A Generic Method to Guide HTN Progression Search with Classical Heuristics

Daniel Höller, Pascal Bercher, Gregor Behnke, Susanne Biundo
Institute of Artificial Intelligence, Ulm University, D-89069 Ulm, Germany
{daniel.hoeller, pascal.bercher, gregor.behnke, susanne.biundo}@uni-ulm.de

Abstract

HTN planning combines actions that cause state transition with grammar-like decomposition of compound tasks that additionally restricts the structure of solutions. There are mainly two strategies to solve such planning problems: decomposition-based search in a plan space and progression-based search in a state space. Existing progression-based systems do either not rely on heuristics (e.g. SHOP2) or calculate their heuristics based on extended or modified models (e.g. GoDeL). Current heuristic planners for standard HTN models (e.g. PANDA) use decomposition-based search. Such systems represent search nodes more compactly due to maintaining a partial order between tasks, but they have no current state at hand during search. This makes the design of heuristics difficult. In this paper we present a progression-based heuristic HTN planning system: We (1) provide an improved progression algorithm, prove its correctness, and empirically show its efficiency gain; and (2) present an approach that allows to use arbitrary classical (non-hierarchical) heuristics in HTN planning. Our empirical evaluation shows that the resulting system outperforms the state-of-the-art in HTN planning.

1 Introduction

Hierarchical Task Network (HTN) planning combines *primitive* tasks that can be executed directly and cause state transitions like actions in classical planning with *compound* tasks. These describe more abstract tasks and have to be decomposed using (decomposition) methods – grammar-like rules on how to divide the task into other tasks that may be compound or primitive. Decomposition is continued until all tasks are primitive. These rules have huge influence on the set of solutions and it has been shown that the plan existence problem in HTN planning is (strictly) semi-decidable (Erol, Hendler, and Nau 1996) and that it is far more expressive than classical planning (Höller et al. 2014; 2016).

There are mainly two approaches to solve HTN planning problems: decomposition-based search in a plan space and progression-based search in a state space. All approaches need to combine the restrictions of state transition *and* decomposition to find a solution. This makes the design of heuristics in HTN planning difficult. To make it easier, some systems rely on extended models (like SHOP2, Nau et al. 2003) or on modified models (like GoDeL, Shivashankar et al. 2013). Current *heuristic* search systems for standard HTN models (e.g. PANDA, Bercher, Keen, and Biundo 2014) are based on a decomposition-based search. These systems benefit from their systematicity and use a compact search node representation that maintains a partial order of tasks, but they have poor information about the current state during search, so they can not combine state and hierarchy properly.

In this paper we show how a progression-based search enables using existing state-based heuristics from classical planning to solve general HTN problems without extending or modifying the model. Our main contributions are:

1. We show how to improve the standard progression algorithm by reducing the number of non-deterministic choice-points.
2. We show how to integrate relaxed hierarchy information into the state of the problem in order to use arbitrary classical heuristics to guide the progression planner. Heuristic values based upon this information estimate the sum of decompositions *and* applied actions needed to generate a solution and can thus be used to guide a progression search in a state space.

We first summarize related work. Then, after introducing the formal framework, we show how to improve the progression algorithm and describe our approach to use classical heuristics to guide its search. The overall system is evaluated afterwards[1].

2 Related Work

The maybe best-known HTN planning system, SHOP2 (Nau et al. 2003), is a progression-based system performing a depth-first search. To control search, it does not rely on heuristics to estimate the goal distance. Instead, search is guided by other features like state-based preconditions that define when a decomposition method can be applied and an if-then-else structure in methods. This hand-modeled *advice* makes it a very efficient domain-*configurable* planning approach (Nau 2007). However, this advice has to be developed from scratch and by hand for every new domain.

[1] Source code as well as newly introduced evaluation domains are available online at www.uni-ulm.de/en/in/ki/panda.

Other approaches extend or modify the model to make it more prone for integrating heuristics and/or other search techniques from classical planning. Waisbrot, Kuter, and Könik (2008) added a goal definition to methods and calculate heuristics based on these goals. Shivashankar et al. introduced a novel formalism, Hierarchical Goal Network (HGN) planning (Shivashankar et al. 2012; Alford et al. 2016b), that does not decompose *tasks*, but *goals* and developed algorithms and heuristics for HGN problems (Shivashankar et al. 2013; 2016; Shivashankar, Alford, and Aha 2017).

We give a domain-*independent* approach based on heuristic search that is intended to work on domains representing physics, no advice – like classical planners (McDermott 2000). This allows the system to solve arbitrary HTN planning problems without encoding the search strategy into the problems. There are several approaches in the literature that address the same objective: Lotem, Nau, and Hendler introduced a combination of a *planning graph* (to represent relaxed state transition) with a so-called *planning tree* (that represents decomposition) (Lotem, Nau, and Hendler 1999; Lotem and Nau 2000). The two graphs are built interleaved and restrict each other. The plan is extracted from this combined structure, i.e. the approach uses a specialized algorithm and does not calculate a heuristic for a standard search. Gerevini et al. (2008) interleave classical heuristic search and HTN decomposition. It allows for task insertion (which lowers expressivity and makes them solve a different problem class) and does no heuristic search in the HTN part of the search. Alford et al. introduced translations into classical planning (Alford, Kuter, and Nau 2009; Alford et al. 2016a). For general HTN problems, iteratively incrementing a bound is needed, like in SAT-based classical planning. We do not aim at providing such a translation, but at using classical heuristics in our HTN planning system.

An example for a *plan space*-based system is FAPE. It has a slightly different focus than the systems given in this section (and ours): It comes with sophisticated support for planning with time at the cost of not supporting recursion (Dvorak et al. 2014). Enforcing non-recursive models lowers the expressivity of the formalism (Höller et al. 2016). Another major difference is the lack of a goal distance estimation. FAPE uses a domain-independent delete-relaxed *pruning* of search nodes combined with blind search (Bit-Monnot, Smith, and Do 2016). The PANDA system also performs a heuristic search in *plan space*, combining decomposition-based search with partial order causal link (POCL) planning (Bercher, Keen, and Biundo 2014). All its recent heuristics are based on the *task decomposition graph* (TDG) – an AND/OR graph that compactly represents the reachability information imposed by the decomposition methods (Bercher et al. 2017, Def. 1). Elkawkagy et al. have shown how hierarchical landmarks, extracted from a TDG, can be used for search strategies (Elkawkagy et al. 2012). In more recent work, heuristics are computed that estimate the number of decompositions and causal link insertions required to find a goal or the cost of (resp. number of) actions that still need to be inserted (Bercher et al. 2017). We propose a *progression*-based search to integrate more state information into our heuristics.

3 Formal Framework

We use the formalism of Höller et al. (2016), which is based on the one by Geier and Bercher (2011), and defines HTN planning as extension of STRIPS.

3.1 STRIPS Planning

A STRIPS planning problem is a tuple $p_c = (L, A, s_0, g, \delta)$, whereas L is a set of propositional environment facts, A the set of action names, $s_0 \in 2^L$ is the initial state, and $g \in 2^L$ the goal description; states $s \supseteq g$ are called *goal states*. The functions $prec$, add, and del map an action to its preconditions, add-, and delete-effects, respectively. These functions are given in a tuple $\delta = (prec, add, del)$ and are all defined as $f : A \to 2^L$. Whether a primitive task a is applicable in a state s is given by the relation $\tau : A \times 2^L$ with $\tau(a, s) \Leftrightarrow prec(a) \subseteq s$. Given that $\tau(a, s)$ holds, the state resulting from the application is given by the state transition function $\gamma : A \times 2^L \to 2^L$ with $\gamma(a, s) = (s \setminus del(a)) \cup add(a)$. A sequence of actions $\langle a_0 a_1 \dots a_l \rangle$ with $a_i \in A$ is applicable in a state s_0 iff $\tau(a_i, s_i)$ holds with $s_i = \gamma(a_{i-1}, s_{i-1})$ for $i > 0$. The state s_{l+1} results from its application. A sequence $\langle a_0 a_1 \dots a_l \rangle$ is a solution to a STRIPS planning problem iff it is applicable in s_0 and results in a goal state.

3.2 HTN Planning

An HTN planning problem is defined as a tuple $p = (L, C, A, M, s_0, tn_I, g, \delta)$. The elements L, A, s_0, g, and δ are defined as given above. C defines the set of compound task names. Task names are organized in *task networks*. A task network is a triple $tn = (T, \prec, \alpha)$. T is a (possibly empty) set of identifiers that are mapped to task names by a function $\alpha : T \to A \cup C$. This enables a task name to be contained in a task network more than once. A set of ordering constraints $\prec : T \times T$ defines a partial order on the identifiers. If two task networks $tn = (T, \prec, \alpha)$ and $tn' = (T', \prec', \alpha')$ differ only in their identifiers, i.e. there is a bijection $\sigma : T \to T'$ so that for all identifiers $t, t' \in T$ holds that $[(t, t') \in \prec] \Leftrightarrow [(\sigma(t), \sigma(t')) \in \prec']$ and $\alpha(t) = \alpha'(\sigma(t))$, they are called to be *isomorphic* ($tn \cong tn'$). tn_I is the initial task network. The definition using an initial task *network* enables the interpretation of every node in the search space as a new planning problem (Alford et al. 2012). However, for some proofs it is beneficial to have a single initial task *name* instead of a task *network*. An initial task network can be compiled away by introducing a new task name (used for the new initial task) and a method that decomposes this task into the original task network (Geier and Bercher 2011). I.e. the two definitions can be regarded equivalent and we will use the variant that is most appropriate for a given paragraph.

The set of decomposition methods M defines how compound tasks may be decomposed. A method $m \in M$ is a pair (c, tn) of a compound task name $c \in C$ and a task network, called the method's *subnetwork*. The tasks in the subnetwork are called the *subtasks* of the method. We use a definition without method preconditions. This is not a restriction since the definitions are equivalent: method preconditions can be compiled away by introducing a new task name that holds

the precondition and is ordered at first position of the subnetwork. When a task t is decomposed, it is removed from the network, the method's subtasks are added and all ordering constraints that hold for t are introduced for the subtasks.

Formally, a method (c, tn) decomposes a task network $tn_1 = (T_1, \prec_1, \alpha_1)$ into a task network $tn_2 = (T_2, \prec_2, \alpha_2)$ if $t \in T_1$ with $\alpha_1(t) = c$ and there is a task network $tn' = (T', \prec', \alpha')$ with $tn' \cong tn$ and $T_1 \cap T' = \emptyset$. The task network tn_2 is defined as

$$
\begin{aligned}
tn_2 =& ((T_1 \setminus \{t\}) \cup T', \prec' \cup \prec_D, (\alpha_1 \setminus \{(t \mapsto c)\}) \cup \alpha') \\
\prec_D =& \{(t_1, t_2) \mid (t_1, t) \in \prec_1, t_2 \in T'\} \cup \\
& \{(t_1, t_2) \mid (t, t_2) \in \prec_1, t_1 \in T'\} \cup \\
& \{(t_1, t_2) \mid (t_1, t_2) \in \prec_1, t_1 \neq t, t_2 \neq t\}
\end{aligned}
$$

When a task network tn can be decomposed into a task network tn' by applying the method m to a task with the identifier t, we write $tn \xrightarrow[t,m]{} tn'$; if it is possible using an arbitrary number of methods in sequence, we write $tn \rightarrow^* tn'$.

A task network $tn = (T, \prec, \alpha)$ that can be reached via decomposing tn_I (i.e. $tn_I \rightarrow^* tn$) is a solution to a planning problem p if and only if all task names are primitive ($\forall t \in T : \alpha(t) \in A$), there exists a sequence $\langle t_1 t_2 \dots t_n \rangle$ of the task identifiers in T that is in line with $\prec$, and the application of $\langle \alpha(t_1)\alpha(t_2) \dots \alpha(t_n) \rangle$ in s_0 results in a goal state.

In HTN planning, it is easy to compile a state-based goal into the hierarchy. Therefore it is often omitted in the definition (though some include it, e.g. Biundo et al. 2011 and Bercher et al. 2016). Our approach smoothly combines task decomposition with state-based goals, so our definition also includes one. However, our approach works perfectly without it (in fact, most evaluation domains do not include one).

4 Avoiding Branching in Progression Search

In this section we first introduce the de-facto standard progression algorithm and show how to improve it afterwards.

Progression-based HTN planning has two characteristics: (1) Only the tasks that have no predecessor in the network are processed and (2) primitive tasks that are processed are removed and cause a state transition. An HTN planning problem is solvable if and only if progression search finds a solution (Alford et al. 2012, Thm. 3).

Algorithm 1 gives the canonic progression algorithm similar to those by Nau et al. (2003, Fig. 5), Ghallab, Nau, and Traverso (2004, p. 243), or Alford et al. (2012, p. 5). A search node is a tuple that includes the current state and a task network, i.e. the elements of the problem definition that change during search. We added the actions progressed so far as third element (i.e. the prefix of the generated solution). Initially, the triple $(s_0, tn_I, \emptyset)$ is inserted into the fringe. While the fringe is not empty, some node $n = (s, tn, \pi)$ with $tn = (T, \prec, \alpha)$ is removed (line 3). A node is a solution if and only if $T = \emptyset$ and $s \supseteq g$. During search, only those tasks that have no predecessors in the network are processed (we denote these tasks to be *unconstrained*). They are assigned to the set U in line 5. For each primitive task in U, there is no (when it is not applicable) or exactly one possible modification: a state transition in combination with the

```
1  fringe ← {(s_0, tn_I, ∅)}
2  while fringe ≠ ∅ do
3      n ← fringe.poll()
4      if isgoal(n) then return n
5      U ← n.unconstrainedNodes
6      for t ∈ U do
7          if isPrimitive(t) then
8              n' ← n.apply(t)
9              fringe.add(n')
10         else
11             for m ∈ t.methods do
12                 n' ← n.decompose(t, m)
13                 fringe.add(n')
```

Algorithm 1: Standard progression-based HTN planning

deletion of the primitive task. The resulting new search node n' is defined as $(s', (T', \prec', \alpha'), \pi')$ with $s' = \gamma(\alpha(t), s)$, $T' = T \setminus \{t\}$, $\prec' = \prec \setminus \{(t, t') \mid (t, t') \in \prec\}$, $\alpha' = \alpha \setminus \{(t \mapsto \alpha(t))\}$, and $\pi' = \pi \circ \alpha(t)$ (the concatenation of π and $\alpha(t)$). The node is added to the fringe in line 9.

For *each* compound task $t \in U$, a *set* of new search nodes is generated, one for every applicable method (loop in line 11). Formally, for each method $m = (\alpha(t), tn) \in M$, a successor (s, tn', π) with $tn \xrightarrow[t,m]{} tn'$ is generated.

The algorithm loops over all unconstrained tasks and adds for each applicable modification a new node to the fringe. For primitive tasks this is necessary: It makes a difference in which order they are applied because this is a commitment to their order in the solution. But consider compound tasks: Here, the order in which two tasks are decomposed implies no commitment to the solution and no branching is required.

Consequently, we can improve Alg. 1 by picking only a single compound task (*no* non-deterministic choice) out of U and decomposing it. Therefore we split U into U_C and U_A that hold the compound and primitive unconstrained tasks, respectively. Apart from that, line 1 to 5 of the algorithm are unchanged. The remaining lines are given in Alg. 2. Though

```
5  ...
6  for t ∈ U_A do
7      n' ← n.apply(t)
8      fringe.add(n')
9  t ← selectCompTask(U_C)
10 for m ∈ t.methods do
11     n' ← n.decompose(t, m)
12     fringe.add(n')
```

Algorithm 2: Our optimization to reduce branching

the ordering does not influence the solution, it may influence the algorithm's efficiency. A suitable strategy for picking the next task is interesting future work. Here, we do it randomly.

It is known that Alg. 1 is sound and complete (Alford et al. 2012). So the *soundness* of Alg. 2 follows directly. However, there is no formal proof published, so we give self-contained

proofs for both soundness and completeness of Alg. 2.

For both proofs, we need to show that a task network can be reached by decomposing the initial task. This is done by providing a valid *decomposition tree* (DT) that leads to it. DTs have been introduced by Geier and Bercher (2011, Def. 7-9). A DT is a tuple $g = (T_g, E_g, \prec_g, \alpha_g, \beta_g)$, where T_g and E_g are the nodes and directed edges of a tree, $\prec_g$ is a strict partial order on the nodes, α_g is a function that labels each node with a task name ($\alpha_g : T_g \to C \cup A$), and β_g labels inner nodes with a method. A DT is *valid* with respect to a planning problem if and only if its root is labeled with the initial task of the planning problem and for any inner node t with $\beta_g(t) = (c, tn)$, it holds that: (1) $\alpha_g(t) = c$; (2) the task network induced in g by the children of t are isomorphic to tn; (3) ordering constraints between the children of t in g are inherited as defined for decomposition; (4) $\prec_g$ includes only ordering constraints demanded by (2) or (3). The $yield$ of a DT is a task network containing the leafs of the tree and the parts of α_g and $\prec_g$ belonging to these tasks. Now, the following holds (Geier and Bercher 2011, Prop. 1): Given a planning problem, for every task network tn holds that there is a valid decomposition tree g with $yield(g) = tn$ if and only if $tn_I \to^* tn$. I.e. a DT is a witness for a valid decomposition leading to a task network.

Theorem 1. *Algorithm 2 is sound.*

Proof. We first show that every path in the progression search space (and thus these that are searched by Alg. 2) corresponds to a valid decomposition of the initial task, i.e., to the construction of a valid DT. To show this, the tree can be maintained during search as follows:

Let $g = (\{t\}, \emptyset, \emptyset, \{(t \mapsto n_i)\}, \emptyset)$ be the tree in the initial search node where n_i is the initial task. A new tree is generated for every decomposition (Alg. 2 line 11): Let $g = (T_g, E_g, \prec_g, \alpha_g, \beta_g)$ be the tree in the current search node $(s, (T, \prec, \alpha), \pi)$. When decomposing a task $t \in T$, $\alpha(t) = c$ with a method $m = (c, (T_m, \prec_m, \alpha_m))$[2] we define the tree of the new search node as $g = (T_g \cup T_m, E_g \cup \{(t, t') \mid t' \in T_m\}, \prec'_g, \alpha_g \cup \alpha_m, \beta_g \cup \{(t \mapsto m)\})$. The ordering relation $\prec'_g$ includes the orderings of the current tree, plus new relations that are defined according to the definition of decomposition.

The created trees have n_i as their root node, the mappings α_g and β_g are consistent, the children of a node n match its method $\beta_g(n)$, the ordering relation matches the decomposition criteria, i.e., the maintained trees are valid. When a search node is returned as a solution, its task network is empty, and the $yield$ of its corresponding tree includes exactly the primitive tasks in the generated solution π. The sequence π is a witness for an executable linearization, and the system has checked that it leads to a goal state. □

Theorem 2. *Algorithm 2 is complete.*

Proof. Let tn_S be an arbitrary (but fixed) solution to the given planning problem. Given that it is a solution, there is a valid decomposition tree $g = (T_g, E_g, \prec_g, \alpha_g, \beta_g)$ with $yield(g) = tn_S$. Given a suitable fringe (e.g. a queue), Alg. 2 will find this solution. As we have seen in the last proof, every path in the search space corresponds to a DT. The algorithm will explore a path corresponding to a tree that is isomorphic to g.

Alg. 2 has two types of modifications, decomposition and progression. It is free to mix them, but in each iteration of its outer loop (line 2), it branches over decomposing *one* compound task (with *every* applicable method) and the application of *all* primitive tasks. From a high level perspective, we will follow in this proof the path that first decomposes compound tasks until no unconstrained one is left. When no unconstrained compound tasks are left, we progress a prefix of the solution away, and start again.

Let n_I be the initial task. The search starts with a single node that contains n_I and the root of g must also be labeled with it. While there is an unconstrained compound task t, Alg. 2 selects one of them (line 9). It applies all methods that are applicable to that specific task (line 10), i.e., it will apply the method $\beta_g(t)$ used in the given solution. The decomposition introduces the subtasks of $\beta_g(t)$. Since it is the same method as in g, these tasks equal the children of t in g.

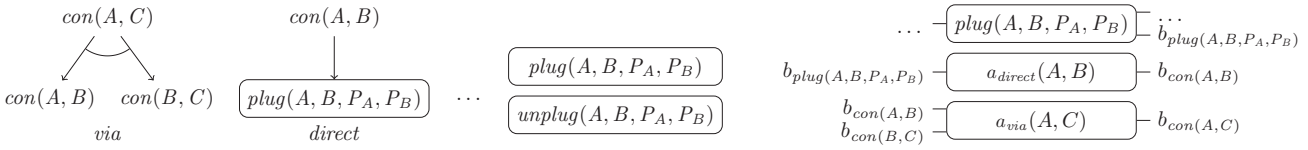
At some point there will be no unconstrained compound task left, i.e. no further decomposition is possible. Since tn_S is a solution, there is an applicable sequence $\bar{a}$ of its primitive tasks that is in line with the ordering constraints introduced by the methods in β_g and leads to a goal state. Alg. 2 applied the same methods, i.e., it ended up with the same tasks, and since no further decomposition is possible, the first task(s) of the solution must be included. This means that Alg. 2 can progress a prefix of (at least one) action(s) that equals the prefix of $\bar{a}$. Since the progression is done via branching, all possible combinations are applied. After a progression, one of the following three cases occurs: (1) an compound task is unconstrained that had been ordered after the progressed task, then we can follow this path as given above; (2) the network is empty, then we have reconstructed the given solution; or (3) another progression is needed. Finally, Alg. 2 will have processed all tasks by applying the same methods as in g to compound tasks and by progressing primitive tasks in the same order as they appear in $\bar{a}$. □

5 Guiding HTN Search with Classical Heuristics

In classical planning, a common way to create search heuristics is to relax the PSPACE-complete problem to a problem solvable in P, and to use the solution of that problem as heuristic estimation. We use a similar approach: We relax the undecidable HTN problem to a (PSPACE-complete) STRIPS problem. This problem is passed to a classical heuristic that will further relax it to solve it in P. The heuristic estimation is used in the HTN search. Our translation includes the non-hierarchical part of the HTN and a relaxed part of the restrictions induced by the hierarchy. It can be combined with arbitrary classical heuristics to calculate heuristic values for the original HTN planning problem.

Using progression search, state is tracked during search and can be the basis for estimating a goal distance. This makes it easier to use such a transformation-based approach.

[2] m is an isomorph copy of the applied method with new IDs



(a) Domain definition. Preconditions and effects are omitted due to readability.

(b) Translation of via, $direct$ and $plug$.

Figure 1: A simple entertainment domain. Boxes represent actions, non-boxed nodes represent compound tasks. Arrows indicate how tasks are decomposed by a method.

However, the following issues have to addressed:

- The hierarchy may have a huge impact on reachable solutions and
- usually there is no state-based goal as basis for the classical heuristic.

We first introduce the ideas behind our transformation based on examples. However, a self-contained formal definition is given in the beginning of Section 5.3.

5.1 Bottom-Up-Reachability

Our transformation tracks which tasks may be reached via the hierarchy. Intuitively, it simulates a (relaxed) task *composition*, i.e. the reverse application of methods, starting with the primitive tasks. For each task $n \in C \cup A$ we introduce a new state feature b_n indicating that this specific task has been reached (it is bottom-up-reachable). Every primitive task makes the corresponding reachability fact true. For each method $m = (c, tn)$, a new action a_m is introduced that has the facts corresponding to the tasks in tn as preconditions and b_c as single effect. The goal is to reach the facts corresponding to the tasks in the current network.

Figure 1a shows a simple domain describing how to connect entertainment devices, e.g. a DVD player and a TV. The connect task (*con*) may be accomplished by the methods $direct$ or via. Devices and cables are treated equal. The *hierarchy* ensures that the signal is transmitted from source to sink, *not* the state part of the problem. Figure 1b shows the transformation. Consider the example of connecting a DVD player with a TV, i.e. the initial task is $con(dvd, tv)$. It can be accomplished by connecting one end of a cable with the DVD player and the other end with the TV (using once the method via and twice $direct$; resulting in two actions). Using our transformation, the goal is to reach $b_{con(dvd,tv)}$, the fact belonging to $con(dvd, tv)$. A solution to the transformation (i.e. a classical plan) could be:

1. $plug(dvd, cable, port_{dvd}, plug1_{cable})$
2. $plug(cable, tv, plug2_{cable}, port_{tv})$
3. $a_{direct(dvd,cable)}$
4. $a_{direct(cable,tv)}$
5. $a_{via(dvd,tv)}$ (with $cable$ as argument for B)

The application of the plug actions (line 1 and 2) fulfills the preconditions of the actions in line 3 and 4 that represent methods from the HTN. The execution of these two actions fulfills the preconditions of $a_{via}(dvd, tv)$ (line 5) that fulfills $b_{con(dvd,tv)}$, the goal of the transformed problem.

In this example, the classical plan provides the optimal goal distance for a progression-based HTN search, and thus a classical heuristic might also come up with it. In general, our transformation makes several relaxations:

- The system may insert tasks apart from the hierarchy – even those that are *never reachable* via decomposition. Such actions may be used (1) to fulfill preconditions of actions needed due to the hierarchy or (2) to fulfill the (state-based) goal of the original problem. This can be seen as an instance of HTN planning with task insertion (Geier and Bercher 2011; Alford, Bercher, and Aha 2015).
- Ordering relations that are enforced by the HTN (i.e. in methods' subnetworks) are entirely ignored.
- Every task must be reached only once, regardless of how often it is enforced via the hierarchy. This leads to an underestimation of the number of needed decompositions. It can be seen as task sharing (see e.g. Alford et al. 2016b).

5.2 Top-Down-Reachability

Imagine a second planning problem in our example domain: The goal is to connect several devices (DVD player, satellite receiver, etc.) with a TV with a single input port – the solution is to use an adapter with multiple input ports, connect that adapter to the TV, and connect the devices to the adapter. However, if the system is free to insert actions apart from the hierarchy, it may as well insert the *unplug* action to make the single port of the TV free again.

In general, there are two distinct cases of inserted actions:

1. The inserted action may be entirely unreachable from the current task network (like the unplug action) or
2. the system came up with a *combination* of actions that is unreachable due to the hierarchy.

The first case can be avoided easily by restricting the set of actions to those that are reachable via decomposition. Given the task network tn in the current search node, a task n is reachable if there is a task network $tn' = (T', \prec', \alpha')$ with $tn \rightarrow^* tn'$ and $\alpha(t) = n$ for some $t \in T'$. We introduce a new fact d_a (top-down-reachable) for every action a that is true if a is reachable. It is added as precondition of a and set before calculating the heuristic. By encoding it into the model, it is visible for heuristics without modification. The second case can not be detected using this technique.

Top-down-reachability (TDR) considers *one* task at a time and the respective task must only be *reachable* via the hierarchy, it must *not* be contained in a solution, there might even not be a decomposition where it is executable. This makes it feasible to compute it. It can even be preprocessed and stored in a look-up table to save search time. We describe in Section 5.4 how to implement it efficiently.

5.3 Formal Definition and Properties

We denote our encoding *Relaxed Composition Encoding* (RC). Formally, it maps an HTN problem to a classical problem. We give the definition for an arbitrary HTN planning problem. Since every search node can be seen as a new planning problem (see Sec. 3.2), it can be applied not only to the initial problem, but also to every node during search. Given an HTN planning problem $p = (L, C, A, M, s_0, tn_I, g, (prec, add, del))$ with $tn_I = (T, \prec, \alpha)$, we define the classical planning problem $\text{RC}(p) = p'$ as:

$$
\begin{aligned}
p' &= \big(L', A', s_0', g', (prec', add', del')\big) \\
L' &= L \cup L_d \cup L_b \\
&\qquad L_b = \{b_n \mid n \in A \cup C\}, \text{ with } L \cap L_d = \emptyset \\
&\qquad L_d = \{d_n \mid n \in A\}, \text{ with } (L \cup L_d) \cap L_b = \emptyset \\
A' &= A \cup A_M, \\
&\qquad A_M = \{a_m \mid m \in M\}, \text{ with } A \cap A_M = \emptyset \\
s_0' &= s_0 \cup \{d_n \mid \exists tn' : tn_I \rightarrow^* tn' = (T', \prec', \alpha'), \\
&\qquad t \in T', \alpha'(t) = n\} \\
g' &= g \cup \{b_n \mid t \in T, \alpha(t) = n\}
\end{aligned}
$$

For all $a \in A'$, the adapted δ-functions are defined as:

$$
prec'(a) = \begin{cases} prec(a) \cup \{d_a\}, & iff\, a \in A \\ \{b_n \mid t \in T, \alpha(t) = n\}, & a \in A_M, a = a_m, \\ & m = (c, (T, \prec, \alpha)) \end{cases}
$$

$$
add'(a) = \begin{cases} add(a) \cup \{b_a\}, & iff\, a \in A \\ \{b_c\}, & a \in A_M, a = a_m, \\ & m = (c, tn) \end{cases}
$$

$$
del'(a) = \begin{cases} del(a), & iff\, a \in A \\ \emptyset, & else \end{cases}
$$

Let p be an HTN planning problem and h a classical heuristic. We denote our new HTN heuristic RC^h and define it as $\text{RC}^h(p) = h(\text{RC}(p))$.

The encoding adds a linear number of new state features and one precondition and one effect to each original action. Each action that mimes a method has the same number of preconditions as the original method had subtasks. Thus, the size of the overall encoding is linear in the size of the input HTN domain.

The primary aim of our transformation is to provide guidance for progression-based HTN planning. Therefore it is important to estimate the steps in the search space needed to reach a goal, i.e. the sum of decompositions and actions (progressions) needed to reach a goal node in that graph, i.e. the number of modifications. For some HTN problem p, let $h^*_m(p)$ be the perfect modification estimation, $\text{RC}(p)$ the transformed problem and h^* the perfect heuristic to a classical planning problem. Then, the following holds:

Theorem 3 ($h^*(\text{RC}(p)) \leq h^*_m(p)$)**.** *The perfect heuristic value of the transformed problem is smaller or equal to the actual number of modifications needed to reach a goal.*

Proof. Let $g^* = (T_g, E_g, \prec_g, \alpha_g, \beta_g)$ be a DT that belongs to a solution with minimal costs. It contains one node for every compound task that has been decomposed while refining the initial task tn_I into primitive tasks and it contains the primitive tasks in the final solution, i.e. it contains exactly $h^*_m(p)$ nodes. Based on this tree, we construct a plan π^* in the transformation that contains $h^*_m(p)$ actions.

All tasks in g^* are reachable from tn_I, i.e. preconditions belonging to TDR are fulfilled for all actions corresponding to primitive tasks in the tree. Since g^* represents a solution to the problem, there is an executable ordering of the primitive tasks that is, by construction, also executable in the transformation. Original preconditions and effects are unchanged, i.e. it transforms s_0 into a state that fulfills the state-based goal. Let π^* start with this sequence. The transformed actions mark the corresponding tasks as executed. Now we go up the tree and pick some node n whose children have all been executed. By construction, $a_{\beta_g(n)}$ is executable and marks $\alpha_g(n)$ as executed. $a_{\beta_g(n)}$ is appended to π^*. That way, each node in g^* results in exactly one action in π^*. Actions in A_M have no delete effects, i.e. the state-based goal will hold after π^* and the actions belonging to methods decomposing tasks in tn_I fulfill the HTN-based goal of the transformation. □

In the following corollaries, let p_c be an arbitrary classical planning problem and p an arbitrary HTN problem.

Corollary 1 (RC preserves safety)**.** *For any safe classical heuristic h, i.e., if $(h(p_c) = \infty) \Rightarrow (h^*(p_c) = \infty)$, holds that $(h(\text{RC}(p)) = \infty) \Rightarrow (h^*_m(p) = \infty)$.*

Corollary 2 (RC preserves goal-awareness)**.** *For any goal-aware classical heuristic h, i.e., if $h(p_c) = 0$ when the goal is fulfilled, holds that $h(\text{RC}(p)) = 0$ when the goal in p is fulfilled.*

So far we pointed out properties interesting to control search. However, by introducing costs in RC and setting the costs of all actions in A_M to 0 (the original actions may have arbitrary positive costs), we can calculate a heuristic that does not estimate the number of modifications necessary to find a goal, but that estimates the action costs in the solution. Then, the following corollary holds:

Corollary 3 (RC preserves admissibility)**.** *For any admissible classical heuristic h, i.e., if $h(p_c) \leq h^*(p_c)$, holds that $h(\text{RC}(p)) \leq h^*_{ac}(p)$, where $h^*_{ac}(p)$ is the optimal HTN heuristic summing action costs.*

5.4 Implementation

We implemented the given approach based on the HTN preprocessing (e.g. grounding, hierarchical reachability analysis) of the PANDA planning system (Bercher, Keen, and Biundo 2014) and called it $\text{PANDA}_{\text{pro}}$. So far, all variants of PANDA are based on a plan space search, so we implemented the progression search and heuristics on top of it.

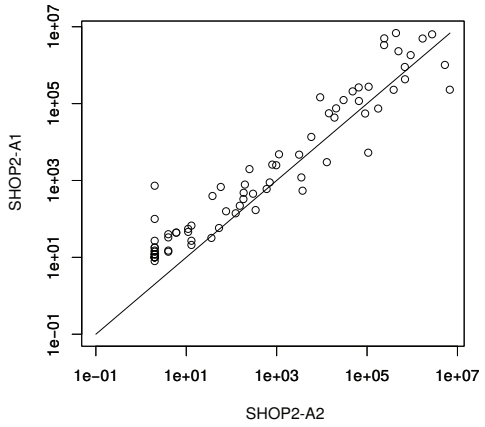


Figure 2: Scatter plot depicting the number of nodes expanded to find a plan ($\text{SHOP2}_{\text{Alg.1}}$ and $\text{SHOP2}_{\text{Alg.2}}$).

In principle, our encoding can be combined with any classical heuristic. However, it has to be considered that not only the state changes during search, but also the goal. Heuristics should therefore be able to adapt it without high costs.

To get an efficient system, it has to be considered that wide parts of the transformation can be calculated in a preprocessing step before search. During search, we need to adapt the state (including TDR of tasks) and the goal (that reflects the current task network). TDR can be calculated in a preprocessing step: First we build a graph where the tasks ($A \cup C$) form the set of nodes and each task is connected with all subtasks of methods that decompose it (this roughly equals a TDG). Then, the strongly connected components (SCCs) of the graph are calculated. By definition, every node in a SCC is reachable from all other nodes of the same SCC, i.e. the reachability of all tasks in a SCC is equal. When the nodes of each SCC are contracted to a single node, we get a directed acyclic graph. On that graph, the transitive reachability can be calculated in a single pass and is stored for each task separately. When creating the state for heuristic calculation, reachability is accumulated over all tasks in the current network.

6 Evaluation

Experiments ran on Xeon E5-2660 v3 CPUs with 2.60 GHz base frequency, 4 GB RAM and 10 minutes time. We included the following planning systems into the evaluation:

- $\text{SHOP2}_{\text{Alg.1}}$ and $\text{SHOP2}_{\text{Alg.2}}$ – We configured our planner to do a depth-first search with the original algorithm ($\text{SHOP2}_{\text{Alg.1}}$) and the modified one ($\text{SHOP2}_{\text{Alg.2}}$). This simulates the SHOP2 strategy (Nau et al. 2003).
- RC^{add}, RC^{FF}, and $\text{RC}^{\text{LM-Cut}}$ – Our new algorithm with the Add (Bonet and Geffner 2001), FF (Hoffmann and Nebel 2001), and LM-Cut (Helmert and Domshlak 2009) heuristics applied to our encoding.
- TDG_{M}, TDG_{M}-R, TDG_{C}, and TDG_{C}-R – The plan space search of the PANDA system with its most recent heuristics (Bercher et al. 2017).
- UMCP – A simulated UMCP algorithm that comes with the PANDA system.
- 2ADL – An approach by Alford et al. (2016a). The HTN problem is translated into a series of classical problems, the given time is the accumulated time until a plan was found. We used the configuration with the best performance in Alford et al. (2016a), the ADL-translation with the Jasper planner (Xie, Müller, and Holte 2014).

There are two remarks regarding the SHOP2 configurations: These strategies are uninformed and designed to be used with hand-tailored problems. If done properly, the resulting system will have a much better performance. The domain-*configurable* system is used on domains intended to be solved with domain-*independent* systems. It can be regarded a base-line. Second, to evaluate the impact of the modified algorithm, it is necessary to use a re-implementation. When evaluating against the original SHOP2, it would not be clear what causes performance differences (implementation or modified algorithm). However, we tested our re-implementation against it and found that $\text{SHOP2}_{\text{Alg.1}}$ outperforms the original implementation by 11 instances.

We included the domains used by Bercher et al. (2017): *UM-Translog* (22 instances), *SmartPhone* (7), *Satellite* (25), *Woodworking* (11) (described by Bercher, Keen, and Biundo (2014)). Because the best planners in this evaluation solve most instances, we added further, more challenging domains: The *Rover* domain (10) that we combined with instances from IPC-3, a *Transport* domain (30 instances); *Entertainment* (12), modeling the installation of a home entertainment system; and *PCP* (17), modeling Post's correspondence problem. With PCP, we included an undecidable problem that can not be modeled using STRIPS, but easily in HTN planning. However, we know that there exist solutions for the instances included in the evaluation.

We combined systems calculating a heuristic for standard search algorithms with Greedy (denoted G), A* and Greedy A* (GA*) search. Figure 3 shows the coverage of all systems. All systems performed best with Greedy A* search.

To investigate the impact of our modification, we compared the standard and improved progression algorithm using the simplest strategy, i.e. SHOP2. Coverage increased by 4 instances. The number of search nodes needed to solve a problem is depicted in Fig. 2 (be aware the log-scale of the figures). A dot stands for an instance solved by the systems and shows which system needed more search nodes. The modification decreases the number of explored nodes.

RC^{add}, RC^{FF}, and $\text{RC}^{\text{LM-Cut}}$ have the highest coverage. RC^{FF} and $\text{RC}^{\text{LM-Cut}}$ have a quite similar profile of solved instances. Figure 4 (left) shows the planning time against the number of solved instances. Due to readability, we only included the best configuration for each heuristic. Our three configurations perform best, followed by 2ADL and TDG_{M}. Using our improved progression algorithm, $\text{SHOP2}_{\text{Alg.2}}$ is level with TDG_{M}. Due to the fast preprocessing of the Fast Downward system underlying 2ADL, it outperforms our system in the first second, but is overtaken afterwards. Apart from this first second, our configurations are always on top of the field, i.e. they have not only the highest coverage, but are also the

	#probl.	2ADL	SHOP2-A1	SHOP2-A2	UMCP			TDG_C			TDG_M			TDG_C-R			TDG_M-R			RC^{add}			$RC^{LM\text{-}Cut}$			RC^{FF}		
					BF	DF	H	G	A*	GA*	G	A*	GA*	G	A*	GA*	G	A*	GA*	G	A*	GA*	G	A*	GA*	G	A*	GA*
UM-Translog	22	19	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**
Satellite	25	23	19	22	18	20	23	**25**	21	21	**25**	**25**	**25**	**25**	21	21	**25**	**25**	**25**	24	24	24	24	21	**25**	**25**	**25**	**25**
Entertainment	12	5	9	9	5	5	6	9	6	9	7	9	9	9	6	9	7	9	9	**12**	11	**12**	**12**	8	11	**12**	11	11
PCP	17	3	10	9	0	0	0	0	5	8	8	8	9	0	5	8	8	8	9	2	10	**12**	1	5	9	4	5	10
Woodworking	11	5	6	6	6	6	6	9	8	**10**	8	8	8	9	8	9	8	8	8	**10**	**10**	**10**	8	8	**10**	8	**10**	**10**
SmartPhone	7	**6**	5	5	4	4	4	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
Transport	30	**19**	1	2	1	0	0	0	2	1	0	2	1	0	2	1	0	1	1	1	6	8	2	3	10	2	3	13
Rover	10	5	3	4	0	0	0	2	1	2	2	2	2	2	1	2	2	2	2	4	4	**6**	**6**	0	4	3	0	3
total	134	85	75	79	56	57	61	72	70	78	77	81	81	72	70	77	77	80	81	80	92	**99**	80	72	96	81	81	**99**

Figure 3: First column contains the number of problem instances per domain, the following the solved instances for a system.

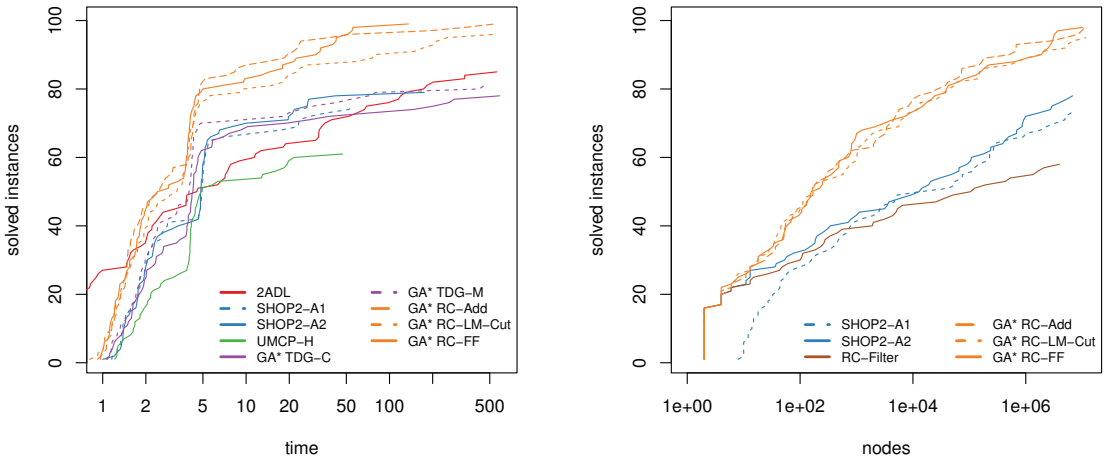


Figure 4: Left: planning time in seconds against number of solved instances (all systems with their best configuration). Right: search nodes against solved instances (all progression-based systems with their best configuration). Be aware the log-scale.

fastest in this evaluation.

Figure 4 (right) shows the number of search nodes against the number of solved instances for *state space*-based planners. This is interesting to investigate how informed the systems are. The modified algorithm ($\text{SHOP2}_{\text{Alg.2}}$) outperforms the standard one ($\text{SHOP2}_{\text{Alg.1}}$), needing less search nodes to find plans. RC^{add}, RC^{FF}, and $\text{RC}^{\text{LM-Cut}}$ are far more informed than $\text{SHOP2}_{\text{Alg.1}}$ and $\text{SHOP2}_{\text{Alg.2}}$. Compared to each other, they perform quite similar without a clear winner.

An interesting question is whether our encoding enables the heuristics to estimate a goal distance, or if it merely filters search nodes by detecting the goal as unreachable (dead ends). To investigate this, we included a heuristic (denoted $\text{RC}^{\text{filter}}$) that returns whether the goal is still reachable in a relaxed planning graph, i.e. it returns ∞ iff $\text{RC}^{\text{add}} = \infty$; or 0, else. It can be seen that the other configurations are far more informed, i.e. that our encoding enables the classical heuristics to extract goal distance information from it.

7 Conclusion

In this work we introduce a generic method to apply classical heuristics in HTN planning. We propose the use of progression-based search, which opens the door for state-based heuristics, because progression algorithms have a concrete state at hand (unlike plan space-based approaches where the order of the actions is not fixed). First, we introduced an improved progression algorithm that increases the performance by reducing the part of the search space searched multiple times. Second, we showed how (relaxed) information about the hierarchy can be incorporated into the non-hierarchical part of the problem. This enables the use of arbitrary state-based classical heuristics to estimate goal distance. Our approach works on standard (i.e. unchanged) HTN models. It further solves a major problem when using classical heuristics in HTN planning: the absence of a state-based goal. Our evaluation shows that the modified algorithm performs better than the original one, that standard state-based (non-hierarchical) heuristics can guide progression search for HTN problems, and that our system outperforms state-of-the-art HTN planning systems.

Acknowledgements

This work was done within the Transregional Collaborative Research Centre SFB/TRR 62 “Companion-Technology for Cognitive Technical Systems” funded by the German Research Foundation (DFG).

References

Alford, R.; Bercher, P.; and Aha, D. 2015. Tight bounds for HTN planning with task insertion. In *Proc. of the 25th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 1502–1508.

Alford, R.; Shivashankar, V.; Kuter, U.; and Nau, D. S. 2012. HTN problem spaces: Structure, algorithms, termination. In *Proc. of the 5th Annual Symposium on Combinatorial Search (SoCS)*, 2–9.

Alford, R.; Behnke, G.; Höller, D.; Bercher, P.; Biundo, S.; and Aha, D. W. 2016a. Bound to plan: Exploiting classical heuristics via automatic translations of tail-recursive HTN problems. In *Proc. of the 26th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 20–28.

Alford, R.; Shivashankar, V.; Roberts, M.; Frank, J.; and Aha, D. W. 2016b. Hierarchical planning: Relating task and goal decomposition with task sharing. In *Proc. of the 25th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 3022–3029.

Alford, R.; Kuter, U.; and Nau, D. S. 2009. Translating HTNs to PDDL: A small amount of domain knowledge can go a long way. In *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 1629–1634.

Bercher, P.; Höller, D.; Behnke, G.; and Biundo, S. 2016. More than a name? on implications of preconditions and effects of compound HTN planning tasks. In *Proc. of the 22nd European Conf. on Artificial Intelligence (ECAI)*, 225–233.

Bercher, P.; Behnke, G.; Höller, D.; and Biundo, S. 2017. An admissible HTN planning heuristic. In *Proc. of the 26th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 480–488.

Bercher, P.; Keen, S.; and Biundo, S. 2014. Hybrid planning heuristics based on task decomposition graphs. In *Proc. of the 7th Annual Symposium on Combinatorial Search (SoCS)*, 35–43.

Bit-Monnot, A.; Smith, D. E.; and Do, M. 2016. Delete-free reachability analysis for temporal and hierarchical planning. In *Proc. of the 22nd European Conf. on Artificial Intelligence (ECAI)*, 1698–1699.

Biundo, S.; Bercher, P.; Geier, T.; Müller, F.; and Schattenberg, B. 2011. Advanced user assistance based on AI planning. *Cognitive Systems Research* 12(3-4):219–236.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1-2):5–33.

Dvorak, F.; Bit-Monnot, A.; Ingrand, F.; and Ghallab, M. 2014. A flexible ANML actor and planner in robotics. In *Planning and Robotics (PlanRob) Workshop (ICAPS)*.

Elkawkagy, M.; Bercher, P.; Schattenberg, B.; and Biundo, S. 2012. Improving hierarchical planning performance by the use of landmarks. In *Proc. of the 26th AAAI Conf. on Artificial Intelligence (AAAI)*, 1763–1769.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence* 18(1):69–93.

Geier, T., and Bercher, P. 2011. On the decidability of HTN planning with task insertion. In *Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 1955–1961.

Gerevini, A.; Kuter, U.; Nau, D. S.; Saetti, A.; and Waisbrot, N. 2008. Combining domain-independent planning and HTN planning: The Duet planner. In *Proc. of the 18th European Conf. on Artificial Intelligence (ECAI)*, 573–577.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning – Theory and Practice*.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. of the 19th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 162–169.

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

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language classification of hierarchical planning problems. In *Proc. of the 21st European Conf. on Artificial Intelligence (ECAI)*, 447–452.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2016. Assessing the expressivity of planning formalisms through the comparison to formal languages. In *Proc. of the 26th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 158–165.

Lotem, A., and Nau, D. S. 2000. New advances in GraphHTN: Identifying independent subproblems in large HTN domains. In *Proc. of the 5th Int. Conf. on Artificial Intelligence Planning Systems (AIPS)*, 206–215.

Lotem, A.; Nau, D. S.; and Hendler, J. A. 1999. Using planning graphs for solving HTN planning problems. In *Proc. of the 16th Nat. Conf. on Artificial Intelligence and Eleventh Conf. on Innovative Applications of Artificial Intelligence (AAAI/IAAI)*, 534–540.

McDermott, D. V. 2000. The 1998 AI planning systems competition. *AI Magazine* 21(2):35–55.

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

Nau, D. S. 2007. Current trends in automated planning. *AI Magazine* 28(4):43–58.

Shivashankar, V.; Alford, R.; and Aha, D. W. 2017. Incorporating domain-independent planning heuristics in hierarchical planning. In *Proc. of the 31st AAAI Conf. on Artificial Intelligence (AAAI)*, 3658–3664.

Shivashankar, V.; Kuter, U.; Nau, D. S.; and Alford, R. 2012. A hierarchical goal-based formalism and algorithm for single-agent planning. In *Proc. of the Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, 981–988.

Shivashankar, V.; Alford, R.; Kuter, U.; and Nau, D. S. 2013. The GoDeL planning system: A more perfect union of domain-independent and hierarchical planning. In *Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2380–2386.

Shivashankar, V.; Alford, R.; Roberts, M.; and Aha, D. W. 2016. Cost-optimal algorithms for planning with procedural control knowledge. In *Proceedings of the 22nd European Conf. on Artificial Intelligence (ECAI 2016)*, 1702–1703.

Waisbrot, N.; Kuter, U.; and Könik, T. 2008. Combining heuristic search with hierarchical task-network planning: A preliminary report. In *Proc. of the 21st Int. Florida Artificial Intelligence Research Society Conf. (FLAIRS)*, 577–578.

Xie, F.; Müller, M.; and Holte, R. 2014. Jasper: The art of exploration in greedy best first search. In *Proc. of the 8th Int. Planning Competition*, 39–42.