the final state achieved by the

* Corresponding author at: Department of Computer Science, University of Toronto, Canada.
E-mail addresses: jabaier@cs.toronto.edu (J.A. Baier), fbacchus@cs.toronto.edu (F. Bacchus), sheila@cs.toronto.edu (S.A. McIlraith).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2008.11.011

plan, while a TEP expresses a desired property of the sequence of states traversed by the plan. For example, a preference that a shift worker work no more than 2 overtime shifts in a week is a temporally extended preference. It expresses a condition on a sequence of daily schedules that might be constructed in a plan. Planning with TEPs has been the subject of recent research (e.g. [6,12,35]). It was also a theme of the 5th International Planning Competition (IPC-5).

The technique we provide in this paper is able to plan with a class of preferences that includes those that can be specified in the planning domain definition language PDDL3 [23]. PDDL3 was specifically designed for IPC-5. It extends PDDL2.2 to include, among other things, facilities for expressing both temporally extended and simple preferences, where the temporally extended preferences are described by a subset of linear temporal logic (LTL). It also supports quantifying the value of achieving different preferences through the specification of a metric function. The metric function assigns to each plan a value that is dependent on the specific preferences the plan satisfies. The aim in solving a PDDL3 planning instance is to generate a plan that satisfies the hard goals and constraints while achieving the best possible metric value, optimizing this value if possible or at least returning a high value plan if optimization is infeasible.

Our technique is a two part approach. The first part exploits existing work [2] to convert planning problems with TEPs to equivalent problems containing only simple preferences defined over an extended planning domain. The second part, and main contribution of our work, is to develop a set of new heuristics, and a search algorithm that can exploit these heuristics to guide the planner towards preferred plans. Many of our heuristics are extracted from a relaxed planning graph a technique that has previously been used to compute heuristics in classical planning. Previous heuristics for classical planning, however, are not well suited to planning with preferences. The heuristics we present here are specifically designed to address the tradeoffs that arise when planning to achieve preferences.

Our search algorithm is also very different from previous algorithms used in planning. As we will show, it has a number of attractive properties, including the ability to find optimal plans without having to resort to admissible heuristics. This is important because admissible heuristics generally lead to unacceptable search performance. Our method is also able to find optimal plans without requiring a restriction on plan length or make-span. This is important because such restrictions do not generally allow the planner to find a globally optimal plan. In addition, the search algorithm is incremental in that it finds a sequence of plans each one improving on the previous. This is important because in practice it is often necessary to trade off computation time with plan quality. The first plans in this sequence of plans can often be generated fairly quickly and provide the user with at least a working plan if they must act immediately. If more time is available the algorithm can continue to search for a better plan. The incremental search process also employs a pruning technique to make each incremental search more efficient. The heuristics and search algorithm presented here can easily be employed in other planning systems.

An additional contribution of the paper is that we have brought all of these ideas together into a working planning system called HPLAN-P. Our planner is built as an extension of the TLPLAN system [1]. The basic TLPLAN system uses LTL formulae to express *domain control knowledge*; thus, LTL formulae serve to prune the search space. However, TLPLAN has no mechanism for providing heuristic guidance to the search. In contrast, our implementation extends TLPLAN with a heuristic search mechanism that guides the planner towards plans that satisfy TEPs, while still pruning those partial plans that violate hard constraints. We also exploit TLPLAN's ability to evaluate quantified formulae to avoid having to convert the preference statements (many of which are quantified) into a collection of ground instances. This is important because grounding the preferences can often yield intractably large domain descriptions. We use our implementation to evaluate the performance of our algorithm and to analyze the relative performance of different heuristics on problems from both the IPC-5 *Simple* and *Qualitative Preferences* tracks.

In the rest of the paper we first provide some necessary background. This includes a brief description of the features of PDDL3 that our approach can handle. In Section 3 we describe the first part of our approach—a method for compiling a domain with temporally extended preferences into one that is solely in terms of simple (i.e., final state) preferences. Section 4 describes the heuristics and search algorithm we have developed. It also presents a number of formal properties of the algorithm, including characterizing various conditions under which the algorithm is guaranteed to return optimal plans. Section 5 presents an extensive empirical evaluation of the technique, including an analysis of the effectiveness of various combinations of the heuristics presented in Section 4. Section 6 presents a discussion of the approach and Section 7 summarizes our contributions and discusses related work after which we provide some final conclusions.

2. Background

This section reviews the background needed to understand this paper. Section 2.1 presents some basic planning definitions and a brief description of the planning domain definition language PDDL. Section 2.2 describes a variation of the well-known approach to computing domain-independent heuristics based on the computation of relaxed plans that is used by our planner to compute heuristics. As opposed to most well-known approaches, our method is able to handle ADL domains directly without having to pre-compile the domain into a STRIPS domain. Section 2.3 describes the planning domain definition language PDDL3, a recent version of PDDL that enables the definition of hard constraints, preferences, and metric functions.

2.1. An overview of planning formalisms and languages

A classical planning instance is a tuple $\mathcal{I} = (Objs, Oper, Init, Goal)$, where *Objs* is a finite set of objects, *Oper* is a finite set of planning operators, *Init* is the initial *state*, i.e., a finite set of ground literals—or simply, *facts*—describing the initial state, and *Goal* describes the set of goal states.

In STRIPS planning instances [19], the set *Oper* contains operator descriptions of the form $(pre(o), add(o), del(o))$, where $pre(o)$ is a list of precondition facts for operator o, $add(o)$—the *add list*—is a list of facts that are positive effects of operator o, and $del(o)$—the *delete list*—is a list of facts that are negative effects of operator o. Finally, *Goal* is a set of goal facts.

In the more expressive ADL formalism [31], operators still describe preconditions and effects, but these can now be more than simple lists of ground literals. ADL preconditions can be arbitrary boolean formulae, existentially or universally quantified over the set of objects *Objs*. ADL effects can be *conditional*, which means that adds and deletes can be conditioned on the satisfaction of arbitrary boolean formulae. Effects can also be *universal* in the sense that they affect *all* objects that satisfy a certain condition. For example, assume we are describing a domain where objects can contain other objects. Further, assume action $move(x, y, z)$ moves object x from location y to location z and in the process moves all objects in x to z as well. The precondition for this action is just $at(x, y)$; i.e., the object x has to be at location y, while its effects can be defined by the list:

$$Eff = \big\{\textbf{add } at(x, z), \forall v\big[in(v, x) \Rightarrow \textbf{add } at(v, z)\big], \textbf{del } at(x, y), \forall v\big[in(v, x) \Rightarrow \textbf{del } at(v, y)\big]\big\}.$$

Thus, the location of the object x and all objects inside x changes to z.

In addition to more expressive preconditions and effects, ADL also allows for the representation of functions. This means that states can contain, in addition to propositional facts, sentences of the form $f(\vec{c}) = z$, where f is a function name, c is a tuple of objects in *Objs*, and z is an object in *Objs*. Actions can change the functions by assigning $f(\vec{c})$ a different value as an add effect.

Finally, in ADL, *Goal* can be any formula (possibly quantified) that describes a condition that must be satisfied by a goal state. For more details on ADL we refer the reader to [31].

Although STRIPS and ADL can be used to provide formal descriptions of classical planning instances, they cannot be used as a standard input language for planners since their precise syntactical form has never been standardized. The Planning Domain Definition Language (PDDL) [30], on the other hand, was specifically designed to provide a uniform syntax for describing planning problems in the context of the 1998 International Planning Competition. PDDL is currently a *de facto* standard for describing planning problems, and it has been extended and used in all subsequent versions of IPC.

Recent versions of PDDL enable the definition of planning instances in a superset of ADL. For example, PDDL2.1 [20] extends ADL by enabling explicit representation of time. Among other features, it allows the specification of actions with duration. On the other hand, PDDL2.2 [16] extends PDDL2.1 by allowing derived predicates (i.e., predicates defined axiomatically), and timed literals (i.e., literals that will become true at a specified time instant). PDDL3, as we describe in Section 2.3, extends PDDL2.2 with hard constraints, preferences, and metric functions.

The planning problem in both the STRIPS and the ADL settings is the problem of finding a legal sequence of actions—ground operators—that, when executed in the initial state, will lead to a state in which the goal condition *Goal* is satisfied.

2.2. Planning as heuristic search

Many state-of-the-art domain-independent planners use domain-independent heuristics to guide the search for a plan. Heuristics estimate the cost of achieving the goal from a certain state. They can be used with standard search algorithms, and are usually key to good performance. They are typically computed by solving a relaxed version of the original problem. One of the most popular domain-independent relaxations corresponds to ignoring the negative effects of actions. This is the approach taken by many planners (e.g., HSP [8] and FF [27], among others). In the STRIPS formalism, this corresponds to ignoring delete lists.

In this paper we exploit heuristic search to plan with preferences. The heuristics presented here are based on the well-known technique of computing a *relaxed planning graph* [27], which is the graph that would be generated by GRAPHPLAN [7] on the STRIPS relaxed planning instance that ignores negative effects. This graph is composed of fact layers—or *relaxed worlds*—and action layers. The action layer at level n contains all actions that are possible in the relaxed world at depth n. The relaxed world at depth $n+1$ contains all the facts that hold at layer $n+1$ and is generated by applying all the positive effects of actions in action layer n. The graph is expanded until the goal is satisfied by the final relaxed world or a fixed point is reached.

Once the graph is expanded, one can compute a *relaxed* plan for the goals by regression from the goal facts in the graph to the initial state. The length of this plan can then be used as a heuristic estimator of the cost for achieving the goal. In the rest of the paper we assume familiarity with the extraction of relaxed plans. For more details we refer the reader to the article by Hoffmann and Nebel [27].

2.2.1. Relaxed plans for function-free ADL domains

To compute heuristics for function-free ADL domains one can first transform the domain to STRIPS, using a well-known procedure described by Gazen and Knoblock [21], and then compute the heuristic as usual. This is the approach taken

by some systems (e.g. FF) but unfortunately this procedure can lead to a considerable blow up in the size of the original instance.

Our planner handles ADL domains, but takes a different approach. In particular, it computes the relaxed planning graph directly from the ADL instance, using an approach similar to that taken by the MARVIN planning system [11]. To effectively handle relaxed ADL domains (in which effects can be conditioned on negative facts), the relaxed worlds represent both the facts that become *true* and the facts that become *false* after executing a set of actions. To that end, the relaxed worlds are divided into two parts: a positive part, that represents added facts, and a negative part, that represents deleted facts.

When computing a relaxed planning graph for a state s, the set of relaxed worlds is a sequence of pairs of fact sets $(F_0^+, F_0^-), \ldots, (F_n^+, F_n^-)$, with $F_0^+ = s$ and $F_0^- = s^c$, where s^c is the set of facts not in s (i.e., the complement of s). Furthermore, if action a appears in the action layer at depth n, all facts that are added by a are included in the positive relaxed world at depth F_{k+1}^+, whereas facts that are deleted by a are added to F_{k+1}^-. Moreover, all facts in layer k are copied to layer $k+1$ (i.e. $F_n^+ \subseteq F_{k+1}^+$ and $F_k^- \subseteq F_{k+1}^-$).

Special care has to be taken in the evaluation of preconditions and conditions in conditional effects for actions, because negations could appear anywhere in those conditions. To evaluate a formula in a relaxed world, we evaluate its *negation normal form* (NNF) instead. In NNF, all negations appear right in front of atomic formulae. A formula can easily be converted to NNF by pushing negations in using the standard rules $\neg\exists.f \equiv \forall.\neg f$, $\neg\forall.f \equiv \exists.\neg f$, $\neg(f_1 \wedge f_2) \equiv \neg f_1 \vee \neg f_2$, $\neg(f_1 \vee f_2) \equiv \neg f_1 \wedge \neg f_2$, and $\neg\neg f \equiv f$.

Now assume we want to determine whether or not the formula ϕ is true in the relaxed state (F_k^+, F_k^-) in the graph with relaxed worlds $(F_0^+, F_0^-) \cdots (F_k^+, F_k^-) \cdots (F_n^+, F_n^-)$. Furthermore, let ϕ' be the NNF of ϕ. To evaluate ϕ we instead evaluate ϕ' recursively in the standard way, interpreting quantifiers and boolean binary operators as usual. When evaluating a positive fact f, we return the truth value of $f \in F_k^+$. On the other hand, when evaluating a negative fact $\neg f$, we return the truth value of $f \in F_k^-$. In short, $\neg f$ is true at depth k if f was deleted by an action or was already false in the initial state. More formally,

Definition 1 *(Truth of an NNF formula in a relaxed state).* Let the relaxed planning graph constructed from the initial state s in a problem where the set of objects of the problem is *Objs* be $(F_0^+, F_0^-) \cdots (F_k^+, F_k^-)$. The following cases define when ϕ is true at level k of the relaxed planning graph, which is denoted as $(F_k^+, F_k^-) \models \phi$.

- If ϕ is an atomic formula then $(F_k^+, F_k^-) \models \phi$ iff $\phi \in F_k^+$.
- If $\phi = \neg f$, where f is an atomic formula, then $(F_k^+, F_k^-) \models \phi$ iff $\phi \in F_k^-$.
- If $\phi = \psi \wedge \xi$, then $(F_k^+, F_k^-) \models \phi$ iff $(F_k^+, F_k^-) \models \psi$ and $(F_k^+, F_k^-) \models \xi$.
- If $\phi = \psi \vee \xi$, then $(F_k^+, F_k^-) \models \phi$ iff $(F_k^+, F_k^-) \models \psi$ or $(F_k^+, F_k^-) \models \xi$.
- If $\phi = \forall x.\psi$, then $(F_k^+, F_k^-) \models \phi$ iff for every $o \in \mathit{Objs}$ $(F_k^+, F_k^-) \models \psi(x/o)$, where $\psi(x/o)$ is the formula ψ with all free instances of x replaced by o.[1]
- If $\phi = \exists x.\psi$, for some $o \in \mathit{Objs}$ $(F_k^+, F_k^-) \models \psi(x/o)$.

The standard relaxed plan extraction has to be modified slightly for the ADL case. Now, because actions have conditional effects, whenever a fact f is made true by action a there is a particular set of facts that is responsible for its addition, i.e. those that made both the precondition of a and the condition in its conditional effect true. When recursing from a subgoal f we add as new subgoals all those facts responsible for the addition of f (which could be in either part of the relaxed world).

As is the case with STRIPS relaxed planning graphs, whenever a fact f is reachable from a state by performing a certain sequence of legal actions, then f eventually appears in a fact layer of the graph. The same happens in these relaxed planning graphs. This is proven in the following proposition.

Proposition 2. *Let s be a planning state, $R = (F_0^+, F_0^-)(F_1^+, F_1^-) \cdots (F_m^+, F_m^-)$ be the relaxed planning graph constructed from s up to a fixed point, and ϕ be an NNF formula. If ϕ is true after performing a legal sequence of actions $a_1 \cdots a_n$ in s, then there exists some $k \leqslant m$ such that $(F_k^+, F_k^-) \models \phi$.*

Proof. See Appendix A. □

This proposition verifies that the relaxed planning graph is in fact a relaxation of the problem. In particular, it says that if the goal is not reachable in the relaxed planning graph then it is not achievable by a real plan.

Besides being a desirable property, this reachability result is key to some interesting properties of our search algorithm. In particular, as we see later, it is essential to proving that some of the bounding functions we employ will never prune an optimal solution (under certain reasonable assumptions).

[1] In our implementation, bounded quantification is used so that this condition can be checked more efficiently. In particular, this means that not every object in *Objs* need be checked.

1.	$s_0 s_1 \cdots s_n \models$ (always ϕ)	iff $\forall i : 0 \leqslant i \leqslant n, s_i \models \phi$
2.	$s_0 s_1 \cdots s_n \models$ (sometime ϕ)	iff $\exists i$: $0 \leqslant i \leqslant n, s_i \models \phi$
3.	$s_0 s_1 \cdots s_n \models$ (at end ϕ)	iff $s_n \models \phi$
4.	$s_0 s_1 \cdots s_n \models$ (sometime-after $\phi\psi$)	iff $\forall i$ if $s_i \models \phi$ then $\exists j$: $i \leqslant j \leqslant n, s_j \models \psi$
5.	$s_0 s_1 \cdots s_n \models$ (sometime-before $\phi\psi$)	iff $\forall i$ if $s_i \models \phi$ then $\exists j$: $0 \leqslant j < i, s_j \models \psi$
6.	$s_0 s_1 \cdots s_n \models$ (at-most-once ϕ)	iff $\forall i$: $0 < i \leqslant n$, if $S_i \models \phi$ then $\exists j$: $j \geqslant i$, $\forall k$: $k > j$, $s_k \models \neg\phi$

Fig. 1. Semantics of PDDL3's temporally extended formulae that do not mention explicit time. The trajectory $s_0 s_1 \cdots s_n$ represents the sequence of states that results from the execution a sequence of actions $a_1 \cdots a_n$.

2.3. Brief description of PDDL3

PDDL3 was introduced by Gerevini and Long [23] for the 5th International Planning Competition. It 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 dependent on the satisfaction of the preferences.

The current version of our planner handles the non-temporal and non-numeric subset of PDDL3, which was the language used for the *Qualitative Preferences* track in IPC-5. In this subset, temporal features of the language such as durative actions and timed fluents are not supported. Moreover, preference formulae that mention explicit times (e.g., using operators such as `within` and `always-within`) are not supported. Numeric functions (PDDL fluents) are not supported either. The rest of this section briefly describes the new elements introduced in PDDL3 that we do support.

2.3.1. Temporally extended preferences and constraints

PDDL3 specifies TEPs and temporally extended hard constraints in a subset of a quantified linear temporal logic (LTL) [32]. These LTL formulae are interpreted over *trajectories*, which in the non-temporal subset of PDDL3 are sequences of states that result from the execution of a legal sequence of actions. Fig. 1 shows the semantics of LTL-based operators that can be used in temporally extended formulae. The first two operators are standard in LTL; the remaining ones are abbreviations that can be defined in terms of standard LTL operators.

2.3.2. Temporally extended preferences and constraints

Preferences and constraints (which can be viewed as being preferences that must be satisfied) are declared using the `:constraints` construct. Each preference is given a name in its 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 cannot be nested in PDDL3. Our approach can however handle the more general case of nested temporal operators.

2.3.3. Precondition preferences

Precondition preferences are atemporal formulae expressing conditions that should ideally hold in the state in which the action is performed. 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.

2.3.4. 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.

2.3.5. 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 have been 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, in the non-temporal subset of PDDL3, returns the plan length, and the actual duration of the plan in more expressive settings. 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.

In this article we focus on metric functions that mention only `total-time` or `is-violated` functions, since we do not allow function symbols in the planning domain.

3. Preprocessing PDDL3

As described in the previous section, PDDL3 supports the definition of temporally extended preferences in a subset of LTL. A brute force method for generating a preferred plan would be to generate all plans that realize the goal and then to rank them with respect to the PDDL3 metric function. However, evaluating plans once they have been generated is not efficient because there could be many plans that achieve the goal. Instead, we need to be able to provide heuristic guidance to the planner to direct it towards the generation of *high-quality* plans. This involves estimating the merit of partial plans by estimating which of the TEPs could potentially be satisfied by one of its extensions (and thus estimating the metric value that could potentially be achieved by some extension). With such heuristic information the planner could then direct the search effort towards growing the most promising partial plans.

To actively guide the search towards plans that satisfy the problem's TEPs we develop a two-part approach. The first component of our approach is to exploit the techniques presented by Baier and McIlraith [2] to convert a planning domain containing TEPs into one containing an equivalent set of simple (final-state) preferences. Simple preferences are quite similar to standard goals (they express soft goals), and thus this conversion enables the second part of our approach, which is to extend existing heuristic approaches for classical goals to obtain heuristics suitable for guiding the planner toward the achievement of this new set of simple preferences. The development and evaluation of these new heuristics for simple preferences is one of the main contributions of our work and is described in the next section. That section also presents a new search strategy that is effective in exploiting these heuristics.

In this section we describe the first part of our approach: how the techniques of Baier and McIlraith [2] can be exploited to compile a planning domain containing TEPs into a domain containing only simple preferences. Besides the conversion of TEPs we also describe how we deal with the other features of PDDL3 that we support (i.e., those described in the previous section).

3.1. Temporally extended preferences and constraints

Baier and McIlraith [2] presented a technique that can construct an automaton A_φ from a temporally extended formula φ. The automaton A_φ has the property that it accepts a sequence of states (e.g., a sequence of states generated by a plan) if and only if that sequence of states satisfies the original formula φ. The technique works for a rich subset of first-order

linear temporal logic formulas that includes all of PDDL3's TEPs. It also includes TEPs in which the temporal operators are nested, which is not allowed in PDDL3. To encode PDDL3 preference formulae, each preference formula is represented as an automaton. Reaching an accepting condition of the automaton corresponds to satisfying the associated preference formula.

The automaton A_φ can then be embedded within the planning domain by extending the domain with new predicates representing the state of the automaton. Thus, in the initial state of the planning problem these predicates will capture the fact that the automaton, starting from its initial state, has just inputed the initial state of the problem. The technique also modifies the domain's actions so that they can properly update the "automata-state" predicates. When a sequence of actions is applied starting in the initial state, the automata-state predicates are updated to capture the progress these actions have made towards satisfying the preference formula to which the automaton corresponds. Hence we can determine if a sequence of actions has satisfied φ by simply testing if the automata-state predicates in the final state arising from these actions indicate that the automaton is in an accepting state. In other words, the technique allows one to convert a temporally extended condition (φ) into a condition on the final state (the automaton state predicates indicate that A_φ is in a accepting state).

One important feature of the compilation technique we exploit is that it can construct *parameterized* automata. That is, we do not need to expand a quantified first-order temporal extended formula φ into a larger propositional formula (by computing all ground instantiations). This means that the technique generates compact domains, by avoiding grounding of quantified preferences. Generating a compact compiled problem is key for good performance, as we will see in Section 5. Although in general the size of the automaton that results from compiling an arbitrary LTL formula φ can be exponential in $|\varphi|$, in case of the restricted subset of LTL allowed by PDDL3 (in which formulae do not allow nestings of temporal operators) an exponential blowup cannot occur.

Baier and McIlraith's original paper was aimed at planning with temporally extended goals, not preferences. Up to the construction of the automata for each temporally extended formula, our approach is identical to that taken by them. However, Baier and McIlraith [2] then propose using *derived predicates* to embed the automata in the planning domain. In our work we have chosen a different approach that is more compatible with the underlying TLPLAN system we employed in our implementation. In the rest of the section, we give some more details on the construction of automata and the way we embed these automata into a planning domain. Further details on automata construction can be found in [2].

3.1.1. Parameterized finite state automata

The compilation process first constructs a parameterized nondeterministic finite-state automaton (PNFA) A_φ for each temporally extended preference or hard constraint expressed as an LTL formula φ. The PDDL3 operators presented in Fig. 1 that are abbreviations are first expanded into standard LTL operators following Gerevini and Long [23].

The PNFA represents a family of nondeterministic finite-state automata. Its transitions are labeled by first-order formulae, and its input language is the set of all strings of plan states. A PNFA A_φ accepts a sequence of plan states iff such a sequence satisfies φ. Fig. 2 shows some examples of PNFA for first-order LTL formulae.

Parameters in the automaton appear when the LTL formula is externally quantified (e.g., Fig. 2(b)). The intuition is that different *objects* (or tuples of objects) can be in different states of the automaton. Tuples of objects can transition from a state q to a state q' when the automaton reads a plan state s iff there is a transition between q and q' that is labeled by a formula that is satisfied in s.

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 Fig. 2(b), we see that both objects start off in the initial state q_0. Then the automaton inputs the initial state of the planning problem. That state satisfies the formula `(implies (loaded ?x) (delivered ?x))` for both packages A and B since neither is loaded in the initial state. Hence the packages transition to state q_2 as well as stay in state q_0 (the automata is nondeterministic). 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 Fig. 2. Now, assume we perform the action $load(A, Truck)$. In the resulting state, B stays in q_0 and moves once again from q_0 to q_2 while A now moves from q_0 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.

A PNFA is useful for computing heuristics because it effectively represents all the different paths to the goal that can achieve a certain property; its states intuitively "monitor" the progress towards satisfying the original temporal formula. Therefore, while expanding a relaxed planning graph for computing heuristics, one is implicitly considering all possible (relaxed) ways of satisfying the property.

3.1.2. Representing the PNFA within the planning problem

After the PNFA has been constructed it must be embedded within the planning domain. This is accomplished by extending the original planning problem with additional predicates that represent the state of the automaton in each plan state. If the planning domain has multiple TEPs (as is usually the case), a PNFA is constructed for each TEP formula and then embedded within the planning domain with automaton-specific automata-state predicates. That is, the final planning problem will contain distinct sets of automata-state predicates, one for each embedded automaton.

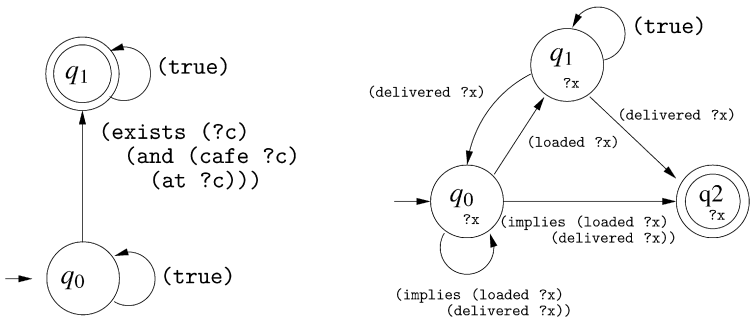

Fig. 2. PNFA for (a) `(sometime (exists (?c) (and (cafe ?c) (at ?c))))`, and (b) `(forall (?x) (sometime-after (loaded ?x) (delivered ?x)))`. In both PNFA q_0 is the initial state and the accepting states are indicated by a double circle border.

To represent an automaton within the domain, 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 states for a particular *tuple of objects*. In our example, the fact `(aut-state q0 A)` represents that object `A` is in automaton state `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 the tuple.

Rather than modify the domain's actions so that the automata state can be properly updated as actions are executed (as was done by Baier and McIlraith [2]) we instead modified the underlying TLPLAN system so that after every action it would automatically apply a specified set of *automata updates*. Automata updates work like pseudo-actions that are performed automatically while a new successor is generated. When generating the successor to s after performing action a, the planner builds the new state s' by adding and deleting the effects of a. When this is finished, it processes the automata updates over s', generating a new successor s''. The state s'' is then regarded as the actual successor of s after performing a. The compilation process can then avoid changes to the domain's actions and instead insert all of the conditions needed to transition the automata state in one self-contained addition to the domain specification.

Syntactically, the automata updates are encoded in the domain as first-order formulae that contain the `add` and `del` keywords, just like regular TLPLAN action effect specifications. For the automata of Fig. 2(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 automaton 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 also need to specify what objects are in what automata states in the initial state of the problem. This means we must augment the problem's initial state by adding a collection of automata facts. Given the original initial state and an automaton, the planner computes the states that every relevant tuple of objects can be in after the automaton has inputed the problem's initial state, 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 temporally extended formula originally described a hard constraint, the accepting condition of the automaton can be treated as an 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.

3.2. 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 family `p`. This function keeps track of how many times the preference has been violated. It is initialized to zero and 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 us 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)
```

3.3. Simple preferences

As with TEPs, we add new *accepting predicates* to the compiled domain, one for each simple preference. We also define updates, analogous to the automata updates for these accepting predicates. Accepting predicates become true iff the preference is satisfied. Moreover, if the preference is quantified, these accepting predicates are parameterized: they can be true of some tuples of objects and at the same time be false for other tuples.

3.4. Metric function

For each preference family *name*, we define a new *domain* function `is-violated-`*name*. The return values of these functions are 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-p` function counts the number of object tuples that fail to satisfy the preference.

By way of illustration, the TLPLAN code that is generated for the preference p-light defined in Section 2.3.2 is:

```
(def-defined-function (is-violated-p-light)
  (local-vars ?x)                    ;; ?x is a local variable
    (and (:= ?x 0)                   ;; ?x initialized to 0
      (forall (?l) (light ?l)
    (implies (not (preference_p-light_satisfied ?l))
       (:= ?x (+ ?x 1))))            ;; increase ?x by 1 if preference not satisfied
      (:= is-violated-p-light ?x)))  ;; return total sum
```

where `preference_p-light_satisfied` is the accepting predicate defined for preference p-light. Note our translation avoids grounding by using quantification to refer to all objects of type `light`.

If the original metric function contains the PDDL3 function `(total-time)`, we replace its occurrence by the TLPLAN function `(plan-length)`, which counts the number of actions in the plan. Thus, actions are implicitly associated a unitary duration.

The metric function in the resulting instance is defined just as in the PDDL3 definition but by 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.

In the remainder of the paper, we use the notation `is-violated`(p, N) to refer to the value of `is-violated-p` in a search node N. We will sometimes refer to the metric function as M, and we will use $M(N)$ to denote the value of the metric in search node N.

4. Planning with preferences via heuristic search

Starting with the work of UNPOP [29], HSP [8], and FF [27], forward-chaining search guided by heuristics has proved to be a powerful and useful paradigm for solving planning problems. As shown above, the automata encoding of temporally extended preferences allows us to automatically augment the domain with additional predicates that serve to keep track of the partial plans' progress towards achieving the TEPs. The central advantage of this approach is that it converts the planning domain to one with simple preferences. In particular, now the achievement of a TEP is marked by the achievement of an accepting predicate for the TEP, which is syntactically identical to a standard goal predicate.

This means that, in the converted domain, standard techniques for computing heuristic distances to goal predicates can be utilized to obtain heuristic distances to TEP accepting predicates. For example, the standard technique based on a relaxed planning graph [27], which approximates the distance to each goal and each TEP accepting predicate can be used to heuristically guide a forward-chaining search.

Nevertheless, although the standard methods can be fairly easily modified in this manner, our aim here is to develop a search strategy that is more suitable to the problem of planning with TEPs. In particular, our approach aims to provide a search algorithm with three main features. First, the planner should find good plans, which optimize a supplied metric function. Second, it should be able to generate optimal plans, or at least be able to generate an improvement over an existing plan. Finally, since in some contexts it might be very hard to achieve an optimal plan—and hence a great deal of search effort could be required—we want the algorithm to find at least one plan as quickly as possible.

Heuristic search with non-admissible heuristics, like the relaxed goal distances employed in planners like FF can be very effective at quickly finding a plan. However, they offer no assurances about the quality of the plan they find. On the other hand, if an admissible heuristic is used, the plan found is guaranteed to be optimal (assuming the heuristic

is admissible with respect to the supplied plan metric). Unfortunately, admissible heuristics typically perform poorly in practice [8]. Hence, with an admissible heuristic the plan often fails to find any plan. This is typically unacceptable in practice.

In this section we develop a heuristic search technique that exploits the special structure of the translated planning domains in order to (a) find a plan fairly rapidly using a non-admissible heuristic and (b) generate a sequence of improved plans that, under some fairly general conditions, terminates with an optimal plan by using a bounding technique. In particular, our search technique allows one to generate better plans—or even optimal plans—if one has sufficient computational resources available. It also allows one to improve on an existing plan and sometimes prove a plan to be optimal.

In the rest of the section we begin by describing a set of different heuristic functions that can serve to guide the search towards satisfying goals and preferences. Then, we describe our search algorithm and analyze some of its properties.

4.1. Heuristics functions for planning with preferences

Our algorithm performs a forward search in the space of states guided by heuristics. Most of the heuristic functions given below are computed at a search node N by constructing a relaxed planning graph as described in Section 2.2.1. The graph is expanded from the planning state corresponding to N and is grown until all *goal* facts and all *preference* facts (i.e., instances of the accepting predicates) appear in the relaxed state or a fixed point is reached. The goal facts correspond to the hard goals, and the preference facts correspond to instantiations of the accepting predicates for the converted TEPs.

Since in our compiled domain we need to update the automata predicates, the procedure in Section 2.2.1 is modified to apply automata updates in action layers after all regular actions have been performed. On the other hand, because our new compiled domain has functions, in addition we modify the procedure in Section 2.2.1 to *ignore* all effects that directly affect the value of a function. This means that in the relaxed worlds, all preference counters will have the same value as in the initial state s. Note that since preference counters do not appear in the conditions of conditional effects or in the preconditions of actions, Proposition 2 continues to hold for relational facts; in particular, it holds for accepting predicates.

Below we describe a suite of heuristics that can be computed from the relaxed planning graph and can be used for planning with preferences. They are designed to guide the search towards (1) satisfying the goal, and (2) satisfying highly valued preferences, i.e., those preferences that are given a higher weight in the metric function. However, highly valued preferences can be very hard to achieve and hence guiding the planner towards the achievement of such preferences might yield unacceptable performance. To avoid this problem, our approach tries to account for the difficulty of satisfying preferences as well as their value, ultimately attempting to achieve a tradeoff between these two factors.

4.1.1. Goal distance function (G)

This function returns an estimate of the number of actions needed to achieve the goal (planning problems often contain a hard "must achieve" goal as well as a collection of preferences). G is the same as the heuristic used by the FF planner but modified for the ADL case. The value returned by G is the number of actions contained in a relaxed plan that achieves the goal.

4.1.2. Preference distance function (P)

This function is a measure of how hard it is to reach the various preference facts. It is based on a heuristic proposed by Zhu and Givan [37] for conjunctive hard goals, but adapted to the case of preferences. Let $\mathcal{P}$ be the set of preference facts that appear in the relaxed planning graph, and let $d(f)$ be the depth at which f first appears during the construction of the graph. Then $P(N) = \sum_{f \in \mathcal{P}} d(f)^k$, for some parameter k. Notice that unreachable preference facts (i.e., those not appearing in the graph) do not affect P's value.

4.1.3. Optimistic metric function (O)

The O function is an estimate of the metric value achievable from a search node N in the search space. O does not require constructing the relaxed planning graph. Rather, we compute it by assuming (1) no further precondition preferences will be violated in the future, (2) TEPs that are violated and that can be proved to be unachievable from N are regarded as false, (3) all remaining preferences are regarded as satisfied, and that (4) the value of `(total-time)` is evaluated to the length of the plan corresponding to N. To prove that a TEP p is unachievable from N, O uses a sufficient condition. It checks whether or not the automaton for p is currently in a state from which there is no path to an accepting state. Examples of LTL formulae that can be detected by this technique as always being falsified in the future are those of the form `(always` φ`)`. Indeed, as soon as φ becomes false, from no state in the automaton's current set of states will it be possible to reach an accepting state.

Although O clearly underestimates the set of preferences that can be violated by any plan extending N it is not necessarily a lower bound on the metric value of any plan extending N. It will be a lower bound when the metric function is non-decreasing in the number of violated preferences. As we will see later, lower bounds for the metric function can be used to soundly prune the search space and speed up search.

Definition 3 *(NDVPL metric functions).* Let $\mathcal{I}$ be a (preprocessed) PDDL3 planning instance, let the set Γ contain its preferences, and let $\texttt{length}(N)$ be the length of the sequence of action that generated N. A metric function M is *non-decreasing in the number of violated preferences and in plan length* (NDVPL) iff for any two nodes N and N' it holds that:

(1) If $\mathit{length}(N) \geqslant \mathit{length}(N')$, and for every $p \in \Gamma$, $\texttt{is-violated}(p, N) \geqslant \texttt{is-violated}(p, N')$, then $M(N) \geqslant M(N')$, and
(2) If $\texttt{(total-time)}$ appears in M, and $\mathit{length}(N) > \mathit{length}(N')$, and for every $p \in \Gamma$, $\texttt{is-violated}(p, N) \geqslant \texttt{is-violated}(p, N')$, then $M(N) > M(N')$.

NDVPL metrics are natural when the objective of the problem is to minimize the metric function (as in our preprocessed instances). Problems with NDVPL metrics are those in which violating preferences never improves the metric of the plan. Furthermore, adding more actions to a plan that fail to satisfy any new preferences can never improve its metric. Below, in Remark 16, we see that *additive* metrics, which were the only metrics used in IPC-5, satisfy this condition.

Proposition 4. *If the metric function is NDVPL, then $O(N)$ is guaranteed to be a lower bound on the metric value of any plan extending N.*

Proof. The optimistic metric only regards as violated those preferences that are provably violated in every successor of N (i.e., in every state reachable from N by some sequence of actions). It regards as satisfied all remaining preferences. That is, O is evaluating the metric in a hypothetical node N_O such that for any node N' reachable from N and for every $p \in \Gamma$ $\texttt{is-violated}(p, N_O) \leqslant \texttt{is-violated}(p, N')$. Furthermore, because O evaluates the plan length to that of N, our hypothetical node is such that $\texttt{length}(N_O) = \texttt{length}(N)$ and hence we have $\texttt{length}(N_O) \leqslant \texttt{length}(N')$. Since the metric function is NDVPL, it follows from Definition 3 that for every successor N' of N, $M(N_O) \leqslant M(N')$. It follows that $O(N)$ returns a lower bound on the metric value of any plan extending N. □

The O function is a variant of the "*optimistic weight*" heuristic in the PPlan planner [6]. PPlan *progresses* LTL preferences (as defined by Bacchus and Kabanza [1]) through every node of the search space. The optimistic weight assumes as falsified only those LTL preferences that have progressed to false.

4.1.4. Best relaxed metric function (B)

The B function is another estimate of the metric value achievable by extending a node N. It utilizes the relaxed planning graph grown from the state corresponding to N 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. The metric function evaluated in a relaxed world w, $M(w)$, evaluates the $\texttt{is-violated}$ functions directly on w, and evaluates $\texttt{(total-time)}$ as the length of the sequence of actions that corresponds to N.

For the case of NDVPL metric functions, B is similar to O, but can return tighter estimates. Indeed, note that the last layer of the relaxed planning graph contains a superset of the preference facts that can be made true by some successor to the current state. Also, because the counters for precondition preferences are not updated while expanding the graph, the value of the $\texttt{is-violated}$ functions for precondition preferences is constant over the relaxed states. This represents the implicit assumption that no further precondition preferences will be violated. The metric value of the relaxed worlds does not increase (and sometimes actually decreases), since the number of preference facts increases in deeper relaxed worlds. As a result, the metric of the deepest relaxed world is the one that will be returned by B. This value corresponds to evaluating the metric function in a relaxed state where: (1) $\texttt{is-violated}$ functions for precondition preferences are identical to the ones in N, (2) preference facts that do not appear in the relaxed planning graph are regarded as violated, and (3) all remaining preferences are regarded as satisfied. This condition (2) is stronger than condition (2) in the definition of O above. Indeed, no preference that is detected as unsatisfiable by the method described for O can appear in the relaxed planning graph, since there is no path to an accepting state of that preference. Hence, no action can ever add the accepting predicate for the preference.

By using the relaxed planning graph, B can sometimes detect preferences that are not satisfiable by any successor of N but that cannot be spotted by O's method. For example, consider we have a preference $\varphi = \texttt{(sometime f)}$, and consider further that fact $\texttt{f}$ is not reachable from the current state. The myopic O function would regard this preference as satisfiable, because it is always possible to reach the final state of the automaton for formula φ (the automaton for $\texttt{f}$ looks like the one in Fig. 2(a)). On the other hand, $\texttt{f}$ might not appear in the graph—because $\texttt{f}$ is unreachable from the current state—and therefore B would regard φ as unsatisfiable.

These observations lead to the conclusion that $B(N)$ will also be a lower bound on the metric value of any successor of N under the NDVPL condition.

Proposition 5. *If the metric function is NDVPL, then $B(N)$ is guaranteed to be a lower bound on the metric value of any plan extending N.*

Proof. Proposition 2 implies that all preference facts that could ever be achieved by some successors of N will eventually appear in the deepest relaxed world. Because the metric is NDVPL, this implies that the metric value of the deepest relaxed

world is also the minimum, and therefore such a value will be returned by the B function. Now we can apply the same argument as in the proof for Proposition 4, since the returned metric value corresponds to evaluating the metric in a hypothetical node in which all `is-violated` counters are lower or equal than those of any plan extending N. □

4.1.5. Discounted metric function ($D(r)$)

The D function is a weighting of the metric function evaluated in the relaxed worlds. Assume $w_0, w_1, \ldots, w_n$ are the relaxed worlds in the relaxed planning graph, where w_i is at depth i and the $w_0 = (s, s^c)$, i.e., the positive and negative facts of the state where $D(r)$ is being evaluated. Then the discounted metric, $D(r)$, is:

$$D(r) = M(w_0) + \sum_{i=0}^{n-1} \big(M(w_{i+1}) - M(w_i)\big) r^i, \tag{1}$$

where $M(w_i)$ is the metric function evaluated in the relaxed world w_i and r is a discount factor ($0 \leqslant r \leqslant 1$).

The D function is optimistic with respect to preferences that appear earlier in the relaxed planning graph (i.e., preferences that seem easy) and pessimistic with respect to preferences that appear later (preferences that seem hard). Intuitively, the D function estimates the metric value of plans extending the current state by "believing" more in the satisfaction of preferences that appear to be easier. Observe that $M(w_{i+1}) - M(w_i)$ is the amount of metric value *gained* when passing from relaxed world w_i to w_{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 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.

Finally, observe that when r is close to 1, the effect of discounting is low, and when it is close to 0, the metric is quickly discounted. When r is close to 0 the D function is myopic in the sense that it discounts heavily those preferences that appear deeper in the graph.

4.2. The planning algorithm

Our planning algorithm searches for a plan in a series of *episodes*. The purpose of each of these episodes is to find a plan for the goal that has a better value than the best found so far. In each planning episode a best-first search for a plan is initiated using some of the heuristics proposed above. The episode ends as soon as it finds a plan whose quality is better than that of the plan found in the previous episode. The search terminates when the search frontier is empty. The algorithm is shown as Algorithm 1.

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.

```
1:  function SEARCH-HPLAN-P(initial state init, goal formula goal, a set of hard constraints hConstraints,
                            metric function METRICFN, heuristic function USERHEURISTIC)
2:    frontier ← INITFRONTIER(init)                                  ▷ initialize search frontier
3:    closed ← ∅
4:    bestMetric ← worst case upper bound
5:    HEURISTICFN ← G
6:    while frontier is not empty
7:      current ← Best element from frontier according to HEURISTICFN
8:      if ¬CLOSED?(current, closed) and current satisfies hConstraints then
9:        if METRICBOUNDFN(current) < bestMetric then                  ▷ pruning by bounding
10:         if current satisfies goal and its metric is < bestMetric then
11:           Output plan for current
12:           if this is first plan found then
13:             HEURISTICFN ← USERHEURISTICFN
14:             frontier ← INITFRONTIER(init)                          ▷ search restarted
15:             Reinitialize closed List
16:           end if
17:           bestMetric ← METRICFN(current)
18:         end if
19:         succ ← successors of current
20:         frontier ← merge succ into frontier
21:         closed ← closed ∪ {current}
22:       end if
23:     end if
24:   end while
25: end function
```

Algorithm 1. HPLAN-P's search algorithm.

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, together 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, it uses the O function to break it. In Section 5, we investigate the effectiveness of several such prioritized heuristics sequences.

4.2.1. Pruning the search space

Once we have completed the first planning episode (using G) we want to ensure that each subsequent planning episode yields a better plan. Whenever a plan is found, it will only be returned if its metric is lower than that of the last plan found (line 10).

Moreover, in each episode we can use the metric value of the previously found plan to prune the search space, and thus improve search performance. In each planning episode, the algorithm prunes from the search space any node N 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(N) must compute or estimate a lower-bound on the metric of any plan extending N.

Pruning is realized by the algorithm in line 1, when the condition in the *if* becomes false. As the value of *bestMetric* gets updated (line 17), the pruning constraint imposes a tighter bound causing more partial plans to be rejected.

The O and B heuristic functions defined above are well-suited to be used as MetricBoundFN(). Indeed, we tried both of them in our experiments. On the other hand, it is also simple to "turn-off" pruning by simply passing a null function as MetricBoundFN().

4.2.2. Discarding nodes in closed list

Under certain conditions, our algorithm will also prune nodes that revisit a plan state that has appeared in a previously expanded node. This is done for efficiency, and allows the algorithm to avoid considering plans with cycles.

The algorithm keeps a list of nodes that have already been expanded in the variable *closed*, just as in standard best-first search. Furthermore, when *current* is extracted from the search frontier, its state is checked against the set of closed nodes (line 8). If there exists a node in the closed list with the same state and a better or equal heuristic value (i.e., Closed?(*current*, *closed*) is true), then the node *current* will be pruned from the search space.

Note that for two states to be identical in the compiled planning instance every boolean predicate has to coincide and, moreover, values assigned to each ground function also have to coincide. In particular, this means that `is-violated`counters in two identical states are also identical, i.e., the preferences are equally satisfied. Nevertheless, two search nodes with identical states can still be assigned different heuristic values. Given the way we have defined UserHeuristic(), different heuristic values will be assigned to nodes with identical states only when the metric function depends on (`total-time`). If the (`total-time`) function appears positively in the metric (i.e., the metric is such that for otherwise equally preferred plans, longer ones are never preferred to shorter ones), then discarding of nodes cannot prune any node that leads to an optimal plan. We discuss this further in the next section.

Finally, note that the cycles we are eliminating are those that occur in the compiled instance, *not* those occurring in the original instance. Indeed, in the original instance there might be LTL preferences that can be satisfied by visiting the same state twice. For example consider the preference: *eventually turn the light switch on and sometime after turn it off*. Any plan that contains the action *turn-on* immediately followed by *turn-off* satisfies the preference but also visits the same state twice. In our compiled domains however such a plan will not produce a cycle, and therefore will not be pruned. This is because the set of current states of the preference's automaton—represented by the automata domain predicates—changes when performing those actions; indeed it changes from a non-accepting state to an accepting state.

4.3. Properties of the algorithm

In this section we show that under certain conditions our search algorithm is guaranteed to return *optimal* and *k-optimal* plans. We will prove this result without imposing any 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.

The first requirement in our proofs is that the pruning performed by the algorithm is *sound*.

Definition 6 *(Sound Pruning).* The pruning performed by Algorithm 1 is *sound* iff whenever a node N is pruned (line 1) the metric value of any plan extending N exceeds the current bound *bestMetric*.

When Algorithm 1 uses sound pruning, no state will be incorrectly pruned from the search space. That is, node N is not pruned from the search space if some plan extending it can achieve a metric-value superior to the current bound. To guarantee that the algorithm performs sound pruning it suffices to provide a lower-bound function as input to the algorithm.

Theorem 7. *If* MetricBoundFN(N) *is a lower bound on the metric value of any plan extending N, then Algorithm* 1 *performs sound pruning.*

Proof. If node N is not in closed and is pruned from the search space then (a) MetricBoundFN(N) $\geqslant$ *bestMetric*. If MetricBoundFN() is a lower bound on the metric value of any plan extending N, then (b) MetricBoundFN(N) $\leqslant M(N_p)$ for any solution node N_p extending N. By putting (a) and (b) together we obtain that if N is not in closed and it is pruned, then $M(N_p) \geqslant$ *bestMetric*, for every solution node N_p extending N, i.e., pruning is sound. □

As proven previously in Section 4.1, if the metric function is NDVPL, O and B will both be lower bound functions, and therefore provide sound pruning. Notice also that "turning off" pruning by having MetricBoundFN() return a value that is always less than *bestMetric*, also provides sound pruning.

The second requirement for optimality has to do with the discarding of closed nodes performed in line 8. To preserve optimality, the algorithm must not remove a node that can lead to a plan that is more preferred than any plan that can be achieved by extending nodes that are not discarded. Formally,

Definition 8 *(Discarding of Closed Nodes Preserves Optimality).* The discarding of nodes by Algorithm 1 preserves optimality iff for any node N that is discarded in line 8, there is either already an optimal node (i.e., plan) N_O in the closed list or there exists a node N in *frontier* that can be extended to a plan with optimal quality.

The condition defined above holds when using NDVPL metrics under fairly general conditions. In particular, it holds for any NDVPL metric that is independent of (`total-time`). It also holds if the NDVPL metric depends on (`total-time`), and O or B is used as a first tie breaker after G or P in UserHeuristic(). Finally, it will hold if D is used as the first tie breaker for NDVPL metric functions that are *additive on total-time*.

Definition 9 *(Additive on total-time (ATT)).* A metric function M is additive on total time (ATT) iff it is such that $M(N) = M_P(N) + M_T(N)$, where $M_P(N)$ is an expression that does not mention the function (`total-time`), and $M_T(N)$ is an expression whose only plan-dependent function is (`total-time`).

Intuitively, an ATT metric is a sum of a function that only depends on the `is-violated` functions, and a function that includes (`total-time`) but does not include any `is-violated` functions. Now we are ready to state our result formally.

Theorem 10. *The discarding of nodes done by Algorithm* 1 *preserves optimality if the algorithm performs sound pruning, the metric function M is NDVPL and*:

(1) *M is independent of* (`total-time`)*, or*
(2) *M is dependent on* (`total-time`) *and O or B are used as the first tie breaker in* UserHeuristic() *after G or P, or*
(3) *M is ATT and D is used as the first tie breaker in* UserHeuristic() *after G or P.*

Proof. See Appendix B. □

An important fact about sound pruning is that it never prunes optimal plans from the search space, unless another optimal plan has already been found. An important consequence of this fact, is that the search algorithm will be able to find optimal plans under fairly general conditions. Our first result says that, under sound pruning, optimality is guaranteed when the algorithm terminates.

Theorem 11. *Assume Algorithm* 1 *performs sound pruning, and that its node discarding preserves optimality. If it terminates, 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 algorithm never prunes or discards a node that can be extended to an optimal unless an optimal plan has already been found then no plan better than the last one returned exists. □

Theorem 11 still does not guarantee that an optimal solution will be found because the algorithm might never terminate. To guarantee this we must impose further conditions that restrict the explored search space to be finite. Once we have these conditions, optimality is easy to prove since the search must eventually terminate.

Theorem 12. *Assume the following conditions hold:*

(1) *The initial value of bestMetric* (*worst case upper bound*) *in Algorithm* 1 *is finite*;
(2) *The set of cycle-free nodes N such that* MetricBoundFN(N) *is less than the initial value of bestMetric is finite*;
(3) *Algorithm* 1 *performs sound pruning*;
(4) *Node discarding in Algorithm* 1 *preserves optimality.*

Then Algorithm 1 *is guaranteed to find an optimal plan, if one exists.*

Proof. Each planning episode only examines nodes with estimated metric value—given by MetricBoundFN—that is less than *bestMetric*. By assumption 2, this is a finite set of nodes, so each episode must complete and the algorithm must eventually terminate. Now the result follows from Theorem 11. □

In Theorem 12, condition (1) is satisfied by any implementation of the algorithm that uses a sufficiently large number for the initial value of *bestMetric*. Moreover, Theorem 7 shows how condition (3) can be satisfied, and Theorem 10 shows how condition (4) can be satisfied. Condition (2), however, can sometimes be falsified by a PDDL3 instance. In particular, the metric function can be defined in such a way that its value *improves* as the number of violated precondition preferences increases. Under such a metric function the plans' metric values might improve without bound as the plan length increases. This would mean that the number of plans with metric value less than the initial bound, *bestMetric*, becomes unbounded, and condition (2) will be violated. We can avoid cases like this when the metric function is *bounded on precondition preferences*.

Definition 13 *(BPP metrics).* Let the individual precondition preferences for a planning instance P be Γ, and let U denote the initial value of *bestMetric*. A metric function is *bounded on precondition preferences* (BPP) if there exists a value r_i for each precondition preference $p_i \in \Gamma$ such that in every node N with MetricBoundFN(N) $< U$, p_i is never violated more than r_i times.

BPP metrics are such that the `is-violated` functions are always smaller than a fixed bound in every node with metric value lower than U. This property guarantees that there are only a finite number of plans with value less than U, and ultimately enables us to prove another optimality result:

Corollary 14. *Assume that the metric function for planning instance P is BPP and assume conditions* (1), (3), *and* (4) *in Theorem* 12 *hold. Then Algorithm* 1 *finds an optimal plan for P.*

Proof. We need only prove that the set of nodes N with MetricBoundFN(N) $<$ *bestMetric* is finite. This will satisfy condition (2) and allow us to apply Theorem 12. The BPP condition ensures that each precondition function p_i in N can only have a value in the range 0–r_i (for some fixed value r_i). Since the precondition functions are the only functions in the planning instance (the remaining elements of the state are boolean predicates), this means that only a finite number of different states can have this property. □

Note that the NDVPL property, which we could use to satisfy condition (4) in Theorem 12, *does not* imply necessarily the BPP property. As an example suppose a domain where `precPref` is a precondition preference, and `goalPref1` and `goalPref2` are final-state preferences. Assume we are using the B function as MetricBoundFN and that the metric for a node N is defined as:

$$M(N) = \texttt{is-violated}(\texttt{goalPref1}, N) * \texttt{is-violated}(\texttt{precPref}, N) + \texttt{is-violated}(\texttt{goalPref2}, N). \quad (2)$$

M is clearly NDVPL since it cannot decrease as plans violate more preferences. However, M does not necessarily *increase* as more preferences are violated, which can lead to situations in which we have an infinite set of goal nodes with the same metric value. Indeed, assume `goalPref2` is an unreachable preference that cannot be detected by the relaxed planning graph (i.e., it is such that it won't be detected by our B bounding function). Moreover, assume the planner has found a node that satisfies `goalPref1`. Assuming `precPref` can be violated by some action in the planning instance, there might be infinite plans that could be generated that violate `precPref` repeatedly while still satisfying `goalPref1`. Because the `is-violated` functions are represented within the state, those plans cannot be eliminated by the algorithm since they will not produce cycles.

The BPP and NDVPL properties are quite natural conditions on the metric function. Indeed, it is reasonable to assume that violated preferences are undesirable. Hence, a plan should become (arbitrarily) worse as the number of preferences it violates becomes (arbitrarily) larger. Such a property is sufficient to guarantee both the NDVPL and the BPP conditions. The *additive* family of metric functions satisfies both conditions, and it is defined as follows.

Definition 15 *(Additive metric function).* A PDDL3 metric function is *additive*, if it has the form $M = \sum_{i=0}^{n} c_i \times \texttt{is-violated}(p_i)$, where $c_i \geqslant 0$.

Remark 16. Additive metric functions satisfy the NDVPL condition and satisfy the BPP condition when MetricBoundFN is either B or O.

Additive metric functions were used in all of the problems in the qualitative preference track of IPC-5. Therefore, our algorithm—when using O or B for pruning—is guaranteed to find an optimal solution for these problems, given sufficient time and memory. In practice, however, due to restrictions of time and memory, the algorithm finds the optimal solution only in the most simple problems. On the other larger problems it returned the best plan its completed planning episodes found in the time alloted.

4.3.1. k-Optimality

Instead of searching for an optimal plan among the set of all valid plans, one might be interested in restricting attention to a subset of the valid plans. For example, there might be resource usage limitations that might further constrain the set of plans that one is willing to accept. This might be the case when a shift worker cannot be asked to work more than one overtime shift in three days, or a plane cannot log more than a certain number of continuous kilometers. If the set of plans one is interested in can be characterized by a temporally extended property, it suffices to add such a property to the set of hard constraints. The optimality results presented above, will allow the planner to find the optimal plan from among the restricted set of plans, regardless of the property used.

For some interesting properties, however, we can find optimal plans under weaker conditions on the metric function than those required in the general case above. This is the case, for example, when we are interested in plans whose length is bounded by a certain value.

Several existing preference planners are able to find plans that are optimal among the set of plans with restricted length or makespan. For example, PPLAN [6] when given a bound k is able to find an optimal plan among those with length k or less. Similarly, both the system by Brafman and Chernyavsky [10] and SATPLAN-P [24] return optimal plans among those plans of makespan n, where n is a parameter. It should be noted, however, that such plans need not be globally optimal. That is, there could be plans of longer length or makespan that have higher value than the plan returned by these systems. Our algorithm, on the other hand, can return the globally optimal plan under conditions described above. If we are interested, however, in plans of restricted length then our algorithm can return k-optimal plans under weaker conditions.

Definition 17 *(k-optimal plan).* A plan is *k-optimal* iff it is the optimal among the set of plans of length $i \leqslant k$.

To achieve k-optimality, we force the algorithm to search in the space of plans whose length is smaller than or equal to k, by imposing an additional hard constraint that restricts the length of the plan.

Theorem 18. *Assume Algorithm* 1 *uses sound pruning, and that the set of initial hard constraints contains the formula* `(total-time)` $\leqslant k$. *Then, the returned plan* (*if any*) *is k-optimal.*

Proof. Since the space of plans of length up to k is finite, each planning episode will terminate with an improved plan (if any exists). Because of sound pruning, no node can be wrongly pruned from the search space. Hence, the last returned plan (if any) is optimal. □

Note that this result does not require restrictions on the metric function such as condition 2 in Theorem 12. Thus, this result is satisfied by a broader family of metric functions than those that satisfy Theorem 12; for example, it is satisfied when using NDVPL metrics such as the one in Eq. (2).

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 of our planner 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 of 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 (for more details, we refer the reader to the IPC-5 booklet [13]). 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 [22] 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 (O), the best relaxed metric (B), and no pruning at all. From our experiments, we conclude that most of the time pruning can only produce better results than no pruning, and that, overall, pruning with B usually produces better results than pruning with O.

To compare the different strategies, we ran all IPC-5 problems with O and no pruning, with a 30-minute timeout. The heuristics used in these experiments were the four top-performing strategies on each domain, under pruning with B.

The impact of pruning varies across different domains. In three of the domains, the impact of pruning is little. In the storage and TPP domains, pruning has no effect, in practice. In the rovers domain, the impact is slim: O performs as good as B does, and no pruning, on average, produces solutions with a 0.05% increase on the metric. An increased impact is observed in the trucks domain, where the top-performing heuristics improve the metric of the first plan found by 30.60% under B pruning, while under O pruning the metric is improved by 28.02% on average, and under no pruning by 21.33% on average. Finally, the greatest impact can be observed on the openstacks domain. Here, B produces 13.63% improvement on average, while both no pruning and pruning with O produce only 1.62% improvement.

In general, pruning has a noticeable impact when, during search, it can be frequently proven that certain preferences will *not* be satisfied. In the case of the openstacks domain for example, most preferences require certain products (which are associated with *orders*) to be *delivered*. On the other hand, the goal usually requires a number of orders to be *shipped*. To ship an order one is required to start the order, and then ship it. However, to deliver a product associated with order o, one needs to *make* the product after o has been started and before the o has been shipped. Thus, whenever an order o is shipped, the B function automatically regards as unsatisfiable all preferences that involved the delivery of an unmade product associated with o. This occurs frequently in the search for plans for this domain. The initial solution, which ignores preferences, produces a plan with no *make-product* actions. As the search progresses, states that finish an order early are constantly pruned away, which in turn favours adding *make-product* actions.

A side effect of pruning is that it can sometimes prove (when the conditions of Theorem 11 are met) 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 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 42 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, and $BD(r)$, $D(r)B$, $OD(r)$, $D(r)O$ for $r \in \{0, 0.01, 0.05, 0.1, 0.3, 0.5, 0.7, 0.9, 1\}$.

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 a 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 planning graph are key to good performance. In all problems, save TPP, the heuristics that used the relaxed planning graph had the best performance. The case of TPP is pathological in the qualitative preference track. However, upon looking at the actual plans traversed during the search we observed that it is not the case that O is a *good* heuristic for this problem, indeed O is almost totally blind since in most states O is equal to 0. Rather, it turns out that heuristics based on the relaxed planning graph are *poor* in this domain, misguiding the search. In Section 6, we explain scenarios in which our heuristics can perform badly, and give more details on why TPP is one of these cases.

5.3. Comparison to other approaches

We entered HPLAN-P in the IPC-5 *Qualitative Preferences* track [22], achieving second place behind SGPlan_5 [28]. 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. This is supported by the fact that HPLAN-P has similar or better performance than SGPlan_5 on simple planning instances, as we see in experiments shown at the end of this section.

HPLAN-P consistently performed better than MIPS-BDD [17] and MIPS-XXL [15]; 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

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 with respect to the first plan found is shown in square brackets.

Domain	1 Plan	>1 Plan	Best heuristics	Worst heuristics
openstacks	18	14	BP[13.77], DO(1)[13.63], DB(1)[13.63], BD(1)[13.63], B[13.63]	D(0)B[7.56], for $r \in \{0.01, 0.05, 0.1\}$: DO($r$)[7.63] and DB($r$)[7.63]
trucks	5	4	D(0)O[30.68], OD(0)[30.68]	PB[5.35], OP[5.35], PO[5.35], O[12.02]
storage	16	9	BO[37], OB[37], B[37], O[37], BD(0.05)[35.62], OD(0.05)[35.55], BD(0)[35.42]	PO[21.04], PB[21.04], BP[24.18], OP[24.18]
rovers	11	9	D(0.1)O[17.15], D(0.1)B[17.15], D(0.3)B[16.91], D(0.3)O[16.91], O(0.01)D[16.47], O(0.05)D[16.47]	BP[6.97], OP[7.16], B[10.85], OB[10.85], BO[10.85], O[10.85]
TPP	20	20	O[40.32], BO[32.02], B[32.02], OB[33.97]	for $r \leqslant 0.9$: BD(r)[9.03], OD(0.9)[10.98]

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. "$*$" means the planner did not compete in the domain.

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	0.78–0.8	11	1.02–1.07	9	0.94–0.99	9	1.68–1.78
openstacks	20	1	20	0.89–0.92	$*$	$*$	2	2.5	18	6.45–6.81
storage	20	1	20	0.74–0.76	5	3.86–3.95	4	1	4	15.41
pathways	20	1	20	0.77	4	1.02	10	0.79	16	1.19–1.21

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 [22].

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 min[2] on the first 20 instances[3] 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. HPLAN-P was able to solve 20 problems in all domains, except trucks, where it could only solve the 5 simpler instances (see Table 3 for details on the trucks domain). 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, whereas values under 1 indicate the opposite. The results for HPLAN-P were obtained on an Intel(R) Xeon(TM) CPU 2.66 GHz machine running Linux, with a timeout of 15 min. Results for other planners were extracted from the IPC-5 official results, which were generated on a Linux Intel(R) Xeon(TM) CPU 3.00 GHz machine, with a 30 min timeout. Memory was limited to 1 GB for all processes.

We conclude that SGPlan_5 typically outperforms HPLAN-P. SGPlan_5, on average, obtains plans that are no more than 25% better in terms of metric value than those obtained by HPLAN-P. Moreover, in the most simple instances usually HPLAN-P does equally well or better than SGPlan_5 (see Table 3). 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 planning graph, and, in particular, the *D* heuristic.

We ran a final comparison between SGPlan_5 and HPLAN-P on the openstacks-nce domain [25]. openstacks-nce is a reformulation of the original openstacks simple-preferences domain that does not include actions with conditional effects. These two domains are essentially equivalent in the sense that plans in one domain have a corresponding plan with equal quality in the other. The results are shown in Table 4. We observe that HPLAN-P consistently outperforms SGPlan_5 across

[2] In IPC-5, planners where given 30 min on a similar machine.
[3] Only the pathways domain has more than 20 problems.

Table 3
Plan quality (metric) of three of HPLAN-P's heuristics compared to the IPC-5 *Simple Preferences* participants on the simpler, non-metric problems. "ns" means that the instance what not solved by the planner. "$*$" means the planner did not compete in the domain.

Instance	*Yochan*$^{\mathcal{PS}}$	MIPS-BDD	MIPS-XXL	SGPlan$_5$	HPLAN-P			
					O	OD($r = 0.5$)	OD($r = 0$)	OD($r = 1$)
TPP-01	22	16	16	16	16	16	16	16
TPP-02	36	24	24	24	24	24	24	24
TPP-03	24	29	29	29	29	29	29	29
TPP-04	45	35	35	35	39	35	35	42
TPP-05	103	89	223	79	103	79	87	105
TPP-06	133	110	275	101	120	118	114	120
TPP-07	124	126	322	100	124	135	135	135
openstacks-01	$*$	12	63	13	6	6	6	6
openstacks-02	$*$	12	63	16	4	4	4	4
openstacks-03	$*$	ns	88	12	36	30	36	30
openstacks-04	$*$	ns	98	26	47	44	45	49
openstacks-05	$*$	ns	133	36	25	21	25	21
openstacks-06	$*$	ns	133	33	21	18	21	18
openstacks-07	$*$	ns	285	67	87	74	87	74
trucks-01	0	0	0	1	0	0	0	0
trucks-02	3	0	0	0	0	0	0	0
trucks-03	0	0	0	0	0	0	0	0
trucks-04	0	0	ns	0	3	1	3	4
trucks-05	1	ns	ns	0	0	0	0	0
storage-01	6	3	18	5	3	3	3	3
storage-02	11	5	37	8	5	5	5	5
storage-03	49	6	158	14	6	6	6	6
storage-04	51	9	197	17	9	9	9	9
storage-05	165	ns	ns	87	97	130	130	97
storage-06	ns	ns	ns	124	161	195	195	161
storage-07	ns	ns	ns	160	274	281	307	274
pathways-01	2	2	3	2	2	2	2	2
pathways-02	3	3	5	3	3	4	4	4
pathways-03	3	3	4.7	3	3	3.7	3.7	3.7
pathways-04	3	2	3	2	2	2	2	2
pathways-05	ns	7	10.2	6.5	8.5	9	10.2	10.2
pathways-06	ns	8	12.9	10	12.9	12.9	12.9	12.9
pathways-07	ns	11	12.5	8	12.5	12.5	12.5	12.5

Table 4
Metric values obtained by four of HPLAN-P's heuristics and SGPlan$_5$ on the openstacks and openstacks-nce [25] domains.

Instance	openstacks-nce					openstacks				
	SGPlan$_5$	HPLAN-P				SGPlan$_5$	HPLAN-P			
		O	OD(.5)	OD(0)	OD(1)		O	OD(.5)	OD(0)	OD(1)
01	70	11	11	11	11	13	6	6	6	6
02	70	7	11	7	11	16	4	4	4	4
03	90	38	42	37	41	12	36	30	36	30
04	100	48	49	46	49	26	47	44	45	49
05	140	48	48	48	48	36	25	21	25	21
06	140	35	41	34	41	33	21	18	21	18
07	300	98	98	98	98	67	87	74	87	74
08	620	140	152	148	148	123	86	78	86	78
09	620	154	155	154	154	121	109	123	109	123
10	120	30	25	30	20	20	19	11	10	13
11	120	36	26	36	22	21	19	22	23	12
12	153	80	81	80	73	23	52	45	45	51
13	223	190	172	181	174	48	171	167	167	167
14	65	47	22	47	24	6	32	23	21	21
15	210	125	123	125	126	0	74	67	67	67
16	210	133	133	133	133	0	74	63	67	63
17	450	224	255	269	254	0	209	179	179	180
18	930	588	558	929	557	0	557	464	464	493
19	1581	1581	1581	1581	1581	254	1581	1581	1581	1581
20	1348	1348	1348	1348	1348	424	1348	1348	1348	1348
	openstacks-nce					openstacks				

all instances of this domain, obtaining plans that are usually at least 50% better in quality. We also observe that the performance of HPLAN-P is consistent across the two formulations, which is not the case with SGPlan_5.

6. Discussion

In previous sections, we proposed a collection of heuristics that can be used in planning with TEPs and simple preferences in conjunction with our incremental search algorithm. In our experimental evaluation we saw that in most domains the heuristics that utilize the relaxed planning graph are those that provide the best performance. Given the limited number of domains in which we have had the opportunity to test the planner, it is hard—and might be even be impossible—to conclude which is the best combination of heuristics to use. It is even hard to give a justified recipe for their use. However, some situations in which our heuristics perform poorly can be identified and analyzed. Below we describe two reasons for potential poor performance.

The first reason for potentially poor performance is due to our choice of using prioritized sequences of heuristics. We have chosen the goal distance G to appear as the first priority to guide the planner towards satisfying the must-achieve goals for a pragmatic reason: the goal is the most important thing to achieve. However, this design decision sometimes makes the search algorithm focus excessively on goal achievement to the detriment of preference satisfaction. This issue becomes particularly relevant when there are interactions between the goal and the preferences. Consider, for example, a situation in which a preference p can *only* be achieved *after* achieving the goal. Furthermore, assume the goal g is the conjunction $f_1 \wedge f_2$, and assume that prior to achieving p one has to make f_2 false. In cases like this, after the algorithm finds a plan for the goal, it can hardly find a plan that also satisfies p. When extending any plan for g, the planner will always choose an action that does not invalidate the subgoal f_2 over an action that invalidates f_2, if such an action is available. This is because the goal distance (G) of any search node in which f_2 is false is strictly greater than the goal distance in which both f_1 and f_2 are true. As a consequence, the algorithm will have trouble achieving p, and actually will only achieve p when extending a plan for g when *no actions* that invalidate f_2 are available. Unfortunately the only way of getting into such a situation implies exhausting the search space of plans that extend a plan for g without invalidating g.

The second source for poor performance is the loss of structure in which we incur by computing our heuristic in a planning instance in which the action's deletes (i.e., negative effects) are ignored. The inaccurate reachability information provided by this relaxation might significantly affect the performance of all our heuristics based on the relaxed planning graph (i.e., P, B, and D). Consider for example an instance in which there are no hard goals and there are two preferences, p_1 and p_2. Assume further that p_2 is a preference that is rather easy to achieve from any state but that has to be violated in order to achieve p_1. Assume that we are in a state in which p_2 is satisfied but p_1 is not, and in which we need to perform at least three actions to achieve both p_1 and p_2. Let those actions be a, b, and c, such that a makes p_2 false and p_1 true, and finally action b followed by c reestablish p_1, as shown in Fig. 3. Moreover, assume that action e is applicable in s, and that it leads to s_2—a state from which p_1 and p_2 can be reached by the same sequence of three actions. Because the D heuristic is computed on the delete relaxation, D will always prefer to expand s_2 instead of s_1. A relaxed solution on s_2 may achieve both preferences at depth 1, since the preference p_2 is already satisfied at depth 0. On the other hand, a relaxed solution on s_1 may achieve both preferences at depth 2, since in s_1 two actions are needed to reestablish p_2. Once the algorithm expands s_2, there could be another action applicable in s_2, analogous to e, that would steer the search away from s_3.

It is precisely a situation similar to that described above that makes the heuristics based on the relaxed planning graph (especially D and P), perform poorly in the TPP domain. TPP is a transportation problem in which trucks can move between markets and depots transporting goods. A good can be put into the truck by performing a *load* followed by a *store*. Stored goods can be unloaded from the truck performing an *unload*. Once in a market, one has to *buy* an object before it becomes ready to load. In problems of the TPP domain there is a preference that states that any good must be eventually loaded on some truck (p_1). On the other hand, there is a preference that states that all trucks should be unloaded at the end of the plan (p_2). Once we have considered moving a truck to a market and bought a certain good, say `good1`, our plan prefix has achieved p_2 but not p_1. A reasonable course of action to achieve both preferences would be to *load* `good1` on the truck, followed by a *store*, and followed by an *unload*. However, the state that results from performing a *load* is never preferred by the planner, since just like in Fig. 3, a *load* invalidates p_2 while making p_1 true. Instead, an action that preserves the

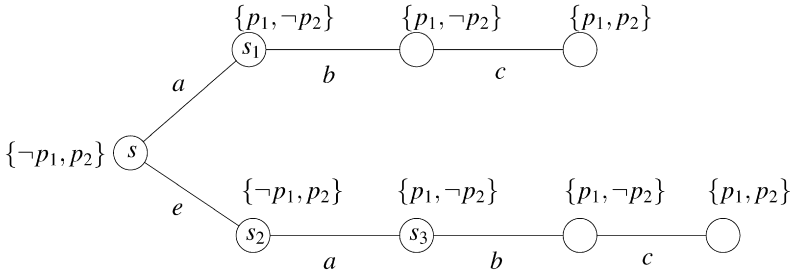

Fig. 3. A situation in which our D heuristics prefers a node that does not lead to the quick satisfaction of both p_1 and p_2.

p_2 property (e.g., a *buy* of another good) is always preferred. This leads the planner to consider all possible combinations of sequences that *buy* a good before considering a *load*. Even worse, after performing all possible buys, for a similar reason the search prefers to use other truck to move to another market to keep on buying products.

7. Related work

There is a significant amount of work on planning with preferences that is related, in varying degrees, to the method we have presented here. We organize this work into two groups: first, planners that are able to plan with preferences in non-PDDL3 preference languages or using soft goals; second, work that focuses on the PDDL3 language. In the rest of the section we review the literature in these two categories.

7.1. Other preference languages

PPLAN [6] is a planning system that exploits progression to plan directly with TEPs using heuristic search. In contrast to HPLAN-P, which is incremental, PPLAN always returns an optimal plan whose length is bounded by a plan-length parameter (i.e., it is k-optimal). Unfortunately, PPLAN uses an admissible heuristic that is far less informative than the heuristics proposed here. As such, it is far less efficient. The heuristic in PPLAN is similar to our O heuristic, and thus does not provide an estimate of the cost to achieving unsatisfied preferences. PPLAN was developed prior to the definition of PDDL3 and exploits its own *qualitative* preference language, LPP, to define preferences. LPP supports rich TEPs, including nested LTL formulae (unlike PDDL3) and rather than specifying a metric objective function, the LPP objective is expressed as a logical formula. PPLAN's LPP language is an extension and improvement over the PP language proposed by Son and Pontelli [35].

The HPLAN-QP planner [3] was proposed as an answer to some of the shortcomings of PPLAN. It is an extension to the HPLAN-P system, allowing planning for *qualitative* TEPs guided by heuristics similar to those that have been proposed in this paper. The preference language is based on LPP, the language used by PPLAN. HPLAN-QP guides the search actively towards satisfaction of preferences (unlike PPLAN), and like HPLAN-P, guarantees optimality of the last plan found given sufficient resources.

Also related is the work on *partial satisfaction planning problems* (PSPs) (over-subscription planning) [34,36]. PSPs can be understood as a planning problem with no hard goals but rather a collection of soft goals each with an associated utility; actions also have costs associated with them. Some existing planners for PSPs [14,33] are also incremental and use pruning techniques. However in general, they do not offer any optimality guarantees. Recently, Benton et al. [5] developed an incremental planner, BBOP-LP, that uses branch-and-bound pruning for PSP planning, similar to our approach. BBOP-LP is able to offer optimality guarantees given sufficient resources. However, in contrast to HPLAN-P, it uses very different techniques for obtaining the heuristics. To compute heuristics it first relaxes the original planning problem and creates an integer programming (IP) model of this new problem. It then computes heuristics from a linear-programming relaxation of the IP model. Lastly, Feldmann et al. [18] propose a planner for PSPs that iteratively invokes METRIC-FF to find better plans.

Bonet and Geffner [9] have proposed a framework for planning with action costs and costs/rewards associated with fluents. Their cost model can represent PSPs as well as the simple preferences subset of PDDL3. They propose admissible heuristics and an optimal algorithm for planning under this model. Heuristics are obtained by compiling a relaxed instance of the problem to d-DNNF, while the algorithm is a modification of A^*. The approach does not scale very well for large planning instances, in part because of its need to employ an admissible heuristic.

Finally, there has been work that casts the preference-based planning problem as an answer set programming problem (ASP), as a constraint satisfaction problem (CSP), and as a satisfiability (SAT) instance. The paper by Son and Pontelli [35] proposed one of the first languages for preference-based planning, PP, and cast the planning problem as an optimization of an ASP problem. Their PP language includes TEPs expressed in LTL. Brafman and Chernyavsky [10] proposed a CSP approach to planning with final-state qualitative preferences specified using TCP-nets. Additionally, Giunchiglia and Maratea [24] proposed a compilation of preference-based planning problems into SAT. None of these approaches exploits heuristic search and thus are fundamentally different form the approach proposed here. The latter two approaches guide the search for a solution by imposing a variable/value ordering that will attempt to produce preferred solutions first. Because these works are recasting the problem into a different formalism, they explore a very different search space than our approach. Note also that the conversion to ASP, CSP or SAT requires assuming a fixed bound on plan length limiting the approach to at best finding k-optimal plans.

7.2. IPC-5 competitors

Most related to our work are the approaches taken by the planners that competed in IPC-5, both because they used the PDDL3 language and because many used some form of heuristic search. *Yochan*$^{\mathcal{PS}}$ [4] is a heuristic planner for simple preferences based on the Sapa^{ps} system [36]. Our approach is similar to theirs in the sense that both use a relaxed planning graph to obtain a heuristic estimate. *Yochan*$^{\mathcal{PS}}$ is also an incremental planner, employing heuristics geared towards classical goals. However, to compute its heuristic, it explicitly selects a subset of preferences to achieve then treats this subset as a classical goal. This process can be very costly in the presence of many preferences.

MIPS-XXL [17] and MIPS-BDD [15] both use Büchi automata to plan with temporally extended preferences. While the approach to compiling away the TEPs also constructs an automata (as in our approach), their translation process generates grounded preference formulae. This makes the translation algorithm prone to unmanageable 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 [26] forcing plans to have decreasing metric values. MIPS-BDD, on the other hand, performs a cost-optimal breath-first search that does not employ a heuristic.

Finally, the winner of the preferences tracks at IPC-5, SGPlan_5 [28], uses a completely different approach. It partitions the planning problem into several subproblems. It then uses a modified version of FF to solve those subproblems and finally integrates these sub-solutions into a solution for the entire problem. During the integration process it attempts to minimize the metric function. SGPlan_5 is not incremental, and seems to suffer from some non-robustness in its performance as shown by the results given in Table 4 (where its performance on an reformulated but equivalent domain changes quite dramatically).

8. Conclusions and future research

In this paper we have presented a new technique for planning with preferences that can deal with simple preferences, temporally extended preferences, and hard constraints. The core of the technique, our new set of heuristics and incremental search algorithm, are both amenable to integration with a variety of classical and simple-preference planners. The compilation technique for converting TEPs to simple preferences can also be made to work with other planners, although the method of embedding the constructed automata we utilize here might need some modification, dependent on the facilities available in that planner. Our method of embedding the constructed automata utilized TLPLAN's ability to deal with numeric functions and quantification. In particular, TLPLAN's ability to handle quantification allowed us to utilize the parameterized representation of the preferences generated by the compilation, leading to a considerably more compact domain encoding.

We have presented a number of different heuristics for planning with preferences. These heuristics have the feature that some of them account for the value that could be achieved from unsatisfied preferences, while others account for the difficulty of actually achieving these preferences. Our method for combining these different types of guidance is quite simple (tie-breaking), and more sophisticated combinations of these or related heuristics could be investigated. More generally, the question of identifying the domain features for which particular heuristics are most suitable is an interesting direction for future work.

We have also presented an incremental best-first search planning algorithm. A key feature of this algorithm is that it can use heuristic bounding functions to prune the search space during its incremental planning episodes. We have proved that under some fairly natural conditions our algorithm can generate optimal plans. It is worth noting that these conditions do not require the algorithm to utilize admissible heuristics. Nor do they require imposing a priori restrictions on the plan size (length or makespan) which would allow the algorithm to only achieve k-optimality rather than global optimality.

The algorithm can also employ different heuristics in each incremental planning episode, something we exploit during the very first planning episode by ignoring the preferences and only asking the planner to search for a plan achieving the goals. The motivation for this is that we want at least one working plan in hand before trying to find a more preferred plan. In our experiments, however, the remaining planning episodes are all executed with one fixed heuristic. More flexible schedules of heuristics could be investigated in future work.

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

Although the proposed heuristics perform reasonably well in many of the benchmarks we have tested, we have identified cases in which they perform poorly. In some cases, computing heuristics over the delete relaxation can provide bad guidance in the presence of preferences. The resolution of some of the issues we have raised above open interesting avenues for future research.

Acknowledgements

We gratefully acknowledge funding from Natural Sciences and Engineering Research Council of Canada (NSERC) and from the Ontario Ministry of Research and Innovation Early Researcher Award. We also thank Christian Fritz for helpful discussions during the development of our planner, and the anonymous reviewers for their useful feedback.

Appendix A. Proof for Proposition 2

In this section we prove Proposition 2. First, we prove three intermediate results that will be used by the final proof.

The first intermediate result says that if an NNF formula ϕ over P is true in a state s (denoted as $s \models \phi$), then ϕ will also be true in a relaxed state (F^+, F^-) if every proposition that is true in s is also true in such a relaxed state. This is proven in the following lemma.

Lemma 19. *Let P be a set of propositions, ϕ be an NNF formula, and $s, F^+, F^- \subseteq P$ be states. Then if $s \models \phi$, and (F^+, F^-) is such that*:

(1) $(F^+, F^-) \models p$, *for every* $p \in s$, *and*
(2) $(F^+, F^-) \models \neg p$, *for every* $p \in s^c$,

then $(F^+, F^-) \models \phi$.

Proof. The proof that follows is by induction on the structure of ϕ.

Base cases ($\phi = p$ *or* $\phi = \neg p$) They both follow directly from the conditions of this lemma.

Induction We have the following cases:

- If $\phi = \psi \wedge \xi$, then $s \models \psi$ and $s \models \xi$. By inductive hypothesis, also $(F^+, F^-) \models \psi$ and $(F^+, F^-) \models \xi$. It follows from Definition 1 that $(F^+, F^-) \models \phi$.
- If $\phi = \psi \vee \xi$, then the proof is analogous to the previous case.
- If $\phi = \forall x.\psi$, then for every $o \in \mathit{Objs}$ we have that $s \models \psi(x/o)$. By inductive hypothesis, for every $o \in \mathit{Objs}$ then $(F^+, F^-) \models \psi(x/o)$, hence by Definition 1, we have that $(F^+, F^-) \models \phi$.
- If $\phi = \exists x.\psi$, the proof is analogous to the previous case. □

The final intermediate result is actually a version of Proposition 2 but for simple facts.

Lemma 20. *Let s be a planning state, $R = (F_0^+, F_0^-)(F_1^+, F_1^-) \cdots (F_{m-1}^+, F_{m-1}^-)(F_m^+, F_m^-)$ be the relaxed planning graph constructed from s up to a fixed point. Moreover, let s_n be the state that results after performing a legal sequence of actions $a_1 \cdots a_n$ in s, then there exists some $k \leqslant m$ such that $(F_k^+, F_k^-) \models f$, for every $f \in s$, and such that $(F_k^+, F_k^-) \models \neg f$ for every $f \in s^c$.*

Proof. Since R has been constructed to a fixed point, $F_{m-1}^+ = F_m^+$ and $F_{m-1}^- = F_m^-$, and $m > 0$. Moreover, assume that the set of states generated by performing the action sequence over s is $s_1 \cdots s_n$ (i.e., state s_i is generated after performing the sequence of actions $a_1 \cdots a_i$ over s). The following proof for the lemma is by induction on the length of the action sequence, n.

Base case ($n = 0$) We prove that in this case we can consider $k = 0$. In this case the sequence of actions performed on s is empty. By definition of the construction of R, $F_k^+ = F_0^+ = s$ and $F_0^- = F_k^- = s^c$. Let f be an arbitrary fact.

(1) $f \in s$. Then, by Definition 1, $(F_k^+, F_k^+) \models f$, for $k = 0$ concluding the proof for this case.
(2) $f \in s^c$. Then, again by Definition 1, we obtain $(F_k^+, F_k^-) \models \neg f$, for $k = 0$.

Induction Let us assume that the theorem is true for $n-1$. We now prove that it is also true for n. We divide this proof into four cases. Again, assume f is an arbitrary fact.

(1) $f \in s_n$ and $f \in s_{n-1}$. This case is trivial, since by inductive hypothesis we have that $(F_k^+, F_k^-) \models f$ for some $k \leqslant m$.
(2) $f \notin s_n$ and $f \notin s_{n-1}$. Again, by induction hypothesis $(F_k^+, F_k^-) \models \neg f$ for some $k \leqslant m$.
(3) $f \in s_n$ and $f \notin s_{n-1}$. Then, a_n must have added fact f when performed in s_{n-1}. We now prove that action a_n is executable at some level $k' \leqslant m-1$ of the relaxed planning graph, and that it will add fact f to the graph at level $k'+1 \leqslant m$.
Let us assume that the precondition of action a_n is φ_P and that the condition of the conditional effect that adds f is φ_c. Then since both formulae are satisfied in s_{n-1}, we have that

$$s_{n-1} \models \varphi_P \wedge \varphi_c. \tag{A.1}$$

Moreover, by inductive hypothesis, we have that there exists a $k' \leqslant m$ such that

$$(F_{k'}^+, F_{k'}^-) \models p, \quad \text{for every } p \in s_{n-1}, \tag{A.2}$$

$$(F_{k'}^+, F_{k'}^-) \models \neg p, \quad \text{for every } p \in s_{n-1}^c. \tag{A.3}$$

At this point, we can safely assume also that $k' < m$, because if k' were equal to m, then (A.2) and (A.3) also hold for $k' = m-1$, because the graph has been constructed to a fixed point.

Now, we combine Eqs. (A.2), (A.3), and (A.1) with Lemma 19 to conclude that $(F^+_{k'}, F^-_{k'}) \models \varphi_P \wedge \varphi_c$. Action a_n is therefore executable at level k' of the relaxed planning graph, and the condition φ_c, which enables the conditional effect that adds f is also true at level k'. Therefore, f is added to the graph at level $k = k' + 1 \leqslant m$, concluding the proof for this case.

(4) $f \notin s_n$ and $f \in s_{n-1}$. Proof is analogous to previous case. □

Now we are ready to prove our result.

Proof for Proposition 2. By Lemma 20, we know that there exists a $k \leqslant m$ such that for each $p \in s_n$, $(F^+_k, F^-_k) \models p$, and for each $p \in s^c$ then $(F^+_k, F^-_k) \models \neg p$. Because $s_n \models \phi$ it follows from Lemma 19 that $(F^+_k, F^-_k) \models \phi$. □

Appendix B. Proof for Theorem 10

Before we start our proof we prove a lemma which establishes that, under the conditions of Theorem 10, if two nodes with exactly the same state have different B, D, or O metric value, then their lengths must also differ analogously.

Lemma 21. *Let N_1 and N_2 be two search nodes that correspond to the same planning state s. Furthermore, let the metric M of the instance be NDVPL and depend on* `(total-time)`*. If $R(N_1) \leqslant R(N_2)$, and*:

(1) *R is either O or B, or*
(2) *M is ATT and R is D.*

then $\texttt{length}(N_1) \leqslant \texttt{length}(N_2)$.

Proof. We divide the proof in two cases.

Case 1: R is either O or B. Then $R(N_1) = M(N'_1)$, where N'_1 is a hypothetical node with the same `length` as N_1 but in which possibly more preferences are satisfied. Analogously, $R(N_2) = M(N'_2)$ for a node N'_2 with the same `length` as N_2. Therefore,

$$M(N'_1) \leqslant M(N'_2). \tag{B.1}$$

Because the planning state associated to N_1 and N_2 are identical, we know that N'_2 and N'_1 are such that they satisfy exactly the same preferences, i.e., if Γ is the set of preferences of the planning instance, for all $p \in \Gamma$ we have that $\texttt{is-violated}(p, N'_1) = \texttt{is-violated}(p, N'_2)$. Now, using the contra-positive of implication (2) in the NDVPL definition (Definition 3) and Eq. (B.1), we have that $\texttt{length}(N'_1) \leqslant \texttt{length}(N'_2)$. This implies that $\texttt{length}(N_1) \leqslant \texttt{length}(N_2)$, and concludes the proof for this case.

Case 2: R is D and M is ATT. Because M is ATT, then by Eq. (1), $D(N_1) = M(N_1) + R_1$, where R_1 is an expression that does not depend on `(total-time)`, i.e. it *only* depends on N_1's state. Likewise, $D(N_2) = M(N_2) + R_2$, where R_2 only depends on the state of N_2. Since both the states corresponding to N_1 and N_2 are equal, we have that $R_1 = R_2$. Hence, because $D(N_1) \leqslant D(N_2)$ we have that $M(N_1) \leqslant M(N_2)$, which by the contra-positive of implication (2) in the NDVPL definition (Definition 3) implies that $\texttt{length}(N_1) \leqslant \texttt{length}(N_2)$. This concludes this case, finishing the proof. □

Now we are ready to prove our result. First, note that the search is restarted from scratch after the first plan is found. This also means that the closed list is reinitialized. Second, note that if two nodes N_1 and N_2 have the same state associated to them then both the G and the P functions evaluated on these nodes return the same value. Therefore, if UserHeuristic$(N_1) \leqslant$ UserHeuristic(N_2), then this means that the tie breaker functions used, say R, is such that $R(N_1) \leqslant R(N_2)$ where R is either O, B or D.

The sketch of the proof is as follows. We assume that a node N that leads to an optimal plan is discarded by the algorithm. Then we prove that if this happens then either the optimal was found or there is a node in the frontier that can be extended to another optimal plan.

Assume there exists an optimal plan $p_1 = a_1 a_2 \cdots a_n$ that traverses the sequence of states $s_0 s_1 \cdots s_n$. Let N_1 be a node formed by applying p_1 on s_0. Because the metric is NDVPL, we assume that this plan contains no cycles (otherwise, had the plan contained any cycles, by removing them we could not make it worse). Suppose further that at some point in the search, there is a node N that is generated by applying $a_1 a_2 \cdots a_j$ in the initial state (with $j < n$) and that is discarded by the algorithm in line 8. This means that there exists another closed node, say N_C that is associated the same state as N, and that is such that

$$\text{UserHeuristic}(N_C) \leqslant \text{UserHeuristic}(N). \tag{B.2}$$

Both nodes are associated the same state s_j, hence the `is-violated` counters are identical for each preference. This means that N_C is constructed from s_0 by a sequence of actions $b_1 b_2 \cdots b_k$. This sequence of actions gets to the same state s_j, hence the sequence $p_2 = b_1 b_2 \cdots b_k a_{j+1} \cdots a_n$ is also a plan.

Let N_2 be a node that would be constructed by applying p_2 in s_0. Now we prove that N_2 also corresponds to an optimal plan. We have two cases.

Case 1: The metric depends on (`total-time`). Because inequality (B.2) implies that $R(N_C) \leqslant R(N)$, where R is either O, D or B, by Lemma 21, we have that $\texttt{length}(N_C) \leqslant \texttt{length}(N)$, and therefore $k \leqslant j$. We clearly have that $\texttt{length}(N_2) \leqslant \texttt{length}(N_1)$, furthermore because all precondition counters are identical, it follows from the NDVPL condition that $M(N_2) \leqslant M(N_1)$. Given that N_1 represents an optimal plan, we conclude that $M(N_2) = M(N_1)$, and therefore N_2 also represents an optimal plan.

Case 2: The metric does not depend on (`total-time`). Therefore, because node N_2 reaches the same state as N_1 does and M only depends on properties encoded in the state, $M(N_1) = M(N_2)$ and hence N_2 also represents an optimal plan. This concludes case 2.

Now, we know that since N_C, a predecessor of N_2 was expanded by the algorithm, one of the following things happen:

(1) A successor of N_C is in *frontier*. In this case, the condition of Definition 8 follows immediately.
(2) N_2 is in the closed list. This implies that the condition of Definition 8 is also satisfied.
(3) A successor of N_C has been discarded by the algorithm. In this case, such a successor also leads to an optimal plan. This means that we could apply the same argument in this proof for such a node, leading to eventually satisfy the condition of Definition 8 since the algorithm has visited finitely many nodes.

References

[1] F. Bacchus, F. Kabanza, Planning for temporally extended goals, Annals of Mathematics and Artificial Intelligence 22 (1–2) (1998) 5–27.
[2] J.A. Baier, S.A. McIlraith, Planning with first-order temporally extended goals using heuristic search, in: Proceedings of the 21st National Conference on Artificial Intelligence (AAAI), Boston, MA, 2006, pp. 788–795.
[3] J.A. Baier, S.A. McIlraith, On domain-independent heuristics for planning with qualitative preferences, in: 7th Workshop on Nonmonotonic Reasoning, Action and Change (NRAC), 2007.
[4] J. Benton, S. Kambhampati, M.B. Do, YochanPS: PDDL3 simple preferences and partial satisfaction planning, in: 5th International Planning Competition Booklet (IPC-2006), Lake District, England, July 2006, pp. 54–57.
[5] J. Benton, M. van den Briel, S. Kambhampati, A hybrid linear programming and relaxed plan heuristic for partial satisfaction problems, in: Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS), Providence, RI, September 2007, pp. 34–41.
[6] M. Bienvenu, C. Fritz, S. McIlraith, Planning with qualitative temporal preferences, in: Proceedings of the 10th International Conference on Knowledge Representation and Reasoning (KR), Lake District, England, 2006, pp. 134–144.
[7] A. Blum, M.L. Furst, Fast planning through planning graph analysis, Artificial Intelligence 90 (1–2) (1997) 281–300.
[8] B. Bonet, H. Geffner, Planning as heuristic search, Artificial Intelligence 129 (1–2) (2001) 5–33.
[9] B. Bonet, H. Geffner, Heuristics for planning with penalties and rewards using compiled knowledge, in: Proceedings of the 10th International Conference on Knowledge Representation and Reasoning (KR), 2006, pp. 452–462.
[10] R. Brafman, Y. Chernyavsky, Planning with goal preferences and constraints, in: Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS), Monterey, CA, June 2005, pp. 182–191.
[11] A.I. Coles, A.J. Smith, Marvin: A heuristic search planner with online macro-action learning, Journal of Artificial Intelligence Research 28 (February 2007) 119–156.
[12] J.P. Delgrande, T. Schaub, H. Tompits, Domain-specific preferences for causal reasoning and planning, in: Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS), Whistler, Canada, June 2004, pp. 63–72.
[13] Y. Dimopoulos, A. Gerevini, P. Haslum, A. Saetti, The benchmark domains of the deterministic part of ipc-5, http://zeus.ing.unibs.it/ipc-5/, July 2006.
[14] M.B. Do, J. Benton, M. van den Briel, S. Kambhampati, Planning with goal utility dependencies, in: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI), Hyderabad, India, 2007, pp. 1872–1878.
[15] S. Edelkamp, Optimal symbolic PDDL3 planning with MIPS-BDD, in: 5th International Planning Competition Booklet (IPC-2006), Lake District, England, July 2006, pp. 31–33.
[16] S. Edelkamp, J. Hoffmann, PDDL2.2: The language for the classical part of the 4th International Planning Competition, Tech. Rep. 195, Computer Science Department, University of Freiburg, 2004.
[17] S. Edelkamp, S. Jabbar, M. Naizih, Large-scale optimal PDDL3 planning with MIPS-XXL, in: 5th International Planning Competition Booklet (IPC-2006), Lake District, England, July 2006, pp. 28–30.
[18] R. Feldmann, G. Brewka, S. Wenzel, Planning with prioritized goals, in: Proceedings of the 10th International Conference on Knowledge Representation and Reasoning (KR), Lake District, England, July 2006, pp. 503–514.
[19] R. Fikes, N.J. Nilsson, STRIPS: A new approach to the application of theorem proving to problem solving, Artificial Intelligence 2 (3/4) (1971) 189–208.
[20] M. Fox, D. Long, PDDL2.1: An extension to PDDL for expressing temporal planning domains, Journal of Artificial Intelligence Research 20 (2003) 61–124.
[21] B.C. Gazen, C.A. Knoblock, Combining the expressivity of UCPOP with the efficiency of graphplan, in: ECP97. Toulouse, France, September 1997, pp. 221–233.
[22] A. Gerevini, Y. Dimopoulos, P. Haslum, A. Saetti, 5th International Planning Competition, http://zeus.ing.unibs.it/ipc-5/, July 2006.
[23] A. Gerevini, D. Long, Plan constraints and preferences for PDDL3, Tech. Rep. 2005-08-07, Department of Electronics for Automation, University of Brescia, Brescia, Italy, 2005.
[24] E. Giunchiglia, M. Maratea, Planning as satisfiability with preferences, in: Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI), Vancouver, British Columbia, 2007, pp. 987–992.
[25] P. Haslum, Openstacks SP-NCE domain, http://users.rsise.anu.edu.au/~patrik/ipc5.html, 2007.
[26] J. Hoffmann, The Metric-FF planning system: Translating ignoring delete lists to numeric state variables, Journal of Artificial Intelligence Research 20 (2003) 291–341.
[27] J. Hoffmann, B. Nebel, The FF planning system: Fast plan generation through heuristic search, Journal of Artificial Intelligence Research 14 (2001) 253–302.
[28] C.-W. Hsu, B. Wah, R. Huang, Y. Chen, Constraint partitioning for solving planning problems with trajectory constraints and goal preferences, in: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI), Hyderabad, India, January 2007, pp. 1924–1929.
[29] D.V. McDermott, A heuristic estimator for means-ends analysis in planning, in: AIPS96, 1996, pp. 142–149.
[30] D.V. McDermott, PDDL—The Planning Domain Definition Language, Tech. Rep. TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[31] E.P.D. Pednault, ADL: Exploring the middle ground between STRIPS and the situation calculus, in: Proceedings of the 1st International Conference of Knowledge Representation and Reasoning (KR), Toronto, Canada, May 1989, pp. 324–332.
[32] A. Pnueli, The temporal logic of programs, in: Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS), 1977, pp. 46–57.
[33] R. Sanchez, S. Kambhampati, Planning graph heuristics for selecting objectives in over-subscription planning problems, in: Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS), Monterey, CA, 2005, pp. 192–201.
[34] D.E. Smith, Choosing objectives in over-subscription planning, in: Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS), Whistler, Canada, 2004, pp. 393–401.
[35] T.C. Son, E. Pontelli, Planning with preferences using logic programming, in: V. Lifschitz, I. Niemela (Eds.), Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR), in: LNCS, vol. 2923, Springer, 2004, pp. 247–260.
[36] M. van den Briel, R.S. Nigenda, M.B. Do, S. Kambhampati, Effective approaches for partial satisfaction (over-subscription) planning, in: Proceedings of the 19th National Conference on Artificial Intelligence (AAAI), 2004, pp. 562–569.
[37] L. Zhu, R. Givan, Simultaneous heuristic search for conjunctive subgoals, in: Proceedings of the 20th National Conference on Artificial Intelligence (AAAI), Pittsburgh, Pennsylvania, USA, July 9–13 2005, pp. 1235–1241.