HTN Planning with Preferences*

Shirin Sohrabi **Jorge A. Baier** **Sheila A. McIlraith**
Department of Computer Science
University of Toronto
{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 Planning Domain Definition Language, PDDL3, to support specification of simple and temporally extended 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 the **SHOP2** planner. We compared our planner with **SGPlan**$_5$ and **HPLAN-P**– the top performers in the 2006 International Planning Competition preference tracks. **HTNPLAN-P** generated plans that in all but a few cases equalled or exceeded the quality of plans returned by **HPLAN-P** and **SGPlan**$_5$. 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 *et al.*, 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 Planning Domain Definition Language, PDDL3 [Gerevini *et al.*, 2009], 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 highly expressive, however they are solely *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*) and preferred parameters to use when choosing a set of subtasks to realize a task (e.g., *I prefer to book a flight with United*). To compute preferred HTN plans, we propose a branch-and-bound algorithm, together with a set of heuristics that leverage HTN structure.

The main contributions of this paper are: (1) a language that supports the specification of temporally extended preferences over complex action- and state-centric properties of a plan, and (2) heuristics and an algorithm that exploit HTN procedural preferences and control to generate preferred plans that under some circumstances are guaranteed optimal. The notion of adding advice to an HTN planner regarding how to decompose a task network was first proposed by Myers (e.g., [Myers, 2000]). Recently, there was another attempt to integrate preferences into HTN planning *without* the provision of action-centric language constructs [Lin *et al.*, 2008].

*This paper will also appear in the Proceedings of IJCAI-09.

We discuss these and other related works in Section 7. PBP has been the topic of much research in recent years, and there has been a resurgence of interest in HTN planning. Experimental evaluation of our planner shows that HTN PBP generates plans that, in all but a few cases, equal or exceed the best PBP planners in plan quality. As such, it argues for HTN PBP as a viable and promising approach to PBP.

2 Background

2.1 HTN Planning

Informally, an HTN planning problem can be viewed as a generalization of the classical planning paradigm. An HTN domain contains, besides regular primitive actions, a set of *tasks* or high-level actions. Tasks can be successively refined or *decomposed* by the application of so-called *methods*. When this happens, the task is replaced by a new, intuitively more specific *task network*. In short, a task network is a set of tasks plus a set of restrictions (often ordering constraints) that its tasks should satisfy. The HTN planning problem consists of finding a primitive decomposition of a given (initial) task network.

Example 1 (Travel Example) Consider the planning problem of arranging travel in which one has to arrange accommodation and various forms of transportation. This problem can be viewed as a simple HTN planning problem, in which there is a single task, "arrange travel", which can be decomposed into arranging transportation, accommodations, and local transportation. Each of these more specific tasks can successively be decomposed based on alternative modes of transportation and accommodations, eventually reducing to primitive actions that can be executed in the world. Further constraints can be imposed to restrict decompositions.

A formal definition of HTN planning with preferences follows. Most of the basic definitions follow Ghallab *et al.* [2004].

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 which consists of a set of operators and methods.*

A domain is a pair $D = (O, M)$ where O is a set of operators and M is a set of methods. An operator is a primitive action, described by a triple *o =(name(o), pre(o), eff(o))*, corresponding to the operator's name, preconditions and effects. In our example, ignoring the parameters, operators might include: *book-train*, *book-hotel*, and *book-flight*.

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.

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. Method m is relevant for a task t if there is a substitution σ such that $\sigma(t)$ =*task*(m). Several 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. An operator o may also accomplish a ground primitive task t if their names match.

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 ∈ U contains a task t_u. If all of the tasks are primitive, then w is called primitive; otherwise it is called nonprimitive.*

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.

Definition 3 (Plan) $\pi = o_1 o_2 \dots o_k$ *is a plan for HTN planning program $\mathcal{P} = (s_0, w_0, D)$ if there is a primitive decomposition, w, of w_0 of which π is an instance.*

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 Planning Domain Definition Language (PDDL) is the de facto standard input language for many planning systems. PDDL3 [Gerevini *et al.*, 2009] extends PDDL2.2 to support the specification of preferences and hard constraints over *state* properties of a trajectory. These preferences form the building blocks for definition of a PDDL3 *metric function* that defines the quality of a plan. In this context PBP necessitates maximization (or minimization) of the metric function. In what follows, we describe those elements of PDDL3 that are most relevant to our work.

Temporally extended preferences/constraints PDDL3 specifies temporally extended preferences (TEPs) and temporally extended hard constraints in a subset of linear temporal logic (LTL). Preferences are given names in their declaration, to allow for later reference. The following PDDL3 code illustrates one preference and one hard constraint.

```
(forall (?l - light)
 (preference p-light (sometime (turn-off ?l))))
(always (forall ?x - explosive)
                    (not (holding ?x)))
```

The `p-light` preference suggests that the agent eventually turn all the lights off. 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, comprising 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.

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.

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 pkg1 depot1)
        (preference p-truck (at truck depot1))))
```

specifies both a hard goal (`pkg1` must be delivered at `depot1`) and a simple preference (that `truck` is at `depot1`). Simple preferences can also be quantified.

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.

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 (+
                     (* 40 (is-violated p-light))
                     (* 20 (is-violated p-truck))))
```

specifies that it is twice as important to satisfy preference `p-light` as to satisfy preference `p-truck`.

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

Finally, we now complete the formal definition for HTN planning with PDDL3 preferences. 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 such that $\pi_1 \preceq \pi_2$ iff $M(\pi_1) \leq M(\pi_2)$.

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 states within plan trajectories to provide preferences over non-functional properties of the planning problem including *how* some planning objective is accomplished. This is particularly useful 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.

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.

Here, we extend PDDL3 to incorporate complex action-centric preferences over HTN tasks. This gives users the ability to express preferences over certain parameterization of a task (e.g., preferring one task grounding to another) and over certain decompositions of nonprimitive tasks (i.e., prefer to apply a certain method over another). To support preferences over task occurrences (primitive and nonprimitive) and task decompositions, we added three new constructs to PDDL3: **occ**(a), **initiate**(x) and **terminate**(x), where a is a primitive task (i.e., an action), and x is either a task or a name of method. **occ**(a) states that the primitive task a occurs in the present state. On the other hand **initiate**(t) and **terminate**(t) state, respectively, that the task t is initiated or terminated in the current state. Similarly **initiate**(n) (resp. **terminate**(n)) states that the application of method named n is initiated (resp. terminated) in the current state. These new constructs can be used within simple and temporally extended preferences and constraints, but not within precondition preferences.

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

```
(preference p1
  (always (not (occ (pay MasterCard)))))
(preference p2 (sometime (occ
  (book-flight SA Eco Direct WindowSeat))))
(preference p3 (imply (close origin dest)
  (sometime (initiate (by-rail-trans)))))
(preference p4
  (sometime-after (terminate (arrange-trans))
                  (initiate (arrange-acc))))
```

The **p1** preference states that the user never pays by Mastercard. The **p2** preference states that at some point the user

[1] For simplicity many parameters have been suppressed.

books a direct economy window-seated flight with a Star Alliance (SA) carrier. The **p3** preference states that the *by-rail-trans* method is applied when origin is close to destination. Finally **p4** states that *arrange-trans* task is terminated before the *arrange-acc* task begins (for example: finish arranging your transportation before booking a hotel).

Semantics: The semantics of the preference language comprises two parts: (1) a formal definition of the satisfaction of individual preference formulae, and (2) a formal definition of the aggregation of preferences through an objective function. The satisfaction of individual preference formulae is defined by mapping HTN decompositions and LTL formulae into the situation calculus [Reiter, 2001]. In so doing, satisfaction of a preference formula is reduced to entailment of the formula in a logical theory. A sketch of the situation calculus encoding is found in Appendix A. Preference formulae are composed into a metric function. The semantics of the metric function, including the aggregation of quantified preferences via the `is-violated` function, is defined in the same way as in PDDL3, following Gerevini *et al.* [2009].

4 Preprocessing HTN problems

Before searching for a most preferred plan, we preprocess the original problem. This is needed in order to make the planning problem more easily manageable by standard planning techniques. We accomplish this objective by removing all of the modal operators appearing in the preferences. The resulting domain, has only final-state preferences, and all preferences refer to state properties.

By converting TEPs into final-state preferences, our heuristic functions are only defined in terms of domain predicates, rather than being based on non-standard evaluations of an LTL formula, such as the ones used by other approaches [e.g. Bienvenu *et al.*, 2006]. Nor do we need to implement specialized algorithms to reason about LTL formulae such as the progression algorithm used by **TLPLAN** [Bacchus and Kabanza, 1998].

Further, by removing the modal operators **occ**, **initiate**, and **terminate** we provide a way to refer to these operators via state predicates. This allows us to use standard HTN planning software as modules of our planner, without needing special modifications such as a mechanism to keep track of the tasks that have been decomposed or the methods that have been applied.

Preprocessing Tasks and Methods Our 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 are two new predicates: $executing$-t and $terminated$-t. If $a_0a_1 \cdots a_n$ 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, and $terminated$-t is true in all states after 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., $n = name(m)$) that occurs in some preference. We use two predicates $executing$-n and $terminated$-n, whose updates are realized analogously to their task versions described above.

Preprocessing the Modal Operators We replace each occurrence of **occ**(t), **initiate**(t), and **terminate**(t) by occ-t when t is primitive. We replace the occurrence of **initiate**(t) by $executing$-t, and **terminate**(t) by $terminated$-t when t is non-primitive. Occurrences of **initiate**(n) are replaced by $executing$-n, and **terminate**(n) by $terminated$-n.

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

Temporally Extended and Precondition Preferences We use an existing compilation technique [Baier *et al.*, 2009] 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. Quantified preferences are compiled into parametric automata for efficiency. Finally, precondition preferences, preferences that should ideally hold in the state in which the action is performed, are compiled away as conditional action costs, as is done in the **HPLAN-P** planner. For more details refer to the original paper [Baier *et al.*, 2009].

5 Preference-based Planning with HTNs

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 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 a planning problem (s_0, w_0, D), a metric function METRICFN, and a heuristic function HEURISTICFN.

```
1:  function HTNPBP(s_0, w_0, D, METRICFN, HEURISTICFN)
2:    frontier ← ⟨s_0, w_0, ∅⟩                          ▷ initialize frontier
3:    bestMetric ← worst case upper bound
4:    while frontier is not empty do
5:      current ← Extract best element from frontier
6:      ⟨s, w, partialP⟩ ← current
7:      lbound ← METRICBOUNDFN(s)
8:      if lbound < bestMetric then                     ▷ pruning by bounding
9:        if w = ∅ and current's metric < bestMetric then
10:         Output plan partialP
11:         bestMetric ← METRICFN(s)
12:       succ ← successors of current
13:       frontier ← merge succ into frontier
```

Figure 1: A sketch of our HTN PBP algorithm.

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 w – $current$'s task network – 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.

Finally, all successors to $current$ are computed using the Partial-order Forward Decomposition procedure (PFD) [Ghallab *et al.*, 2004], and 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-decreasing in the number of violated preferences. This is the function used as METRICBOUNDFN in our planner. OM is a variant of "optimistic weight" [Bienvenu *et al.*, 2006].

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 (e.g., those non-decreasing in the number of violated preferences), PM is monotonically decreasing as more actions are added to $partialP$. PM provides good guidance because it is a measure of assured progress towards the satisfaction of the preferences.

Lookahead Metric Function (LA) This function is an estimate of the metric of the *best successor* to the current node. It is computed by conducting a two-phase search. In the first phase, a search for all possible decompositions of w is performed, up to a certain depth k. In the second phase, for each of the resulting nodes, a single primitive decomposition is computed, using depth-first search. The result of LA is the best metric value among all the fully decomposed nodes. Intuitively, LA estimates the metric value of a node by first performing an exhaustive search for decompositions of the current node, and then by approximating the metric value of the resulting nodes by the metric value of the the first primitive decomposition that can be found, a form of sampling of the remainder of the search space.

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** also use depth or depth-first search to guide the search to find a plan quickly.

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

Figure 2: 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.

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 sequences we have used in our experiments.

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 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 some 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. 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.

Proposition 1 *The OM function provides sounds pruning if the metric function is non-decreasing in the number of satisfied preferences, non-decreasing in plan length, and independent of other state properties.*

A metric is non-decreasing in plan length if one cannot make a plan better by increasing its length only (without satisfying additional preferences).

Theorem 1 *If the algorithm performs sound pruning, then the last plan returned, if any, is optimal.*

Proof sketch: Follows the proof of optimality for the **HPLAN-P** planner [Baier *et al.*, 2009].

6 Implementation and Evaluation

Our implemented HTN PBP planner, **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. The planner itself is a modification of the LISP version of **SHOP2** [Nau *et al.*, 2003] that implements the algorithm and heuristics described above.

We had three objectives in performing our experimental evaluation: to evaluate the relative effectiveness of our heuristics, to compare our planner with state-of-the-art PBP planners, and to compare our planner with other HTN PBP planners. Unfortunately, we were unable to achieve our third objective, since we could not obtain a copy of **SCUP**, the only HTN PBP planner we know of [Lin *et al.*, 2008]. (See Section 7 for a qualitative comparison.)

		HTNPLAN-P				SGPlan$_5$		HPLAN-P	
		No-LA		*LA*					
	#Prb	#S	#Best	#S	#Best	#S	#Best	#S	#Best
travel	41	41	3	41	**37**	41	1	41	17
rovers	20	20	4	20	**19**	20	1	11	2
trucks	20	20	6	20	**15**	20	11	4	2

Figure 3: Comparison between two configurations of **HTNPLAN-P**, **HPLAN-P**, and **SGPlan$_5$** on **rovers**, **trucks**, and **travel** domains. Entries show number of problems in each domain (#Prb), number of solved instances in each domain (#S) by each planner, and number of times each planner found a plan of equal or better quality to those found by all other planners (#Best). All planners were ran for 60 minutes, and with a limit of 2GB per process.

We used three domains for the evaluation: the **rovers** domain, the **trucks** domain, both standard IPC benchmark domains; and the **travel** domain, which is a domain of our own making. Both the **rovers** and **trucks** domains comprised the preferences from IPC-5. In **rovers** domain we used the HTN designed by the developers of **SHOP2** for IPC-2 and in **trucks** we created our own HTN. We modified the HTN in **rovers** very slightly to reflect the true nondeterminism in our **HTNPLAN-P** planner: i.e., if a task could be decomposed using two different methods, then both methods would be considered, not just the first applicable one. We also modified the IPC-5 preferences slightly to ensure fair comparison between planners. The **rovers** and **trucks** problems sets comprised 20 problems. The number of preferences in these problem sets ranged in size, with several having over 100 preferences per problem instance.

The **travel** domain is a PDDL3 formulation of the domain introduced in Example 1. Its problem set was designed in order to evaluate the PBP approaches based on two dimensions: (1) scalability, which we achieved by increasing the branching factor and grounding options of the domain, and (2) the complexity of the preferences, which we achieved by injecting inconsistencies (i.e., conflicts) among the preferences. In particular, we created 41 problems with preferences generated automatically with increasing complexity. For example problem 3 has 27 preferences with 8 conflicts in the choice of transportation while problem 40 has 134 preferences with 54 conflicts in the choice of transportation.

Our experiments evaluated the performance of four planners: **HTNPLAN-P** with the *No-LA* heuristic, and **HTNPLAN-P** with the *LA* heuristic, **SGPlan$_5$** [Hsu *et al.*, 2007], and **HPLAN-P**– the latter two being the top PBP performers at IPC-5. Results are summarized in Figure 3, and show that **HTNPLAN-P** generated plans that in all but a few cases equalled or exceeded the quality of plans returned by **HPLAN-P** and **SGPlan$_5$**. The results also show that **HTNPLAN-P** performs better on the three domains with the *LA* heuristic.

Conducting the search in a series of episodes does help in finding better-quality plans. To evaluate this, we calculated the *percent metric improvement* (PMI), i.e., the percent difference between the metric of the first and the last plan returned by our planner (relative to the first plan). The average PMI is

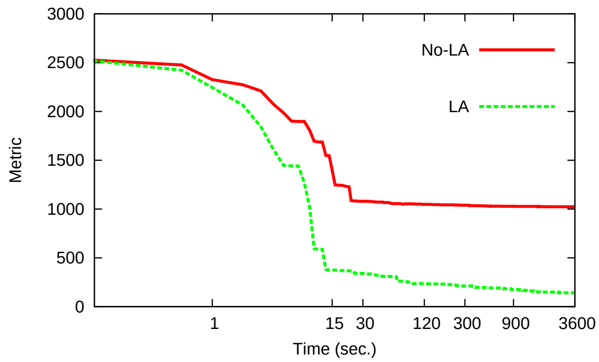

Figure 4: Added metric vs. time for the two strategies in the trucks domain. Recall that a low metric value means higher quality plan. When a problem is not solved at time t, we add its worst possible metric value (i.e. we assume no satisfied preferences).

40% in **rovers**, 72% in **trucks**, and 8% in **travel**.

To compare the relative performance between *LA* and *No-LA*, we averaged the percent metric difference (relative to the worst plan) in problems in which the configurations found a different plan. This difference is 45% in **rovers**, 60% in **trucks**, and 3% in **travel**, all in favour of *LA*.

We created 18 instances of the **travel** domain where we tested the performance between *LA* and *No-LA* on problems that have preferences that use our HTN extension of PDDL3. The average PMI for these problems is 13%, and the relative performance between the two is 5%.

Finally Figure 4 shows the decrease of the sum of the metric value of all instances of the trucks domain relative to solving time. We observe a rapid improvement during the first seconds of search, followed by a marginal one after 900 seconds. Other domains exhibit similar behaviour.

7 Discussion and Related Work

PBP has been the subject of much interest recently, spurred on by three IPC-5 tracks on this subject. A number of planners were developed, all based on the competition's PDDL3 language. Our work is distinguished in that it employs HTN domain control extending PDDL3 with HTN-inspired constructs. The planner itself then employs heuristics and algorithms that exploit HTN-specific preferences and control. Experimental evaluation of our planner shows that **HTNPLAN-P** generates plans that, in all but a few cases, equal or exceed the best PBP in plan quality. As such, it argues generally for HTN PBP as a viable and promising approach to PBP.

With respect to advisable HTN planners, Myers was the first to advocate augmenting HTN planning with hard constraints to capture advice on *how to* decompose HTNs, extending the approach to conflicing advice in [Myers, 2000]. Their work is similar in vision and spirit to our work, but different with respect to the realization. In their work, preferences are limited to consistent combinations of HTN advise; they do no include the rich temporally extended state-centric preferences found in PDDL3, nor do they support the weighted combination of preferences into a metric function that defines plan quality. With respect to computing HTN PBP, Myers' algorithm does not exploit lookahead heuristics or sound pruning techniques.

The most notable related work is that of Lin *et al.* [2008] who developed a prototype HTN PBP planner, **SCUP**, tailored to the task of web service composition. Unfortunately, **SCUP** is not available for experimental comparison, however there are fundamental differences between the planners, that limit the value of such a comparison. Most notably, Lin *et al.* [2008] do not extend PDDL3 with HTN-specific preference constructs, a hallmark of our work. Further, their planning algorithm appears to be unable to handle conflicting user preferences since they note that such conflict detection is performed manually prior to invocation of their planner. Optimization of conflicting preferences is common in most PBP's, including ours. Also, their approach to HTN PBP 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. This is well motivated by the task of web service composition, but not a practice found in classical HTN planning.

Also related is the **ASPEN** planner [Rabideau *et al.*, 2000], which 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.

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, observe that we approached HTN PBP by integrating PBP into HTN planning. An alternative approach would be to integrate HTN into PBP. Kambhampati *et al.* [1998] hints at how this might be done by integrating HTN into their plan repair planning paradigm. For the integration of HTN into PBP to be effective, heuristics would have to be developed that exploited the special compiled HTN structure. Further, such a compilation would not so easily lend itself to mixed-initiative PBP, a topic for future investigation.

Acknowledgements: We thank our colleague Christian Fritz for helpful discussion. We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Ontario Ministry of Innovations Early Researcher Award (ERA).

References

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

[Baier *et al.*, 2009] J. A. Baier, F. Bacchus, and S. A. McIlraith. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence*, 173(5-6):593–618, 2009.

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

[Gabaldon, 2002] A. Gabaldon. Programming hierarchical task networks in the situation calculus. In *AIPS'02 Workshop on On-line Planning and Scheduling*, April 2002.

[Gabaldon, 2004] A. Gabaldon. Precondition control and the progression algorithm. In *Proc. of the 9th Int'l Conference on Knowledge Representation and Reasoning (KR)*, 634–643. AAAI Press, 2004.

[Gerevini *et al.*, 2009] A. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6):619–668, 2009.

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

[Hsu *et al.*, 2007] C.-W. Hsu, B. Wah, R. Huang, and Y. Chen. Constraint partitioning for solving planning problems with trajectory constraints and goal preferences. In *Proc. of the 20th Int'l Joint Conference on Artificial Intelligence (IJCAI)*, 1924–1929, 2007.

[Kambhampati *et al.*, 1998] S. Kambhampati, A. D. Mali, and B. Srivastava. Hybrid planning for partially hierarchical domains. In *Proc. of the 15th National Conference on Artificial Intelligence (AAAI)*, 882–888, 1998.

[Kuter *et al.*, 2004] U. Kuter, E. Sirin, D. S. Nau, B. Parsia, and J. A. Hendler. Information gathering during planning for web service composition. In *Proc. of the 3rd Int'l Semantic Web Conference (ISWC)*, 335–349, 2004.

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

[Myers, 2000] K. L. Myers. Planning with conflicting advice. In *Proc. of the 5th Int'l Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, 355–362, 2000.

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

[Rabideau *et al.*, 2000] G. Rabideau, B. Engelhardt, and S. A. Chien. Using generic preferences to incrementally improve plan quality. In *Proc. of the 5th Int'l Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, 236–245, 2000.

[Reiter, 2001] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, Cambridge, MA, 2001.

[Sohrabi and McIlraith, 2008] S. Sohrabi and S. A. McIlraith. On planning with preferences in HTN. In *Proc. of the 12th Int'l Workshop on Non-Monotonic Reasoning (NMR)*, 241–248, 2008.

A Sketch of the Semantics

The satisfaction of all constraint and preference formulae is defined by a translation of formulae into the Situation Calculus (SC), a logical language for reasoning about action and change [Reiter, 2001] Formulae are satified if their translations are entailed by the SC logical theory representing the HTN planning problem and plan. The translation of our HTN constructs are more complex, so we begin with the original elements of PDDL3.

In the SC, primitive actions a are instantaneous. A situation s is a *history* of primitive actions performed at a distinguished initial situation S_0. The logical function $do(a, s)$ returns the situation that corresponds to performing action a in s. In the SC, the *state* of the world is expressed in terms of functions and relations (fluents) relativized to a particular situation s, e.g., $F(\vec{x}, s)$.

The translation to SC proceeds as follows. Since we are operating over finite domains, all universally quantified PDDL3 formulae are translated into individual grounded instances of the formulae. Simple preferences (resp. constraints) are translated into corresponding SC formulae. Temporally extended preferences (resp. constraints) are translated into SC formulae following the translation of LTL formulae into SC by Gabaldon [2004] and Bienvenu *et al.* [2006].

To define the semantics of our HTN extension, we appeal to a translation of HTN planning into SC entailment of a ConGolog program that is again credited to Gabaldon [2002]. ConGolog is a logic programming language built on top of the SC that supports the expression of complex actions. In short, the translation defines a way to construct a logical theory and formula $\Psi(s)$ such that $\Psi(s)$ is entailed by the logical theory iff the sequence of actions encoded by s is a solution to the original HTN planning problem.

More specifically, the initial HTN state s_0 is encoded as the initial situation, S_0. The HTN domain description maps to a corresponding SC domain description, $\mathcal{D}$, where for every operator o there is a corresponding primitive action a, such that the preconditions and the effects of o are axiomatized in $\mathcal{D}$. Every method and nonprimitive task together with constraints is encoded as a ConGolog procedure. $\mathcal{R}$ is the set of procedures in the ConGolog domain theory.

In addition to this translation, we need to deal with the new elements of PDDL3 that we introduced: **occ**(a), **initiate**(X), and **terminate**(X). To this end, following Gabaldon's translation we add two new primitive actions *start*$(P(\vec{v}))$, *end*$(P(\vec{v}))$, to each procedure P that cornrensponds to an HTN task or method. In addition, we add the fluents *executing*$(P(\vec{v}), s)$ and *terminated*(X, s), where $P(\vec{v})$ is a ConGolog procedure and X is either $P(\vec{v})$ or a primitive action a. *executing*$(P(\vec{v}), s)$ states that $P(\vec{v})$ is executing in situation s, *terminated*(X, s) states that X has terminated in s. *executing*(a, s) where a is a primitive action, is defined to be false.

occ(a), **initiate**(X), and **terminate**(X) are translated into the situation calculus by building SC formulae that are evaluated when they appear in a preference formula. Below we define these formulae, using a notation compatible with Gabaldon's translation, in which $\varphi[s', s]$ denotes that the (temporal) expression φ holds over the situation fragment s, that starts in situation s'.

occ(a) tells us the first action executed is a:

$$\textbf{occ}(a)[s', s] = do(a, s') \sqsubseteq s$$

initiate(X) and **terminate**(X) are interpreted as follows:

$$\textbf{initiate}(X)[s', s] = \begin{cases} do(X, s') \sqsubseteq s & \text{if X} \in \mathcal{A} \\ do(start(X), s') \sqsubseteq s & \text{if X} \in \mathcal{R} \end{cases}$$

$$\textbf{terminate}(X)[s', s] = \begin{cases} do(X, s') \sqsubseteq s & \text{if X} \in \mathcal{A} \\ do(end(X), s') \sqsubseteq s & \text{if X} \in \mathcal{R}, \end{cases}$$

where $s' \sqsubseteq s$ denotes that situation s' is a predecessor of situation s, and $\mathcal{A}$ is a set containing all primitive actions.

Space precludes a full exposition of the translation. Details provided here and in Section 3, together with the details in Gabaldon [2002] and Bienvenu *et al.* [2006] provide all the pieces.