Applying Anytime Heuristic Search to Cost-Optimal HTN Planning

Alexandre Menif[3], Christophe Guettier[1], Éric Jacopin[2], and Tristan Cazenave[3]

[1] Safran Electronics & Defense, 100 Avenue de Paris, 91300 Massy Cedex, France
[2] MACCLIA, CREC Saint Cyr, Écoles de Coëtquidan, F-56381 GUER Cedex, France
[3] LAMSADE, Université Paris-Dauphine, 75016, Paris, France

Abstract. This paper presents a framework for cost-optimal HTN planning. The framework includes an optimal algorithm combining a branch-and-bound with a heuristic search, which can also be used as a near-optimal algorithm given a time limit. It also includes different heuristics based on weighted cost estimations and different decomposition strategies. The different elements from this framework are empirically evaluated on three planning domains, one of which is modeling a *First-Person Shooter* game.
The empirical results establish the superiority on some domains of a decomposition strategy that prioritizes the most abstract tasks. They also highlight that the best heuristic formulation for the three domains is computed from linear combinations of optimistic and pessimistic cost estimations.

1 Introduction

We study cost-optimal planning in the context of Hierarchical Task Network (HTN) planning [6]. An HTN is a specification of a planning problem based on the encoding of compound tasks and decomposition methods. A legal solution is a plan made of applicable actions that satisfies the constraints enforced by the HTN structure. In cost-optimal HTN planning, we are interested in those solution plans with minimum action costs. We aim at applying HTN planning for the animation of virtual agents in simulated environments, and these agents could express more consistent behaviors when optimizing their plans.

Several works have already investigated the issue of cost-optimal search for HTN planning. SHOP2 [13], a state-based HTN planner, includes a branch-and-bound mechanism to improve the quality of the best solution found so far. This approach provides an *anytime* mechanism for optimizing plans, which is a valuable feature for computer game agents [9]. However, the search space is still explored in depth-first, and guidance relies solely on the ordering of the decomposition methods in the domain description. Other extensions of SHOP2 [3, 17] have tackled the problem of finding a more preferred plan. The planner implements a best-first search, but the preference criterion is not based on the action costs. Another approach related to HTN planning, Angelic Hierarchical Planning (AHP), is also subject to cost-optimal planning [10]. A search procedure called AHA* (Angelic Hierarchical A*) is implemented as an A*-based algorithm and uses user-defined estimations of the cost of high-level actions. Optimistic estimations provide an admissible heuristic to guide the search, while pessimistic estimations are used as a tie breaking rule or to prune dominated high level plans.

The approach described in this article follows the idea of defining cost estimations for compound tasks introduced by AHP. The first contribution is a framework that includes these estimations into a more general form of HTN planning, and proposes new types of non-admissible weighted heuristics [15] based on these estimations. The second contribution is the description of a best-first search algorithm adapted to HTN planning, which combines a branch-and-bound with a best-first search algorithm guided by non admissible heuristics. The third contribution is an empirical study of optimal search with different weighted heuristics and task decomposition strategies, and the evaluation of their performances on different planning domains.

This article is organized as follows. Section 2 presents a general formalism for HTN planning with action costs. Section 3 provides the implementation of the search algorithm, and introduces the definitions and properties of the cost based heuristics and task decomposition strategies. Section 4 presents the different planning domains used in our experiments the results obtained on these domains for different cost heuristics and selection strategies for task decomposition, as well as a discussions on those results. Finally, Section 5 concludes on the application of heuristics based on weighted cost estimations for cost-optimal HTN planning.

2 General HTN Formalism

This section presents a basic and general formalism for HTN planning with action costs. This formalism is designed to be general enough to cover most HTN formalisms, including SHOP-like planners (referred to as "Simple Task Network Planning" in [7]) or the more general HTN framework developed with UMCP [5]. For this purpose, it is strongly inspired by the formalism proposed by [1]. This formalism abstracts away the description of states as well as the nature of logical constraints carried by task networks and decomposition methods, and focuses on the network decomposition process common to all HTN systems.

2.1 HTN Domain

For the sake of generality, a task network is simply considered here as a directed acyclic graph, with nodes labeled with task symbols:

Definition 1 (Task network). *A task network ν over a set of tasks T is a pair $(N, \prec)$ such that:*

- *N is a set of nodes, and each node $n \in N$ is associated with a task $t_n \in T$.*
- *$\prec$ is a partial order on the nodes of N.*

An HTN planning domain includes a state transition system for actions (also called primitive tasks), a set of compound tasks and decomposition methods, as well as a state-independent cost function for actions:

Definition 2 (Domain). *An HTN planning domain is a tuple $\mathcal{D} = (S, A, C, M, \mathcal{T}, \mathcal{C})$ such that:*

- *S is a set of states.*
- *A is a set of actions.*
- *C is a set of compound tasks such that $A \cap C = \emptyset$.*
- *M is a set of decomposition methods. A decomposition method is a triple $m = (c_m, N_m, \prec_m)$ such that $c_m \in C$ and $(N_m, \prec_m)$ forms a task network.*
- *$\mathcal{T} \in S \cdot A \to S$ is a state transition function.*
- *$\mathcal{C} \in A \to \mathbb{R}^+$ is a cost function.*

When a task network does not contain compound tasks, it is said to be *primitive*. Considering a primitive task network $\nu = (N, \prec)$, it is possible to linearize this task network into a sequence of actions (a plan) $a_1...a_{|N|}$, such that the linear order is consistent with $\prec$. Then, a primitive task network is said to be *executable* on some state if it can be linearized into a plan executable in this state:

Definition 3 (Executability). *A primitive task network $\nu = (N, \prec)$ is said to be executable if and only if there exists a linearization $a_1...a_{|N|}$ of ν such that:*

$$\forall i \in [\![1, |N|]\!], a_i \textit{ is applicable to } s_{i-1} \textit{ and } s_i = \mathcal{T}(s_{i-1}, a_i).$$

A task network that contains at least one node associated to a compound task can be *refined* into a new task network. To obtain this new task network, a compound task is selected and replaced by the task network defined by a relevant decomposition method:

Definition 4 (Decomposition). *Let $\nu_1 = (N_1, \prec_1)$ be a task network with a node $n \in N_1$ such that $t_n \in C$, and let $m = (c_m, N_m, \prec_m)$ be a decomposition method such that $c_m = t_n$. The decomposition of node n in ν_1, according to method m, produces a task network $\nu_2 = (N_2, \prec_2)$ such that:*

$$\begin{aligned}
&N_2 = (N_1 \setminus \{n\}) \cup N_m \\
&\prec_2 = (\prec_1 \setminus \{(n_1, n_2) \in \prec_1 \mid n_1 = n \vee n_2 = n\}) \cup \prec_m \cup \\
&\quad \{(n_1, n_2) \mid n_1 \prec_1 n, n_2 \in N_m\} \cup \\
&\quad \{(n_1, n_2) \mid n \prec_1 n_2, n_1 \in N_m\}
\end{aligned}$$

Finally, we consider the *primitive cost* of a task network. This cost is the value that an optimal solution has to minimize. For a given task network, it is defined as the sum of its action costs:

Definition 5 (Primitive cost). *Let $\nu = (N, \prec)$ be a task network. The primitive cost of ν, denoted $g(\nu)$, is defined as:*

$$g(\nu) = \sum_{n \in N : t_n \in A} \mathcal{C}(t_n)$$

2.2 HTN Problem and Solution

The definition of an HTN problem includes an HTN domain, a state representing the initial configuration of the world, and a network of tasks to be achieved:

Definition 6 (Problem). *An HTN planning problem is a triple $\mathcal{P} = (\mathcal{D}, s_0, \nu_0)$ such that:*

- *$\mathcal{D}$ is an HTN planning domain.*
- *$s_0 \in S$ is the initial state.*
- *ν_0 is the initial task network.*

The nature of an HTN solution differs from that of classical planning. In HTN planning, a solution is not any plan reaching a goal state, but is a primitive task network that fits into the specification described by the set of decomposition methods:

Definition 7 (Solution to an HTN problem). *A primitive task network ν is a solution to an HTN planning problem $\mathcal{P}$ if and only if it is executable in s_0 and can be obtained from a finite succession of refinements from ν_0. A solution ν^* is said to be optimal if and only if:*

$$g(\nu^*) = min\{g(\nu) \mid \nu \textit{ is a solution of } \mathcal{P}\}$$

3 Search

This section describes the search mechanism of our HTN framework. It introduces the algorithm, discusses its properties, and finally focuses on the two features available to control the search: cost-based heuristics, and task decomposition strategies.

3.1 Algorithm

The search procedure is described in Algorithm 1. The implementation is inspired by *Anytime A** [8]. In this algorithm, the optimal solution is not expected to be the first solution returned. A non-admissible heuristic guides the search, and a branch-and-bound step uses the cost of the best solution found so far as an upper bound to prune the search space.

Algorithm 1 uses three elements to control the search: an admissible heuristic h_1 to prune task networks, a non-admissible heuristic h_2 to guide the search, and a decomposition strategy. The decomposition strategy chooses the task to decompose when refining a task network (in the step at line 15 of the algorithm description).

Because of the existence of recursive tasks, some task networks can be infinitely expanded, and thus termination cannot be guaranteed for any HTN domains. The search space of Algorithm 1 corresponds with the *decomposition problem space* defined by [1]. For this space, they have proved that the search space is finite if and only if the initial task networks is "$\leq_1 -stratifiable$" (we refer the reader to [1] for more insight on this property). With a finite search space, proof of termination for Algorithm 1 is straightforward, as the $Closed$ set prevents the search from looping on the same task network. However, the "$\leq_1 -stratifiable$" property is quite restrictive, and none of our planning

Algorithm 1 Anytime Optimal HTN Algorithm

```
Input: A planning problem $\mathcal{P} = (\mathcal{D}, s_0, \nu_0)$
Output: Best solution found.
 1: $f_1(\nu_0) \leftarrow g(\nu_0) + h_1(\nu_0)$
 2: $f_2(\nu_0) \leftarrow g(\nu_0) + h_2(\nu_0)$
 3: $Open \leftarrow \{\nu_0\}$
 4: $Closed \leftarrow \emptyset$
 5: $solution, cost \leftarrow \emptyset, +\infty$
 6: while $Open \neq \emptyset$ and not interrupted do
 7:     $\nu \leftarrow$ select $\nu$ in $Open$ with lowest $f_2(\nu)$
 8:     $Open \leftarrow Open \setminus \{\nu\}$
 9:     $Closed \leftarrow Closed \cup \{\nu\}$
10:     if $\nu$ is primitive then
11:         if $\nu$ is executable on $s_0$ then
12:             $solution, cost \leftarrow \nu, g(\nu)$
13:         end if
14:     else
15:         $n \leftarrow$ select a node associated to a compound task in $\nu$
16:         for all $m \in M$ such that $c_m = t_n$ do
17:             $\nu' \leftarrow$ the decomposition of $\nu$ by $m$ in $n$
18:             $f_1(\nu') \leftarrow g(\nu') + h_1(\nu')$
19:             $f_2(\nu') \leftarrow g(\nu') + h_2(\nu')$
20:             if $f_1(\nu') < cost$ and $\nu' \notin Closed$ then
21:                 $Open \leftarrow Open \cup \{\nu'\}$
22:             end if
23:         end for
24:     end if
25: end while
26: return $solution$
```

domains could satisfy it. In this situation, our implementation of this algorithm does not include the $Closed$ set, as it requires an extra memory usage without guaranteeing termination anyway. How we eventually deal with recursive tasks to ensure the algorithm termination is explained later in this article. The proof for completeness and correctness is also straightforward : the branch-and-bound only prunes task networks when at least one solution has been found, and the algorithm only returns primitive and executable task networks obtained from a succession of refinements from the initial task network, which is exactly the definition of a solution. When Algorithm 1 runs, it returns a succession of solutions with improving costs, and if it is not interrupted, eventually returns an optimal solution. The proof for this last statement can be found in [8].

3.2 Heuristics

So far we can only compute the primitive cost of a task network. In order to obtain an accurate evaluation of a task network, it is also necessary to estimate the cost of the non-primitive tasks. To proceed, one can simply assign some costs on compound tasks. In the general case, these costs cannot be exact: indeed, compound tasks can eventually be decomposed in different action sequences, and each sequence can have a different total cost. A possible estimation can underestimate the final cost of any primitive decomposition obtained from a compound task. Following AHP, we refer to this estimation as an *optimistic* cost.

Definition 8 (Optimistic cost). *A cost assignment function $\mathcal{C}_{opt} : C \to \mathbb{R}^+$ is optimistic if and only if for all t $\in C$ and for each task network ν such that ν is a primitive decomposition of t:*

$$\mathcal{C}_{opt}(t) \leq g(\nu)$$

We also extend this optimistic cost to primitive tasks, where it is simply equal to the primitive cost : $\forall a \in A, \mathcal{C}_{opt}(a) = \mathcal{C}(a)$.

For a task network $\nu = (N, \prec)$ and with an optimistic cost assignment function, the sum of all compound task costs in ν underestimates the primitive cost of any primitive refinement of ν. Therefore, the admissible heuristic h_1 can be computed according to this optimistic cost function:

$$h_1(\nu) = \sum_{n \in N : t_n \in C} \mathcal{C}_{opt}(t_n)$$

The second heuristic h_2 could also be computed in this way (this is actually the heuristic used by AHA*). However, using an admissible heuristic to guide the search is counterproductive in the context of Algorithm 1: in this case, the first solution returned by the algorithm is optimal, the branch-and-bound mechanism is not used to reduce the search space, and we make no advantage of the anytime feature.

A usual way to make this heuristic not admissible is to apply a weight $\omega > 1$ on h_1: $h_2 = \omega \cdot h_1$. However, this simple weight has a drawback in HTN planning: as long as expanded task networks do not contain primitive tasks, their primitive cost is equal to 0 and the weight does not modify the expansion order. To resolve this issue, we can rather use a dynamic weight [16]. A dynamic weight is a decreasing function that converges to 1 when the depth of the search increases. To achieve a similar effect in HTN planning, one can assign a different weight ω_c to each compound task c, with higher weights at the top of the task hierarchy than at lower levels. Therefore, we can now define h_2 as:

$$h_2(\nu) = \sum_{n \in N : t_n \in C} \omega_{t_n} \cdot \mathcal{C}_{opt}(t_n)$$

A cost assignment function can also be *pessimistic*. As such, it always overestimates the final cost of a compound task:

Definition 9 (Pessimistic cost). *A cost assignment function $\mathcal{C}_{pes} : C \to \mathbb{R}^+$ is pessimistic if and only if for all t $\in C$ and for all task networks ν such that ν is a primitive decomposition of t:*

$$\mathcal{C}_{pes}(t) \geq g(\nu)$$

Again, we extend this pessimistic cost to primitive tasks, where it is equal to the primitive cost : $\forall a \in A, \mathcal{C}_{pes}(a) = \mathcal{C}(a)$. We will explain how to deal with potential infinite pessimistic costs latter in this paper.

A heuristic computed from a pessimistic cost function could not be admissible, but could still be used to guide the search. With such a heuristic, the search would be more "depth-first-like". Actually the search will exactly be a depth-first search, if for every compound task, its pessimistic cost strictly overestimates the cost of all its refinements.

Definition 10 (strictly pessimistic monotonic cost). *A cost assignment function $\mathcal{C}_{pes} : C \to \mathbb{R}^+$ is strictly pessimistic and monotonic if and only if for all $c \in C$ and for all $m \in M$ such that $c_m = c$:*

$$\mathcal{C}_{pes}(c) > \sum_{n \in N_m} \mathcal{C}_{pes}(t_n)$$

Proposition 1. *Let $\mathcal{C}_{pes}$ be a strictly pessimistic and monotonic cost assignment function. If for all task networks ν, $h_2(\nu) = \sum_{n \in N : t_n \in C} \mathcal{C}_{pes}(t_n)$, then Algorithm 1 performs a depth-first search.*

Proof. *Let ν be the task network with the lowest f_2 value in the $Open$ list. Any refinement of ν has a strictly lower f_2 value than ν, so it also has a strictly lower value than any other task network in the $Open$ list. Therefore the next selected task network from the $Open$ list is a refinement of ν.*

This last observation motivates a new definition for h_2 as a weighted sum of an optimistic cost and a strictly pessimistic and monotonic cost with a family of weights (λ_c) such that $\forall c \in C, \lambda_c \in [0, 1]$. The idea is to set a cursor between a pure admissible search ($\lambda = 1$), and a pure depth-first search ($\lambda = 0$):

$$h_2(\nu) = \sum_{n \in N : t_n \in C} (1 - \lambda_{t_n}) \cdot \mathcal{C}_{opt}(t_n) + \lambda_{t_n} \cdot \mathcal{C}_{pes}(t_n)$$

At this point, the reader might be interested in knowing how we obtain the optimistic and strictly pessimistic costs required to formulate the different heuristics we have introduced. The general idea behind the procedure we use is presented by algorithm 2. For all compound tasks, the sum of the optimistic and pessimistic costs are first computed for every associated decomposition method. Then, the optimistic cost is set to be the minimum optimistic sum, while the pessimistic estimation is set according to the maximum pessimistic sum. This procedure ensures that both optimistic and pessimistic costs are monotonic. To ensure that the pessimistic cost is strictly monotonic, a negligible value ϵ (for instance $\epsilon = 0.001$) is added to the maximal pessimistic sum. This step is repeated until a fix point is reached. This simple procedure helps to capture the general idea about how these costs are computed, but it has at least two flaws that do not make it appropriate for a direct use.

First, this procedure runs on the whole set of tasks. This is a problem because in a planning domain description, actions and compound tasks are specified through parameterized schemas. These schemas can be instantiated by substituting their parameters with all the constants associated with each object from the domain of discourse. As this collection of objects is specified specifically for each problem instance, every instance generates a different set of instantiated tasks, and the procedure should be executed before every search. When planning is used for the animation of a character in a real-time simulated environment, this procedure would consume a share of the limited amount of time available for planning. In fact, we would rather use a procedure based on a static analysis of a lifted planning domain and save the results in order to reuse it at every planning request. This can be made possible with a few adjustments when action costs are specified in the planning domains with ground numerical values. In this case, ground tasks in algorithm 2 can simply be replaced by their lifted versions, and this algorithm will return ground numerical cost estimations for every task schema.

The second flaw is more serious. Algorithm 2 theoretically works for computing optimistic costs, as their values can only decrease and are bounded by 0. This property guarantees the occurrence of a fixed point for optimistic costs after a finite number of iterations. However, convergence cannot occur for pessimistic costs if the planning domain contains recursive tasks that introduce solutions of arbitrary size, because pessimistic estimations are unbounded and could increase to infinity. However, these very long (and costly) plans can be rejected as, at some point, they have no chance to be involved in any feasible optimal solution. A pessimistic estimation that does not overestimate these decompositions would still preserve the effect of strictly monotonic estimations (ensuring depth-first exploration of the search space), as it would still overestimate the evaluation of these refinements that potentially lead to feasible solutions with reasonable costs. Therefore, this issue can simply be solved by providing algorithm 2 with reasonable upper bounds for all tasks with pessimistic costs tending to infinity. Another solution, which is the one we have implemented, adds an additional integer parameter to recursive tasks. This parameter is set to 0 at the first occurrence of the task, and is incremented at each recursive call, until reaching a recursion threshold beyond which the task cannot be decomposed anymore (again, this threshold is set at a reasonable value to maintain completeness and optimality). Both solutions implement the same logic in different ways, however the second one has an additional advantage : it also limits the amount of recursive decompositions at planning time, ensuring termination for algorithm 1.

Algorithm 2 Algorithm for computing cost estimations

```
Input: A planning domain $D$ with a finite set of tasks $A \cup C$ and a constant $\epsilon$.
Output: A pair of optimistic and pessimistic $(\mathcal{C}_{opt}, \mathcal{C}_{pes})$.
 1: for all $t \in A \cup C$ do
 2:     $\mathcal{C}_{opt}(t), \mathcal{C}_{pes}(t) \leftarrow (\mathcal{C}(t), \mathcal{C}(t))$ if $t \in A$ else $(0, \infty)$
 3: end for
 4: while a fixed point has not been reached do
 5:     for all $c \in C$ do
 6:         $\mathcal{C}_{opt}(c) \leftarrow \min\{\sum_{n \in N_m} \mathcal{C}_{opt}(t_n) \mid m \in M : c_m = c\}$
 7:         $\mathcal{C}_{pes}(c) \leftarrow \max\{\sum_{n \in N_m} \mathcal{C}_{pes}(t_n) \mid m \in M : c_m = c\} + \epsilon$
 8:     end for
 9: end while
10: return $(\mathcal{C}_{opt}, \mathcal{C}_{pes})$
```

3.3 Decomposition Strategies

Search control in algorithm 1 is not restricted to the evaluation functions f_1 and f_2. Indeed, search efficiency is also influenced by the policy used to select the task to decompose in a task network (at line 15 in the algorithm). We refer to this kind of policy as a *decompostion strategy*, because it dictates the decomposition of tasks during the planning process.

A popular decomposition strategy consists in selecting a task that is not constrained to be ordered after any other non-primitive tasks. This is the strategy implemented in all planners derived from SHOP. In this case, the tasks are decomposed in the order of their execution and a linear plan prefix made of primitive tasks can be obtained to compute all intermediate states. Therefore, expressive preconditions encoded in actions and methods can be efficiently evaluated on these states. This decomposition strategy has been very successful in the application of HTN planning to real-world applications [11, 14], and it would be interesting to evaluate its performance with a heuristic search algorithm. In the remainder of this paper, this strategy will be referred to as *first* (because it selects one of the "first" non primitive tasks).

But *first* may not be the most appropriate strategy to obtain task networks with accurate cost estimations. Indeed, the most abstract tasks will not be decomposed until all the previous tasks have reached the primitive level. Because these tasks are likely to be the ones with the least accurate cost estimations, the overall evaluation function may not be accurate enough to guide the search efficiently. Therefore, we also consider a decomposition strategy that prioritizes the decomposition of tasks located at the highest levels in the task hierarchy. The intuition behind this strategy is to minimize the amount of planning steps required to achieve an accurate cost estimation. To implement this strategy, we assign a "level" to every task according to the following rule: if a task t_1 belongs to any successive decompositions of a task t_2, then the level assigned to t_1 cannot be greater than the one assigned to t_2. This decomposition strategy is referred here as *highest* (because it selects one of the tasks with the "highest" level).

4 Empirical evaluation

In this section, we present an empirical study of the algorithm performance according to different heuristics and decomposition strategies. We first provide a brief description of our implemented HTN planning system. Then, we introduce the three planning domains used in the experiments, as well as the different planning profiles forming the benchmark. Eventually, we present and discuss the results.

4.1 Planning System

So far, we have introduced our ideas on a general HTN formalism. Our intention was to demonstrate that these ideas can be applied to many different HTN frameworks : non-linear HTN planners (such as UMCP [5], NONLIN [18], SIPE [20] or O-PLAN [4]...), planners following the simple, totally ordered, scheme of SHOP, or planners from the AHP framework. But we have to commit to one particular system in order to implement and evaluate the different heuristics and decomposition strategies described above. Our system has been initially designed as an implementation of the algorithm of SHOP [12], using Python as a programming language. As a result, decomposition methods are described with preconditions formulas that must be evaluated on states, and use totally ordered sequences of subtasks. Moreover, the choice of the decomposition strategy was originally restricted to *first*, as it is for SHOP and its derivatives. To remove this limitation, we have extended compound tasks with a state transition semantic based on abstract effects, comparable to the optimistic effect descriptions of the AHP framework. However, the description of this semantic is out of the scope of this paper.

4.2 Planning Domains

To guide the search efficiently, an HTN domain must contain some tasks that can accurately estimate the final cost of their decompositions, and be reachable at an early level. Hence, it is possible to compute accurate heuristics at an early stage of the search, when only a reasonable amount of task networks have been expanded. These “abstract actions” are a common feature of all domains used in the experiment.

The first one is *Robotbox* [2]. In this domain, a (virtual) robot moves boxes in a house-like environment made of rooms connected by doors. Doors can be open, closed or locked. The robot can open unlocked doors and move boxes through them. Two different move actions are available: “`CarryThruDoor`” and “`PullThruDoor`”. The first action requires that the box has been previously loaded by the robot, while the second one requires that it has been attached to the robot. These two actions can be aggregated under a compound task “`MoveThruDoor`”. Using this “abstract action”, one can obtain a good estimation of the cost of the final plan without having yet to commit to one or the other actual move action. Table 1 provides a list of all the tasks we have modeled for this domain, along with their level in the task hierarchy and the values of their cost estimations. The task “`AchieveAt`” is recorded for each instantiation of its depth parameter, with a range varying from 0 (first call) to 5 (maximal depth). Therefore, this recursive task can introduce at most 5 steps for navigating on the graph of rooms. 6 being the maximum number of rooms in our generated examples, an optimal plan cannot contain a sequence of more than 5 move actions to navigate between two rooms. Hence, a value of 5 for the depth limit is sufficient to ensure completeness and optimality.

Instances for this domain are randomly generated with 6 rooms, an average of 3 doors per rooms, a probability of 0.66 for a door to be unlocked and a probability of 0.5 for an unlocked door to be open. Only the number of boxes to move is left as a parameter to characterize instances of different sizes.

The next planning domain is an HTN extension of the PDDL *Logistics* domain, where packages are transported by trucks and airplanes between locations in different cities. Locations in cities are linked according to connected graphs of roads, and airports are linked together by a connected graph of airways. Actions are assigned different costs, with a major cost for air travel. The task hierarchy reflects the hierarchical structure of the domain, and contains “abstract actions” for flying, loading and unloading packages at the city level. These tasks provide accurate cost estimations at a level that still ignore traveling inside cities. Again, a list of all tasks with their level

Task	Level	$\mathcal{C}_{ops}$	$\mathcal{C}_{pes}$	λ_t
CarryThruDoor(b, d, r1, r2)	0	1	1	1
PullThruDoor(b, d, r1, r2)	0	1	1	1
LoadBox(b)	0	1	1	1
AttachBox(b)	0	1	1	1
OpenDoor(d)	0	1	1	1
AchieveLoaded(b)	1	0	1.001	1
AchieveAttached(b)	1	0	1.001	1
MoveThruDoor(b, d, r1, r2)	2	1	2.002	1
AchieveOpen(d)	3	0	1.001	1
AchieveAt(b, r, 5)	4	0	0.001	0.5
AchieveAt(b, r, 4)	4	0	3.005	0.5
AchieveAt(b, r, 3)	4	0	6.007	0.5
AchieveAt(b, r, 2)	4	0	9.010	0.5
AchieveAt(b, r, 1)	4	0	12.013	0.5
AchieveAt(b, r, 0)	4	0	15.016	0.5

Table 1. A list of all tasks modeled for *Robotbox*, along with their associated level, optimistic and pessimistic costs, and value for the distributed linear weighted heuristic.

and cost estimations is available in Table 2. For this domain, two recursive tasks have been modeled : "`AchieveTruckAt`", which encode the navigation of trucks inside a city, and "`AchieveAirplaneAtCity`", which encode the navigation of airplanes on the graph of airways. The sizes of these two types of graphs are bounded by 5 in the generated examples, hence both maximal depths are set to 4.

For this domain, instances are randomly generated with 5 cities, 5 locations per city (with one location being an airport), 7 trucks (with at least 1 per city) and 2 planes. The average degree of the roads and airways graphs is 1.5 and the instance size is parameterized by the number of packages to be delivered.

The last planning domain is derived from *SimpleFPS* [19]. It is intended to represent FPS-like (First Person Shooter) video game mechanics, and has been selected for its relevance to our application domain. In *SimpleFPS*, a game level is represented as a set of areas connected by waypoints. Each area contains different points of interest (medikits, weapons, coverpoints, control boxes to turn off or on the lights in areas...). The purpose of planning is to find an appropriate behavior for an NPC (Not Playable Character) who must potentially heal himself and attack a player. Different actions are available for attacking the player, and these actions are assigned costs reflecting their safety level: attacking the player in the dark with a gun equipped with night vision, and from a coverpoint, costs less than using a gun without cover, which again costs less than attacking the player with a knife in close combat. In this domain, "abstract actions" are high level versions of the base actions that solely rely on the occurrence of a type of point of interests in a given room or in the NPC inventory, and postpone the choice of the exact item at a lower level. Once more, a summary of all the tasks with their levels and cost estimations is available in Table 3, with "`Navigate`" being the only recursive task.

Task	Level	$\mathcal{C}_{ops}$	$\mathcal{C}_{pes}$	ω_t	λ_t
Load(c, p, l, c)	0	1	1	1	1
Unload(c, p, l, c)	0	1	1	1	1
Drive(t, l1, l2, c)	0	2	2	1	1
Fly(a, l1, l2, c1, c2)	0	10	10	1	1
AchieveTruckAt(t, l, c, 4)	1	0	0.001	1	0.5
AchieveTruckAt(t, l, c, 3)	1	0	2.002	1	0.5
AchieveTruckAt(t, l, c, 2)	1	0	4.003	1	0.5
AchieveTruckAt(t, l, c, 1)	1	0	6.004	1	0.5
AchieveTruckAt(t, l, c, 0)	1	0	8.005	1	0.5
Pickup(t, p, l, c)	2	0	9.006	12	1
Deliver(t, p, l, c)	2	1	9.006	12	1
AchieveInCityAt(p, c, l)	3	0	18.013	1	1
LoadAtCity(a, p, c)	4	1	19.014	24	1
UnloadAtCity(a, p, c)	4	1	1.001	2	1
FlyToCity(a, c1, c2)	4	10	10.001	1.5	1
AchieveAirplaneAtCity(a, c, 4)	5	0	0.001	1	0.3
AchieveAirplaneAtCity(a, c, 3)	5	0	10.003	1	0.3
AchieveAirplaneAtCity(a, c, 2)	5	0	20.005	1	0.3
AchieveAirplaneAtCity(a, c, 1)	5	0	30.007	1	0.3
AchieveAirplaneAtCity(a, c, 0)	5	0	40.009	1	0.3
PickupAtCity(a, p, c)	6	1	59.024	80	0.3
DeliverToCity(a, p, c)	6	1	41.011	80	0.3
AchieveAtCity(p, c)	7	0	100.036	1	0.3
AchieveAt(p, l)	8	0	118.050	1	0.3

Table 2. A list of all tasks modeled for *Logistics*, along with their associated level, optimistic and pessimistic costs, and value for the distributed weighted heuristics.

In *SimpleFPS*, instances are generated with a graph of 5 areas and an average degree of 1.5, and the instance size is parameterized by the number of points of interest.

4.3 Benchmark

For every domain, the algorithm is evaluated with an "admissible" heuristic, modeled with a unique weight $\omega = 1$, as well as a linear heuristic with a single weight $\lambda = 1$ to emulate a depth-first search (according to proposition 1). Different values for heuristics with a single weight ω have also been evaluated, and each domain is associated with one distributed weighted heuristic (ω_c), and one distributed linear weighted heuristic (λ_c). A detailed presentation of every distributed heuristic is available in Table 1, Table 2 and Table 3 for each domain. All these heuristics have been human-tailored. We have experimented different sets of values and have eventually reported the samples with the best performance. In addition, each heuristic has been evaluated twice : once with the decomposition strategy *first*, which prioritizes the decomposition of the "first" compound tasks, and a second time with the strategy *highest*, which prioritizes the decomposition of the tasks with the highest levels. We refer to the combination of a heuristic and a decomposition strategy as a "planning profile".

We have decided not to include external algorithms in our benchmark, despite the existence of cost-optimal HTN planners such as SHOP2 and AHA*. However, some planning profiles still enable to position these algorithms.

In fact, SHOP2 explores the search space in depth-first, uses the decomposition strategy *first*, and a branch-and-bound step based on the primitive cost of a task network. Therefore, the planning profile that combines a uniform linear weight equal to 1 (emulating a depth-first search), and the decomposition strategy *first*, theoretically explores the same space than SHOP2, but benefits from a more accurate bound for the branch-and-bound procedure. As a consequence, this profile provides an optimistic estimation of SHOP2's performance.

Adding an algorithm from the AHP framework would have required to model an additional amount of domain knowledge : indeed our high-level semantic for compound tasks is not as precise as AHP optimistic effects. AHP descriptions allow to model disjunctive effects for compound tasks, while our high level semantic simply abstracts these disjunctive effects and only entails effects supported by all decompositions. Moreover, they do not cover the pessimistic part of AHP high-level descriptions either. The optimistic and pessimistic costs considered here are also simpler than the one that can be modeled for AHP. We have restricted our scope to state-independent estimations, which are more practical for automatic generation, while AHP allows the user to model high-level cost depending on the current state. However it would be straightforward to extend an AHP algorithm such as AHA* with a branch-and-bound step and use one of the non-admissible heuristics presented in this work. Considering the additional amount of user knowledge expected by the AHP framework, an AHP planner would be expected to dominate our algorithm when using the same heuristic. Therefore, the admissible heuristic, while not benefiting from the branch-and-bound mechanism, has been included in the benchmark in order to position the new heuristics in comparison with the one used by AHA*.

Task	Level	$\mathcal{C}_{ops}$	$\mathcal{C}_{pes}$	ω_t	λ_t
MoveToPoint(pt, a)	0	1	1	1	1
MoveBetweenPoints(pt1, pt2, a)	0	1	1	1	1
Move(a1, a2, w)	0	0.5	0.5	1	1
TakeCover(cp, a)	0	1	1	1	1
Uncover	0	1	1	1	1
MakeAccessible(a1, a2, w, t)	0	1	1	1	1
GetItem(i, a)	0	2	2	1	1
Reload(g, am)	0	3	3	1	1
TurnLightOn(a, cb)	0	2	2	1	1
TurnLightOff(a, cb)	0	2	2	1	1
UseMedikit(m)	0	5	5	1	1
AttackMelee(k, p, a)	0	15	15	1	1
AttackRanged(g, p, a)	0	10	10	1	1
AttackRangedCovered(g, p, a)	0	7	7	1	1
SneakKill(g, p, a)	0	5	5	1	1
SneakKillCovered(g, p, a)	0	2	2	1	1
AchieveCloseTo(pt, a)	1	0	1.001	1	1
TakeCover(a)	2	1	2.002	5	1
TurnLightOn(a)	2	2	3.002	5	1
TurnLightOff(a)	2	2	3.002	5	1
UseMedikit	2	5	5.001	5	1
ReloadGun	2	3	3.001	5	1
ReloadNvGun	2	3	3.001	5	1
MakeAccessible(a1, a2, w)	2	1	1.001	5	1
AttackMelee(p, a)	2	15	16.002	5	1
AttackRanged(p, a)	2	10	10.001	5	1
AttackRangedCovered(p, a)	2	7	7.001	5	1
SneakKill(p, a)	2	5	5.001	5	1
SneakKillCovered(p, a)	2	2	2.001	5	1
GetMedikit(a)	2	2	3.002	5	1
GetAmmo(a)	2	2	3.002	5	1
GetKnife(a)	2	2	3.002	5	1
GetGun(a)	2	2	3.002	5	1
GetTool(a)	2	2	3.002	5	1
GetNvGun(a)	2	2	3.002	5	1
GetLoadedGun(a)	2	2	3.002	5	1
GetLoadedNvGun(a)	2	2	3.002	5	1
AchieveUncovered	3	0	1.001	1	1
AchieveCovered(a)	3	0	2.003	1	1
AchieveLighted(a)	4	0	4.004	1	1
AchieveDark(a)	4	0	4.004	1	1
Navigate(a1, a2, 3)	5	0.5	1.502	10	0.2
Navigate(a1, a2, 2)	5	0.5	3.004	10	0.2
Navigate(a1, a2, 1)	5	0.5	4.506	10	0.2
Navigate(a1, a2, 0)	5	0.5	6.008	10	0.2
AchieveAt(a)	6	0	7.01	1	0.2
AchieveHoldingMedikit	7	0	14.017	1	0.2
AchieveHoldingKnife	7	0	14.017	1	0.2
AchieveHoldingAmmo	7	0	14.017	1	0.2
AchieveHoldingGun	7	0	14.017	1	0.2
AchieveHoldingNvGun	7	0	14.017	1	0.2
AchieveHoldingLoadedGun	7	0	31.036	1	0.2
AchieveHoldingLoadedNvGun	7	0	31.036	1	0.2
AchieveFullHealth	8	0	19.019	1	0.2
AchieveWounded(p)	8	0	41.034	1	0.2

Table 3. A list of all tasks modeled for *SimpleFPS*, along with their associated level, optimistic and pessimistic costs, and value for the distributed weighted heuristics.

4.4 Results

The results are presented in Table 4. For every instance type, a collection of 100 random solvable problems is generated. For each of these collections, we have recorded the amount of instances solved, as well as data from two time points in the search with a time-out of 180 seconds. The first time point corresponds to the time when the first solution is found. Knowing the solution quality at this point is a valuable information, especially for real-time applications, which need a near-optimal solution with a short run-time. The second time point corresponds with the best solution found within the time-out. For both time points, the results provide the amount of task networks expanded, the elapsed CPU time and the cost of the best solution found so far. Every planning profile is described with its decomposition strategy followed by its corresponding heuristic.

As expected, the weighted heuristics with single value ($\omega = 5$ for example), does not perform well here : they fail to solve the integrity of several collections. Despite the high value of some of these weights, their effect on search is limited. Indeed they cannot be applied on tasks with 0 optimistic cost (for example, this is the case for 4 types of compound tasks over 5 in *Robotbox*). Moreover, as long as the task network is only made of compound tasks, they do not alter search compared to an admissible heuristic. As a result, the first solution returned with these heuristics is already close to the optimal one, and they barely benefit from the branch and bound. The distributed version of these weights (ω_c) have been designed to solve the last issue. With these heuristics, tasks at the lowest levels of the hierarchy are associated with lower weights. Therefore, every decomposition of a task reduces the overestimation and guides the search toward deeper task networks. We can see that they often perform better than the single value one.

The planning profiles using the depth-first search ($\lambda = 1$) always manage to solve every problem instance. They return a first solution with a minimum amount of search, but at the expense of plan quality. In addition, they require a significant amount of search to improve this quality. Planning profiles using distributed linear weight heuristics (λ_c) return a first solution with an amount of search comparable to depth-first search. However, they achieve much better plan quality in general, and they are also fast at improving this plan quality. Combined with the decomposition strategy *highest*, this heuristic dominates all the other profiles in term of search time, while achieving the best plan quality.

As for the decomposition strategies, $highest$ outperforms $first$ in most situations, especially in *Robotbox* and *Logistics*. In these domains, graph navigation plays the major role (whatever it is a graph of rooms, or a graph of city). The navigation relies on recursive tasks with very inaccurate estimations. The strategy $highest$ focuses on the decomposition of these recursive tasks, and quickly produces plans made of abstract actions with accurate cost estimations. On the opposite side, the strategy *first* maintains these recursive tasks in the task networks until all their predecessors have been refined to the primitive levels, and does not benefit from good cost evaluations. In the case of SimpleFPS, the cost of navigation is less significant, and there does not appear to be a clear advantage between both decomposition strategies.

Every heuristic evaluated here has been configured and selected manually. We have done our best to identify good combinations of weights in order to reach a satisfying compromise between getting a first solution in a short time and improving plan quality as fast as possible. Therefore, we have no guaranty that these are the best configurations to achieve this goal. For instance, there could still be a set of weights for a distributed heuristic (ω_c) that could compete against the selected linear heuristic (λ_c). However, finding an efficient linear heuristic turned out to be a simpler task compared to weighted heuristics. Indeed, the combination between optimistic and strictly pessimistic costs that characterize the heuristic (λ_c) provides a convenient tool to reason about depth-first or admissible exploration of the search space on a task-by-task basis. We consider this feature as an additional advantage on behalf of linear weighted heuristics.

5 Conclusions and future work

In this article, we have applied a general HTN framework to the context of anytime search where we aim at improving solution quality over time. We have presented and evaluated different weighted heuristics based on optimistic and pessimistic cost estimations computed at the level of compound tasks. We have empirically demonstrated that for three planning domains, a heuristic consisting of a linear combination of the optimistic and pessimistic estimations is able to return a first solution in a very short time, and also provides the best rate for improving plan quality. The experiments also reveal that an additional speed-up can generally be obtained with a decomposition strategy that assigns priorities to more abstract tasks.

The heuristics and decomposition strategies achieving the best performance can directly be applied in the case of nonlinear HTN planning, where the flexible search operations support any decomposition strategy. This is also the case for a progression based planner with a state transition semantic for compound tasks, such as AHP or our extension of SHOP based on abstract effects. In addition, a SHOP-based planner without any particular extension, could still benefit of this heuristic search framework to improve solution quality, but in a limited scope, as the decomposition strategy would be restricted to *first*.

Considering that all weighted heuristics must be configured manually, a future extension for this work would consist in developing a system to derive them automatically. We believe that the relevance of a given heuristic is bound to the properties of the problem instances the planner is intended to solve. Therefore, a promising direction for future work could be to apply machine learning techniques to learn better weights from previously solved problems with similar heuristics. However, we have not made any step in this direction yet.

instance type	planning profile	% solved	first solution			best solution		
			# expanded	CPU time (ms)	cost	# expanded	CPU time (ms)	cost
	first $\omega = 1$	100.0	37	0.073	1.80	37	0.073	1.80
	highest $\omega = 1$	100.0	25	0.084	1.80	25	0.084	1.80
	first $\omega = 1.5$	100.0	22	0.042	1.80	22	0.042	1.80
	highest $\omega = 1.5$	100.0	12	0.035	1.80	12	0.035	1.80
	first $\omega = 5$	100.0	22	0.041	1.80	22	0.041	1.80
Robotbox 1 box	highest $\omega = 5$	100.0	8	0.024	1.80	8	0.024	1.80
	first $\lambda = 1$	100.0	16	0.034	2.70	41	0.073	1.80
	highest $\lambda = 1$	100.0	10	0.031	2.70	50	0.118	1.80
	first (λ_c)	100.0	16	0.033	2.28	32	0.061	1.80
	highest (λ_c)	100.0	6	0.020	1.80	6	0.020	1.80
	first $\omega = 1$	100.0	386	1.095	3.99	386	1.095	3.99
	highest $\omega = 1$	100.0	268	1.002	3.99	268	1.002	3.99
	first $\omega = 1.5$	100.0	289	0.768	3.99	289	0.768	3.99
	highest $\omega = 1.5$	100.0	74	0.271	3.99	74	0.271	3.99
	first $\omega = 5$	100.0	275	0.787	3.99	275	0.787	3.99
Robotbox 2 boxes	highest $\omega = 5$	100.0	36	0.170	4.00	38	0.178	3.99
	first $\lambda = 1$	100.0	40	0.115	6.25	224	0.602	3.99
	highest $\lambda = 1$	100.0	23	0.101	6.25	2829	8.030	3.99
	first (λ_c)	100.0	44	0.132	5.01	120	0.322	3.99
	highest (λ_c)	100.0	15	0.054	4.03	18	0.065	3.99
	first $\omega = 1$	100.0	1340	4.988	5.09	1340	4.988	5.09
	highest $\omega = 1$	93.0	1904	8.522	4.65	1904	8.522	4.65
	first $\omega = 1.5$	100.0	931	3.895	5.09	931	3.895	5.09
	highest $\omega = 1.5$	100.0	198	0.996	5.09	198	0.996	5.09
	first $\omega = 5$	100.0	869	3.677	5.09	869	3.677	5.09
Robotbox 3 boxes	highest $\omega = 5$	100.0	65	0.445	5.09	65	0.445	5.09
	first $\lambda = 1$	100.0	39	0.149	7.57	629	2.233	5.09
	highest $\lambda = 1$	100.0	27	0.120	7.57	3641	11.712	6.00
	first (λ_c)	100.0	39	0.150	6.23	237	0.862	5.09
	highest (λ_c)	100.0	16	0.082	5.09	16	0.082	5.09
	first $\omega = 1$	100.0	220	1.913	35.14	220	1.913	35.14
	highest $\omega = 1$	100.0	127	1.202	35.14	127	1.202	35.14
	first $\omega = 4$	100.0	171	1.183	35.16	172	1.184	35.14
	highest $\omega = 4$	100.0	79	0.528	35.20	79	0.529	35.14
	first $\omega = 10$	100.0	130	0.881	35.42	131	0.886	35.14
	highest $\omega = 10$	100.0	65	0.431	35.66	75	0.482	35.14
Logistics 1 package	first (ω_c)	100.0	39	0.262	40.16	90	0.539	35.14
	highest (ω_c)	100.0	56	0.352	40.16	95	0.555	35.14
	first $\lambda = 1$	100.0	41	0.271	56.76	183	0.963	35.14
	highest $\lambda = 1$	100.0	38	0.257	56.76	173	0.941	35.14
	first (λ_c)	100.0	63	0.622	35.52	67	0.647	35.14
	highest (λ_c)	100.0	38	0.371	35.66	45	0.414	35.14
	first $\omega = 1$	93.0	2176	22.356	64.84	2176	22.356	64.84
	highest $\omega = 1$	99.0	1091	14.628	66.91	1091	14.628	66.91
	first $\omega = 4$	96.0	2709	23.531	65.75	2709	23.531	65.75
	highest $\omega = 4$	99.0	430	5.779	67.01	435	5.825	66.91
	first $\omega = 10$	96.0	2344	19.372	66.08	2347	19.385	65.75
	highest $\omega = 10$	100.0	305	4.347	68.32	393	5.085	67.24
Logistics 2 packages	first (ω_c)	100.0	1324	8.709	73.36	1409	9.367	67.24
	highest (ω_c)	100.0	242	2.907	75.92	628	7.014	67.44
	first $\lambda = 1$	100.0	81	0.744	105.88	2348	19.772	68.72
	highest $\lambda = 1$	100.0	75	1.070	105.88	1568	17.909	67.44
	first (λ_c)	100.0	554	4.728	67.62	558	4.761	67.24
	highest (λ_c)	100.0	100	1.696	68.42	174	2.541	67.24
	first $\omega = 1$	38.0	4510	53.798	81.21	4510	53.798	81.21
	highest $\omega = 1$	58.0	3264	56.432	91.24	3264	56.432	91.24
	first $\omega = 4$	45.0	5679	45.040	89.38	5679	45.040	89.38
	highest $\omega = 4$	96.0	2954	40.944	111.54	3073	42.514	111.10
	first $\omega = 10$	48.0	6105	47.478	91.25	6108	47.487	90.75
	highest $\omega = 10$	98.0	1725	27.153	114.78	2747	37.647	112.71
Logistics 3 packages	first (ω_c)	60.0	6206	40.227	103.33	6354	41.186	95.37
	highest (ω_c)	98.0	1224	15.613	128.65	3744	44.806	116.39
	first $\lambda = 1$	100.0	132	1.255	172.80	9070	66.666	122.82
	highest $\lambda = 1$	100.0	120	1.880	172.80	6839	76.012	127.64
	first (λ_c)	84.0	2180	23.682	104.00	2195	23.782	103.50
	highest (λ_c)	100.0	324	9.354	115.52	1133	21.551	113.02
	first $\omega = 1$	99.0	341	36.411	20.77	341	36.411	20.77
	highest $\omega = 1$	81.0	609	46.653	19.25	609	46.653	19.25
	first $\omega = 5$	100.0	109	9.697	21.79	211	18.856	20.83
	highest $\omega = 5$	95.0	181	16.164	21.01	300	23.043	20.38
	first (ω_c)	100.0	100	8.613	21.79	199	17.286	20.83
SimpleFPS 20 poi	highest (ω_c)	100.0	147	15.007	21.54	280	23.570	20.88
	first $\lambda = 1$	100.0	37	2.466	33.38	353	23.685	20.82
	highest $\lambda = 1$	100.0	37	3.194	33.38	780	45.606	21.27
	first (λ_c)	100.0	76	6.938	21.52	107	9.229	20.79
	highest (λ_c)	100.0	47	5.613	21.64	81	8.032	20.79
	first $\omega = 1$	89.0	271	52.241	16.26	271	52.241	16.26
	highest $\omega = 1$	76.0	317	35.116	15.26	317	35.116	15.26
	first $\omega = 5$	100.0	117	17.876	17.86	153	23.205	17.14
	highest $\omega = 5$	97.0	152	25.660	17.27	177	28.346	16.76
	first (ω_c)	100.0	101	16.587	17.89	135	21.159	17.14
SimpleFPS 40 poi	highest (ω_c)	98.0	106	18.675	17.30	129	21.072	16.85
	first $\lambda = 1$	100.0	32	4.583	30.07	328	41.460	17.26
	highest $\lambda = 1$	100.0	31	5.042	30.07	451	42.546	18.66
	first (λ_c)	100.0	59	11.572	17.94	95	16.258	17.00
	highest (λ_c)	100.0	37	9.790	18.14	61	12.351	17.01
	first $\omega = 1$	67.0	196	58.619	13.36	196	58.619	13.36
	highest $\omega = 1$	79.0	299	55.678	13.75	299	55.678	13.75
	first $\omega = 5$	96.0	106	35.962	15.12	127	42.881	14.61
	highest $\omega = 5$	93.0	93	28.410	14.96	142	34.462	14.22
	first (ω_c)	99.0	90	30.093	15.06	111	36.357	14.66
SimpleFPS 80 poi	highest (ω_c)	95.0	84	27.391	14.94	118	31.276	14.41
	first $\lambda = 1$	100.0	25	8.678	26.53	226	57.941	16.17
	highest $\lambda = 1$	100.0	23	7.504	26.53	360	56.027	16.21
	first (λ_c)	97.0	40	20.472	15.04	61	24.706	14.51
	highest (λ_c)	100.0	27	14.305	15.45	60	18.728	14.78

Table 4. Results obtained from solving different collections of problems in 3 HTN planning domains with different heuristics and decomposition strategies. (ω_c) and (λ_c) refer to distributed weighting heuristics (a different weight is assigned to each compound task) and uniform weights are described with their value.

References

1. Alford, R., Shivashankar, V., Kuter, U., Nau, D.S.: HTN Problem Spaces: Structure, Algorithms, Termination. In: SOCS (2012)
2. Bacchus, F., Yang, Q.: Downward Refinement and the Efficiency of Hierarchical Problem solving. Artificial Intelligence 71(1), 43–100 (1994)
3. Baier, S.S.J.A., McIlraith, S.A.: HTN Planning With Preferences. In: Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence. pp. 1790–1797 (2009)
4. Currie, K., Tate, A.: O-Plan: the Open Planning Architecture. Artif. Intell. 52(1), 49–86 (1991)
5. Erol, K., Hendler, J.A., Nau, D.S.: UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning . In: AIPS. vol. 94, pp. 249–254 (1994)
6. Erol, K., Hendler, J.A., Nau, D.S.: Semantics for Hierarchical Task-network Planning. Tech. rep., DTIC Document (1995)
7. Ghallab, M., Nau, D., Traverso, P.: Hierarchical Task Network Planning. In: Automated Planning: Theory & Practice, pp. 229–261. Morgan Kaufmann Publishers Inc. (2004)
8. Hansen, E.A., Zhou, R.: Anytime Heuristic Search. J. Artif. Intell. Res.(JAIR) 28, 267–297 (2007)
9. Hawes, N.: An Anytime Planning Agent for Computer Game Worlds. In: Proceedings of the Workshop on Agents in Computer Games at The 3rd International Conference on Computers and Games. pp. 1–14 (2002)
10. Marthi, B., Russell, S.J., Wolfe, J.A.: Angelic Hierarchical Planning: Optimal and Online Algorithms. In: ICAPS. pp. 222–231 (2008)
11. Nau, D.S., Aha, D.W., Muoz-Avila, H.: Ordered Task Decomposition. In: AAAI Workshop on Representational Issues for Real-World Planning Systems. AAAI Press (2000)
12. Nau, D., Cao, Y., Lotem, A., Munoz-Avila, H.: SHOP: Simple Hierarchical Ordered Planner. In: Proceedings of the 16th international joint conference on Artificial intelligence. vol. 2, pp. 968–973. Morgan Kaufmann Publishers Inc. (1999)
13. Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: SHOP2: An HTN Planning System. J. Artif. Intell. Res.(JAIR) 20, 379–404 (2003)
14. Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Muoz-Avila, H., Murdock, J.W., Wu, D., Yaman, F.: Applications of SHOP and SHOP2. IEEE Intelligent Systems 20(2), 34–41 (2005)
15. Pohl, I.: Heuristic Search Viewed as Path Finding in a Graph. Artificial Intelligence 1(3), 193–204 (1970)
16. Pohl, I.: The Avoidance of (Relative) Catastrophe, Heuristic competence, Genuine Dynamic Weighting and Computational Issues in Heuristic Problem Solving. In: Proceedings of the 3rd international joint conference on Artificial intelligence. pp. 12–17. Morgan Kaufmann Publishers Inc. (1973)
17. Sohrabi, S., McIlraith, S.A.: On Planning With Preferences in HTN. In: Proc. of the 12th Intl Workshop on Non-Monotonic Reasoning (NMR). pp. 241–248 (2008)
18. Tate, A.: Generating Project Networks. In: Proceedings of the 5th international joint conference on Artificial intelligence-Volume 2. pp. 888–893. Morgan Kaufmann Publishers Inc. (1977)
19. Vassos, S., Papakonstantinou, M.: The SimpleFPS Planning Domain: A PDDL Benchmark for Proactive NPCs. In: Intelligent Narrative Technologies (2011)
20. Wilkins, D.E.: Practical Planning - Extending the Classical AI Planning Paradigm . Morgan Kaufmann series in representation and reasoning, Morgan Kaufmann (1988)