An Admissible HTN Planning Heuristic

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

Abstract

Hierarchical task network (HTN) planning is well-known for being an efficient planning approach. This is mainly due to the success of the HTN planning system SHOP2. However, its performance depends on hand-designed search control knowledge. At the time being, there are only very few domain-independent heuristics, which are designed for differing hierarchical planning formalisms. Here, we propose an admissible heuristic for standard HTN planning, which allows to find optimal solutions heuristically. It bases upon the so-called task decomposition graph (TDG), a data structure reflecting reachable parts of the task hierarchy. We show (both in theory and empirically) that rebuilding it during planning can improve heuristic accuracy thereby decreasing the explored search space. The evaluation further studies the heuristic both in terms of plan quality and coverage.

1 Introduction

In contrast to classical planning, the goal in hierarchical planning is not to find a plan that satisfies a goal formula that holds in the state produced by executing the plan, but to find an executable refinement of an initial partial plan (a partially ordered set of primitive and/or abstract tasks). This initial plan is thus the "goal" – the set of tasks one wants to have achieved. Partial plans consist of two kinds of tasks: primitive and abstract ones. As it is the case in classical planning, only primitive ground tasks (so-called actions) are directly executable in a state, while abstract tasks need to be refined into more primitive ones until an executable primitive (solution) plan is obtained.

The motivation for choosing a hierarchical problem class as underlying framework is manifold, since the task hierarchy can be exploited in several ways. In case a user is integrated into the plan generation process, a hierarchical planning approach seems more natural, since it shows many similarities with the way in which humans solve their problems [Byrne, 1977; Fox, 1997; Marthi *et al.*, 2008]. For the same reason, plan explanations ("Why do I have to perform this action?") can make use of the hand-designed hierarchy to give intuitive explanations [Seegebarth *et al.*, 2012; Bercher *et al.*, 2014a]. The task hierarchy also allows to capture constraints restricting the set of plans that are regarded solutions, which cannot be expressed without the hierarchy [Höller *et al.*, 2014; 2016] (if no other expressive constructs such as functions are available). This higher expressivity of hierarchical planning problems comes at the cost of higher computational complexities. That is, deciding whether there is a solution is generally harder than in non-hierarchical planning – the precise complexity depends on structural properties of the task hierarchy and is at most undecidable [Erol *et al.*, 1994b; Alford *et al.*, 2015a; Bercher *et al.*, 2016] – and even deciding whether a partial plan is a solution is harder than in non-hierarchical planning [Behnke *et al.*, 2015; Bercher *et al.*, 2016]. Contrary to these seemingly negative results, the hierarchy can also be exploited to reduce the time required to find a solution. That is, instead of (or: in addition to) exploiting the hierarchy to pose additional constraints on solutions, it can be used to encode search control as exploited by the well-known SHOP2 system [Nau *et al.*, 2003].

Like in classical, non-hierarchical, planning, in some application domains one wants to find *optimal* solutions or solutions of a certain quality. Particularly in real-world applications this is of crucial importance, as plan quality often translates to costs in the real world, e.g., in terms of money, time, or resource consumption. Several hierarchical planning approaches have addressed the issue of finding optimal solutions [Lotem *et al.*, 1999; Marthi *et al.*, 2008; Sohrabi *et al.*, 2009; Shivashankar *et al.*, 2016; 2017]. To find such solutions, it suffices to use a suitable algorithm in combination with an admissible heuristic. We propose such an admissible heuristic for standard HTN planning.

Our heuristic is based upon the so-called task decomposition graph (TDG) – a representation of the AND/OR structure underlying the task hierarchy [Elkawkagy *et al.*, 2012]. It exploits the TDG to find a least-cost set of primitive actions into which the given abstract tasks can be decomposed. We show after which plan modifications during search rebuilding the TDG might lead to improved heuristic estimates. In the empirical evaluation, we show that the heuristic performs slightly worse than inadmissible ones when using A*, but finds plans of better quality. With Greedy-A*, it becomes one of the best configurations. The TDG-recomputation reduces the search space in almost every problem instance, but also increases the runtime for several instances.

2 Related Work

Independent of the purpose that hierarchical planning is deployed for, it is always desirable to solve the respective problems as fast as possible. This is especially important if the hierarchy is used for "physics" instead of "advice" (i.e., if the domain is modeled in a hierarchical manner, but it does not reflect search guidance). Several approaches were proposed to find solutions fast. Some of these approaches pursue the same basic approach than we do in this paper: using a standard search procedure (such as progression search) in combination with a domain-independent heuristic. Other approaches do not (only) rely on domain-independent heuristics, but instead the information encoded in the task hierarchy is exploited by the algorithm itself.

Marthi *et al.* [2008] propose several algorithms for their *angelic hierarchical planning* semantics. They are concerned with generating provably optimal "high-level plans" that may still contain abstract tasks. In their framework, abstract tasks are associated with preconditions and effects, the latter describing the set of states reachable by some refinement of the respective task. They further assume a totally ordered problem (making the problem decidable, because it prevents intertwining plans [Erol *et al.*, 1994b; Alford *et al.*, 2015a]). To find optimal plans, they deploy A* that exploits optimistic and pessimistic estimates of abstract tasks, which estimate the cost of their reachable refinements. They note: *"we will assume that the descriptions are given along with the hierarchy. However, we note that it is theoretically possible to derive them automatically from the structure of the hierarchy."* We believe that our admissible heuristic values can serve as their optimistic estimates that they rely on. Shivashankar *et al.* [2016; 2017] developed an admissible landmark-based heuristic for finding optimal solutions in *Hierarchical Goal Network (HGN) planning*, a formalism closely related to standard HTN planning (the relationship is investigated in detail by Alford *et al.* [2016b]). In standard HTN planning, decomposition methods specify into which task networks (a partially ordered set of tasks) an abstract task may be decomposed. In HGN planning, methods specify into which goal networks (a partially ordered multiset of goals) a goal may be decomposed. The proposed heuristic exploits the close relationship between goal networks and the (partially ordered) landmarks of a problem. Sohrabi *et al.* [2009] are concerned with finding high-quality plans for *HTN planning with preferences*. They propose a branch-and-bound algorithm that first finds some solution quickly (guided by inadmissible heuristics) and then tries to find better solutions by pruning plans that will lead to solutions of equal or worse quality (based on an admissible heuristic). Most heuristics are specific to both progression search (as they exploit that the current state is changing) and to their approach how the preferences are compiled away. One of their heuristics, lookahead metric function, is closer related to ours as it is an estimate of the metric of the best successor to the current partial plan. It does so by first calculating all refinements up to a certain depth. For each of them, a single primitive partial plan is computed based on Depth First search. The best metric serves as heuristic. The *MME heuristic* for hybrid planning [Bercher *et al.*, 2014b] is closely related to the one proposed here. We will detail the precise relationship later in the paper.

The planning system *Duet* [Gerevini *et al.*, 2008] combines the HTN planner SHOP2 [Nau *et al.*, 2003] with LPG [Gerevini *et al.*, 2003], a stochastic search planner for non-hierarchical problems. Duet is not directly concerned with calculating heuristics for partial plans, but it shows how hierarchical problems can be solved efficiently without relying on hand-coded search control. It is also notable that Duet is not concerned with solving standard HTN problems, since it allows to insert actions into partial plans (for achieving goals) that do not stem from task decomposition as allowed in TI-HTN planning (HTN planning with task insertion [Geier and Bercher, 2011; Alford *et al.*, 2015b]). The planner *GraphHTN* [Lotem *et al.*, 1999] solves standard HTN problems via combining the so-called *planning tree* with the planning graph. The planning tree is an AND/OR tree that represents all decompositions of the initial partial plan up to a certain depth. We rely on a similar graph representation. GraphHTN works by extending both the planning graph and the planning tree until a solution is found. It guarantees to find a solution with shortest makespan.

3 Problem Formalization

We rely on a problem formalization that extends standard *HTN planning* [Erol *et al.*, 1994b; Geier and Bercher, 2011] with concepts known from *partial order causal link (POCL) planning* [McAllester and Rosenblitt, 1991]. In accordance to previous work, we refer to the respective framework as *hybrid planning* [Biundo and Schattenberg, 2001; Bercher *et al.*, 2016]. Hybrid models extend standard HTN models in two directions: First, abstract tasks syntactically look like primitive ones, i.e., they also have preconditions and effects. That way, the planning model can be restricted to methods that adhere the semantics intended by the modeler [Bercher *et al.*, 2016]. Second, the model's methods can contain causal links that annotate which (abstract or primitive) task's precondition has been achieved by which other task of that method.

We want to emphasize that the the proposed heuristic is neither inherently making use of the preconditions and effects of abstract tasks nor of the causal links, both of which distinguish hybrid models from standard HTN models. As a consequence, the proposed heuristic can cope both with hybrid and with HTN models – i.e., it can be regarded both an HTN as well as a hybrid planning heuristic.

In hybrid planning, both primitive and abstract tasks are 3-tuples $\langle t(\bar{\tau}), pre(\bar{\tau}), \mathit{eff}(\bar{\tau})\rangle$ consisting of a parametrized name $t(\bar{\tau})$, a precondition $pre(\bar{\tau})$, and effects $\mathit{eff}(\bar{\tau})$ – the latter two are conjunctions of literals and depend on the task's parameter variables $\bar{\tau}$. We will often refer to a task by just mentioning its name $t(\bar{\tau})$. Partial plans are partially ordered sets of primitive and/or abstract tasks. A partial plan P is given by a tuple $\langle PS, \prec, CL, VC\rangle$ consisting of its plan steps PS, ordering constraints $\prec$, causal links CL, and variable constraints VC. A plan step $l : t(\bar{\tau}) \in PS$ is a uniquely labeled task. Labeling is required because P maintains a *partial* order of its plan steps, so multiple occurrences of the same task are differentiated relying on the labels. The set $\prec$ is a strict

partial order on PS. A causal link $ps \rightarrow_\varphi ps' \in CL$ represents that the precondition φ of plan step ps' is achieved by the plan step ps. The set VC is defined over the variables $\bar{\tau}$ of the tasks of PS. It can co- or non-co-designate variables with each other or with constants. With $Ground_{VC}(P)$ we denote the set of all groundings of P under consideration of the additional variable constraints VC.

Whereas primitive tasks have the same semantics as in classical planning, abstract tasks are abstractions of several primitive or abstract tasks. They can thus be regarded representations of high-level activities that need to be refined into more specific courses of action. This is accomplished by a set of so-called *(decomposition) methods*. A method is a tuple $m = (t(\bar{\tau}), P_m, VC)$ that maps an abstract task to its predefined partial plan P_m. VC_m denotes a set of variable constraints that relates the variables $\bar{\tau}$ of $t(\bar{\tau})$ with the variables of the partial plan P_m. In our formalism, methods are not associated with preconditions as it is the case in other hierarchical planning formalisms (as exploited, e.g., by SHOP2). That is, given a partial plan P contains a plan step $l : t(\bar{\tau}') \in PS$ and $\bar{\tau}$ and $\bar{\tau}'$ can be unified, then m is applicable to P. Applying m to P results in a successor plan P' in which $t(\bar{\tau})$ has been removed and replaced by P_m with the according variable constraints. Any of the causal links involving a precondition or effect of the decomposed abstract task is inherited down to suitable tasks in P_m. Adhering to legal methods ensures that this is always possible [Bercher *et al.*, 2016].

A hybrid planning domain $\mathcal{D} = \langle T_p, T_a, M \rangle$ contains the primitive and abstract tasks T_p and T_a, respectively, and a set of methods M. A hybrid planning problem $\mathcal{P} = \langle \mathcal{D}, P_{init}, C \rangle$ is then given by a domain $\mathcal{D}$, an initial partial plan P_{init}, and a set of constants C. The problem's initial state and, if given, its goal description, are encoded in P_{init} as additional actions as done in POCL planning.

A partial plan P is a solution (plan) to a hybrid planning problem, if and only if the following two criteria hold:

- P is a refinement of P_{init} with respect to the decomposition of abstract tasks and the insertion of ordering constraints, variable constraints, and causal links.
- P is executable in the initial state in the sense of the standard POCL solution criteria. That is, all tasks are primitive and ground, for each precondition φ of some plan step ps' there is a causal link $ps \rightarrow_\varphi ps'$ from a plan step ps in P, and there are no causal threats. A plan step ps'' threatens a causal link $ps \rightarrow_\varphi ps'$ if and only if it has some effect $\neg\psi$, such that ψ and φ can be unified and ps'' could be ordered between ps and ps'.

Note that we do not allow the insertion of tasks (cf. first solution criterion), except via decomposition of abstract tasks, in order to stick to the standard HTN solution criteria [Erol *et al.*, 1994b; Alford *et al.*, 2015a] as opposed to a relaxation thereof, called HTN planning with task insertion, TIHTN planning [Geier and Bercher, 2011; Alford *et al.*, 2015b].

4 On the Task Decomposition Graph

The so-called *task decomposition graph (TDG)* [Elkawkagy *et al.*, 2012] represents the AND/OR structure of the task hierarchy. Since it is a canonical representation of hierarchical problems, similar structures are used for various purposes (e.g., as the basis for HTN planning systems [Lotem *et al.*, 1999]). We use it to ground the domain model and to estimate the remaining effort of turning a given abstract task into a primitive plan. We first give a formal definition of TDGs, which is equivalent to the one given by Elkawkagy *et al.* [2012], but simplified and therefore more intuitive.

Definition 1 (Task Decomposition Graph (TDG)). *Let $\mathcal{P} = \langle \mathcal{D}, P_{init}, C \rangle$ be a hybrid planning problem with domain $\mathcal{D} = \langle T_p, T_a, M \rangle$. Without loss of generality, we assume that P_{init} contains just a single ground abstract task* TOP *for which there is exactly one method in M.*[1]

The bipartite graph $\mathcal{G} = \langle V_T, V_M, E_{T \rightarrow M}, E_{M \rightarrow T} \rangle$, consisting of a set of task vertices V_T, method vertices V_M, and edges $E_{T \rightarrow M}$ and $E_{M \rightarrow T}$ is called the TDG of $\mathcal{P}$ if it holds:

1. ***base case*** *(task vertex for the given task)*
   TOP $\in V_T$*, the TDG's root.*
2. ***method vertices*** *(derived from task vertices)*
   Let $v_t \in V_T$ with $v_t = t(\bar{c})$ and $(t(\bar{\tau}), P_m, VC_m) \in M$. Then, for all $v_m \in Ground_{VC_m \cup \{\bar{\tau} = \bar{c}\}}(P_m)$ holds:
   • $v_m \in V_M$ • $(v_t, v_m) \in E_{T \rightarrow M}$.
3. ***task vertices*** *(derived from method vertices)*
   Let $v_m \in V_M$ with $v_m = \langle PS, \prec, CL, VC \rangle$. Then, for all plan steps $l : t(\bar{c}) \in PS$ with $v_t = t(\bar{c})$, holds:
   • $v_t \in V_T$ • $(v_m, v_t) \in E_{M \rightarrow T}$.
4. ***tightness***
   $\mathcal{G}$ is minimal, such that 1. to 3. hold.

The definition works inductively by first requiring that the problem's initial ground task TOP[1] is part of the TDG as its root. The second criterion, one of the inductive steps, requires that for all methods that are applicable to any task vertex of the TDG, their respective ground partial plans are method vertices of the TDG. The third criterion, the second inductive step, requires any task in any of the TDG's method vertices to be a task vertex of the TDG. Finally, the last criterion ensures minimality of the graph, so that no vertexes or edges are in the TDG other than the ones demanded by the previous criteria. Graphical illustrations of example TDGs are provided by Lotem *et al.* [1999], Elkawkagy *et al.* [2012], and in Fig. 1.

Due to the undecidability of HTN planning [Erol *et al.*, 1994b; Geier and Bercher, 2011] and hybrid planning [Bercher *et al.*, 2016], there cannot always be a limit on the number of method applications to find a solution. However, since the TDG contains each decomposition method and task at most once, the TDG is always finite.

Def.1 incorporates *all* tasks that can be reached via decomposing the initial partial plan. Instead, we deploy the technique by Elkawkagy *et al.* [2010] that removes parts of the TDG which are unreachable when considering a delete-relaxed reachability analysis of the primitive tasks.

[1] If the problem specifies an initial partial plan P_{init} we can obtain the required form by adding a new artificial (parameter-free) abstract task TOP that decomposes exactly into P_{init}.

5 An Admissible Heuristic Based on the TDG

In case TDG-recomputation is not enabled, the proposed heuristic is a pre-processing heuristic, since it calculates the TDG only once before planning and assigns cost estimates to each of its vertices that also do not change. During search, these values are retrieved for a given search node. We assume that we have given a TDG and show how these cost estimates are calculated. To ensure termination, we assume that every primitive ground task $t(\bar{c})$ has a non-negative action cost $cost(t(\bar{c})) \in \mathbb{R}^+$ and that the TDG contains only abstract tasks that can be refined into a set of primitive tasks. Abstract tasks that do not fulfill the latter property can easily be identified in polynomial time by relying on a bottom-up reachability analysis (proof of Thm. 3.1 by Alford *et al.* [2014]).

We estimate the effort of refining an abstract task vertex by minimizing over the estimated effort of its method vertices. Analogously, the effort of refining a method vertex can be estimated by summing over the estimates of the tasks it contains. We do this for each task in the TDG. The semantics behind this calculation can be regarded as the cost of the least-expensive set of primitive tasks into which a given abstract task can be refined. This implies that these tasks must not necessarily form an executable plan, but they can all be made applicable using delete-relaxed primitive tasks, as they would not be in the TDG otherwise.

Definition 2 (TDG Cost Estimates)**.**
Let $\langle V_T, V_M, E_{T\to M}, E_{M\to T}\rangle$ be a TDG.

$$h_T(v_t) := \begin{cases} cost(v_t) & \text{if } v_t \text{ is primitive} \\ \min\limits_{(v_t,v_m)\in E_{T\to M}} h_M(v_m) & \text{else} \end{cases} \quad (1)$$

For a method vertex $v_m = \langle PS, \prec, CL, VC\rangle$, we set:

$$h_M(v_m) := \sum_{(v_m,v_t)\in E_{M\to T}} h_T(v_t) \quad (2)$$

We want to emphasize that each cost estimate of the TDG's vertices is finite even though the TDG itself might be cyclic. Intuitively spoken, it can never be optimal (or required) to run into a cycle for the sake of minimizing the estimates.

We use the following algorithm to compute the cost estimates. First, we identify all strongly connected components (SCCs) of the TDG, which is possible in polynomial time. All primitive tasks form an SCC on their own, since they do not have outgoing edges. They can be assigned their cost value according to h_T. We then sort all SCCs topologically, which is possible since their dependencies form a directed acyclic graph. We then process them in their topological (possibly partial) order, starting with the SCCs containing primitive tasks. For the remaining SCCs, we use an iterate-until-fixpoint procedure. We start by initializing the estimates for all vertices in an SCC with ∞. Then we iterate over these vertices and use the formulae for h_T and h_M to update the respective estimates. This iteration is repeated until no value in the SCC has changed, i.e., until the correct estimates according to Def. 2 have been computed. This fixpoint is reached after a polynomial number of steps, because in each iteration at least one vertex in the SCC is assigned its final value, which can be proven as follows. If the SCC contains one vertex, the claim is trivial. Otherwise, there are one or more vertices in it with outgoing edges. Consider a *method vertex* in an SCC: since it has been included, all other vertices are reachable from this one, i.e., at least one of its edges points to a vertex in the SCC. Thus, one of the outgoing edges of the SCC must originate from a task vertex (otherwise there were an infinite recursion and the SCC would have been pruned). Since the costs of a method vertex are additive with respect to its children, the method vertices will be at least as costly as the cheapest task vertex in the SCC. One of the outgoing edges of the SCC originating from a *task vertex* v_t will lead to a method vertex with minimal cost estimate c. As noted, every recursion must leave the SCC via a task vertex, and we have picked the cheapest task vertex, therefore this is the vertex with minimal costs (in the fixpoint). Since all estimates in the SCC must be greater (or equal) to c, the vertex v_t gets assigned c as its fixpoint cost in the first round of iterations. Since we have found the edge that determines the value of v_t, all other outgoing edges can be ignored from now on. Ignoring these edges, v_t falls out of the SCC. As such, the size of the SCC would decrease and, by induction, n iterations suffice to reach the fixpoint for an SCC with n vertices.

Def. 2 and the procedure above further forms an improvement of the so-called *minimal modification effort (MME)* heuristic, which is designed for hybrid planning systems that rely on POCL search techniques [Bercher *et al.*, 2014b]. MME is also based on the idea to exploit a TDG via minimizing over different methods and summing within the same method. But instead of action costs, it estimates the number of required decompositions and causal link insertions a hybrid planner needs to perform: in (1), primitive tasks are estimated by the number of their preconditions and abstract tasks by the minimum as given here plus 1 for performing a decomposition. In (2), we subtract $|CL|$ from the given sum to account for causal links that are already in a partial plan. But in contrast to Def. 2, MME relies on a visited list of abstract tasks that are taken into account when calculating the TDG estimates. This list ensures termination in the presence of cycles (cf. Def. 3 by Bercher *et al.* [2014b]), but it also produces smaller and less accurate estimates in these cases. We give the respective improved version of the MME heuristic later in this section.

During search, given a current partial plan and one of its abstract plan steps $l : t(\bar{\tau})$, we retrieve the estimate of one of its compatible groundings in the TDG, $comp(t(\bar{\tau}))$. When using the TDG cost estimates given in Def. 2, the resulting heuristic is called *cost-aware TDG heuristic*, *TDG$_c$*.

Definition 3 (TDG$_c$ Heuristic)**.**
Let $P = \langle PS, \prec, CL, VC\rangle$ be a partial plan. Then,

$$h_{TDG_c}(P) := \sum_{\substack{l:t(\bar{\tau})\in PS \\ t(\bar{\tau}) \text{ abstract}}} \left(\min_{v_t \in comp(t(\bar{\tau}))} h_T(v_t) \right)$$

TDG$_c$ basically relies upon the same mechanics as the definition of the method vertex estimate in Def. 2, since both inputs are partial plans. The heuristic for the costs of any missing actions can be estimated by summing over the heuristic estimates of its abstract tasks (which were pre-calculated,

cf. Def. 2). Since these tasks might still be lifted in the given partial plan, we minimize over the possible groundings.

Since Def. 2 minimizes for each abstract task over its available methods, and again Def. 3 minimizes over all possible groundings, the resulting heuristic is clearly admissible.

Prop 1. *The cost-aware TDG heuristic h_{TDG_c} is admissible with respect to action costs.*

Note that removing unreachable parts of the TDG [Elkawkagy *et al.*, 2010] does not influence admissibility. In fact, such pruning makes the heuristic more accurate, which means that it profits from any future research that is concerned with identifying and removing more unreachable parts of the TDG.

When using the modification-aware estimates described above, we obtain an improved variant of the MME heuristic, which we call *modification-aware TDG heuristic*, *TDG$_m$*.

Definition 4 (TDG$_\text{m}$ Heuristic)**.**
Let $P = \langle PS, \prec, CL, VC\rangle$ be a partial plan. Then,

$$h_{TDG_m}(P) := \sum_{l:t(\bar{\tau})\in PS} \left(\min_{v_t \in comp(t(\bar{\tau}))} h_T(v_t) \right) - |CL|$$

5.1 Recomputing the TDG During Planning

All TDG-based search strategies or heuristics developed so far are pre-processing heuristics [Elkawkagy *et al.*, 2012; Bercher *et al.*, 2014b]. That is, they rely on a TDG that is computed only *once* – prior search. However, decompositions that are performed during search influence the TDG and, consequently, the accuracy of any heuristic that is based upon it. Therefore, we show after which plan modifications the TDG could possibly change and after which it can not.

In order to understand in which situations a recomputation of the TDG can result in a changed TDG, we need to explain the TDG construction process. We do this only *very briefly* and refer to the paper by Elkawkagy *et al.* [2010] for any further details. Starting from the given initial partial plan, all primitive tasks (i.e., actions) that are reachable via decomposition are identified. Then, using only these actions, a relaxed reachability analysis is performed. Afterwards, the TDG is constructed and limited to partial plans in which all actions are reachable via the relaxed reachability analysis.

Applying methods implies that certain actions may not be reachable via decomposition anymore. The non-availability of these actions might even prove further actions unreachable via the relaxed reachability analysis. Thus, recomputing a TDG after decompositions can improve heuristic estimates. In the following, we only explain the technique for the proposed heuristic, i.e., we assume $h = h_{TDG_c}$ – but the recomputation can also be done for the TDG$_\text{m}$ heuristic.

Consider the example given in Fig. 1. When assuming $cost(p_3) = i$ and $h_M(P_{m_4}) = h_T(p_4) + h_T(a_3) = j > i$, we get $h_T(a_2) = i$. We now consider how the heuristic estimates with and without TDG-recomputation differ after the abstract task a_1 in P_{init} is decomposed. After decomposing a_1 we get two possible successor plans, P_1 and P_2. Since they both contain the same abstract task a_2, we get the same heuristic value $h(P_1) = h(P_2) = i$ without TDG-recomputation. When recomputing the TDG, we can exploit updated reachability information as follows. When assuming that the only

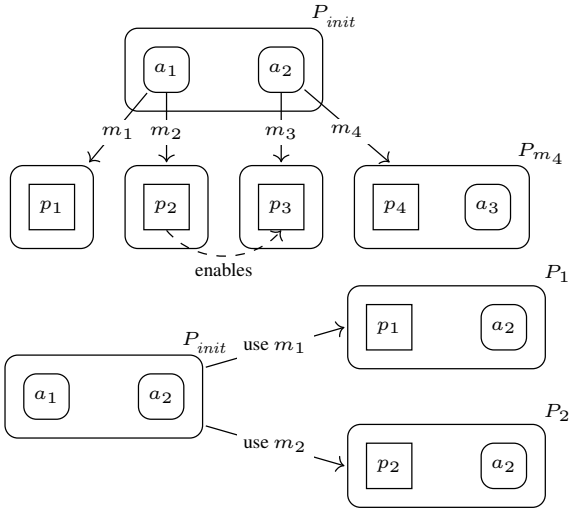

Figure 1: On top, we show a fragment of a TDG and at the bottom we show a fragment of a search space. Partial plans are denoted by surrounding boxes, abstract tasks by round boxes, and primitive tasks by square boxes.

action that enables the executability of p_3 is p_2 and further assume that p_2 cannot be reached via decomposing a_3, then the TDG for P_1 will not include the sub graph that is introduced via m_3. Thus, we get $h(P_1) = j$ and $h(P_2) = i$.

We can thus simply recompute the TDG after each decomposition. In lifted planning, any new variable binding might also influence the set of reachable tasks, but performing the reachability analysis in all these cases turned out to be too inefficient, so we assume a ground model. So, we can set $h(P') = h(P)$ for a partial plan P and its successor P' that is obtained by a modification other than a decomposition.

Even when a decomposition has been performed, there is a special case in which we can reuse the parent node's heuristic value. This is the case if an abstract task has just a single decomposition method in its initial TDG for all its groundings, since applying such a method does not influence the set of reachable actions. In previous work, we analyzed the structural properties of the planning benchmarks that we also use in this evaluation. We observed that the average branching factor of the TDG is often smaller than 2 [Bercher *et al.*, 2014b], which means that this special case does occur in practice. Further details about how often this occurs in our problem set is given in the empirical evaluation.

Let t be the decomposed task and $v_m = \langle PS_m, \prec_m, CL_m, VC_m\rangle$ its single method vertex in the initial TDG. Due to Def. 2, we get that $h_T(t) = h_M(v_m) = \sum_{l:t'\in PS_m} h_T(t')$. Since all primitive tasks that are introduced via decomposing t were previously accounted for by the heuristic for P, but are now part of P itself and therefore contribute to its cost, we get the new heuristic value $h(P') = h(P) - \sum_{l:t'\in PS_m, t' \text{primitive}} cost(t')$.

Note that this incremental heuristic computation, which calculates a heuristic $h(P')$ based on its parent's heuristic $h(P)$, also improves runtime of the TDG-based heuristics that do not perform TDG-recomputation.

6 Evaluation

This section introduces the benchmark set, the search strategies and heuristics, and presents the empirical results.

6.1 Domains

The empirical evaluation is done based on four different HTN planning domains. Originally, these domains are hybrid planning domains. We converted them into standard HTN domains by removing preconditions and effects from the abstract tasks and by further removing all causal links that are present in the methods' partial plans.

The domains include all the ones used by Elkawkagy *et al.* [2010; 2012] and Bercher *et al.* [2014b] for their evaluations, which were also used by Alford *et al.* [2016a]. These are the *UM-Translog* domain (originally designed specifically for HTN planning), the *SmartPhone* domain (originally designed for hybrid planning), and the *Satellite* and *Woodworking* domains (both were originally designed for the International Planning Competitions (IPCs), which were adapted to hybrid planning). For further information about these four domains, we refer to previous work [Bercher *et al.*, 2014b]. In total, we have 59 problem instances. Note that all domains use unit costs. Thus, for these domains, the proposed heuristic estimates the length of a shortest solution.

6.2 Experimental Setting

We evaluate the proposed TDG_c heuristic in terms of its heuristic guidance power (in form of coverage) and investigate whether its admissibility creates plans of better quality for the evaluated problem set.

We implemented all strategies within the same system to allow a fair comparison. We use the hybrid planner PANDA [Bercher *et al.*, 2014b, Alg. 1], which is capable of solving HTN and hybrid planning problems. Its code will be made available online (www.uni-ulm.de/en/in/ki/panda).

Search Strategies We evaluate the blind strategies Uniform cost, Breadth First (BF), and Depth First (DF) search.

We also compare our heuristic search approach with other search techniques (i.e., systems) from the literature. We have simulated the *UMCP* algorithm [Erol *et al.*, 1994a, Fig. 2] within the used system. It always chooses some partial plan according to a plan selection function based on BF, DF, or a "heuristic" that greedily selects a partial plan with a least number of abstract tasks. In case the selected partial plan is primitive, UMCP turns it into a solution or dismisses it altogether in case this is impossible. We also include a technique for solving HTN problems that is based on a translation into classical planning [Alford *et al.*, 2016a]. Since HTN planning is more expressive than classical planning, any such translation requires a bound (for Alford *et al.*'s translation the maximum size of a task network under progression). If the resulting problem is solvable so is the original one (and a solution can be extracted from the classical solution), but if not, there might still be a solution requiring a higher bound. Alford *et al.* [2016a] provided a mechanic to compute a lower bound on the progression bound and showed empirically that the smallest bound allowing for a solution is often near this lower bound. We have therefore started with the translation using Alford *et al.*'s lower bound and increased the bound by one if no solution was found. The translated classical problems are solved by existing PDDL planners. We use the best-performing (non-optimal) planner of the evaluation by Alford *et al.* [2016a], JASPER [Xie *et al.*, 2014]. In addition, we use the (optimal) SymBAStar*-2 planner [Torralba *et al.*, 2014], the winner of the optimal track of the IPC 2014. Since the compilation creates additional actions, action costs for the original primitive tasks are unaltered and the costs for the new ones are set to zero. Note that JASPER can handle an ADL model, whereas SymBAStar*-2 handles only STRIPS. The two configurations are referred to as Compile and $\text{Compile}_\text{opt}$, respectively. We have set the time limit for runs of the planner of each translated problem to two minutes (increasing this bound does not improve the results of the planner). The times reported are the sum of run times for all runs until a solution has been found. Please note that we did not include a comparison to SHOP2, because our models do not encode search guidance and, consequently, their decomposition methods do not specify preconditions on which the success of SHOP2 is based. For these models, SHOP2 would basically perform a blind DF progression search.

Additionally, we use both A* as well as Greedy-A*, where the heuristic is accounted for by factor 2, denoted by A^*_2. Because PANDA is a hybrid planner (i.e., it performs plan-space-based search relying on POCL concepts), we can also use the well-known POCL heuristics h_{add} and h^r_{add} [Younes and Simmons, 2003] as well as h_{relax} and h_{OC} [Nguyen and Kambhampati, 2001]. Since these heuristics are designed for (non-hierarchical) POCL problems, they only take the primitive tasks in a given partial plan into account, but make up for being less informed about the hierarchy by being extremely fast. In addition to the TDG_c heuristic, we evaluate our improved version of the hybrid planning heuristic MME [Bercher *et al.*, 2014b], TDG_m. We refer to the variants with recomputation as TDG_c-rec and TDG_m-rec, respectively.

For our experiments, we ground the models prior search and deploy the TDG construction and domain model reduction technique by Elkawkagy *et al.* [2010].

Hardware We used a machine with Xeon E5-2660 v3 CPUs with 2.60 GHz base frequency, a memory limit of 10 GB, and a time limit of 10 minutes (CPU time) per run.

6.3 Results

Coverage Results are given in Tab. 1. We can observe that already the uninformed, blind search strategies perform impressively well with solving between 47 and 53 problems out of 59. This can in part be attributed to our grounding procedure [Elkawkagy *et al.*, 2010], which reduces the model based on the given problem instances. Bercher *et al.* [2014b] showed that almost all resulting (grounded) problem instances become acyclic.

The UMCP strategies perform similarly to the uninformed ones. They solve between 47 and 52 problems. The two configurations based on the compilation technique are performing quite differently. The compilation in combination with

Table 1: Per domain and strategy, we present the number of solved problem instances (*#s*), the number of optimally solved instances (*#o*), and the maximal plan cost over all solved problem instances relative to the optimal solution (*cost*).

	Strategy	UM-Tr. (21 inst.) #s	#o	cost	SmartPh. (5 inst.) #s	#o	cost	Satellite (22 inst.) #s	#o	cost	Woodw. (11 inst.) #s	#o	cost	Summary (59 inst.) #s	#o	cost
blind	Uniform	21	21	1.00	4	4	1.00	17	17	1.00	8	8	1.00	50	50	**1.00**
	BF	21	21	1.00	4	4	1.00	15	15	1.00	7	7	1.00	47	47	**1.00**
	DF	21	21	1.00	5	1	1.60	19	7	2.09	8	4	1.44	53	33	2.09
systems	UMCP$_{BF}$	21	21	1.00	4	4	1.00	15	15	1.00	7	7	1.00	47	47	**1.00**
	UMCP$_{DF}$	21	21	1.00	4	1	1.60	17	6	2.09	6	4	1.29	48	32	2.09
	UMCP$_{h}$	21	21	1.00	5	4	1.40	19	11	1.50	7	7	1.00	52	43	1.50
	Compile	18	18	1.00	5	5	1.00	21	18	1.10	5	5	1.00	49	46	1.10
	Compile$_{opt}$	16	16	1.00	5	5	1.00	9	9	1.00	5	5	1.00	35	35	**1.00**
A*	ADD	21	21	1.00	4	1	1.20	21	21	1.00	10	9	1.17	56	52	1.20
	ADD-r	21	21	1.00	5	5	1.00	19	18	1.08	9	4	1.25	54	48	1.25
	Relax	21	21	1.00	5	5	1.00	18	18	1.00	10	8	1.17	54	52	1.17
	OC	21	21	1.00	4	4	1.00	21	21	1.00	10	7	1.17	56	53	1.17
	TDG$_m$/-rec	21	21	1.00	5	5	1.00	22	21	1.31	9	9	1.00	**57**	56	1.31
	TDG$_c$/-rec	21	21	1.00	5	5	1.00	18	18	1.00	8	8	1.00	52	52	**1.00**
A$^*_2$	ADD	21	21	1.00	4	0	1.20	21	20	1.09	10	9	1.17	56	50	1.20
	ADD-r	21	21	1.00	5	5	1.00	20	17	1.10	10	4	1.25	56	47	1.25
	Relax	21	21	1.00	5	5	1.00	18	15	1.10	10	4	1.25	54	45	1.25
	OC	21	21	1.00	4	4	1.00	22	21	1.09	10	7	1.22	**57**	53	1.22
	TDG$_m$/-rec	21	21	1.00	5	5	1.00	22	17	1.31	9	8	1.08	**57**	51	1.31
	TDG$_c$	21	21	1.00	5	5	1.00	20	20	1.00	10	10	1.00	56	56	**1.00**
	TDG$_c$-rec	21	21	1.00	5	5	1.00	20	20	1.00	11	11	1.00	**57**	**57**	**1.00**

JASPER performs similar to the previously discussed configurations: it solves 49 problems. The compilation that uses the optimal planner SymBAStar*-2 is performing worst with solving only 35 instances. This can be attributed to two facts: First, because it is an optimal planner, which are known to incur extra search effort to guarantee optimality. Second, the compilation used for SymBAStar*-2 relies on a basic STRIPS model, which is much larger than the ADL model that is used for JASPER [Alford *et al.*, 2016a].

Concerning the evaluated heuristics, we can make several observations. First, they all perform very good, solving between 54 and 57 problems. We did not expect that the four POCL heuristics perform so impressively well, since they are completely unaware of the hierarchy and ignore all abstract tasks (this would be different if we were using the original hybrid formulations, in which also abstract tasks use preconditions and effects). Similar to the good performance of the uninformed strategies, we assume that this can in part be attributed to our grounding procedure that eliminates much of the problems' difficulty. With A*, all heuristics except for TDG$_c$ perform similarly well. TDG$_c$ shows the lowest coverage solving only 52 problems, which we attribute to its admissibility. Best performing is the TDG$_m$ heuristic, which solves 57 problems. With A$^*_2$, TDG$_m$ remains being among the best configurations, but additionally the TDG$_c$ heuristic is among the best ones, which solves 56 problems when TDG-recomputation is disabled and 57 if it is enabled.

Plan Quality In Tab. 1, we give for each strategy and domain the number of problems that were solved optimal as well as the maximal solution cost relative to the optimal one. We assume that many of our problem instances only allow for solutions with the same number of actions, which makes the comparison of plan quality for these instances meaningless. We assume this being the case because for many instances, all strategies found solutions of the same quality. This is the case for all problems in UM-Translog, for 1 of 5 problems in SmartPhone, for 6 of 22 problems in Satellite, and for 2 of 11 problems in the Woodworking domain.

The overall worst-quality plans are produced by DF, which produces supoptimal solutions in 20 out of 53 cases. Their size is up to a factor of 2.09 as large as the optimal one. We can see that A* with the admissible TDG$_c$ heuristic is the least successful A* variant in terms of coverage, but guarantees to find optimal solutions. Other heuristics sometimes find solutions of suboptimal quality, ranging up to 31%. However, in total, non-optimal solutions are only found in a few cases (see Tab. 1, summary column). E.g., TDG$_m$, although being inadmissible, solves 56 of 57 problems optimal with A*. Using A$^*_2$, TDG$_c$/-rec belong to the best-performing heuristics – solving 56 and 57 problems, respectively, all of them optimal.

Table 2: Per domain, we present the number of recomputations divided by number of decompositions (*rec/dec*) and the number of improved heuristic estimates divided by number of recomputations (*h-im/rec*). For each of these values we report the minimum (*min*), maximum (*max*), and mean of means (μ).

Strategy	rec/dec min	max	μ	h-im/rec min	max	μ	rec/dec min	max	μ	h-im/rec min	max	μ
	UM-Translog						SmartPhone					
A* TDG$_m$	.027	.188	.086	0.00	.333	.032	.300	.691	.476	0.00	.117	.023
A* TDG$_c$	.027	.188	.086	0.00	.333	.032	.300	.647	.473	0.00	.041	.008
A$^*_2$ TDG$_m$	.027	.188	.086	0.00	.333	.032	.300	.713	.484	0.00	.121	.024
A$^*_2$ TDG$_c$	.027	.188	.086	0.00	.333	.032	.300	.647	.471	0.00	.041	.008
	Satellite						Woodworking					
A* TDG$_m$	.857	1.00	.956	.110	.608	.248	.294	.932	.581	0.00	.548	.246
A* TDG$_c$	.750	1.00	.913	.087	.592	.264	.294	.943	.600	0.00	.592	.330
A$^*_2$ TDG$_m$	.857	1.00	.953	.110	.617	.268	.294	.961	.611	0.00	.721	.306
A$^*_2$ TDG$_c$	.814	1.00	.934	.049	.609	.256	.294	.943	.615	0.00	.587	.333

TDG-Recomputation As indicated in Tab. 1, TDG$_m$ and TDG$_m$-rec produce exactly the same results in terms of coverage and plan quality for A* and A$^*_2$. The same holds for TDG$_c$ and TDG$_c$-rec for A*, but for A$^*_2$, TDG$_c$-rec solves one problem instance more than TDG$_c$. The impact of the recomputation can be summarized as follows: it increases heuristic accuracy resulting into more-informed heuristics (Tab. 2), which in turn results into smaller search spaces that sometimes come at the cost of increased computation times (Tab. 3) due to the overhead of recomputation.

Tab. 2 summarizes how often the TDG is recomputed and how often these recomputations are beneficial in terms of more accurate heuristic estimates. We report rec/dec, which gives the ratio of TDG-recomputations to performed decompositions. A ratio of 1 indicates that the TDG is rebuilt each time an abstract task is decomposed. We can see that in the UM-Translog domain, the recomputation is almost never performed, which goes back to the simplicity of the problem instances, where the TDG often has an average branching fac-

Table 3: For each domain, we summarize in how many problem instances the search space or time was deduced (<), unchanged (=), or increased (>), due to TDG recomputation.

Strategy	space			time			space			time			space			time			space			time		
	<	=	>	<	=	>	<	=	>	<	=	>	<	=	>	<	=	>	<	=	>	<	=	>
	UM-Translog						SmartPhone						Satellite						Woodworking					
A^* TDG_m	2	19	0	1	15	5	1	4	0	1	4	0	22	0	0	0	17	5	7	2	0	1	6	2
A^* TDG_c	2	19	0	0	13	8	1	4	0	0	4	1	18	0	0	0	10	8	6	2	0	4	3	1
A^*_2 TDG_m	2	19	0	3	16	2	1	4	0	0	4	1	22	0	0	0	13	9	4	5	0	1	6	2
A^*_2 TDG_c	2	19	0	4	11	6	1	4	0	1	4	0	20	0	0	0	11	9	5	5	0	4	3	3

tor near 1 [Bercher *et al.*, 2014b]. All problems are solved in less than 7 s by all strategies with only little deviation between them – so this domain does not provide much insights. In the SmartPhone and Woodworking domains, recomputation is done in approximately 50% and 60% of all decompositions. In Satellite, this percentage ranges up to 95.6%. We are also interested in the fact whether these recomputations are beneficial, i.e., whether the heuristic estimates based on a recomputed TDG are more accurate than compared to using the TDG of its parent. That value is given by h-im/rec, the ratio of improved heuristic estimates to performed recomputations. A ratio of 1 means that every recomputation produced an improved heuristic value. For UM-Translog, the mean is relatively small, but this is due to the fact that only in 2 of 21 instances the recomputation improves the heuristic (in both instances, the ratio is 33.3% for all strategies). Similarly for SmartPhone: Here, the reduction increases heuristic accuracy in only one problem instance (see table for its ratio). For the others, we see a mean heuristic improvement in about 25% to 33% of all cases. This tells us that further research should investigate the issue of predicting when a recomputation will result into smaller TDGs thereby increasing the ratio h-im/rec (i.e., by reducing the number of non-beneficial recomputations). Interestingly, in nearly all cases the heuristic was increased to ∞ if it was increased at all. This means that in our problem set, if any action that belongs to the set of least-cost reachable actions becomes unreachable in the TDG, then so does every action and the entire set becomes empty.

Tab. 3 summarizes how often the recomputation pays off in terms of search space and search time. We can see that the search space never becomes larger and that the impact of recomputation depends severely on the specific domain. In the UM-Translog and Smartphone domains the search space is reduced in only a few instances, which is non-surprising given the data reported above. In the respective SmartPhone instance, the search space is reduced by 35% (both A^* and A^*_2) for TDG_m-rec and by 26% (A^*) and 28% (A^*_2) for TDG_c-rec, whereas the runtimes are unaffected compared to the runs without recomputation. In the Satellite and Woodworking domains, search space reductions occur more frequently. Even though every search space gets reduced in Satellite, these reductions are usually quite small and do therefore not pay off in terms of search time. E.g., in this domain, the highest relative runtime increase is from 10 s to 20 s with a search space reduction from 49.754 to 47.048 search nodes (5.4%). In Woodworking, the reductions are usually larger. However, also in this domain, they are not always severe enough to pay off in terms of runtime. The worst result in terms of runtime, e.g., is given in a problem instance where TDG_m reduces the search space from 121.661 nodes to 96.824 (by 20%). Due to the overhead of recomputing the TDG, the runtime increases here from 32 s to 71 s (factor 2.22). In total, the reduction pays off in the majority of all instances in terms of runtime in this domain (and it allows to solve one additional problem, not reflected in Tab. 3). In one instance, search space was reduced from 86.935 to 15.213 (by 83%) coinciding with runtime reduction from 15 s to 8 s (factor 0.46) for TDG_m and for TDG_c from 2.360 to 158 (by 0.93%) coinciding with runtime reduction from 5 s to 3 s (factor 0.4).

7 Summary and Conclusion

We presented one of the first admissible heuristics for hierarchical planning. Our empirical evaluation reveals that, despite its admissibility, the heuristic shows good performance in terms of coverage when deployed with Greedy-A^*. With A^*, it performs slightly worse than other evaluated heuristics in terms of coverage, but it guarantees to find optimal solutions – whereas other strategies also produce suboptimal solutions. We further proposed a technique that improves the heuristic quality. The evaluation shows that it often reduces the explored search space, but often at the cost of increased solving time. The search space reductions vary significantly between the different problem instances. The highest achieved search space reduction is 93%, which coincides with a search time reduction of 40%.

Acknowledgements

We thank the reviewers for their comments and suggestions. We also want to thank Mario Schmautz for his support with the evaluation. 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 *et al.*, 2014] Ron Alford, Vikas Shivashankar, Ugur Kuter, and Dana Nau. On the feasibility of planning graph style heuristics for HTN planning. In *ICAPS*, pages 2–10. AAAI Press, 2014.

[Alford *et al.*, 2015a] Ron Alford, Pascal Bercher, and David Aha. Tight bounds for HTN planning. In *ICAPS*, pages 7–15. AAAI Press, 2015.

[Alford *et al.*, 2015b] Ron Alford, Pascal Bercher, and David Aha. Tight bounds for HTN planning with task insertion. In *IJCAI*, pages 1502–1508. AAAI Press, 2015.

[Alford *et al.*, 2016a] Ron Alford, Gregor Behnke, Daniel Höller, Pascal Bercher, Susanne Biundo, and David Aha. Bound to plan: Exploiting classical heuristics via automatic translations of tail-recursive HTN problems. In *ICAPS*, pages 20–28. AAAI Press, 2016.

[Alford *et al.*, 2016b] Ron Alford, Vikas Shivashankar, Mark Roberts, Jeremy Frank, and David W. Aha. Hierarchical planning: relating task and goal decomposition

with task sharing. In *IJCAI*, pages 3022–3029. AAAI Press, 2016.

[Behnke *et al.*, 2015] Gregor Behnke, Daniel Höller, and Susanne Biundo. On the complexity of HTN plan verification and its implications for plan recognition. In *ICAPS*, pages 25–33. AAAI Press, 2015.

[Bercher *et al.*, 2014a] Pascal Bercher, Susanne Biundo, Thomas Geier, Thilo Hörnle, Florian Nothdurft, Felix Richter, and Bernd Schattenberg. Plan, repair, execute, explain - how planning helps to assemble your home theater. In *ICAPS*, pages 386–394. AAAI Press, 2014.

[Bercher *et al.*, 2014b] Pascal Bercher, Shawn Keen, and Susanne Biundo. Hybrid planning heuristics based on task decomposition graphs. In *SoCS*, pages 35–43. AAAI Press, 2014.

[Bercher *et al.*, 2016] Pascal Bercher, Daniel Höller, Gregor Behnke, and Susanne Biundo. More than a name? On implications of preconditions and effects of compound HTN planning tasks. In *ECAI*, pages 225–233. IOS Press, 2016.

[Biundo and Schattenberg, 2001] Susanne Biundo and Bernd Schattenberg. From abstract crisis to concrete relief (a preliminary report on combining state abstraction and HTN planning). In *ECP*, pages 157–168. AAAI Press, 2001.

[Byrne, 1977] Richard Byrne. Planning meals: Problem solving on a real data-base. *Cognition*, 5:287–332, 1977.

[Elkawkagy *et al.*, 2010] Mohamed Elkawkagy, Bernd Schattenberg, and Susanne Biundo. Landmarks in hierarchical planning. In *ECAI*, pages 229–234. IOS Press, 2010.

[Elkawkagy *et al.*, 2012] Mohamed Elkawkagy, Pascal Bercher, Bernd Schattenberg, and Susanne Biundo. Improving hierarchical planning performance by the use of landmarks. In *AAAI*, pages 1763–1769. AAAI Press, 2012.

[Erol *et al.*, 1994a] Kutluhan Erol, James Hendler, and Dana S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *AIPS*, pages 249–254. AAAI Press, 1994.

[Erol *et al.*, 1994b] Kutluhan Erol, James A. Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. In *AAAI*, pages 1123–1128. AAAI Press, 1994.

[Fox, 1997] Maria Fox. Natural hierarchical planning using operator decomposition. In *ECP*, pages 195–207. Springer, 1997.

[Geier and Bercher, 2011] Thomas Geier and Pascal Bercher. On the decidability of HTN planning with task insertion. In *IJCAI*, pages 1955–1961. AAAI Press, 2011.

[Gerevini *et al.*, 2003] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. Planning through stochastic local search and temporal action graphs. *JAIR*, 20:239–290, 2003.

[Gerevini *et al.*, 2008] Alfonso Gerevini, Ugur Kuter, Dana S. Nau, Alessandro Saetti, and Nathaniel Waisbrot. Combining domain-independent planning and HTN planning: The Duet planner. In *ECAI*, pages 573–577. IOS Press, 2008.

[Höller *et al.*, 2014] Daniel Höller, Gregor Behnke, Pascal Bercher, and Susanne Biundo. Language classification of hierarchical planning problems. In *ECAI*, pages 447–452. IOS Press, 2014.

[Höller *et al.*, 2016] Daniel Höller, Gregor Behnke, Pascal Bercher, and Susanne Biundo. Assessing the expressivity of planning formalisms through the comparison to formal languages. In *ICAPS*, pages 158–165. AAAI Press, 2016.

[Lotem *et al.*, 1999] Amnon Lotem, Dana S. Nau, and James A. Hendler. Using planning graphs for solving HTN planning problems. In *AAAI*, pages 534–540. AAAI Press, 1999.

[Marthi *et al.*, 2008] Bhaskara Marthi, Stuart Russell, and Jason Wolfe. Angelic hierarchical planning: Optimal and online algorithms. In *ICAPS*, pages 222–231. AAAI Press, 2008.

[McAllester and Rosenblitt, 1991] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *AAAI*, pages 634–639. AAAI Press, 1991.

[Nau *et al.*, 2003] Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *JAIR*, 20:379–404, 2003.

[Nguyen and Kambhampati, 2001] XuanLong Nguyen and Subbarao Kambhampati. Reviving partial order planning. In *IJCAI*, pages 459–466. Morgan Kaufmann, 2001.

[Seegebarth *et al.*, 2012] Bastian Seegebarth, Felix Müller, Bernd Schattenberg, and Susanne Biundo. Making hybrid plans more clear to human users – a formal approach for generating sound explanations. In *ICAPS*, pages 225–233. AAAI Press, 2012.

[Shivashankar *et al.*, 2016] Vikas Shivashankar, Ron Alford, Mark Roberts, and David W. Aha. Cost-optimal algorithms for planning with procedural control knowledge. In *ECAI*, pages 1702–1703. IOS Press, 2016.

[Shivashankar *et al.*, 2017] Vikas Shivashankar, Ron Alford, and David Aha. Incorporating domain-independent planning heuristics in hierarchical planning. In *AAAI*, pages 3658–3664. AAAI Press, 2017.

[Sohrabi *et al.*, 2009] Shirin Sohrabi, Jorge A. Baier, and Sheila A. McIlraith. HTN planning with preferences. In *IJCAI*, pages 1790–1797. AAAI Press, 2009.

[Torralba *et al.*, 2014] Álvaro Torralba, Vidal Alcázar, Daniel Borrajo, Peter Kissmann, and Stefan Edelkamp. SymBA*: A symbolic bidirectional A* planner. In *The 8th IPC*, pages 105–108, 2014.

[Xie *et al.*, 2014] Fan Xie, Martin Müller, and Robert Holte. Jasper: The art of exploration in greedy best first search. In *The 8th IPC*, pages 39–42, 2014.

[Younes and Simmons, 2003] Håkan L. S. Younes and Reid G. Simmons. VHPOP: Versatile heuristic partial order planner. *JAIR*, 20:405–430, 2003.