Delete- and Ordering-Relaxation Heuristics for HTN Planning

Daniel Höller[1,4], **Pascal Bercher**[2,4] and **Gregor Behnke**[3,4]
[1]Saarland University, Saarland Informatics Campus
[2]The Australian National University, College of Engineering and Computer Science
[3]University of Freiburg
[4]Ulm University, Institute of Artificial Intelligence
hoeller@cs.uni-saarland.de, pascal.bercher@anu.edu.au, behnkeg@cs.uni-freiburg.de

Abstract

In HTN planning, the hierarchy has a wide impact on solutions. First, there is (usually) no state-based goal given, the objective is given via the hierarchy. Second, it enforces actions to be in a plan. Third, planners are not allowed to add actions apart from those introduced via decomposition, i.e. via the hierarchy. However, no heuristic considers the interplay of hierarchy and actions in the plan exactly (without relaxation) because this makes heuristic calculation NP-hard even under delete relaxation. We introduce the problem class of delete- and ordering-free HTN planning as basis for novel HTN heuristics and show that its plan existence problem is still NP-complete. We then introduce heuristics based on the new class using an integer programming model to solve it.

1 Introduction

Planning is the task of finding a course of action that transforms a given state of the world into one where certain desirable conditions hold. This allows systems to act goal-directed and in a situation-adaptive way. It is commonly solved based on a model of the environment and how it can be changed. The maybe most widely used approaches are *classical* planning and *hierarchical task network* (HTN) planning.

In classical planning, the environment is described via propositional state variables. Actions include state features that need to hold for the action to be applicable, its *preconditions*, and state features added and removed by the action, its *effects*. The objective is to fulfill some state features.

HTN planning incorporates a grammar-like decomposition structure: *abstract tasks* cannot be executed directly and are decomposed into other tasks until only actions are left. These are defined in the same way as in classical planning. The hierarchy enables the description of complex behavior [Höller *et al.*, 2014; Höller *et al.*, 2016] like the grammar intersection problem of context-free languages [Erol *et al.*, 1996]. Usually there is no state-based goal given – the objective is to execute the tasks resulting from decomposition.

The hierarchy has a wide impact on the set of solutions: (1) The objective is defined via the hierarchy, (2) the planner needs to integrate the tasks resulting from it in an executable sequence, and (3) is not allowed to add other tasks, i.e., it restricts which tasks are included in a solution.

Heuristic search is a common technique to solve HTN planning problems.[1] However, the interplay of state-transition and hierarchy makes the design of heuristics difficult. So far, there is no heuristic in the literature that captures it exactly (without relaxation). This is NP-hard even under delete-relaxation [Alford *et al.*, 2014]. A common way to enable a calculation in P is to additionally allow for task insertion [Alford *et al.*, 2014], i.e. heuristic functions may insert additional actions apart from those that have been introduced via decomposition while computing their relaxed plan (done e.g. by Bercher *et al.* [2017], or Höller *et al.* [2018]). This, however, can severely relax the underlying problem.

Towards the aim of taking the interplay of state-transition and decomposition better into account, we (1) introduce the class of *delete- and ordering-free (DOF) HTN planning* and show that its plan existence problem is NP-complete, thereby generalizing a result from the literature [Alford *et al.*, 2014]. Based on an integer programming (IP) model for classical planning [Imai and Fukunaga, 2015], we (2) introduce a model to solve DOF problems and present several heuristics based on it.[2] Calculation based on an IP model provides a straightforward way to approximate the result in P by using the relaxation to a linear programming (LP) model. We show that our heuristics are competitive with state-of-the-art heuristics in terms of coverage, but much more informed.

2 Formal Framework

Throughout the paper we use the HTN formalism of Geier and Bercher [2011] that is introduced in this section.

Let C be the set of abstract (also called compound) tasks, A the set of actions, and $N = A \cup C$ the set of all tasks. Tasks are maintained in *task networks*. A task network is a triple $tn = (T, \prec, \alpha)$, where T is a set of unique task identifiers (ids) that are mapped to N by the mapping function $\alpha : T \to N$. That way, a single task can be contained more than once. $\prec$ defines a partial ordering on the task ids.

[1]Heuristic search is also successful in Hierarchical Goal Network (HGN) planning [Shivashankar *et al.*, 2017], which differs severely from HTN planning in that hierarchies are defined among goals (facts) rather than tasks (cf. overview by Bercher *et al.* [2019]).

[2]Source code is available at *panda.hierarchical-task.net*

Abstract tasks are decomposed by using *(decomposition) methods*. A method m is a pair (c, tn) of a compound task $c \in C$ and a task network tn. The task c defines the abstract task the method is applicable to, and tn the tasks c can be decomposed into, the method's *subtasks*. The set of methods is denoted M. Formally, a task network $tn_1 = (T_1, \prec_1, \alpha_1)$ is decomposed into a task network $tn_2 = (T_2, \prec_2, \alpha_2)$ by a method (c, tn) if and only if there is a task $t \in T_1$ with $\alpha_1(t) = c$ and a task network $tn' = (T', \prec', \alpha')$ equal to tn but with ids not contained in the decomposed network (i.e. $T_1 \cap T' = \emptyset$); then, tn_2 is defined as follows:

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

Let L be a set of propositional state features. A state s is represented by the subset of features that hold in it, i.e., $s \in 2^L$. Preconditions, add and delete effects are defined by the functions $prec$, add, and del with $\{prec, add, del\} : A \to 2^L$. An action a is applicable (also called executable) in a state s if and only if $s \supseteq prec(a)$. When a is applicable in a state s, the state s' resulting from applying a is defined as $s' = (s \setminus del(a)) \cup add(a)$. Applicability and state transition of action sequences are defined accordingly. $g \subseteq L$ is the goal definition. A state s is a goal state if and only if $s \supseteq g$. The planning process starts with the initial network tn_I. An HTN planning problem p is defined as a tuple $p = (L, C, A, M, s_0, tn_I, g, \delta)$ with $\delta = (prec, add, del)$. $tn' = (T', \prec', \alpha')$ is a solution to p if and only if: (1) tn' can be obtained from tn_I by applying a sequence of decompositions, (2) all task ids in T' are (mapped to) actions, and (3) there is a sequence $\langle t_1 t_2 \dots t_n \rangle$ of all ids in T' in line with $\prec'$ such that $\langle \alpha'(t_1)\alpha'(t_2) \dots \alpha'(t_n) \rangle$ is applicable in s_0 and results in a goal state.

In the following we need to show that a task network tn can be reached by decomposing tn_I. This is the case if and only if there is a (valid) Decomposition Tree encoding the methods leading from tn_I to tn [Geier and Bercher, 2011].

Definition 1 (Decomposition Tree)**.** *Given an HTN problem, a Valid Decomposition Tree (DT) is a tuple $g = (T, E, \prec, \alpha, \beta)$, where (T, E) is a directed tree, and $\prec$ a strict partial order on the nodes. Nodes are labeled with task names by the function $\alpha : T \to N$. Those labeled with abstract tasks are further labeled with methods by $\beta : T \to M$.*

The tree's root is labeled with the initial task[3] *and for any node t labeled with an abstract task c the following holds:*

1. *It is labeled with a method (c, tn_m)*
2. *Let $ch(g, t)$ be the children of t in g. The task network induced by $ch(g, t)$ differs from tn_m only in the task ids.*
3. *For all $t' \in T$ and for all $t'' \in ch(g, t)$, it holds that*
   (a) if $(t, t') \in \prec$ then $(t'', t') \in \prec$
   (b) if $(t', t) \in \prec$ then $(t', t'') \in \prec$
4. *$\prec$ contains only orderings introduced by 2 and 3.*

In the remaining paper we deal with HTN problems not including ordering relations, so 3 and 4 are not needed.

3 Delete & Ordering-Free HTN Planning

We first introduce a new subclass of HTN planning called *delete- and ordering-free* HTN planning where

- the set of delete-effects of all actions is empty and
- the set of ordering relations of all task networks defined in methods as well as in the initial task network is empty.

Our intention is not to use this class to model problems, but to use it to create heuristics by relaxing the HTN planning problem induced by a given search node into a DOF HTN planning problem. Such heuristics are defined in Section 4.

Definition 2 (Delete- and Ordering-free HTN Planning Problems)**.** *An HTN problem $p = (L, C, A, M, s_0, tn_I, g, \delta)$ with $tn_I = (T_I, \prec_I, \alpha_I)$ and $\delta = (prec, add, del)$ is called delete- and ordering-free (DOF) if and only if $\prec_I = \emptyset$, $\forall (c, (T, \prec, \alpha)) \in M : \prec = \emptyset$, and $\forall a \in A : del(a) = \emptyset$.*

The plan existence problem in delete-relaxed HTN planning is NP-hard [Alford *et al.*, 2014]. The problem class we introduced further does not allow to include ordering relations into the HTN model, so the question is whether this result still holds. The following theorem answers this question.

Theorem 1. *The plan existence problem in DOF HTN planning is NP-hard.*

Proof Sketch. We construct an DOF HTN problem that has a solution if and only if a given 3-SAT problem has a solution. For every variable in the SAT formula, we define (1) two state features indicating it to be *true* or *false*; (2) two actions setting either the first or the second state feature and (3) a single abstract task that has two methods, decomposing it into one of the two actions. Let C_{set} be the set of these abstract tasks.

For every clause of the SAT formula we introduce an abstract task with three methods decomposing it into distinct actions, each having a precondition checking that one term of the clause is fulfilled. Let C_{check} be these abstract tasks.

The planning process is started with one instance of each task from C_{set} and C_{check} in tn_I. Each variable is set once, so we get an assignment. When an executable plan is found, at least one term of each clause holds. □

Alford *et al.* have shown NP membership for delete-relaxed HTN planning [2014, Thm. 5.11], so for DOF HTN planning it follows directly and the following theorem holds:

Corollary 1. *The plan existence problem in DOF HTN planning is NP-complete.*

4 Delete- & Ordering-Relaxation Heuristics

For heuristic calculation we relax the HTN planning problem induced by a search node into a DOF planning problem, calculate the goal distance in the DOF problem, and use it as

[3] Be aware of the equivalence of HTN problems with initial *task* and problems with initial *task network* (as used here). Given a problem with initial task network tn_I, we can introduce a new task c_I and a new method (c_I, tn_I) to compile it into a problem with initial task. I.e. we can use this definition also for our formalism.

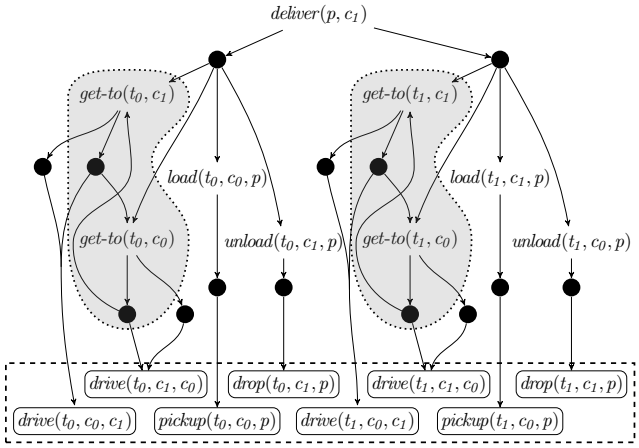

Figure 1: Black dots indicate method nodes, text without frame indicates abstract nodes, and text with frame primitive task nodes.

heuristic value in the original problem. Analog to the decision problem defined before, we define the following distance as the delete- and ordering-relaxation (h^{dor}) heuristic. We will introduce several variations later on in the evaluation section.

Definition 3 (The h^{dor} Heuristic)**.** *The delete- and ordering-relaxation (h^{dor}) heuristic for a search node n is the minimal goal distance for the planning problem p induced by n when setting all actions' delete effects to $\emptyset$ and all ordering relations of tn_I and all methods to $\emptyset$.*

A DT contains one node per action in a solution and one node per abstract task (that needs to be decomposed to reach a solution), i.e. the goal distance is equivalent to the number of nodes in a DT. The DTs of solutions to the original problem are still valid for the DOF problem, i.e. we cannot overestimate the nodes of a tree (i.e. the goal distance):

Corollary 2 (Properties of h^{dor})**.** *The h^{dor} heuristic is admissible (in the goal distance), goal aware, and safe.*

To calculate our heuristics efficiently, we decided to use IP solvers for the calculation. Before we come to our IP model, we want to give an intuition about the constraint system.

A data structure that has often been used to calculate heuristics in HTN planning is the Task Decomposition Graph (TDG) [Bercher *et al.*, 2017]. It compactly represents the decomposition hierarchy of the problem.

Definition 4 (Task Decomposition Graph)**.** *Let $G = (V, E)$ be a directed graph. Given an HTN planning problem $p = (L, C, A, M, s_0, tn_I, g, \delta)$, we define $V = C \cup A \cup M$[4]. Let n be an element of N and m of M. E contains the edge (n, m) if and only if $m = (n, tn)$, i.e., when m decomposes the task n. E contains the edge (m, n) if and only if $m = (c, (T, \prec, \alpha))$ and there is a task id $t \in T$ with $\alpha(t) = n$, i.e., when the task n is contained in the subtasks of m.*

An example is given in Fig. 1. It captures the decomposition structure of a simple transport problem with two cities c_0 and c_1; two trucks t_0 and t_1; and a single package p that is located at c_0 and shall be delivered to c_1. The deliver task at the top can be fulfilled either by using transporter t_0 or t_1.

A TDG is an AND/OR graph. The initial task given at the top needs to be decomposed either by the left or by the right method, i.e., this node is an OR node as the planner needs to select one option. Method nodes represent AND nodes as all its children have to be included in a solution. From a planner's perspective, a TDG could be used directly for planning by starting at the top, assigning the OR decisions, and generating the decomposition *tree*, i.e. the witness for a solution[5]. Every DT can be built that way. The structure of the constraints and their relationship in our IP model is similar to a TDG. They capture the relationship of the number of abstract tasks, methods, and actions in a solution (i.e., in a valid DT).

4.1 Task Decomposition Constraints

We first introduce constraints describing the decomposition hierarchy and then combine it with the model of Imai and Fukunaga [2015] for classical planning that encodes a delete-relaxed planning graph (making sure that there is an applicable sequence of the actions). I.e. the constraints C1-C6 are from related work[6] and adapted to our needs, while C7-C13 are contributions of this paper.

The definition is quite simple for acyclic TDGs, but becomes more complicated for cyclic ones. First we calculate the TDG's strongly connected components (SCCs).

Definition 5 (TDG SCCs)**.** *We define the set of SCCs of the given TDG as $SCC = \{scc_0, scc_1, \ldots, scc_n\}$ with $scc_i \cap scc_j = \emptyset$ for all $0 \leq i, j \leq n$ and $i \neq j$.*

Each scc_i contains nodes from the TDG, i.e. tasks and methods from the problem. In the following definitions we are not interested in the method nodes. Therefore we define $SCC^{>1} = \{scc \cap N \mid scc \in SCC, |scc| > 1\}$. For each cyclic SCC, i.e. containing more than one node, it includes a set containing the tasks in the SCC.

We now introduce our IP model. Its variable set is defined in Fig. 2. Our objective function minimizes the goal distance, i.e. the number of applied actions (i.e. the sum over all variables UA_a belonging to actions a) and applied methods (i.e. the sum over all variables M_m belonging to methods m).

$$min \sum_{a \in A} UA_a + \sum_{m \in M} M_m \qquad \text{(O)}$$

Next we introduce the constraints belonging to the decomposition structure. We need the following function.

Definition 6 (mst)**.** *Let $mst(n)$ be the multiset[7] of methods where the task n is contained as a subtask. A method $m \in M$ is as often in $mst(n)$ as n is a subtask of m.*

A certain instance of a task is in a DT if and only if it is contained in tn_I or in the subtasks of an applied method.

$$\forall n \in N : UA_n = TNI_n + \sum_{m \in mst(n)} M_m \qquad \text{(C7)}$$

[4]WLOG, we assume that the three sets are disjunct.

[5]One would have to consider some special cases we omitted.

[6]We start with constraint C7 to leave the labeling of Imai and Fukunaga [2015] unchanged.

[7]Defining mst as multiset is necessary to be correct when a task is contained in the subtasks of a method more than once.

- $\{UF_f \mid f \in L\}$ (bool) – flag indicating whether a state feature is eventually set or not.
- $\{UA_n \mid n \in N\}$ (int) – value indicating how often a certain primitive or abstract task is in the solution.
- $\{E_{a,f} \mid a \in A, f \in add(a)\}$ (bool) – flag indicating if the given action is the first achiever of the state feature f.
- $\{TP_f \mid f \in L\}$ (int) – time step when a given state feature is made true.
- $\{TA_a \mid a \in A\}$ (int) – time step when an action is added to the relaxed planning graph.
- $\{M_m \mid m \in M\}$ (int) – value indicating how often a certain method is in the solution.
- $\{TNI_n \mid n \in N\}$ (int) – value indicating how often a certain task is in the initial task network.
- $\{R_c^s \mid \forall scc \in SCC^{>1} : c \in scc, s \in \{0 \dots |scc|\}\}$,
- $\{I^s_{c' \to c} \mid \forall scc \in SCC^{>1} : c, c' \in scc, s \in \{1 \dots |scc|\}\}$ both (bool) – reachability bits; the former indicating which tasks inside an SCC are reachable by a directed path of included tasks/methods from outside the SCC or from tn_I, the latter whether a method from c' to c is set.

Figure 2: Variable set of our IP model. The variables on the left are the same as those used by Imai and Fukunaga. We had to adapt the types of some of them as will be discussed later in this section. On the right are the newly introduced variables.

Each abstract task included in a DT needs to be decomposed. Therefore the number of all methods that might decompose it is equal to the number of instances of that task.

Definition 7 ($mdec$)**.** *Let $mdec(c)$ be the set of methods decomposing the abstract task c.*

$$\forall c \in C : UA_c = \sum_{m \in mdec(c)} M_m \tag{C8}$$

For acyclic problems, these constraints specify the relationship between tasks and methods in a DT. However, consider the TDG given in Fig. 1 containing cycles (marked in gray). Here, it would be possible for the IP solver to introduce tasks (by building a self-sustaining circle) that form a component in the DT that is disconnected from the initial tasks.

To prevent this, we encode the reachability of tasks in an SCC in a structure that can be seen as an instance of the Floyd-Warshall algorithm. For each task c we introduce a new variable R_c^0 that is set if and only if the node is reached from the outside, i.e., by a method decomposing a task not included in the same SCC.

$$\forall scc \in SCC^{>1} : \forall c \in scc : R_c^0 \leq \sum_{m \in mst(c), \text{ s.t. } m \notin scc} M_m + TNI_c \tag{C9}$$

C10-C13 ensure that methods and tasks inside the SCC that are set are connected to tasks reached from outside or to tasks in tn_I. The encoding is cubic in the number of tasks in the SCC. s indicates a time step. A task c' might be reached in time step s with $s > 0$ if and only if it was set in $s-1$ (C10) or if there is a task c'' set in $s-1$ (C10, C11) and a method that is set in $s-1$ that decomposes c'' into c' (C12). C13 connects C9-C12 to the other constraints (c_l is a large constant).

Definition 8 (mto)**.** *Let $mto(c \to n)$ be the methods decomposing the task c and where n is included in the subtasks.*

$$\forall scc \in SCC^{>1} : \forall c \in scc : \forall s \in \{1 \dots |scc|\} :$$

$$R_c^s \leq \sum_{c' \in scc} I^s_{c' \to c} + R_c^{s-1} \tag{C10}$$

$$I^s_{c' \to c} \leq R_{c'}^{s-1} \tag{C11}$$

$$I^s_{c' \to c} \leq \sum_{m \in mto(c' \to c)} M_m \tag{C12}$$

$$UA_c \leq c_l \times R_c^{|scc|} \tag{C13}$$

4.2 Planning Graph Constraints

To ensure (delete-relaxed) executability of the resulting actions, we use the encoding of Imai and Fukunaga [2015] with minor adaptations. State features that are contained in the state-based goal definition are eventually made true.

$$\forall f \in g : UF_f = 1 \tag{C1}$$

The precondition of an action has to be fulfilled when the action is included in the plan. In this constraint we had to add the multiplication with a large constant c_l, because the action variables UA_a are no boolean values anymore since the HTN can enforce an action to be in a plan more than once. Due to the delete-relaxed setting, this is no problem. When an action is in a delete-relaxed plan it can be included multiple times.

$$\forall a \in A : \forall f \in prec(a) : c_l \times UF_f \geq UA_a \tag{C2}$$

The variable $E_{a,f}$ is set if and only if a is the first achiever of f. To make this possible a must be set.

$$\forall a \in A : \forall f \in add(a) : UA_a \geq E_{a,f} \tag{C3}$$

A state feature is set when there is a first achiever. In the original encoding, a separate variable indicates if it is set in s_0. We reduce the model before the IP encoding and delete all state features already fulfilled in s_0.

$$\forall f \in L : UF_f = \sum_{a \in \{a' \mid a' \in A, f \in add(a')\}} E_{a,f} \tag{C4}$$

State features contained in the precondition of an action have to be fulfilled before the action.

$$\forall a \in A : \forall f \in prec(a) : TP_f \leq TA_a \tag{C5}$$

When a is the action first achieving the state feature f, the action must be in the planning graph before the add effect.

$$\forall a \in A : \forall f \in add(a) : TA_a + 1 \leq TP_f + (|A| + 1) \times (1 - E_{a,f}) \tag{C6}$$

Theorem 2. *For every solution of a DOF HTN planning problem, there is a valid assignment of the IP model.*

Proof. Given a solution to a DOF HTN planning problem, there exists a DT that represents it. We create a valuation $\beta : V \mapsto \mathbb{R}$ such that $\beta(UA_t)$ is equal to the number of nodes in the DT labeled with t and $\beta(M_m)$ equal to the number of vertices decomposed with m. The values of $\beta(TNI_t)$ are chosen according to tn_I. This valuation fulfills C7 and C8. C9–C13 encode reachability. Since β is based on a tree, every node is reachable from the root(s), i.e. it is possible to set the $I^s_{t' \to t}$ and R_t^s variables appropriately. □

	#instances	h^{dor}_{lp} GA*	h^{dor} GA*	$h^{dor}_{r,lp}$ GA*	h^{dor}_{r} GA*	RC^{ADD} GA*	RC^{FF} GA*	RC^{LMC} GA*	SAT'19	SAT'18	TDG_m GA*	TDG_c GA*	UMCP	2ADL-Jasper	2ADL-LAPKT	JSHOP2	FAPE
ENTERTAINMENT	12	10	10	11	10	11	**12**	11	**12**	**12**	9	9	6	5	4	5	5/5
PCP	17	**14**	**14**	**14**	**14**	**14**	**14**	**14**	12	12	9	8	0	3	3	0	0/0
SATELLITE	25	23	**25**	23	**25**	24	**25**	23	**25**	**25**	24	21	23	23	12	22	23/25
SMARTPHONE	7	5	5	5	5	5	5	5	**7**	6	6	5	4	6	5	4	4/4
UM-TRANSLOG	23	**23**	**23**	**23**	**23**	**23**	**23**	**23**	**23**	**23**	**23**	**23**	**23**	20	18	**23**	**23**/23
WOODWORKING	11	9	10	9	10	10	10	10	**11**	**11**	9	9	6	5	5	8	9/11
ROVER	20	4	5	4	5	4	4	5	**10**	4	5	5	0	5	4	3	4/20
TRANSPORT	30	9	10	9	10	6	15	14	**22**	**22**	2	1	0	19	13	0	0/0
total	144	97	102	98	102	97	108	105	**122**	115	87	81	62	86	64	65	68/88

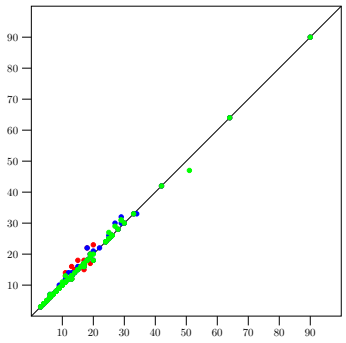

Figure 3: Left: Coverage table of several HTN planning systems. Right: Comparison of solution costs.

Theorem 3. *For every valid assignment of the IP model, there is a solution of the underlying DOF HTN problem.*

Proof. Let $\beta : V \mapsto \mathbb{R}$ be a valuation of the IP satisfying all constraints. We have to show that there exists a solution to the DOF problem, i.e. that a DT exists for it, such that the (delete-free) actions in the yield have an executable order.

The encoding of Imai and Fukunaga ensures that the actions in the yield are executable. What remains to show is that there is a DT with this yield. We construct the DT inductively in a top down fashion. We start with a tree (technically a forest) only containing the tasks of tn_I as its root nodes. In every step we systematically (1) extend the DT and (2) reduce the valuation of the variables representing the tasks added to the tree in a way that (i) maintains a valuation fulfilling our constraint system and (ii) ensuring that we end up with a DT including all tasks represented by our initial valuation.

In each step, we treat a single SCC scc of the TDG in which at least one UA_t is non-zero and for which for no incoming method $m \in \{m = (c, tn) \mid \exists t \in scc, m \in mst(t), c \notin scc\}$ the value $\beta(M_m)$ is greater than 0. If there is no such SCC, then all UA_t are 0 and we have finished our construction. Note that there cannot be a situation where some SCCs have $UA_t > 0$ for some t, but every SCC has active incoming methods, since this would constitute a cycle between SCCs (and there would be a single SCC instead). Let scc be such a SCC without an incoming method where at least for one t holds $\beta(UA_t) > 0$. Assume that for all tasks $t' \in scc$ holds $\beta(TNI_{t'}) = 0$, then C9 can only be fulfilled by setting all R^0_t to 0. Then, all $I^s_{t'\to t}$ (C11) and R^s_t (C10) are 0. C13 can only be fulfilled if for all tasks $t' \in scc$ holds $\beta(UA_{t'}) = 0$, which is a contradiction. There must be at least one task $t^* \in scc$ that is in the current tn_I and thus a leaf of the current DT.

By constraint C7 it holds that $\beta(UA_{t^*}) > 0$. There are some $\beta(M_m)$ for $m \in mdec(t^*)$ greater than 0 (C8). We choose one of them and "apply" it to the leaf t^* in the DT. Choosing it arbitrarily is not correct, as it might end a necessary recursion prematurely, but it is sufficient to apply some method that can lead to recursion, and if none exists, any method. To determine what "leads to recursion" means, consider the graph $G = (V, E)$, $V = N$ and $E = \{(a, b) \mid \exists m \in mst(b) \cap mdec(a) : \beta(M_m) > 0\}$, i.e. the tasks connected by the methods that decompose them in the IP. The task t^* is recursively decomposed if there is a path from t^* to t^* in G.

Let m^* be the selected method. We decrease $\beta(UA_{t^*})$, $\beta(TNI_{t^*})$, and $\beta(M_{m^*})$ by 1 and increase $\beta(TNI_{t'})$ by 1 for every subtask t' in m. This modification still satisfies the constraints C7 and C8. Note that this cannot change the values of any UA_a for $a \in A$ and thus the set yield of the DT.

Next, we update the reachability tracked in the $I^s_{t'\to t}$ and R^s_t variables. R^s_t encodes that the task t is reachable with (at most) s decompositions from either a task in tn_I or from outside – which cannot be the case for the scc, as it has no incoming methods. Before our modifications of $UA_{t^*}, M_{m^*}, \ldots$, all tasks t in the SCC for which $\beta(UA_t) > 0$ have been reachable (C13). Assume that afterwards there is a task t with $\beta(UA_t) > 0$ not reachable from any t' with $\beta(TNI_{t'}) > 0$. Suppose t was reachable from any such task other than t^* before the modification. Then it is either still reachable, or the only path π which made it reachable went through the method m^*. After the modification, all subtasks t' of m^* have $\beta(TNI_{t'}) > 0$ and t will be reachable from at least one of them (starting π at the subtask of m^* contained next in π).

Now, suppose that t was only reachable from t^*. So there is a (shortest) sequence of methods $\pi = (m_1, \ldots, m_n)$ connecting t^* and t in G. If π contains m^*, then t was reachable through a subtask t' of m^*. t is still reachable from t' – which will have $\beta(TNI_{t'}) > 0$ through the modification. Now, suppose π does not contain m^*. If m^* lies on a cycle for t^*, there is at least one subtask of m^* (i.e. the one on the cycle back to t^*) from which t^* is still reachable via edges in G. Also this task will have $\beta(TNI_{t'}) > 0$. Thus t is still reachable from t' by first following the path from t' to t^* and then the original path π. Conversely, if m^* does not lie on a cycle, no other method m' for t^* with $\beta(M_{m'}) > 0$ lies on a cycle, as we prefer methods that are on cycles in G. Since π does not start with m^*, m_1 is a second method decomposing t^*, i.e. $\beta(M_{m_1}) > 0$. Since t was only reachable from t^*, $\beta(TNI_{t*}) > 2$. After the modifications, $\beta(TNI_{t*}) > 1$ and thus t is still reachable from t^*. Consequently, it is possible to set $I^s_{t'\to t}$ and R^s_t s.t. the modified β satisfies all constraints. We repeat these modifications until $\forall t' \in scc : \beta(UA_{t'}) = 0$.

We then repeat the process for the next SCC. By induction, we end up with $\forall c \in C : \beta(UA_c) = 0$ and a DT that has the current task network as its leafs. Since then $\forall m \in M : \beta(M_m) = 0$ we have $\forall a \in A : \beta(UA_a) = \beta(TNI_a)$. Hence the yield of the constructed DT is equal to the set of actions for which the IP has shown delete-relaxed executability. □

5 Discussion

The first difference between our heuristics and those from the literature is that we take the interplay of hierarchy and actions in the solution into account exactly. We want to make this more clear with the following example.

Example 1. *Consider the logical formula $F = x \wedge \neg x$. Using the encoding given in the proof of Thm. 1 we encode it into a DOF HTN problem with the abstract tasks $C = \{setX, checkCl_1, checkCl_2\}$. $setX$ can be decomposed by one method into the action $setXtoTrue$ and by a second method into $setXtoFalse$. $checkCl_1$ checks whether the first clause is fulfilled. It can be decomposed into the action $checkXisTrue$; $checkCl_2$ into $checkXisFalse$. The actions set and check two state features that model whether the variable x is true or false. tn_I contains all tasks from C.*

An HTN planner cannot find a solution, since either $checkXisTrue$ or $checkXisFalse$ is not executable. Consider heuristic calculation: when we allow for task insertion during heuristic calculation, heuristic functions can include both $setXtoTrue$ and $setXtoFalse$ into their solution, making the heuristic value finite. This is prevented in our approach.

A second property of our approach is that the problem that is actually solved to calculate the heuristic value is a relaxed class of HTN planning problems. This makes it easy to understand what is actually calculated.

We now discuss the relationship of our heuristics to specific HTN heuristics from related work in more detail.

Bercher *et al.* [2017] introduced heuristics based on the TDG. The heuristic value of an action is set to its own costs; the one of a method to the sum of the costs of its subtasks and the one of an abstract task to the minimum of methods decomposing it. Costs are updated until a fixpoint is reached. To improve the heuristic, a planning graph is used to only consider actions still reachable. However, this reachability analysis includes all actions hierarchically reachable, i.e. the interplay of hierarchy and state-based reachability is not considered exactly. An action whose costs is included in the heuristic value might have preconditions fulfilled by actions not included in the heuristic value. This is a kind of task insertion: when we would extract a plan with the given costs (not done by the heuristic), we would need to insert actions to make it executable. Even worse, the costs of the inserted actions are not included in the heuristic value. Like ours, the heuristic value does not reflect the ordering constraints from the HTN model and the heuristic is calculated on a delete-relaxed model.

Höller *et al.* [2018] introduced an approach that relaxes the HTN planning problem to a classical planning problem. In contrast to the translation-based approaches by Alford *et al.* [2009; 2016] that do not change the set of solutions and use a classical planner for search, the translation of Höller *et al.* increases the set of solutions, i.e. it *relaxes* the problem. The relaxed model is not used for search, but to compute classical heuristics on it. The resulting heuristic values are then used to guide the HTN search. The relaxation includes ordering relaxation like ours, but also task insertion. In contrast to Bercher *et al.* [2017], it includes the costs of inserted actions into the heuristic value. Whether it uses delete-relaxation depends on the classical heuristic used. Interestingly, what is encoded in the classical model can be seen as the building process of a DT [Höller *et al.*, 2019].

Bit-Monnot *et al.* [2016] introduced a translation from HTN-like models into a temporal (non-hierarchical) planning model that is used for a reachability analysis during search, which prunes wide parts of the search space and is combined with blind search (so technically, no heuristic is used). The translation marks via state features which abstract tasks are currently processed, allowing only the insertion of tasks belonging to methods that decompose these tasks; this minimizes task insertion. However, the used relaxation makes it possible to decompose a single task multiple times, which also leads to inserted tasks.

6 Evaluation

We integrated our heuristic into the PANDA framework [Bercher *et al.*, 2014] and combined it with the progression algorithm by Höller *et al.* [2020]. However, it is not restricted to this setting and can also be combined with other systems or search methods. We recreate the IP model in each search node. To make it as small as possible, we perform a reachability analysis similar to the one done during grounding [Behnke *et al.*, 2020] in every search node. We included 4 configurations into the evaluation. Two of them relax the IP to a LP model by dropping the requirement that assignments of integer and Boolean variables are whole numbers. This enables solving the model in P.

- $\mathrm{h}^{\mathrm{dor}}$ – Our full model with NP-hard IP calculation.
- $\mathrm{h}^{\mathrm{dor}}_{\mathrm{lp}}$ – Full model, but relaxed to LP.
- $\mathrm{h}^{\mathrm{dor}}_{\mathrm{r}}$ – Model without the constraints C9-C13 (relaxed) with NP-hard IP calculation.
- $\mathrm{h}^{\mathrm{dor}}_{\mathrm{r,lp}}$ – Model without C9-C13 relaxed to a LP.

Since we are presenting a novel heuristic, we consider *heuristic search*-based systems from related work as our main competitors. However, since SAT-based approaches currently perform best, we also include those into the evaluation. $\mathrm{RC}^{\mathrm{ADD}}$, $\mathrm{RC}^{\mathrm{FF}}$, and $\mathrm{RC}^{\mathrm{LMC}}$ use progression search with heuristics based on a relaxation by Höller *et al.* [2018; 2019; 2020] combined with the Add [Bonet and Geffner, 2001], FF [Hoffmann and Nebel, 2001], and LM-Cut [Helmert and Domshlak, 2009] heuristics from classical planning, $\mathrm{TDG_c}$ and $\mathrm{TDG_m}$ use plan space search with the TDG-based heuristics introduced by Bercher *et al.* [2017], and UMCP is the search strategy by Erol *et al.* [1994] implemented in the PANDA system. SAT'18 and SAT'19 are translations into propositional logic as introduced by Behnke *et al.* [2018; 2019]. JSHOP2 is the system by Nau *et al.* [2003]. The FAPE system [Bit-Monnot *et al.*, 2016] does not support recursion. Since some instances are recursive in the lifted model but become non-recursive during grounding we produced groundings with PANDA and ran FAPE on these instances (this increased its performance). The number of instances is given for each domain. 2ADL-Jasper and 2ADL-LAPKT use the translation of HTN problems into ADL by Alford *et al.* [2016] combined with JASPER [Xie *et al.*, 2014] and LAPKT-BFWS [Francès *et al.*, 2018].

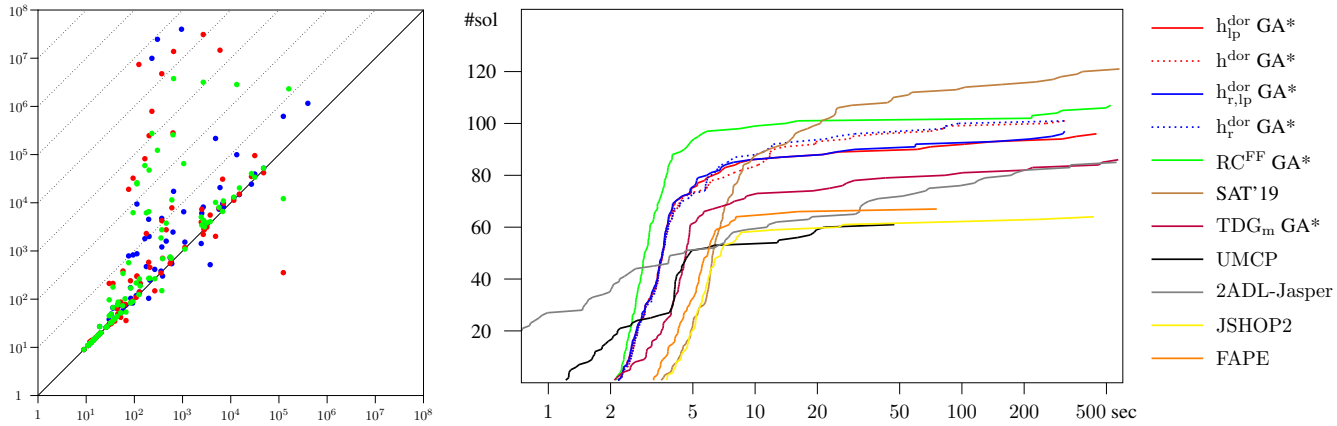

Figure 4: Left: Search nodes needed to solve a problem for our system (x-axis) and the RC heuristics (y-axis) with LM-Cut (green), FF (blue), and Add (red). Right: Sum of solved problems (y-axis) relative to search time (log-scale).

Heuristic search-based HTN systems have been combined with Greedy Best First search, A* search, and Greedy A* search (GA*) with weight 2. Due to lack of space, we only discuss GA* search (the configuration with the highest coverage). We used a server with Xeon E5-2660 CPUs (2.60 GHz), 4 GB RAM and 10 minutes time limit. Our IP model was solved using the CPLEX solver (version 12.8, restricted to 1 CPU core). We used the same problem set used in related work [Höller *et al.*, 2018; Behnke *et al.*, 2018; Behnke *et al.*, 2019] including 144 instances.

The coverage of all systems is given in Figure 3 (left). The SAT-based systems have the highest coverage. From the search-based systems, the RC heuristics solve most problems. $\mathrm{RC}^{\mathrm{FF}}$ and $\mathrm{RC}^{\mathrm{LMC}}$ perform nearly identical, while $\mathrm{RC}^{\mathrm{ADD}}$ solves 11 (8) instances less. Our best configurations are placed between the RC heuristics. Interestingly, the IP version performs better than the one relaxed to a LP model. From the IP configurations, the one not preventing cycles unconnected to the tasks of the current task network performs slightly better. The best TDG-based configuration solves 10 ($\mathrm{RC}^{\mathrm{ADD}}$) to 21 ($\mathrm{RC}^{\mathrm{FF}}$) instances less than the RC configurations and performs similar to 2ADL-Jasper. All other systems have a by far lower coverage.

Figure 4 (right) shows the number of solved instances relative to the search time (be aware of the log-scale). The 2ADL-Jasper system solves simple problems most quickly, followed by UMCP, the RC-based systems, and our configurations. Interestingly, all our systems are very close together, but the NP-hard configurations solve more problems (i.e. NP calculation pays off). The SAT-based systems need more time for simple problems, but solve more instances in total.

Figure 4 (left) shows the number of search nodes needed to find a plan, comparing our $\mathrm{h}^{\mathrm{dor}}$ system and the three RC heuristics (since these are based on the same search algorithm). Figure 3 (right) compares the action costs of the generated plans. It can be seen that our system needs less search nodes (be aware of the log-scale) but finds plans with equal costs. This is similar when comparing $\mathrm{h}^{\mathrm{dor}}$ to variants of our system where the calculation is relaxed to P, i.e. $\mathrm{h}^{\mathrm{dor}}$ is the most informed heuristic of the ones given here.

To sum up, like in evaluations in related work, the SAT-based HTN planners performed best, followed by the best configuration of the RC heuristic ($\mathrm{RC}^{\mathrm{FF}}$). Our heuristics performed slightly worse than $\mathrm{RC}^{\mathrm{FF}}$, but better than $\mathrm{RC}^{\mathrm{ADD}}$. Both RC-based and our heuristics perform much better than the other search-based systems. Preventing self-sustaining cycles seems not to pay off on the given problem set. Solving the NP-hard IP model instead of the relaxed LP model payed off. The analysis of the number of explored search nodes showed that our IP-based heuristics are more informed both than the LP configurations and the RC heuristics.

7 Conclusion

We introduced a new subclass of HTN planning and showed that its plan existence problem is NP-complete. We defined heuristics based on this type of model and showed how to compute them efficiently using an IP model/solver. These heuristics form a new line of research on HTN heuristics. These are the first heuristics avoiding task insertion in the calculation, i.e. taking the interplay of decomposition and actions in the solution into account *exactly*. They are more informed than HTN heuristics from the literature and competitive with these with respect to coverage. In difference to other HTN heuristics, they have easy to grasp semantics since they are based on a well-defined problem class. The declarative model opens promising lines of research, e.g. enhanced IP heuristics incorporating other information like landmarks or net-change constraints [Pommerening *et al.*, 2014]. This will also help the configurations relaxing the model to a LP to find more accurate heuristic values. It can also be adapted to find HTN plans with optimal action costs.

Acknowledgments

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 232722074 – SFB 1102 / Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 232722074 – SFB 1102.

References

[Alford *et al.*, 2009] Ron Alford, Ugur Kuter, and Dana S. Nau. Translating HTNs to PDDL: A small amount of domain knowledge can go a long way. In *Proc. of IJCAI*, pages 1629–1634. AAAI Press, 2009.

[Alford *et al.*, 2014] Ron Alford, Vikas Shivashankar, Ugur Kuter, and Dana S. Nau. On the feasibility of planning graph style heuristics for HTN planning. In *Proc. of ICAPS*, pages 2–10. AAAI Press, 2014.

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

[Behnke *et al.*, 2018] Gregor Behnke, Daniel Höller, and Susanne Biundo. Tracking branches in trees – A propositional encoding for solving partially-ordered HTN planning problems. In *Proc. of ICTAI*, pages 73–80. IEEE, 2018.

[Behnke *et al.*, 2019] Gregor Behnke, Daniel Höller, and Susanne Biundo. Bringing order to chaos – A compact representation of partial order in SAT-based HTN planning. In *Proc. of AAAI*, pages 7520–7529. AAAI Press, 2019.

[Behnke *et al.*, 2020] Gregor Behnke, Daniel Höller, Alexander Schmid, Pascal Bercher, and Susanne Biundo. On succinct groundings of HTN planning problems. In *Proc. of AAAI*. AAAI Press, 2020.

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

[Bercher *et al.*, 2017] Pascal Bercher, Gregor Behnke, Daniel Höller, and Susanne Biundo. An admissible HTN planning heuristic. In *Proc. of IJCAI*, pages 480–488. AAAI Press, 2017.

[Bercher *et al.*, 2019] Pascal Bercher, Ron Alford, and Daniel Höller. A survey on hierarchical planning – One abstract idea, many concrete realizations. In *Proc. of IJCAI*, pages 6267–6275. ijcai.org, 2019.

[Bit-Monnot *et al.*, 2016] Arthur Bit-Monnot, David E. Smith, and Minh Do. Delete-free reachability analysis for temporal and hierarchical planning. In *Proc. of ECAI*, pages 1698–1699. IOS Press, 2016.

[Bonet and Geffner, 2001] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

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

[Erol *et al.*, 1996] Kutluhan Erol, James A. Hendler, and Dana S. Nau. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, 18(1):69–93, 1996.

[Francès *et al.*, 2018] Guillem Francès, Hector Geffner, Nir Lipovetzky, and Miquel Ramírez. Best-first width search in the IPC 2018: Complete, simulated, and polynomial variants. In *Proc. of IPC*, 2018.

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

[Helmert and Domshlak, 2009] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. of ICAPS*, pages 162–169. AAAI Press, 2009.

[Hoffmann and Nebel, 2001] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.

[Höller *et al.*, 2014] Daniel Höller, Gregor Behnke, Pascal Bercher, and Susanne Biundo. Language classification of hierarchical planning problems. In *Proc. of 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 *Proc. of ICAPS*, pages 158–165. AAAI Press, 2016.

[Höller *et al.*, 2018] Daniel Höller, Pascal Bercher, Gregor Behnke, and Susanne Biundo. A generic method to guide HTN progression search with classical heuristics. In *Proc. of ICAPS*, pages 114–122. AAAI Press, 2018.

[Höller *et al.*, 2019] Daniel Höller, Pascal Bercher, Gregor Behnke, and Susanne Biundo. On guiding search in HTN planning with classical planning heuristics. In *Proc. of IJCAI*, pages 114–122. ijcai.org, 2019.

[Höller *et al.*, 2020] Daniel Höller, Pascal Bercher, Gregor Behnke, and Susanne Biundo. HTN planning as heuristic progression search. *JAIR*, 67:835–880, 2020.

[Imai and Fukunaga, 2015] Tatsuya Imai and Alex Fukunaga. On a practical, integer-linear programming model for delete-free tasks and its use as a heuristic for cost-optimal planning. *JAIR*, 54:631–677, 2015.

[Nau *et al.*, 2003] Dana S. 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.

[Pommerening *et al.*, 2014] Florian Pommerening, Gabriele Röger, Malte Helmert, and Blai Bonet. LP-based heuristics for cost-optimal planning. In *Proc. of ICAPS*, pages 226–234. AAAI press, 2014.

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

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