A Heuristic for Optimal Total-Order HTN Planning Based on Integer Linear Programming

Conny Olz[a,*], **Alexander Lodemann**[a] **and Pascal Bercher**[b]

[a]Ulm University
[b]The Australian National University (ANU)

Abstract. Heuristic Search is still the most successful approach to hierarchical planning, both for finding any and for finding an optimal solution. Yet, there exist only a very small handful of heuristics for HTN planning – so there is still huge potential for improvements. It is especially noteworthy that there does not exist a single heuristic that's tailored towards special cases. In this work we propose the very first specialized HTN heuristic, tailored towards totally ordered HTN problems. Our heuristic builds on an existing NP-complete and admissible delete-and-ordering relaxation ILP heuristic, but partially incorporates ordering constraints while reducing the number of ILP constraints. It exploits inferred preconditions and effects of compound tasks and is also admissible thus allowing to find optimal solutions. Our heuristic demonstrates improved performance (ILP) or comparable performance (LP) to the previous heuristic, suggesting the success of the model reduction. Compared to the current state-of-the art heuristic for optimal HTN planning, our heuristic is less efficient on average, but more informed and dominates it in roughly as many cases as it gets dominated by the other, making it a more efficient alternative in several domains.

1 Introduction

As witnessed by already having the second track on Hierarchical Task Network (HTN) Planning in the International Planning Competition (IPC), solving HTN problems quickly or optimally is a prominent research field. Collectively, ten HTN planners participated at the IPCs (not counting various configurations per planner), and further planners exist as well. Among all the various approaches, heuristic progression search [15, 19] is still the most efficient approach, both for optimal and for suboptimal planning – as witnessed by the most recent IPC [8, 9, 21].

The success of these search methods is tied directly to the quality of the heuristics deployed. Despite the success of heuristic search, only a very small number of HTN heuristics exist. One of the first was a TDG-based heuristic that estimates the minimal number of tasks that can be obtained by decomposing the compound tasks in the current search node [5], one finds refinements to delete- and ordering-relaxed problems encoded by an Integer Linear Program (ILP) [14], another bases on landmarks [11], and the last – but most successful – heuristic is the relaxed composition heuristic [12, 15], which encodes each search node into a classical problem allowing to deploy classical heuristics.

* Corresponding Author. Email: conny.olz@uni-ulm.de.

All of these heuristics are designed to cope with the most general case of an arbitrary partial order. However, there are significantly more specialized total-order (TO) planners than planners for general partial order planning, yet no heuristic for this important special case exists. It is however notable that there exists a pruning technique for total-order HTN problems [19], which further shows the potential of this special case as its exploitation further improved the total-order planner on top of which the pruning was implemented for the extend that it won the total-order HTN track of the IPC 2023 [21].

In this paper we propose the – to the best of our knowledge – first HTN planning heuristic tailored towards totally ordered problems. It exploits inferred preconditions and effects of compound tasks [20] (which also serve as the basis for the TO pruning technique mentioned before [19, 21]) and deploys them in a simplified variant of the ILP-based (NP-complete) delete- and ordering relaxation heuristic (DOR) by Höller et al. [14]. Like the original heuristic, our variant is admissible. Similarly, it can also be computed in polytime (by relaxing the integer variables to real-valued ones), but it naturally loses some of its informedness and hence pruning power if that is done.

We compare our new heuristic against the original (I)LP heuristic [14] and the currently best-performing admissible one, the RC heuristic [12, 15] with the (admissible) LM-cut [7] as inner classical heuristic.

2 Theoretical Background

We start with providing the necessary definitions for total-order HTN planning and inferred effects of compound tasks.

2.1 HTN Planning Formalism

Our heuristic for totally ordered HTN planning is grounded in the formalisms introduced by Geier and Bercher [6] and Behnke et al. [3]. A total-order HTN planning domain is defined as a tuple $\mathcal{D} = (F, A, C, M)$, which includes a finite set of facts F, finite sets of *primitive tasks* A (also called actions) and *compound tasks* C (also referred to as *abstract tasks*), and *(decomposition) methods* $M \subseteq C \times T^*$.[1] The collective set of tasks, both primitive and compound, is denoted by $T = A \cup C$. Actions $a = (prec, add, del) \in A$ are characterized by their *preconditions* $prec(a) \subseteq F$ and their *effects* $add(a), del(a) \subseteq F$ (namely, the *add and delete effects*). An action $a \in A$ is *applicable* in a state $s \in 2^F$ if $prec(a) \subseteq s$. Upon

[1] The Kleene star notation T^* represents the set that includes the empty sequence and all finite sequences of tasks from T.

application, it transitions the state s to a successor state $\delta(s, a) = (s \setminus del(a)) \cup add(a)$. This concept extends to action sequences $\bar{a} = \langle a_0 \dots a_n \rangle$ with each $a_i \in A$, deemed applicable in an initial state s_0 if a_0 is applicable in s_0 and sequentially for each $1 \leq i \leq n$, a_i is applicable in the resultant state $s_i = \delta(s_{i-1}, a_{i-1})$. Compound tasks within HTN planning, denoted by $c \in C$, represent a higher-level abstraction of primitive and/or compound tasks, further specified by (decomposition) methods $m = (c, \bar{t}) \in M$. These methods *decompose* a compound task c within a given task network $tn_1 = \langle \bar{t}_1\, c\, \bar{t}_2 \rangle \in T^*$ into another task network $tn_2 = \langle \bar{t}_1\, \bar{t}\, \bar{t}_2 \rangle$, denoted as $tn_1 \rightarrow_{c,m} tn_2$, where *task networks* are finite (possibly empty) sequences of tasks. A sequence of methods that transforms tn into tn' is represented as $tn \rightarrow tn'$, with tn' being called a refinement of tn. For a compound task $c \in C$ we also write $c \rightarrow tn'$ if we refer to $\langle c \rangle \rightarrow tn'$. A *TOHTN planning problem* $\Pi = (\mathcal{D}, s_I, c_I, g)$ is defined by the domain $\mathcal{D}$, an *initial state* $s_I \in 2^F$, an *initial task* $c_I \in C$, and a goal description $g \subseteq F$. A solution is a sequence of actions $tn = \langle a_0 \dots a_n \rangle \in A^*$ if $c_I \rightarrow tn$ holds, tn is applicable in s_I, and it leads to a goal state $s \supseteq g$.

2.2 Preconditions and Effects of Compound Tasks

In our version of the ILP heuristic we incorporate the concept of inferred negative effects of compound tasks as introduced by Olz et al. [20]. Compound tasks, both according to the formalism we base upon [6, 3], and as described by HDDL [13], lack direct effects and serve primarily as placeholders for task networks to which they decompose into during the planning process. A detailed examination of the potential decompositions of compound tasks allows for the inference of state features required prior to any task refinement execution and the state features that result from all possible refinements. Olz et al. [20] categorize such effects into several types, including possible and guaranteed effects, as well as positive and negative ones, in addition to preconditions. Our focus here is specifically on guaranteed negative effects, hence we limit our recap to them.

The set of *executability-enabling states* for a compound task $c \in C$ is defined as $E(c) = \{s \in 2^F \mid \exists\, \bar{a} \in A^* : c \rightarrow \bar{a}$ and $\bar{a}$ is applicable in $s\}$. Moreover, the set of states that could result from executing task c in state $s \in 2^F$ is $R_s(c) = \{s' \in 2^F \mid \exists\, \bar{a} \in A^* : c \rightarrow \bar{a}, \bar{a}$ is applicable in s and leads to $s'\}$.

Now, facts that are deleted after every successful execution of a refinement of a compound task c are called *state-independent negative effects* (cf. Def. 4, [20]) and are defined as follows:

$$\mathit{eff}^-_*(c) := \cap_{s \in E(c)} (F \setminus \cap_{s' \in R_s(c)} s')$$

if $E(c) \neq \emptyset$, otherwise $\mathit{eff}^-_*(c) := \mathit{undef}$.

Computing these "precise" effects for compound tasks is often too computationally expensive for the exploitation in heuristics, as it essentially requires solving certain planning problems [20]. However, a more computationally feasible approach exists, based on *precondition-relaxation*. The *precondition-relaxed effects*, denoted as $\mathit{eff}^{\emptyset-}_*(c)$, are defined similarly to the original effects but rely on a precondition-relaxed version of the domain $\mathcal{D}' = (F, A', C, M)$, where $A' = \{(\emptyset, add, del) \mid (prec, add, del) \in A\}$. This approach considers only the presence and sequence of primitive tasks in the computation. Procedures for computing these effects in polynomial time have been provided by Olz et al. [20] and are also implemented in the PANDADealer planning system [19, 21], which we employ for our evaluation.

- $\{CA_t \mid t \in T\}$ (int) – value indicating how often a certain primitive or abstract task is in the solution.
- $\{M_m \mid m \in M\}$ (int) – value indicating how often a certain method is in the solution.
- $\{TNI_t \mid t \in T\}$ (bool) – flag indicating whether a certain task is the initial task.

Figure 1. The variable set used in our ILP model. It is a subset of the ones by Höller et al. [14]. The first variable is renamed, the last adapted to initial task instead of task network.

3 ILP Encoding for Delete-relaxed TO-HTN Search Nodes

Höller et al. [14] introduced the first HTN planning heuristic based on an ILP. They encode a delete- and ordering-relaxed (DOR) HTN planning problem, for which the plan existence problem is NP-complete to decide. The encoding can be divided into two parts: Constraints that ensure the successful execution of a sequence of actions (or a relaxed version of it) and constraints to ensure the proper decomposition leading to it. For the first part, Höller et al. use the encoding by Imai and Fukunaga [16] (for classical planning) representing a delete-relaxed planning graph. However, we introduce a different approach, which we outline further below. The latter part, we take from the work by Höller et al. without changes, which we recap next.

Figure 1 outlines the set of ILP variables. The model by Höller et al. requires five types of variables to encode the executability of actions, while we reduce this to just *one* – the first one. (In both models, the second and third types are only used for encoding the decomposition).

One can consider two objective functions. We focus on the first one (O1), which minimizes the number of primitive actions in the solution. This function estimates the length of the resulting plan and should be used in combination with an appropriate search strategy if the goal is to find plans of minimal length. More details and explanations regarding admissibility are provided in Section 4.1.

In the context of satisficing planning, where the aim is to find any solution as quickly as possible, the second objective function (O2) has proven to yield better results in combination with GBFS, among others. It minimizes the number of primitive actions that need to be applied and the number of method applications leading to those actions. This function estimates the goal distance in the modification space but can not be used with A* to find optimal plans. We mentioned (O2) for completeness but do not evaluate it later.

$$min \sum_{a \in A} CA_a \tag{O1}$$

$$min \sum_{a \in A} CA_a + \sum_{m \in M} M_m \tag{O2}$$

In order to simplify our constraints, we encode the current state s and goal description as actions, known from partial-order causal link (POCL) planning [4, 17]. Specifically, we introduce $a_I = (\emptyset, s, \emptyset)$ and $a_g = (g, \emptyset, \emptyset)$. Since they need to be ordered before and after, respectively, all other tasks, we additionally add a new compound task c_I with one method $m_I = (c_I, \langle a_I, tn_I, a_g \rangle)$, which replaces the current task network of the search node.

3.1 Task Decomposition Constraints

A solution to an HTN planning problem needs to be a refinement of the initial task. The following two constraints by Höller et al. [14]

encode this criterion, which we also add to our ILP model. If a (primitive) task is contained in the solution, then it is the initial task and/or a method has been applied, which introduced the task into the plan:

Definition 1 (mst)**.** *Let $mst(t)$ be the multiset of methods where the task $t \in T$ is contained as a subtask. A method $m \in M$ is as often in $mst(t)$ as t is a subtask in m.*

$$\forall t \in T : CA_t = TNI_t + \sum_{m \in mst(t)} M_m \qquad \text{(C7)}$$

However, methods can not be applied arbitrarily, there also needs to be a suitable abstract task for every applied method.

Definition 2 ($mdec$)**.** *Let $mdec(c)$ be the set of methods decomposing the abstract task $c \in C$.*

$$\forall c \in C : CA_c = \sum_{m \in mdec(c)} M_m \qquad \text{(C8)}$$

According to Höller et al. [14] the two constraints are sufficient to encode the proper decomposition (more precisely, encoding a so-called decomposition tree) leading to a sequence of tasks for acyclic problems. For cyclic domains further constraints are necessary to handle strongly connected components (SCCs). This means that the encoding without those additional constraints can also be used for cyclic domains but the solutions can encode (shorter) plans that can not be obtained by a proper sequence of decompositions. To simplify the presentation of this paper, we do not repeat these additional constraints here or include them in our theoretical considerations, as our evaluation results show weaker performance when they are included.

3.2 Achiever Constraints

The ILP model by Höller et al. [14] uses the constraints by Imai and Fukunaga [16] to simulate a delete-relaxed planning graph. Therefore, for every time point there exist ILP variables for every action and fact, stating whether an action is executed at that time point and the facts being true or false. If an action is applied at some time point, its preconditions need to be true beforehand. If a fact needs to be true at some time point, there must be an action adding it if it is not already true in the initial state. The ILP solver tries to find an order of the actions so that preconditions and goal facts are satisfied (under delete relaxation). The resulting order might not meet the ordering constraints of tasks within methods. Thus, especially in total-order domains a lot of information gets lost.

Our intention was to improve the existing ILP heuristic in terms of accuracy by incorporating (at least some of) the ordering constraints imposed by the methods. We observed that adding additional constraints could improve the heuristic value but the additional time needed to solve the ILP did not pay off (this was done in a preevaluation, not reported here). To encode the planning graph, already quite a lot of variables and constraints are necessary. Therefore, in our proposed approach, we made the model more simple by calculating necessary information outside of the ILP upfront. Instead of using the constraints by Imai and Fukunaga [16], we ended up with only one (new) constraint (c_l is a large constant):

$$\forall a \in A, \forall f \in prec(a) : c_l \cdot \sum_{p \in achiever(a,f)} CA_p \geq CA_a \qquad \text{(C1)}$$

Algorithm 1 Calculating Predecessor Actions

Input: A problem $\Pi = (\mathcal{D}, s_I, c_I, g)$
Output: Sets of possible predecessors $pred(a) \, \forall \, a \in A$

```
pred(a) = ∅ for all a ∈ A
for all methods m = (c, ⟨t_0, ..., t_n⟩) ∈ M do
    for i = 1 to n do
        for all a ∈ reachable(t_i) do
            for j = i − 1 to 0 do
                pred(a) = pred(a) ∪ reachable(t_j)
```

This constraint ensures that for every action $a \in A$ in the plan and every of its preconditions there is an action "achieving" the precondition. So, the influence of this constraint heavily depends on the definition of the set $achiever(a, f)$. A naive approach might be the following: Let $a \in A$ be a primitive task and $f \in F$ a precondition, then the set of possible achievers is $achiever(a, f) = \{a' \in A \mid f \in add(a')\}$. However, in this case the solutions of the ILP are not very restricted. Neither are the methods' ordering constraints taken into account nor does it guarantee that there is an executable ordering of the actions (even under delete relaxation). In the next section we present algorithms to restrict the set of achievers for an action a further so that it only contains actions that appear before a according to the method's total order. Moreover, by exploiting the inferred effects, we can even partially consider delete effects. Thus, we do not present further changes to the ILP model, we only discuss several options of how to calculate the set of possible achievers and their impact on the set of ILP solutions. To summarize, the ILP model of our new heuristic is composed of the objective function (O1), subject to the constraints (C1), (C7), and (C8).

4 Determining Achiever Actions

Given a total-order HTN planning problem, we can determine for a given action the set of actions that can be ordered before that action in a possible refinement of the initial task. We will see that we can calculate this with different levels of accuracy.

To start, we define the set of reachable actions $reachable(c) = \{a \in A \mid \exists \, \bar{t} \in T^* : c \to \bar{t} \text{ and } a \in \bar{t}\}$ of a compound tasks $c \in C$, which are the actions reachable via decomposition. For primitive actions $a \in A$, we define $reachable(a) = \{a\}$. The sets can be calculated in polynomial time, e.g., by a depth-first search with a closed list of already visited compound tasks to prevent infinite cycles. The RC heuristic does this in a preprocessing step; it's a one-time computation. If we are given these sets for each compound task, we can compute, for every primitive action, the set of actions that can possibly appear as predecessors in a refinement of the initial compound task (and are thus candidates for achiever actions) as shown in Algorithm 1.

The algorithm considers every method once. So let $m = (c, \langle t_0, \ldots, t_n \rangle) \in M$ be a method. For each task t_i ($i > 1$) in the method, all of its reachable actions are considered. All reachable actions of preceding tasks $t_j, j < i$ are added to their sets of predecessors.

Proposition 1. *Let $\Pi = (\mathcal{D}, s_I, c_I, g)$ be a total-order HTN planning problem, $a \in A$, and $pred(a)$ be computed by Algorithm 1. Then it holds $a' \in pred(a)$ if and only if there exists a refinement $\bar{a}$ of c_I (not necessarily executable) so that $a, a' \in \bar{a}$ and $a' \prec a$.*

Proof Sketch. We assume that all methods are reachable by decomposition from the initial task, otherwise the unconnected methods should not be considered in the algorithm.

Algorithm 2 Calculating Achiever Actions

Input: A problem $\Pi = (\mathcal{D}, s_I, c_I, g)$
Output: $achiever(a, f)$ for all $a \in A, f \in prec(a)$

1: $achiever(a, f) = \emptyset$ for all $a \in A, f \in prec(a)$
2: **for all** methods $m = (c, \langle t_0, \dots, t_n \rangle) \in M$ **do**
3: **for** $i = 1$ to n **do**
4: **for all** $a \in reachable(t_i)$ **do**
5: **for all** $f \in prec(a)$ **do**
6: **for** $j = i - 1$ to 0 **do**
7: **if** $f \in \mathit{eff}^-_*(t_j)/del(t_j)$ **then**
8: **break**
9: $achiever(a, f) = achiever(a, f) \cup$
$\{a' \in reachable(t_j) \mid f \in add(a')\}$

"$\Rightarrow$" Let $a, a' \in A$ and $a' \in pred(a)$. Consider the method $m = (c, \langle t_0, \dots, t_n \rangle) \in M$ in line 2 for which a' was added to $pred(a)$ in line 6. Since there there are $0 \leq j, i \leq n$ with $j < i$, $a \in reachable(t_i)$, and $a' \in reachable(t_j)$ there must be a refinement of $\langle t_0, \dots, t_n \rangle$ in which a' is ordered before a. Moreover, by assumption, there must be a refinement of c_I that contains c which can be decomposed using m, which proves the first direction.

"$\Leftarrow$" Let $\bar{a}$ be a refinement of c_I and $a, a' \in \bar{a}$ two primitive tasks with $a' \prec a$. If we consider the two sequences of decompositions (more specifically, the used methods) leading from c_I to a and from c_I to a' in $\bar{a}$, then the two sequences have the same prefix of methods. The suffix may differ. However, the last common method $m = (c, \langle t_0, \dots, t_n \rangle) \in M$ must have two tasks $t_j \prec t_i$ with $a' \in reachable(t_j)$ and $a \in reachable(t_i)$ so that a' will get added to $pred(a)$ in line 6. □

Algorithm 1 can be extended to restrict the set of possible achievers for the preconditions of an action $achiever(a, f)$ based on the total-order of the method set and inferred negative effects, given in Algorithm 2.

Again, every method $m = (c, \langle t_0, \dots, t_n \rangle) \in M$ is considered once. Now, for each task t_i ($i > 1$), all of its reachable actions and their preconditions are considered. Preceding tasks $t_j, j < i$ are checked for reachable actions that can add these preconditions, updating the achiever sets. If any of these preceding task $t_j, j < i$ deletes a precondition (according to its inferred delete effects $\mathit{eff}^-_*(t_j)$ if t_j is abstract or its delete effects $del(t_j)$ if it is primitive), we do not consider its reachable actions nor the reachable actions of its predecessors (line 8).

Algorithm 2 runs in polynomial time, namely in $\mathcal{O}(|M| \cdot n^2 \cdot (reach_{max})^2 \cdot prec_{max})$, where n is the size of the largest task network within methods, $reach_{max} = \max_{t \in T} |reachable(t)|$ and $prec_{max} = \max_{a \in A} |prec(a)|$.

Proposition 2. *Let $\Pi = (\mathcal{D}, s_I, c_I, g)$ be a problem, $a \in A$ a primitive task, and $achiever(a, f)$, $pred(a)$ be calculated according to Algorithms 1 and 2. Then it holds*

- $\bigcup_{f \in prec(a)} achiever(a, f) \subseteq pred(a)$ *and*
- *for all refinements $\bar{a}$ of c_I it holds if $a, a' \in \bar{a}, a' \prec a$, $f \in prec(a) \cap add(a')$ and f is not deleted in between then $a' \in achiever(a, f)$.*

Proof Sketch. Since Algorithm 2 collects only actions that add one of the preconditions, the set of achievers is a subset of the predecessors calculated by Algorithm 1. Since for the achievers some of the delete effects are taken into account in line 8 even less actions are added to that set.

For the second point, let us consider a refinement $\bar{a}$ of c_I with $a, a' \in \bar{a}, a' \prec a$, $f \in prec(a) \cap add(a')$, where no other action deletes f in between. According to Proposition 1 we know that $a' \in pred(a)$. Since no action in between deletes f, the condition for line 8 is not satisfied. So, a' is also added to $achiever(a, f)$ since $f \in prec(a)$ and $f \in add(a')$. □

So, we define our heuristic based on the ILP presented in the last section, with the set of possible achievers $achiever(a, f)$ calculated by Algorithm 2. The last proposition essentially shows that we restricted the set of possible achievers without losing any solutions:

Theorem 3. *For every solution of a TOHTN planning problem there exists a valid assignment of the ILP model.*

Proof. Höller et al. [14] already showed that for every solution of a DOR HTN planning problem, there is a valid assignment of their ILP model. Since every solution of a TOHTN planning problem is also a solution under delete and ordering relaxation, we know that there is a valid assignment of our model that satisfies the task decomposition constraints C7 and C8. So we only need to check C1. For all primitive tasks $t \in A$ the variables CA_t are set according to how often the task is in the solution. Consider a primitive task a with $CA_a > 0$ and precondition f. Since the plan is executable there must be an action a' in the plan adding f and no action deletes f in between. According to Proposition 2 it holds $a' \in achiever(a, f)$ and therefore C1 is satisfied since also $CA_{a'} > 0$. □

The next question is whether the heuristic performs relaxations or if all valid assignments of the ILP model correspond to some solution of a TOHTN problem. From a theoretical point of view this is "unlikely" (or even impossible, depending on the exact relationship of complexity classes, which are still unknown) since TOHTN planning is EXPTIME-complete in general and still PSPACE-complete for acyclic domains [2], while ILPs can only solve problems in NP. Essentially, two relaxations are performed: Since we only check for achievers of preconditions and ignore most of the delete effects, we perform some delete-relaxation. Moreover, the total-order of tasks is partially relaxed. For example, consider a method containing a compound task c twice. Assume that c has two methods $m_1 = (c, \langle a_1 \rangle)$ and $m_2 = (c, \langle a_2 \rangle)$, where a_1 can support a precondition of a_2 and vice versa. Then the two primitive actions are in the achiever sets of each other and in a solution of a corresponding ILP the two actions could support each other. However, actually only one of the preconditions is satisfied because one action is applied before the other, so the first one needs another action adding its precondition. Therefore, we denote our heuristic as P-TODR (partial total-order, partially delete-relaxed).

To overcome the limitation of not fully adhering to the total order, one could duplicate primitive and compound tasks, ensuring that each task occurs only once across all methods' task networks. So, in the example above, we then have two compound tasks c and c', m_1 and m_2 unchanged, but two additional methods $m'_1 = (c', \langle a'_1 \rangle)$ and $m'_2 = (c', \langle a'_2 \rangle)$, where a'_1 and a'_2 have the same preconditions and effects as a_1 and a_2, respectively. In the worst case such a transformation introduces exponential many new tasks. If a transformation is possible with polynomially many new tasks, we can actually encode an acyclic, delete-relaxed TOHTN problem. Therefore, we call a TOHTN planning problem $\Pi = (\mathcal{D}, s_I, c_I, g)$ a *unique tasks* problem if for all tasks $t \in A \cup C$ there is at most one method $m = (c, \bar{t}) \in M$ with $t \in \bar{t}$ and additionally t is contained only once in $\bar{t}$.

Theorem 4. *Consider an acyclic, delete-relaxed, total-order HTN planning problem with unique tasks. Then, for every valid assignment of the ILP model, there exists a corresponding solution to the underlying delete-relaxed, total-order HTN problem.*

Proof Sketch. Consider an acyclic, delete-relaxed, unique tasks TOHTN problem and a valid assignment of the corresponding ILP model. According to Höller et al. [14] the constraints C7 and C8 ensure that there is a refinement of the initial task that contains exactly the number of primitive tasks as indicated by the variables CA_t. Since the constraints C1 are satisfied, for all actions in the plan and their preconditions there is an action adding it. So we only need to verify that the actions appear in the correct order so that all preconditions are satisfied. Since every task (primitive or compound) appears exactly once in all methods there is only one sequence of decompositions leading to that task. This implies that if for two actions a, a' it holds $a' \in pred(a)$, then $a \notin pred(a')$. This does also hold for the achiever sets since they are subsets of the predecessors. So, the achievers already encode some total-order over all tasks and the refinement of the initial task is executable under delete-relaxation. □

Since we take some of the (inferred) delete effects into account when we calculate the achievers (line 8, Alg. 2), not every solution of an acyclic, delete-relaxed, unique tasks TOHTN problem has a valid assignment of the ILP model. Some of the non-executable ones (when considering delete effects) are missing, which is beneficial for the heuristic since they are recognized as not being correct solutions.

If we remove line 8 (what we should not do in practice) the ILP can exactly encode acyclic, delete-relaxed, unique tasks TOHTN problems, which is the first encoding for this class so far. This is also in line with the result by Alford et al. [1] that acyclic and delete-relaxed (total-order) HTN planning is NP-complete. The result by Alford et al. actually tells us that there must be an encoding in general without relying on the unique tasks property. The hardness proof by Alford et al. does not rely on unique tasks but we can adapt a reduction by Olz and Bercher [18] to show NP-hardness of acyclic and regular, delete- and precondition-relaxed problems to unique tasks so that we can conclude that:

Theorem 5. *The plan existence problem of acyclic, delete-relaxed, and total-order HTNs with unique tasks is NP-complete.*

Proof. The problem is in NP because acyclic and delete-relaxed HTN planning is NP-complete [1].

For hardness we adapt the reduction from 3SAT by Olz and Bercher [18, Proof of Thm. 1] to unique tasks: Consider a 3SAT formula ϕ in conjunctive normal form consisting of the propositional variables $x_1 \dots x_n$ and clauses $C_1 \dots C_k$. We construct the following planning problem Π: For every clause C_j ($1 \leq j \leq k$) there is a fact f_j. For every variable x_i ($1 \leq i \leq n$) there is one compound tasks u_i and two primitive tasks $x_i^\top$, $x_i^\bot$ with add effects $add(x_i^\top) = \{f_l \mid x_i \in C_l\}$ and $add(x_i^\bot) = \{f_l \mid \neg x_i \in C_l\}$, respectively. The preconditions and delete effects are empty. Each compound task u_i ($1 \leq i \leq n$) has two decomposition methods $m_i^1 = (u_i, \langle x_i^\top \rangle)$ and $m_i^2 = (u_i, \langle x_i^\bot \rangle)$. At last there is the initial compound task c_I with one method $m_I = (c_I, \langle u_1 \dots u_n \rangle)$. Since the methods of the compound tasks u_i all contain different tasks that are additionally primitive, the problem is acyclic and fulfills the unique task property. The goal condition is $g = \{f_1, \dots, f_k\}$.

The choice of decomposing u_i to either $x_i^\top$ or $x_i^\bot$ corresponds to setting the SAT variable x_i to true or false, respectively. The add effects of these primitive actions represent the satisfaction of the respective clauses. Thus, we can claim that ϕ is satisfiable if and only if Π is solvable. □

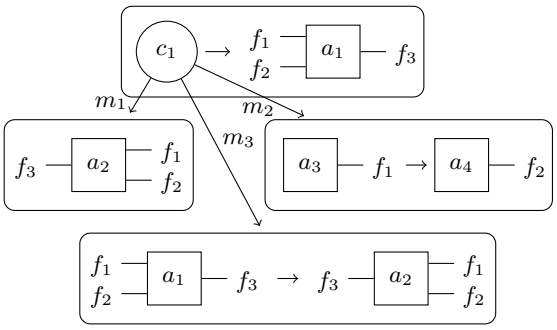

Figure 2. An example demonstrating dominance properties.

4.1 Admissibility and Dominance

We examine the typical theoretical properties: admissibility, safety, and goal-awareness ($h(n) = 0$ for every goal search node n).

Theorem 6. *The heuristic is admissible, goal-aware, and safe with the objective function O1: min* $\sum_{a \in A} CA_a$.

Proof. As demonstrated in Theorem 3, for every solution of a TOHTN planning problem, there exists a valid assignment of the ILP model, ensuring that the heuristic is safe.

The objective function (O1) minimizes the number of primitive tasks, so the heuristic value is bounded above by the length of an optimal plan, making it admissible. The artificial actions encoding the initial state and goal should be excluded from that function. Note that admissibility does not hold for the second function (O2), which minimizes the number of primitive actions plus methods that need to be applied. Thus, it estimates the distance to the goal rather than the length of a minimal plan, making it inadmissible but often more efficient for quickly finding any plan.

For goal-awareness, note that in a progression search, a search node is a goal node if and only if its task network is empty. In this case, the objective function trivially evaluates to 0. □

We assume unit action costs throughout this paper. However, it is easy to adapt the objective function to arbitrary action costs: $min \sum_{a \in A} cost(a) \cdot CA_a$, where $cost(a) \in \mathbb{R}_{\geq 0}$ are the costs associated with an action. In this case, the heuristic value is bounded above by the cost of a cost optimal plan.

According to Theorem 4 we know that P-TODR calculates the length of an optimal plan if the TOHTN planning problem is acyclic and delete-relaxed with unique tasks. The DOR heuristic additionally relaxes ordering constraints so that the underlying encoded plan could be shorter, i.e., the heuristic value can be smaller. This means that P-TODR dominates DOR in such cases.

Corollary 7. *For acyclic, delete-relaxed total-order HTN planning problems with unique tasks* P-TODR *dominates* DOR.

For the general case of TOHTN planning, however, none of the two heuristics dominates the other. Consider the example in Figure 2, showing a search node n consisting of the task network $\langle c_1, a_1 \rangle$ and the decomposition methods of the compound task c_1. The actions $a_1, \dots, a_4$ are primitive with their preconditions depicted on the left, effects on the right. Assume that the current state is $s = \emptyset$. Then,

the solution of the DOR heuristic would encode the only correct plan $\langle a_3, a_4, a_1 \rangle$ using m_2 to decompose c_1 leading to a heuristic value of $\text{DOR}(n) = 3+1 = 4$ with objective function O2. On the other hand, the solution of the P-TODR uses m_1 and only the actions a_2 and a_1 leading to $\text{P-TODR}(n) = 2 + 1 = 3$ (the two actions are possible achievers of each other because of their occurrence in m_3 and the current task network). So the DOR heuristic dominates P-TODR in this particular search node.

5 Evaluation

We evaluated our proposed heuristic for optimal planning. We integrated the heuristic into the progression-based version of the PANDA$_\pi$ system [15].[2,3] We used the currently best-performing configuration according to the last IPC in 2023 (total-order track), which is A* with loop detection [10] and dead-end analysis with look-aheads and early refinements (Dealer) [19, 21]. For completeness reasons we also included some results without the latter. The source code is available in the repository [22], which also contains the generated data and additional evaluations, including results from additional configurations and the time required for the preprocessing step of calculating achiever actions (1.18% of total time on avg.).

We run the evaluation on a machine with a Xeon E5-2660 v3 with 2.60GHz and 40 CPUs. As a benchmark set, we used all problems of the 24 domains of the IPC 2020 and the two additional ones from 2023.[4] Each planning problem was granted one core, a maximum of 8 GiB RAM, and a time limit of 1800 seconds.

We compared our heuristic P-TODR against the ILP-based heuristic DOR by Höller et al. [14]. Since we aim to find optimal plans, we used objective function O1. For each search node, both heuristics perform a reachability analysis to determine which tasks are still reachable, and then construct a new ILP model; hence, there are no incremental computations. The ILPs are solved by CPLEX using its standard parameters, but with the number of parallel threads set to one. In addition to solving the NP-complete ILP problems exactly, we also evaluated the performance when relaxing the ILP variables to real numbers, resulting in a Linear Program (LP), which is known to be solvable in P. To summarize, we evaluated (similar for DOR):

- P-TODR^{ilp} – The NP-complete ILP model solved exactly
- P-TODR^{lp} – The ILP model relaxed to an LP
- P-TODR$^{lp/ilp}_{scc}$ – Additional constraints for handling SCCs
- P-TODR$^{lp/ilp}_{noD}$ – Without using the Dealer technique

Additionally, we included the currently best-performing admissible HTN heuristic, which is the Relaxed Composition (RC) heuristic [15] in combination with the admissible classical LM-cut heuristic [7], denoted as RC(lmc).

In Table 1 we report the number of solved instances within the time and memory limits (coverage); normalized coverage, where equal significance is assigned to all domains, ensuring that domains with a multitude of instances do not overshadow those with fewer instances; and the IPC score, which is computed by $min\{1, 1 - log(t)/log(1800)\}$, where t is the time required to solve the problem in seconds. It rewards solving problems quickly.

P-TODR vs. DOR We first compare the P-TODR and DOR heuristics. Based on Table 1, we observe that both heuristics perform best when the (I)LP model is allowed to take real numbers. P-TODR^{lp} has

[2] http://panda.hierarchical-task.net
[3] https://github.com/ipc2023-htn/PandaDealer
[4] https://ipc2020.hierarchical-task.net/, https://ipc2023-htn.github.io/

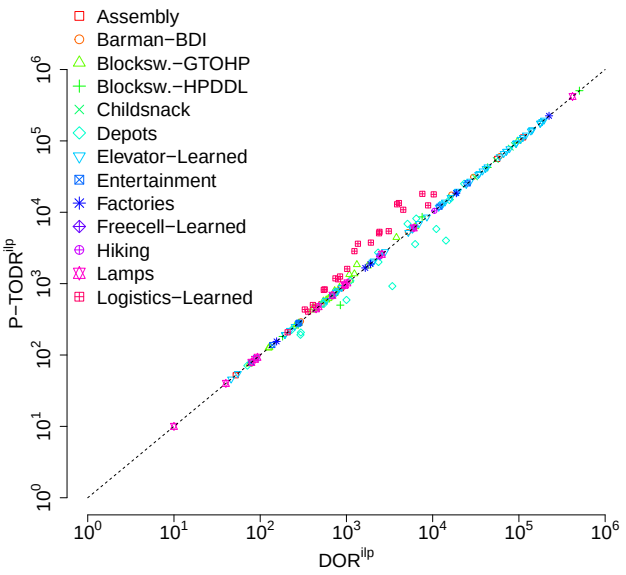

Figure 3. Number of generated search nodes using P-TODR^{ilp} versus DOR^{ilp}. Be aware of the log scale. Shows only domains 1 to 12.

a slightly higher IPC score while DOR^{lp} solves two instances more (337 vs. 339). When the NP-hard ILP model is solved (P-TODR^{ilp} vs. DOR^{ilp}), P-TODR could solve 15% more problems in total. The IPC score was improved by approx. 16%. Looking at individual domains, we can see that P-TODR^{ilp} dominates DOR^{ilp} in every single domain, both in terms of coverage as well as in terms of IPC score, though not always strictly. Looking at some specific domains, we see the largest difference in performance in the Elevator, Hiking, and Towers domains. Using the additional SCC constraints lead to a lower performance in all cases. The same holds for not using Dealer.

Despite having considerably fewer variables and constraints, P-TODR^{ilp}'s performance, in terms of calculated search nodes, is comparable to that of DOR^{ilp}, as illustrated in Figures 3 and 4. We can see that only a few problems produce more search nodes. This suggests that P-TODR's unique constraint ensuring executability, coupled with the precalculated task ordering, delivers results of similar quality to those of the DOR constraints but with faster computation, due the elimination of ILP variables. This elimination of constraints naturally shows the biggest impact in the ILP variant of the heuristic, as their evaluation is here more expensive than in the poly-computable LP variant.

P-TODR vs. RC(lmc) When comparing the P-TODR heuristic against the currently best-performing admissible HTN heuristic, results are not that clear. When looking at the total number of solved instances and the sum of IPC scores, then the RC(lmc) heuristic outperforms P-TODR$^{lp/ilp}$. RC(lmc) solved 384 problems, whereas P-TODR^{lp} solved only 337. RC(lmc) has a sum of IPC scores of 7.77, whereas P-TODR^{lp} has 6.40. However, when looking at individual domains, we can see that no heuristic dominates the other. More precisely, in 6 of 26 domains, both heuristics have the same coverage. In 13 domains RC(lmc) has higher coverage than P-TODR^{lp}, so consequently P-TODR^{lp} has higher coverage than RC(lmc) on 7 domains. The Childsnack domain stands out, where the RC heuristic failed to solve any problems, whereas P-TODR^{lp} was able to solve 6. Looking at IPC scores, values were identical in three case. Otherwise, RC(lmc) has higher scores in 15 cases, and, consequently,

Table 1. Coverage and IPC score for optimal planning, sorted by IPC score

Domain		$RC(lmc)$		P-TODR^{lp}		DOR^{lp}		P-TODR$^{lp}_{scc}$		P-TODR^{ilp}		P-TODR$^{lp}_{noD}$		DOR$^{lp}_{scc}$		P-TODR$^{ilp}_{scc}$		DOR$^{lp}_{noD}$		DOR^{ilp}		DOR$^{ilp}_{scc}$	
		Cov	IPC	Cov	IPC	Cov	IPC	Cov	IPC	Cov	IPC	Cov	IPC	Cov	IPC	Cov	IPC	Cov	IPC	Cov	IPC	Cov	IPC
Assembly	30	4	0.11	3	0.07	3	0.07	3	0.07	**5**	**0.12**	3	0.07	3	0.07	4	0.10	3	0.07	**5**	0.10	4	0.09
Barman-BDI	20	**10**	**0.32**	8	0.20	8	0.17	8	0.20	6	0.14	8	0.15	8	0.18	6	0.14	6	0.14	5	0.11	6	0.11
Blocksw.-GTOHP	30	**26**	0.68	25	**0.75**	**26**	**0.75**	22	0.60	25	0.73	25	0.70	22	0.60	22	0.58	25	0.67	24	0.71	22	0.58
Blocksw.-HPDDL	30	5	**0.12**	**6**	**0.12**	5	0.11	**6**	**0.12**	5	0.10	5	0.11	5	0.11	5	0.11	5	0.10	5	0.10	5	0.10
Childsnack	30	0	0.00	6	0.06	5	0.05	**7**	**0.07**	5	0.05	4	0.03	5	0.05	5	0.05	3	0.01	5	0.03	5	0.03
Depots	30	18	**0.55**	18	0.53	18	0.49	18	0.53	**19**	0.51	18	0.50	18	0.48	**19**	0.51	18	0.45	18	0.38	18	0.38
Elevator-Learned	147	**92**	**0.33**	73	0.27	72	0.26	77	0.27	67	0.24	64	0.22	73	0.26	70	0.25	53	0.19	55	0.20	61	0.21
Entertainment	12	5	0.42	5	0.42	5	0.42	5	0.42	**9**	**0.57**	**9**	**0.57**	5	0.42	**9**	0.51	8	0.52	8	0.53	8	0.50
Factories	20	**6**	**0.23**	**6**	0.21	**6**	0.20	**6**	0.21	5	0.20	5	0.18	**6**	0.20	5	0.20	5	0.17	5	0.18	5	0.18
Freecell-Learned	60	**0**	**0.00**	**0**	**0.00**	**0**	**0.00**	**0**	**0.00**	**0**	**0.00**	**0**	**0.00**	**0**	**0.00**	**0**	**0.00**	**0**	**0.00**	**0**	**0.00**	**0**	**0.00**
Hiking	30	6	0.05	11	0.09	9	0.08	10	0.10	12	0.14	10	0.10	9	0.08	**13**	**0.21**	9	0.08	3	0.03	9	0.07
Lamps	30	**15**	**0.46**	14	0.42	14	0.42	14	0.42	14	0.41	13	0.41	14	0.42	**15**	0.42	13	0.41	14	0.41	14	0.41
Logistics-Learned	80	**27**	**0.25**	22	0.23	22	0.22	24	0.23	22	0.20	22	0.21	22	0.22	22	0.20	22	0.20	22	0.20	22	0.20
Minecraft Pl.	20	**2**	**0.03**	1	0.01	1	0.01	1	0.00	1	0.01	1	0.01	0	0.00	0	0.00	1	0.01	1	0.00	0	0.00
Minecraft Reg.	59	33	0.33	**38**	**0.36**	37	0.32	**38**	**0.36**	**38**	0.34	**38**	**0.36**	37	0.32	**38**	0.34	37	0.32	33	0.27	33	0.27
Monroe FO	20	**19**	**0.39**	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00
Monroe PO	20	**10**	**0.16**	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00
Multiarm-Blocksw.	74	12	0.13	**17**	**0.14**	**17**	0.13	**17**	**0.14**	13	0.11	14	0.12	**17**	0.13	13	0.10	13	0.11	9	0.08	9	0.08
Robot	20	**11**	**0.55**	**11**	0.53	**11**	0.52	**11**	0.53	**11**	0.51	**11**	0.52	**11**	0.52	**11**	0.52	**11**	0.51	**11**	0.51	**11**	0.51
Rover	30	**8**	**0.22**	**8**	**0.22**	**8**	**0.22**	**8**	0.21	**8**	0.21	**8**	**0.22**	**8**	0.21	**8**	0.20	**8**	0.21	**8**	0.20	**8**	0.19
Satellite	20	6	0.21	**10**	0.28	**10**	0.28	**10**	0.28	**10**	**0.34**	**10**	0.27	**10**	0.27	**10**	0.32	9	0.26	**10**	0.29	**10**	0.27
Sharp Sat	21	**9**	**0.34**	8	0.29	8	0.28	8	0.29	6	0.20	8	0.25	8	0.28	6	0.21	8	0.24	6	0.18	6	0.18
Snake	20	**20**	**0.76**	7	0.18	13	0.28	6	0.16	4	0.13	5	0.12	11	0.24	4	0.12	9	0.20	3	0.07	3	0.07
Towers	20	**13**	**0.48**	9	0.28	10	0.32	9	0.28	6	0.19	8	0.26	10	0.32	5	0.19	9	0.29	3	0.10	3	0.12
Transport	40	10	0.15	**16**	**0.30**	15	0.29	14	0.29	15	0.28	15	**0.30**	14	0.28	14	0.26	15	0.28	13	0.24	12	0.23
Woodworking	30	**17**	**0.51**	15	0.43	16	0.43	15	0.43	16	0.40	16	0.42	16	0.43	16	0.40	16	0.43	15	0.37	15	0.37
Overall	943	**384**	**7.77**	337	6.40	339	6.34	337	6.20	322	6.17	320	6.10	332	6.08	320	5.93	306	5.86	281	5.30	289	5.17
Normalized Coverage		**10.93**		8.98		9.21		8.83		8.74		8.85		8.91		8.57		8.69		7.74		7.86	

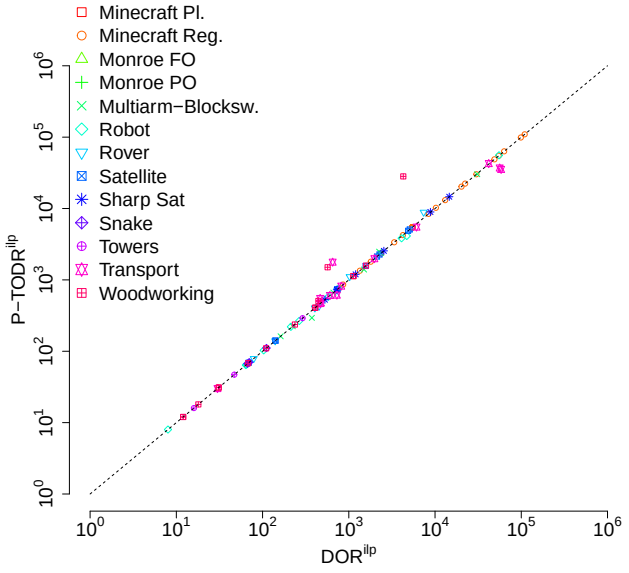

Figure 4. Number of generated search nodes using P-TODR^{ilp} versus DOR^{ilp}. Be aware of the log scale. Shows only domains 13 to 24.

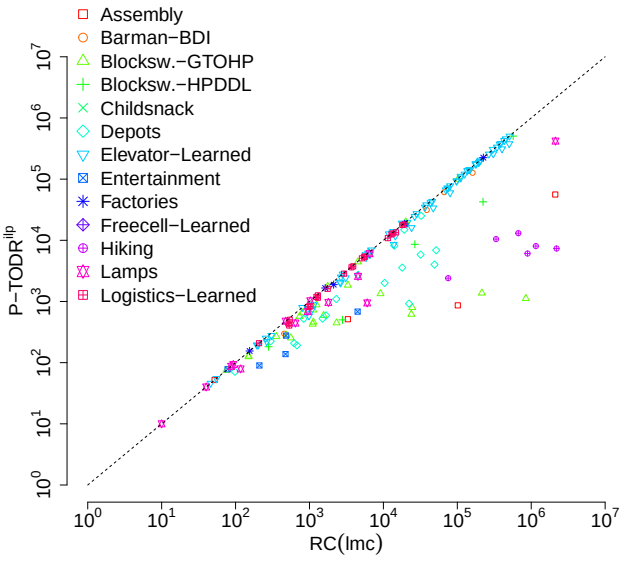

Figure 5. Number of generated search nodes using P-TODR^{ilp} versus $RC(lmc)$. Be aware of the log scale. Shows only domains 1 to 12.

P-TODR^{lp} had higher scores in 8 cases.

When looking at the informedness of the heuristics, Figures 5 and 6 reveal that P-TODR^{ilp} generally requires fewer search nodes than RC(lmc) for most problems, with significant differences observed in several cases (be aware of the log scale). This indicates that the P-TODR^{ilp} heuristic is more informed at the cost of increased computation time as expected given that the RC(lmc) heuristic runs in polynomial time, while the ILP-based heuristics are capable of encoding and solving NP-hard problems resulting in more precise heuristic values.

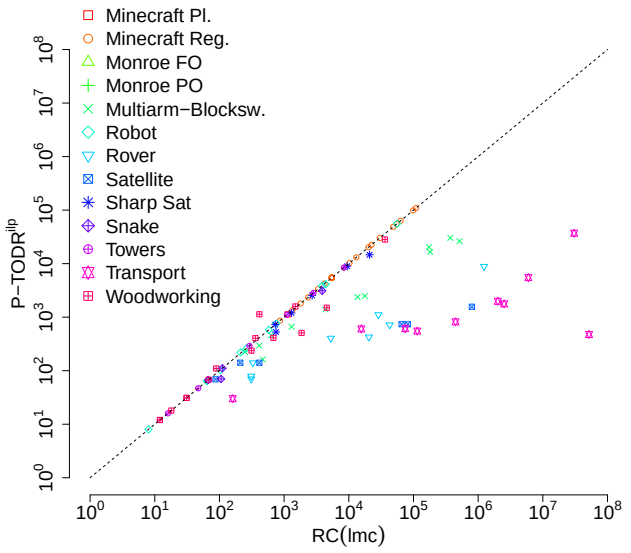

Figure 6. Number of generated search nodes using P-TODR^{ilp} versus $RC(lmc)$. Be aware of the log scale. Shows only domains 13 to 24.

6 Conclusion and Discussion

We proposed a novel (I)LP-based HTN planning heuristic tailored to total-order domains. The ordering information is calculated in advance and integrated into the (I)LP model, significantly reducing the number of constraints compared to an existing (I)LP heuristic, which ignores ordering. Empirical results indicate that the new heuristic outperforms the original one when the ILP is solved, dominating it in terms of solved instances (coverage) and IPC score on every existing total-order domain. If the ILP is relaxed to a linear programming (LP) model, the performance results are comparable with no heuristic clearly dominating the other. When comparing our NP-complete heuristic against the currently best-performing admissible HTN heuristic, RC(lmc), which can be computed in polynomial time, the results are mixed: While RC(lmc) performs better overall in terms of the sum of solved instances and IPC score, neither heuristic clearly dominates the other. There are approximately as many domains where RC(lmc) performs better as there are domains where our proposed heuristic is superior. This indicates that the higher informedness of our heuristic pays off in several domains but is too costly in others. This might pay off when deploying portfolio planners or choosing heuristics based on specific domains. It will also be interesting to see how the proposed heuristic performs with further progress on the research question on how to compute inferred preconditions and effects of compound tasks. The informedness of the heuristic depends on the amount of effects computed, so the heuristic will *automatically* become more informed when more negative effects can be identified in the preprocessing step that this heuristic and the Dealer technique depend upon.

Acknowledgements

Pascal Bercher is the recipient of an Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA), project number DE240101245, funded by the Australian Government.

References

[1] R. Alford, V. Shivashankar, U. Kuter, and D. Nau. On the feasibility of planning graph style heuristics for HTN planning. In *Proc. of ICAPS*, pages 2–10. AAAI Press, 2014.

[2] R. Alford, P. Bercher, and D. W. Aha. Tight bounds for HTN planning. In *Proc. of ICAPS*, pages 7–15. AAAI Press, 2015.

[3] G. Behnke, D. Höller, and S. Biundo. totSAT – Totally-ordered hierarchical planning through SAT. In *Proc. of AAAI*, pages 6110–6118. AAAI Press, 2018.

[4] P. Bercher. A closer look at causal links: Complexity results for delete-relaxation in partial order causal link (POCL) planning. In *Proc. of ICAPS*, pages 36–45. AAAI Press, 2021.

[5] P. Bercher, G. Behnke, D. Höller, and S. Biundo. An admissible HTN planning heuristic. In *Proc. of IJCAI*, pages 480–488. IJCAI, 2017.

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

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

[8] D. Höller. The PANDA λ system for HTN planning in the 2023 IPC. In *11th International Planning Competition: Planner and Domain Abstracts (IPC)*, 2023.

[9] D. Höller. The PANDA progression system for HTN planning in the 2023 IPC. In *11th International Planning Competition: Planner and Domain Abstracts (IPC)*, 2023.

[10] D. Höller and G. Behnke. Loop detection in the PANDA planning system. In *Proc. of ICAPS*, pages 168–173. AAAI Press, 2021.

[11] D. Höller and P. Bercher. Landmark generation in HTN planning. In *Proc. of AAAI*, pages 11826–11834. AAAI Press, 2021.

[12] D. Höller, P. Bercher, G. Behnke, and S. Biundo. A generic method to guide HTN progression search with classical heuristics. In *Proc. of ICAPS*, pages 114–122. AAAI Press, 2018.

[13] D. Höller, G. Behnke, P. Bercher, S. Biundo, H. Fiorino, D. Pellier, and R. Alford. HDDL: An extension to PDDL for expressing hierarchical planning problems. In *Proc. of AAAI*, pages 9883–9891. AAAI Press, 2020.

[14] D. Höller, P. Bercher, and G. Behnke. Delete- and ordering-relaxation heuristics for HTN planning. In *Proc. of IJCAI*, pages 4076–4083. IJCAI, 2020.

[15] D. Höller, P. Bercher, G. Behnke, and S. Biundo. HTN planning as heuristic progression search. *JAIR*, 67:835–880, 2020.

[16] T. Imai and A. 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.

[17] D. A. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proc. of AAAI*, pages 634–639. AAAI Press, 1991.

[18] C. Olz and P. Bercher. Can they come together? A computational complexity analysis of conjunctive possible effects of compound HTN planning tasks. In *Proc. of ICAPS*, pages 314–323. AAAI Press, 2023.

[19] C. Olz and P. Bercher. A look-ahead technique for search-based HTN planning: Reducing the branching factor by identifying inevitable task refinements. In *Proc. of SoCS*, pages 65–73. AAAI Press, 2023.

[20] C. Olz, S. Biundo, and P. Bercher. Revealing hidden preconditions and effects of compound HTN planning tasks – A complexity analysis. In *Proc. of AAAI*, pages 11903–11912. AAAI Press, 2021.

[21] C. Olz, D. Höller, and P. Bercher. The PANDADealer system for totally ordered HTN planning in the 2023 IPC. In *11th International Planning Competition: Planner and Domain Abstracts (IPC)*, 2023.

[22] C. Olz, A. Lodemann, and P. Bercher. Experimental results for the ECAI 2024 paper "A heuristic for optimal total-order HTN planning based on integer linear programming", 2024. doi: 10.5281/zenodo.13269291.