HTN Planning with Quantitative Preferences via Heuristic Search

Shirin Sohrabi **Jorge Baier** **Sheila A. McIlraith**
Department of Computer Science,
University of Toronto,
Toronto, Canada.
{shirin, jabaier, sheila}@cs.toronto.edu

Abstract

In this paper, we address the problem of generating preferred plans by combining the procedural control knowledge specified by Hierarchical Task Networks (HTNs) with rich user preferences. To this end, we extend the popular Plan Domain Description Language, PDDL3, to support specification of preferences over HTN constructs. To compute preferred HTN plans, we propose a branch-and-bound algorithm, together with a set of heuristics that, leveraging HTN structure, measure progress towards satisfaction of preferences. Our preference-based planner, **HTNPLAN-P**, is implemented as an extension of **SHOP2**. We evaluated a variety of search strategies with respect to the quality of the plans generated. At least one of our strategies consistently equalled or outperformed **SGPlan**$_5$, winner of the 2006 International Planning Competition preference tracks. While our implementation builds on **SHOP2**, the language and techniques proposed here are relevant to a broad range of HTN planners.

1 Introduction

Hierarchical Task Network (HTN) planning is a popular and widely used planning paradigm, and many domain-independent HTN planners exist (e.g., **SHOP2**, SIPE-2, I-X/I-PLAN, O-PLAN) (Ghallab, Nau, and Traverso 2004). In HTN planning, the planner is provided with a set of tasks to be performed, possibly together with constraints on those tasks. A plan is then formulated by repeatedly decomposing tasks into smaller and smaller subtasks until primitive, executable tasks are reached. A primary reason behind HTN's success is that its task networks capture useful procedural control knowledge—advice on how to perform a task—described in terms of a decomposition of subtasks. Such control knowledge can significantly reduce the search space for a plan while also ensuring that plans follow one of the stipulated courses of action.

While HTNs specify a family of satisfactory plans, they are, for the most part, unable to distinguish between successful plans of differing quality. Preference-based planning (PBP) augments a planning problem with a specification of properties that constitute a high-quality plan. For example, if one were generating an air travel plan, a high-quality plan might be one that minimizes cost, uses only direct flights, and flies with a preferred carrier. PBP attempts to optimize the satisfaction of these preferences while achieving the stipulated goals of the plan. To develop a preference-based HTN planner, we must develop a specification language that references HTN constructs, and a planning algorithm that computes a preferred plan while respecting the HTN planning problem specification.

In this paper we extend the Plan Domain Description Language, PDDL3 (Gerevini and Long 2005), with HTN-specific preference constructs. This work builds on our recent work on the development of $\mathcal{LPH}$ (Sohrabi and McIlraith 2008), a *qualitative* preference specification language designed to capture HTN-specific preferences. PDDL3 preferences are *state centric*, identifying preferred states along the plan trajectory. To develop a preference language for HTN we add *action-centric* constructs to PDDL3 that can express preferences over the occurrence of primitive actions (operators) within the plan trajectory, as well as expressing preferences over complex actions (tasks) and how they decompose into primitive actions. For example, we are able to express preferences over which sets of subtasks are preferred in realizing a task (e.g., *When booking inter-city transportation, I prefer to book a flight*), preferred parameters to use when choosing a set of subtasks to realize a task (e.g., *I prefer to book a flight with Air Canada*), and so on. To compute preferred HTN plans, we propose a branch-and-bound algorithm, together with a set of heuristics that leverage HTN structure.

In Section 2, we review HTN planning, PDDL3, and provide a definition for HTN PBP. In Section 3, we augment PDDL3 with HTN-specific preference constructs. Section 4 describes a preprocessing algorithm that transforms a temporally extended PBP problem into one where preferences are expressed in terms of the final state of the plan. The resultant representation is more amenable to heuristic search. We discuss our preference-based planner's algorithm and heuristics in Section 5 and results of our experimental evaluation in Section 6. We discuss related work in Section 7.

2 Background

2.1 HTN Planning

In this section, we provide a brief overview of HTN planning, following Ghallab, Nau, and Traverso (2004).

Example 1 (Travel Example) Consider a simple HTN planning problem to address the task of arranging travel. This task can be decomposed into arranging transportation, accommodations, and local transportation. Each of these tasks can again be decomposed based on alternate modes of transportation and accommodations, reducing eventually to primitive actions that can be executed in the world. Further constraints can be imposed to restrict decompositions.

Definition 1 (HTN Planning Problem) *An HTN planning problem is a 3-tuple $\mathcal{P} = (s_0, w_0, D)$ where s_0 is the initial state, w_0 is a task network called the initial task network, and D is the HTN planning domain. $\mathcal{P}$ is a total-order planning problem if w_0 and D are totally ordered; otherwise it is said to be partially ordered.*

A *task* consists of a task symbol and a list of arguments. A task is primitive if its task symbol is an operator name and its parameters match, otherwise it is *nonprimitive*. In our example, *arrange-trans* and *arrange-acc* are nonprimitive tasks, while *book-flight* and *book-car* are primitive tasks.

Definition 2 (Task Network) *A task network is a pair w=(U, C) where U is a set of task nodes and C is a set of constraints. Each task node $u \in U$ contains a task t_u. If all of the tasks are ground then w is ground; If all of the tasks are primitive, then w is called primitive; otherwise it is called nonprimitive. Task network w is totally ordered if C defines a total ordering of the nodes in U.*

In our example, we could have a task network (U, C) where $U = \{u_1, u_2\}$, u_1 =*book-car*, and u_2= *pay*, and C is a precedence constraint such that u_1 must occur before u_2 and a before-constraint such that at least one car is available for rent before u_1.

A domain is a pair $D = (O, M)$ where O is a set of operators and M is a set of methods. Operators are essentially primitive actions that can be executed in the world. They are described by a triple *o =(name(o), pre(o), eff(o))*, corresponding to the operator's name, preconditions and effects. Preconditions are restricted to a set of literals, and effects are described as STRIPS-like Add and Delete lists. An operator o can accomplish a ground primitive task in a state s if their names match and o is applicable in s. In our example, ignoring the parameters, operators might include: *pay, book-train, book-car, book-hotel,* and *book-flight*.

A method, m, is a 4-tuple (*name(m), task(m),subtasks(m), constr(m))* corresponding to the method's name, a nonprimitive task and the method's task network, comprising subtasks and constraints. A method is *totally ordered* if its task network is *totally ordered*. A domain is a total-order domain if every $m \in M$ is *totally ordered*. Method m is relevant for a task t if there is a substitution σ such that $\sigma(t)$ =*task*(m). Several different methods can be relevant to a particular nonprimitive task t, leading to different decompositions of t. In our example, the method with *name by-flight-trans* can be used to decompose the *task arrange-trans* into the *subtasks* of booking a flight and paying, with the constraint (*constr*) that the booking precede payment.

Definition 3 (Solution to HTN Planning Problem) *Given an HTN planning problem $\mathcal{P} = (s_0, w_0, D)$, a plan $\pi = (o_1, ..., o_k)$ is a solution for $\mathcal{P}$ iff one of the following cases hold.* Case 1*: w_0 is primitive and there exists a ground instance of (U', C') of (U, C) and a total ordering $(u_1, ..., u_k)$ of the nodes in U' such that for all $1 \leq i \leq k$, name(o_i) = t_{u_i}, the plan π is executable in the state s_0, and all the constraints hold.* Case 2*: w_0 is nonprimitive, and there exists a sequence of task decompositions that can be applied to w_0 to produce a primitive task network w', where π is a solution for w'.*

Finally, to define the notion of *preference-based* planning we assume the existence of a reflexive and transitive relation $\preceq$ between plans. If π_1 and π_2 are plans for $\mathcal{P}$ and $\pi_1 \preceq \pi_2$ we say that π_1 is *at least as preferred as* π_2. We use $\pi_1 \prec \pi_2$ as an abbreviation for $\pi_1 \preceq \pi_2$ and $\pi_2 \not\preceq \pi_1$.

Definition 4 (Preference-based HTN Planning) *An HTN planning problem with user preferences is described as a 4-tuple $\mathcal{P} = (s_0, w_0, D, \preceq)$ where $\preceq$ is a preorder between plans. A plan π is a solution to $\mathcal{P}$ if and only if: π is a plan for $\mathcal{P}' = (s_0, w_0, D)$ and there does not exists a plan π' for $\mathcal{P}'$ such that $\pi' \prec \pi$.*

The $\preceq$ relation can be defined in many ways. Below we describe PDDL3, which defines $\preceq$ quantitatively through a metric function.

2.2 Brief Description of PDDL3

The Plan Domain Description Language (PDDL) is the de facto standard input language for many planning systems. PDDL3 (Gerevini and Long 2005) extends PDDL2.2 by enabling the specification of preferences and hard constraints. It also provides a way of defining a *metric function* that defines the quality of a plan. The rest of this section briefly describes these new elements.

Temporally extended preferences/constraints PDDL3 specifies temporally extended preferences (TEPs) and temporally extended hard constraints in a subset of linear temporal logic (LTL). Both are declared using the `:constraints` construct. Preferences are given names in their declaration, to allow for later reference. By way of illustration, the following PDDL3 code defines two preferences and one hard constraint.

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

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

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

Temporal operators cannot be nested in PDDL3. Our approach can however handle the more general case of nested temporal operators.

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.

Simple Preferences

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

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

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

Metric Function

The metric function defines the quality of a plan, generally depending on the preferences that have been achieved by the plan. To this end, the PDDL3 expression `(is-violated name)`, returns the number of individual preferences in the `name` family of preferences that 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 returns the plan length. Finally, it is also possible to define whether we want to maximize or minimize the metric, and how we want to weigh its different components. For example, the PDDL3 metric function:

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

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

Since it is always possible to transform a metric that requires maximization into one that requires minimization, we will henceforth assume that the metric is always being *minimized*.

Finally, the formal definition for HTN planning with PDDL3 preferences is an instance of our previous definition: Given a PDDL3 metric function M the *HTN preference-based planning problem with PDDL3 preferences* is defined by Definition 4, where the relation $\preceq$ is defined by:

$$\pi_1 \preceq \pi_2 \text{ if and only if } M(\pi_1) \leq M(\pi_2).$$

3 PDDL3 Extended to HTN

In this section, we extend PDDL3 with the ability to express preferences over HTN constructs. As argued in Section 1, supporting preferences over how tasks are decomposed, their preferred parameterizations, and the conditions underwhich these preferences hold, is compelling. It goes beyond the traditional specification of preferences over the properties of plan trajectories to provide preferences over non-functional properties of the planning problem. This is particularly compelling when HTN methods are realized using web service software components, because these services have many non-functional properties that distinguish them (e.g., credit cards accepted, country of origin, trustworthiness, etc.) and that influence user preferences (Sohrabi, Prokoshyna, and McIlraith 2006).

In designing a preference specification language for HTN planning, we made a number of strategic design decisions. We first considered adding our preference specifications directly to the definitions of HTN methods. This seemed like a natural extension to the hard constraints that are already part of method definitions. Unfortunately, this precludes easy contextualization of methods relative to the task the method is realizing. For example, in the travel domain, many methods may eventually involve the primitive operation of *paying*, but a user may prefer different methods of payment dependent upon the high-level task being realized (e.g., *When booking a car, pay with amex to exploit amex's free collision coverage, when booking a flight, pay with my Aeroplan-visa to collect travel bonus points*, etc.). We also found the option of including preferences in method definitions unappealing because we wished to separate domain-specific, but user-independent knowledge, such as method definitions, from user-specific preferences. Separating the two, enables users to share method definitions but individualize preferences. We also wished to leverage the popularity of PDDL3 as a language for preference specifications.

This work builds on our recent work on $\mathcal{LPH}$ (Sohrabi and McIlraith 2008), a *qualitative* preference specification language designed to capture HTN-specific preferences. $\mathcal{LPH}$ supports specification of action-centric preferences (i.e., preferences over the occurrence of actions, and in the case of HTNs, tasks) as well as state-centric preferences, whereas PDDL3 is strictly state centric. Further, $\mathcal{LPH}$ aggregates preferences into one (qualitative) preference formula using Boolean connectives, whereas PDDL3 aggregates preferences into a (quantitative) metric function. In this paper, we capture some of the merits of both languages by extending PDDL3 to incorporate the action-centric preferences of $\mathcal{LPH}$. This gives users the ability to express preferences over certain parameterization of a task (e.g., preferring one task grounding to another), over a certain decomposition of nonprimitive tasks (i.e., prefer to apply a certain method over another), and a soft version of the before, after, and in between constraints.

To support preferences over task occurrences (primitive and nonprimitive) and task decompositions, we added two new constructs to PDDL3, **occ**(t) and **apply**(n), where t is a task and n is the name of a method. **occ**(t) states that the task t occurs in the present state, and **apply**(n) states that a method whose name is n is applied to decompose a nonprimitive task.

The following are a few temporally extended preferences from our travel domain [1] that use of the above extension.

```
(:constraints
 (and
  (preference  p1
        (always (not (occ (pay MasterCard)))))
  (preference  p2 (sometime (occ
     (book-flight SA Eco Direct WindowSeat))))
  (preference  p3
    (sometime (apply (by-car-local SUV Avis))))
  (preference  p4
    (sometime-before (occ (arrange-trans))
                     (haveHotelReservation)))
  (preference  p5
    (sometime-before  (occ (arrange-trans))
                      (occ (arrange-acc))))))
```

The *p1* preference states that the user never pays by Mastercard. The *p2* preference states that at some point the user books a direct economy window-seated flight with a Star Alliance carrier. *P3* states that at some point, the *by-car-local* method is applied to book an SUV from Avis. *P4* states that the hotel is reserved before transportation is arranged. Finally *P5* states that the *arrange-trans* task occurs before the *arrange-acc* task.

4 Preprocessing HTN problems

Before searching for a most preferred HTN plan, we preprocess the original problem. The resulting problem facilitates the exploitation of our heuristics and our PBP algorithm.

The preprocessing is done in two phases. In the first phase we convert the problem into a more standard one, in the sense that it contains fewer elements that are not seen in classical planning problems. In our preprocessed problems, for example, all temporally extended preferences are transformed into final-state preferences, and thus, reasoning about temporally extended preferences is no different from reasoning about standard predicates. Thus, we do not need to implement specialized algorithms to reason about LTL formulae such as the progression algorithm used by TLPLAN (Bacchus and Kabanza 1998), and which we also exploited in (Sohrabi and McIlraith 2008). Also the compilation enables our heuristic functions to be defined only in terms of domain predicates, rather than being based on non-standard evaluations of an LTL formulae, such as the ones used by other approaches (e.g. Bienvenu, Fritz, and McIlraith, 2006). In the second phase of the preprocessing, we extract a subset of the effects from recursive tasks, and generate a set of extra *heuristic operators* that have those effects. These new heuristic operators are not used during search to decompose a task, but are exploited by our heuristics as an abstraction of the tasks. Now we describe in more detail each of these phases.

[1]To simplify the examples many parameters have been suppressed.

4.1 Preprocessing HTN-PDDL3 Preferences

For a given problem, this phase compiles away many of the HTN-PDDL3 elements, generating an equivalent problem that does not contain temporally extended or precondition preferences. Some of the techniques below are specific to HTN-PDDL3 preferences, whereas others are borrowed from previous approaches.

Preprocessing Tasks and Methods HTN-PDDL3 preferences can refer to the occurrence of tasks and the application of methods. In order to reason about task occurrences and method applications, we preprocess the methods of our HTN problem. In the compiled problem, for each non-primitive task t that occurs in some preference of the original problem, there is a new predicate $executing$-t. If $a_0 a_1 \cdots a_n$ is a plan for the problem, and a_i and a_j are respectively the first and last primitive actions that resulted from decomposing t, then $executing$-t is true in all the states in between the application of a_i and a_j. This is accomplished by adding new actions at the beginning and end of each task network in the methods that decompose t.

Further, for each primitive task (i.e., operator) t occurring in the preferences, we extend the compiled problem with a new occ-t predicate, such that occ-t is true iff t has just been performed.

Finally, we modify each method m whose name n (i.e., $n = name(m)$) occurs in some preference. We introduce a new fluent *hasBeenApplied*-n which is true iff method m has been applied to decompose a task. To achieve this, we add special actions to the beginning of the task networks associated with each method.

Preprocessing occ and apply Modal Operators Now that we have the $executing$ and the $hasBeenApplied$ fluents, we replace each occurrence of **occ**(t) by $executing$-t when t is non-primitive and by occ-t when t is primitive. Occurrences of **apply**(n) are replaced by *hasBeenApplied*-n.

Up to this point all our HTN-PDDL3 preferences exclusively reference fluents of the HTN problem, enabling us to apply standard techniques to simplify the problem further.

Temporally Extended Preferences We use an existing compilation technique (Baier and McIlraith 2006) to encode the satisfaction of temporally extended preferences into predicates of the domain. For each LTL preference φ in the original problem, we generate additional predicates for the compiled domain that encode the various ways in which φ can become true. Indeed, the additional predicates represent a finite-state automaton for φ, where the accepting state of the automaton represents satisfaction of the preference. In our resulting domains, we axiomatically define an *accepting predicate for* φ, which represents the accepting condition of φ's automaton. The accepting predicate is true at a state s if and only if φ is satisfied at s.

In the case of quantified preferences, the predicates are parametric. By way of illustration, consider the `p-light` preference defined above. For this quantified preference, our compiled problem will contain a `p-light-satisfied` predicate, such the fact `(p-light-satisfied l1)` is true iff `l1` has been turned on. The operators in the compiled domain are modified in such a way that the predicate will be updated correctly. In the case of our example, as soon as the action *switch-on* is applied on a light ℓ, the accepting predicate for `p-light` is updated accordingly for ℓ, and will remain true thereafter.

Precondition Preferences Precondition preferences are encoded as conditional costs for actions. For each precondition preference, we associate a counter function in the compiled domain, that is incremented whenever an action has been performed violating some of its precondition preferences. This process is exactly the same as the one used in the **HPLAN-P** planner (Baier, Bacchus, and McIlraith 2007).

4.2 Abstracting Recursive Tasks

For each recursive task t in the original problem we generate an additional, heuristic operator h-t. The new operator captures only *some* of the effects of t, and is used by our *lookahead* technique to quickly estimate an upperbound on the quality of decomposing a task with a certain method.

Our technique can currently preprocess a subclass of recursive tasks: those that are *self-recursive* and that have a *single, void endpoint* (SRSVE) tasks. An SRSVE task t is such that the methods that decompose t can only use t as a non-primitive task, and such that there is only one primitive decomposition, which is also empty. More formally,

Definition 5 (SRSVE) *Let t be a non-primitive task, and let $\mathcal{M}_t$ be the set of methods that may decompose it. Then, t is* self-recursive *and has a* single, void endpoint *(SRSVE) iff M_t is such that (1) it contains one and only one method with no tasks (i.e., a void method) which may only contain a precondition; and (2) all its remaining methods contain the (non-primitive) task t and no other non-primitive task (self-recursive).*

Example 2 The `move-robot-to-depot` task – shown below in SHOP2 syntax – can be decomposed by two methods. One of them is self-recursive (`case2`), and the other one (`case1`) is void (does not contain any tasks). This task is thus SRSVE.

```
(:method move-robot-to-depot
 case1 ((at robot depot)) ; robot at depot?
        ()                 ; do nothing
 case2 ((at robot ?x)
        (closest-to-depot ?x ?y))
       ((move ?x ?y)
        (move-robot-to-depot))) ;recursive call
```

Here the precondition associated with method `case1` is `(at robot depot)`. §

The h-t operator is a relaxation of t, which has as an effect the condition that has to be satisfied in order to end the decomposition of t, which is generally the intended effect of the task t. Formally, let $t(\vec{x})$ be an SRSVE task, and let $C(\vec{x})$ be the precondition associated with the void method that decomposes it. Then, our preprocessing phase generates the operator h-$t(\vec{x})$, with a positive effect $C(\vec{x})$, no negative effects, and with no preconditions (i.e, always executable). Since h-t is a new operator, and is not a member of any task, it will never be used to construct a plan; only our heuristic techniques will use it.

```
1:  function HTNPLAN-P($s_0$, $w_0$, $D$, METRICFN, HEURISTICFN)
2:    $frontier \leftarrow$ INITFRONTIER($s_0, w_0, \emptyset$)  ▷ initialize frontier
3:    $bestMetric \leftarrow$ worst case upper bound
4:    while $frontier$ is not empty do
5:      $current \leftarrow$ Extract best element from $frontier$
6:      $\langle s, w, partialP \rangle \leftarrow current$
7:      $lbound \leftarrow$ METRICBOUNDFN($s$)
8:      if $lbound < bestMetric$ then      ▷ pruning by bounding
9:        if $w = \emptyset$ and $current$'s metric $< bestMetric$ then
10:         Output plan $partialP$
11:         $bestMetric \leftarrow$ METRICFN($s$)
12:       end if
13:       $succ \leftarrow$ successors of $current$
14:       $frontier \leftarrow$ merge $succ$ into $frontier$
15:     end if
16:   end while
17: end function
```

Figure 1: A sketch of our HTN PBP algorithm.

5 Preference-based Planning with HTN

We address the problem of finding a most preferred decomposition of an HTN by performing a best-first, incremental search in the plan search space induced by the initial task network. The search is performed in a series of *episodes*, each of which returns a sequence of ground primitive operators (i.e., a plan that satisfies the initial task network). During each episode, the search performs branch-and-bound pruning—a search node is pruned from the search space, if we can prove that it will not lead to a plan that is better than the one found in the previous episode. In the first episode no pruning is performed. In each episode, search is guided by *inadmissible heuristics*, designed specifically to guide the search quickly to a reasonably good decomposition. The remainder of this section describes the heuristics we use, and the planning algorithm.

5.1 Algorithm

Our HTN PBP algorithm outlined in Figure 1, performs a best-first, incremental search in the space of decompositions of a given initial task network. It takes as input an initial planning state s_0, an initial task network w_0, an HTN planning domain D, a metric function METRICFN, and a heuristic function HEURISTICFN.

The main variables kept by the algorithm are $frontier$ and $bestMetric$. $frontier$ contains the nodes in the search frontier. Each of these nodes is of the form $\langle s, w, partialP \rangle$, where s is a plan state, w is a task network, and $partialP$ is a partial plan. Intuitively, a search node $\langle s, w, partialP \rangle$ represents the fact that task network w remains to be decomposed in state s, and that state s is reached from the initial state of the planning problem s_0 by performing the sequence of actions $partialP$. $frontier$ is initialized with a single node $\langle s_0, w_0, \emptyset \rangle$, where $\emptyset$ represents the empty plan. Its elements are always sorted according to the function HEURISTICFN. On the other hand, $bestMetric$ is a variable that stores the metric value of the best plan found so far, and it is initialized to a high value representing a worst case upper bound.

Search is carried out in the main **while** loop. In each iteration, **HTNPLAN-P** extracts the best element from the $frontier$ and places it in $current$. Then, an estimation of a lowerbound of the metric value that can be achieved by decomposing $current$'s task network, w, is computed (Line 7) using the function METRICBOUNDFN. Function METRICBOUNDFN will be computed using the *optimistic metric* function described in the next subsection.

The algorithm *prunes* $current$ from the search space if $lbound$ is greater than or equal to $bestMetric$. Otherwise, **HTNPLAN-P** checks whether or not $current$ corresponds to a plan (this happens when its task network is empty). If $current$ corresponds to a plan, the sequence of actions in its tuple is returned and the value of $bestMetric$ is updated.

The search then continues by computing all the successors to $current$ using the Partial-order Forward Decomposition procedure (PFD) (Ghallab, Nau, and Traverso 2004). The successors of $current$ will be a new list of nodes in the form $\langle s', w', partialP' \rangle$, one for each legal task decomposition that can be reached by performing $current$'s task network using a PFD procedure. Then the successors of $current$ will be merged into the $frontier$. The algorithm terminates when $frontier$ is empty.

5.2 Heuristics

Our algorithm searches for a plan in the space of all possible decompositions of the initial task network. HTNs that have been designed specifically to be customizable by user preferences may contain tasks that could be decomposed by a fairly large number of methods. In this scenario, it is essential for the algorithm to be able to evaluate which methods to use to decompose a task in order to get to a reasonably good solution quickly. The heuristics we propose in this section are specifically designed to address this problem. All heuristics are evaluated in a search node $\langle s, w, partialP \rangle$.

Optimistic Metric Function (OM) This function is an estimate of the best metric value achievable by any plan that can result from the decomposition of the current task network w. Its value is computed by evaluating the metric function in s but assuming that (1) no further precondition preferences will be violated in the future, (2) temporally extended preference that are violated and that can be proved to be unachievable from s are regarded as false, (3) all remaining preferences are regarded as satisfied. To prove that a temporally extended preference p is unachievable from s, OM 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. Recall that an accepting state is reached when the preference formula is satisfied.

OM provides a lower bound on the best plan extending the partial plan $partialP$ assuming that the metric function is non-increasing in the number of achieved preferences. It can therefore be safely used as the METRICBOUNDFN function in our algorithm. In fact, we use OM as a bounding function in our experiments.

OM is a variant of "optimistic weight" (Bienvenu, Fritz, and McIlraith 2006; Sohrabi and McIlraith 2008).

Pessimistic Metric Function (PM) This function is the dual of OM. While OM regards anything that is not provably violated (regardless of future actions) as satisfied, PM regards anything that is not provably satisfied (regardless of future actions) as violated. Its value is computed by evaluating the metric function in s but assuming that (1) no further precondition preferences will be violated in the future, (2) temporally extended preferences that are satisfied and that can be proved to be true in any successor of s are regarded as satisfied, (3) all remaining preferences are regarded as violated. To prove that a temporally extended preference p is true in any successor of s, we check whether in the current state of the world the automaton for p would be in an accepting state that is also a sink state, i.e., from which it is not possible to escape, regardless of the actions performed in the future.

For reasonable metric functions (i.e., functions that are non increasing in the number of satisfied preferences), PM is monotonically decreasing as more actions are added to $partialP$. PM can provide good guidance because it is a measure of assured progress towards the satisfaction of the preferences.

Lookahead Pessimistic Metric Function (LA) This function is an estimate of the pessimistic metric of the *best successor* to the current node. To compute this we do a lookahead search, in which we search for all possible *relaxed* decompositions of the current task network w, up to a certain depth k. Decompositions of w are relaxed because when decomposing some of the non-primitive tasks in w we do not use their methods but the heuristic operators introduced previously in Section 4.2. By doing this, we allow the search to introduce some of the effects of t in one step, instead of having to perform several decompositions of t.

Once we have obtained all the relaxed successors of the current node up to depth k, LA evaluates the pessimistic metric (PM) in each node, and returns the lowest value.

Depth (D) We use the depth as another heuristic to guide the search. This heuristic does not take into account the preferences. Rather, it encourages the planner to find a decomposition soon. Since the search is guided by the HTN structure, guiding the search toward finding a plan using depth is natural. Other HTN planners such as **SHOP2** (Nau et al. 2003) also use depth or depth-first-search to guide the search to find a plan quickly.

The HEURISTICFN function we use in our algorithm corresponds to a *prioritized sequence* of the above heuristics, in which D is always considered first. As such, when comparing two nodes we look at their depths, returning the one that has a higher depth value. If the depths are equal, we use the other heuristics in sequence to break ties. Figure 2 outlines the three prioritized sequences we have experimented with. For example, LA-First breaks a tie between two nodes at equal depth by first comparing the LA function of those nodes, then by comparing the value of OM, and if still tied, by comparing the value of PM.

5.3 Optimality and Pruning

Since we are using inadmissible heuristics, we cannot guarantee that the plans we generate are optimal. The only way to do this is to run our algorithm until the complete HTN

Strategy	Check whether	If tied	If tied
No-LA	$OM_1 < OM_2$	$PM_1 < PM_2$	-
LA-First	$LA_1 < LA_2$	$OM_1 < OM_2$	$PM_1 < PM_2$
LA-Last	$OM_1 < OM_2$	$PM_1 < PM_2$	$LA_1 < LA_2$

Figure 2: Three strategies to determine whether a node n_1 is better than a node n_2. OM is the optimistic-metric, PM is the pessimistic-metric, and LA is the look-ahead heuristic. All three strategies use D, the depth, as their first criteria for the comparison.

search space is exhausted. In this case, the final plan returned is guaranteed to be optimal.

Exhaustively searching the search space is not reasonable in most planning domains, however here we are able to exploit properties of our planning problem to make this achievable much of the time. Specifically, most HTN specifications severely restrict the search space so that, relative to a classical planning problem, the search space is exhaustively searchable, much of the time. Further, in the case where our preference metric function is additive, our OM heuristic function enables us to soundly prune partial plans from our search space. Specifically, we say that a pruning strategy is sound if and only if whenever a node is pruned (line 8) the metric value of any plan extending this node will exceed the current bound $bestMetric$. This means that no state will be incorrectly pruned from the search space. Soundly pruning the search space of an HTN PBP problem with an additive metric function for preferences guarantees that if search terminates, the final plan returned is optimal.

6 Implementation and Evaluation

We implemented our proof-of-concept preference-based HTN planner, **HTNPLAN-P**. **HTNPLAN-P** has two modules: a preprocessor and a preference-based HTN planner. The preprocessor reads PDDL3 problems and generates a **SHOP2** planning problem with only simple (final-state) preferences. Additionally, it generates the relaxed operators for recursive tasks, and the OM, PM, and LA functions described in Section 5.2. On the other hand, the planner is a modified version of the LISP version of **SHOP2** (Nau et al. 2003) that is able to compute the heuristic functions and implements the algorithm described in Section 5.1.

We evaluated the effectiveness of our various heuristics shown in Figure 2 in obtaining plans with good quality (i.e., with low metric value). We tested the planners on the *Rovers* domain from the qualitative preferences track of the 5th International Planning Competition (IPC-5). We have used a modified version of the HTN Rovers specification used by **SHOP2** in IPC-2. Our modification ensures true non-determinism: if a task can be decomposed using two different methods, then both of these methods are considered, not just the first applicable one. In the Rovers domain, the goal is to sample scientific data such as rock and soil, or take images from different objects by navigating rovers between the different surfaces. However, a rover can only traverse between different waypoints if there is a visible path from the source to the destination. In addition, each rover is equipped for either soil, rock, image, or some combination of them.

#	**No-LA**	**LA-First**	**LA-Last**	**SGPlan$_5$**
1	58.93	47.58	47.58	68.49
2	38.11	23.33	21.78	35.00
3	34.11	23.40	23.39	39.31
4	48.85	37.99	37.99	43.43
5	362.95	266.85	155.54	317.82
6	128.13	46.18	46.18	83.41
7	88.58	79.49	50.17	85.61
8	932.00	782.00	716.00	736.00
9	981.62	538.44	516.18	981.63
10	1020.18	566.80	181.66	559.73
11	1270.98	847.28	840.06	1003.57
12	706.81	465.05	256.31	361.52
13	3899.50	4965.31	1516.46	2873.85
14	828.33	411.31	388.23	507.96
15	3466.00	2619.64	2233.52	2243.96
16	2507.00	2825.00	1449.00	1973.00
17	4228.33	-	2467.40	2443.80
18	4700.00	-	2679.00	3348.00

Figure 3: The metric value of the solutions generated by **SGPlan**$_5$ and 3 strategies for **HTNPLAN-P** on 18 Rover problems.

The problems used in our tests are a modified version of the ones used in IPC-5. After careful consideration, we realized that in order to have a fair comparison we needed to remove some of the preferences from some of the problems. This is because these problems contain preferences that will never be satisfied because of how HTN methods for this domain are defined. For example, if the goal is to only collect rock from location *waypoint3*, a preference indicates rock collection from other locations will never be met because the HTN structure will never consider sampling rock from other non-specified locations. The modified problems still have a fairly high number of preferences. For example, problems 9, 13, and 18, have 71, 129, and 133 preferences respectively.

Figure 3 shows the metric values of the last plan returned by our planner under each of our heuristics. It also shows the metric value of the plans returned by **SGPlan**$_5$, the winner of IPC-5. We ran both planners for 60 minutes, and with a limit of 2 GB per process. The dash entries indicate that no plan was found within the time and memory limit.

Under the LA-Last heuristic, our planner consistently found plans with better quality than those found by **SGPlan**$_5$. The LA-Last heuristic, which uses the lookahead heuristic as the last criterion for breaking ties seems to outperform other strategies. We conjecture that this is partly because the lookahead heuristic may guide the search away from good quality plans by ignoring some of the preconditions of the operators (i.e., by relaxing the methods). Lookahead is more effective if it's used by giving it a lower priority than the optimistic and pessimistic metrics.

Our planner is not able to solve the two most complex problems (19 and 20). The poor performance in these two instances can be explained by two main reasons. First, the current version of our planner is not optimized for speed or memory (it is a LISP prototype). Second, because we have focused our efforts on addressing the problem of finding high-quality plans, the current version of our planner does not implement any heuristics oriented to *hard goal* satisfaction, an arena in which **SGPlan$_5$** currently excels.

In many cases, **SGPlan$_5$** generates a plan significantly faster than HTNPlan-P. This is not surprising since HTNPlan-P's code has not been optimized. Optimization of HTNPlan-P is possible.

7 Discussion and Related Work

Preference-based planning has been the subject of much interest recently, spurred on by two International Planning Competition (IPC) tracks on this subject in 2006. A number of planners were developed, all based on the competition's PDDL3 language. Our work is distinguished in that it exploits *procedural* (action-centric) domain control knowledge in the form of an HTN, extending PDDL3 with action-centric constructs to achieve this objective.

There is a variety of related work. Most notable is our recent work (Sohrabi and McIlraith 2008) in which we similarly extend the $\mathcal{LPP}$ (Bienvenu, Fritz, and McIlraith 2006) *qualitative* preference language with HTN constructs. The resulting language, $\mathcal{LPH}$, is more expressive in that it allows arbitrarily nested LTL formulae, but less expressive in that it doesn't allow counting of violations as do PDDL3 precondition preferences. The associated planning algorithm is also fundamentally different. Optimal plans are found using heuristic search with an *admissible* heuristic. In contrast to the work presented here, no heuristics exploited HTN structure, nor was any precompilation performed. Preferences were evaluated for (partial) satisfaction using progression.

In a similar vein, in 2006, we introduced the problem of integrating user preferences into web service composition (Sohrabi, Prokoshyna, and McIlraith 2006). To that end, we developed a Golog-based composition engine that also exploits heuristic search and an admissible heuristic. Once again, the language used in that work was $\mathcal{LPP}$. In contrast to the work presented here and by Sohrabi and McIlraith (2008), there are no web-service or Golog-specific extensions for complex actions (the Golog analogues of tasks). This paper's HTN-tailored language and HTN-based planner are significantly different, but also provide a planner that can be exploited for web service composition.

Also related is very recent work by Lin, Kuter, and Sirin (2008) which exploits PDDL3, to perform preference-based HTN planning, applied to web service composition. Like Sohrabi, Prokoshyna, and McIlraith (2006), they do not extend their preference language with web service specific/task-specific constructs. Their approach to preference-based HTN planning is quite different from ours. In particular, they translate user preferences into HTN constraints and preprocess the preferences to check if additional tasks need to be added to w_0. Their search is best-first search but uses *ontological* reasoning over the user preferences. Their reasoning assumes that each method has a fixed set of effects or well defined post-conditions associated with it.

It is interesting and important to note that the HTN planners **SHOP2** (Nau et al. 2003) and **ENQUIRER** (Kuter et al. 2004) can be seen to handle some simple user preferences. In particular the order of methods and sorted preconditions in a domain description specifies a user preference over which method is more preferred to decompose a task. Hence users may write different versions of a domain description to specify simple preferences. However, unlike **HTNPLAN-P** the user constraints are treated as hard constraints and (partial) plans that do not meet these constraints will be pruned from the search space.

Finally, the **ASPEN** planner (Rabideau, Engelhardt, and Chien 2000) performs a simple form of preference-based planning, focused mainly on preferences over resources. It can plan with HTN-like task decomposition, but its preference language is far less expressive than ours. In contrast to **HTNPLAN-P**, **ASPEN** performs local search for a local optimum. It does not perform well when preferences are interacting, nested, or not local to a specific activity.

References

Bacchus, F., and Kabanza, F. 1998. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence* 22(1-2):5–27.

Baier, J. A., and McIlraith, S. A. 2006. Planning with first-order temporally extended goals using heuristic search. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, 788–795.

Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2007. A heuristic search approach to planning with temporally extended preferences. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, 1808–1815.

Bienvenu, M.; Fritz, C.; and McIlraith, S. A. 2006. Planning with qualitative temporal preferences. In *Proceedings of the 10th International Conference on Knowledge Representation and Reasoning (KR)*, 134–144.

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

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Hierarchical Task Network Planning. Automated Planning: Theory and Practice*. Morgan Kaufmann.

Kuter, U.; Sirin, E.; Nau, D. S.; Parsia, B.; and Hendler, J. A. 2004. Information gathering during planning for web service composition. In *Proceedings of the 3rd International Semantic Web Conference (ISWC)*, 335–349.

Lin, N.; Kuter, U.; and Sirin, E. 2008. Web service composition with user preferences. In *Proceedings of the 5th European Semantic Web Conference (ESWC)*, 629–643.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Rabideau, G.; Engelhardt, B.; and Chien, S. A. 2000. Using generic preferences to incrementally improve plan quality. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, 236–245.

Sohrabi, S., and McIlraith, S. A. 2008. On planning with preferences in HTN. In *Fourth Multidisciplinary Workshop on Advances in Preference Handling (M-Pref)*, 103–109.

Sohrabi, S.; Prokoshyna, N.; and McIlraith, S. A. 2006. Web service composition via generic procedures and customizing user preferences. In *Proceedings of the 5th International Semantic Web Conference (ISWC)*, 597–611.