An ILP Heuristic for Total-Order HTN Planning

Conny Olz[1], **Alexander Lodemann**[1], **Pascal Bercher**[2]

[1]Ulm University
[2]The Australian National University (ANU)
conny.olz@uni-ulm.de, alexander.lodemann@uni-ulm.de, pascal.bercher@anu.edu.au

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 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. Our current heuristic proves to be more efficient than the one we build on, though it still performs worse than other existing (admissible) heuristics.

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 (Höller et al. 2020; Olz and Bercher 2023b) is still the most efficient approach, both for optimal and for suboptimal planning – as witnessed by the most recent IPC (Höller 2023a,b; Olz, Höller, and Bercher 2023).

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 (Bercher et al. 2017), one finds refinements to delete- and ordering-relaxed problems encoded by an Integer Linear Program (ILP) (Höller, Bercher, and Behnke 2020), another bases on landmarks (Höller and Bercher 2021), and the last – but most successful – heuristic is the relaxed composition heuristic (Höller et al. 2018, 2020), which encodes each search node into a classical problem allowing to deploy classical heuristics.

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 (Olz and Bercher 2023b), 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 and won the total-order HTN track of IPC 2023 (Olz, Höller, and Bercher 2023).

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 (Olz, Biundo, and Bercher 2021) (which also serve as the basis for the TO pruning technique (Olz and Bercher 2023b; Olz, Höller, and Bercher 2023)) and deploys them in a simplified variant of the ILP-based (NP-complete) delete- and ordering relaxation heuristic (DOF) by Höller, Bercher, and Behnke (2020). 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 ILP heuristic and the currently best-performing RC heuristic (Höller et al. 2020, 2018), using Add (Bonet and Geffner 2001) as the inner classical heuristic in satisficing planning and the RC heuristic with the admissible LM-cut (Helmert and Domshlak 2009) in optimal planning since our proposed heuristic is admissible. Our results show that while the ILP-based heuristics are not competitive with the RC(Add) heuristic[1], our new version generally outperforms DOF and also RC(lmc) in some domains in optimal planning

We hope that future work – further improvements to our technique – could bring our novel heuristic on par or even beat the RC(LM-cut) heuristic as well and discuss ideas for doing so at the end of the paper.

[1]When the DOF heuristic was published, it was on par with RC(Add) but a different (smaller) set of domains was used in the evaluation.

Theoretical Background

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

HTN Planning Formalism

Our heuristic for totally ordered HTN planning is grounded in the formalisms introduced by Geier and Bercher (2011) and Behnke, Höller, and Biundo (2018). 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 and *compound tasks* C (alternatively referred to as *abstract tasks*), and *decomposition methods* $M \subseteq C \times T^*$.[2] The collective set of tasks, both primitive and compound, is denoted by $T = A \cup C$. Actions or primitive tasks $a = (prec, add, del) \in A$ 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 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 $s \supseteq g$.

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, Biundo, and Bercher (2021). Compound tasks, both according to the formalism we base upon (Geier and Bercher 2011; Behnke, Höller, and Biundo 2018), and as described by HDDL (Höller et al. 2020), 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, Biundo, and Bercher (2021) 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, (Olz, Biundo, and Bercher 2021)) and are defined as follows:

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

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 (Olz, Biundo, and Bercher 2021). 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, Biundo, and Bercher (2021) and are also implemented in the PANDADealer planning system (Olz and Bercher 2023b; Olz, Höller, and Bercher 2023), which we employ for our evaluation.

ILP Encoding for Delete-relaxed TO-HTN Search Nodes

Höller, Bercher, and Behnke (2020) introduced the first HTN planning heuristic based on an ILP. They encode a delete and ordering free (DOF) 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, Bercher, and Behnke use the encoding by Imai and Fukunaga (2015) (for classical planning) representing a delete-relaxed planning graph. Here, we introduce a different idea outlined further below. The latter part we take from the work by Höller, Bercher, and Behnke without changes, which we recap next.

Figure 1 outlines the set of ILP variables. The model by Höller, Bercher, and Behnke for the executability of actions needs five types of variables, but we could restrict our version to just one (the first one; the second and third are used to encode the decomposition). The objective function is the same, calculating the goal distance by minimizing the number of applied primitive actions and method applications:

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

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

- $\{CA_t \mid t \in T\}$ (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, Bercher, and Behnke (2020). The first variable is renamed, the last adapted to initial task instead of task network.

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 (McAllester and Rosenblitt 1991; Bercher 2021). 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.

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, Bercher, and Behnke (2020) encode this criterion. 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, Bercher, and Behnke (2020) 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. 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 or use them since the evaluation results by Höller, Bercher, and Behnke do not show significant improvements by them.

Achiever Constraints

The ILP model by Höller, Bercher, and Behnke (2020) uses the constraints by Imai and Fukunaga (2015) 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 pre-evaluation, 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. We ended up with only one (new) constraint (c_l is a large constant):

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

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 $achiever(f, a)$. 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 $achiever(f, a) = \{a' \in A \mid f \in add(a')\}$. 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.

Determining Achiever Actions

Given a TOHTN 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 $reachable(c) = \{a \in A \mid \exists\, \bar{t} \in T^* : c \to \bar{t} \text{ and } a \in \bar{t}\}$

Algorithm 1: Calculating Predecessor Actions

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

1: $pred(a) = \emptyset$ for all $a \in A$
2: **for all** methods $m = (c, \langle t_0, \ldots, t_n \rangle) \in M$ **do**
3: **for** $i = 1$ to n **do**
4: **for all** $a \in reachable(t_i)$ **do**
5: **for** $j = i - 1$ to 0 **do**
6: $pred(a) = pred(a) \cup reachable(t_j)$

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. Given that set for every compound task, we can compute for every primitive action the set of actions that can possibly appear as predecessor 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 an 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.

"⇒" Let $a, a' \in A$ and $a' \in pred(a)$. Consider the method $m = (c, \langle t_0, \ldots, t_n \rangle) \in M$ 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 $\langle t_0, \ldots, t_n \rangle$ 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.

"⇐" 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 $m = (c, \langle t_0, \ldots, t_n \rangle) \in M$ 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 $achiever(f, a)$ based on the total-order of the method set and inferred negative effects, given in Algorithm 2.

Algorithm 2: Calculating Achiever Actions

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

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

Again, every method $m = (c, \langle t_0, \ldots, t_n \rangle) \in M$ 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 some preceding task $t_k, k < i$ deletes a precondition (according to its (inferred) delete effects), we do not consider its reachable actions nor the reachable actions of its predecessors (line 8).

Algorithm 2 runs in polynomial time 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 $achiever(a, p), pred(a)$ be calculated according to Algorithms 1 and 2. Then it holds*

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

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.

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

We can now define our first version of our heuristic based on the ILP presented in the last section with the set of possible achiever $achiever(a, p)$ calculated by Algorithm 2, which we denote h^{TOILP}.

Theorem 1. *For every solution of a TOHTN planning problem there exists a valid assignment of the ILP model.*

Proof. Höller, Bercher, and Behnke (2020) already showed for every solution of a DOF 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 to ask is whether h^{TOILP} performs some relaxations or whether all valid assignments of the ILP model correspond to some solution of an 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 (Alford, Bercher, and Aha 2015). ILPs can only solve problems in NP. Basically 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. Consider for example 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.

To overcome this limitation one could duplicate primitive and compound tasks so that every task occurs just once over 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 polynomial 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 2. *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 TOHTN problem.*

Proof Sketch. Consider an acyclic, delete-relaxed, unique tasks TOHTN problem and a valid assignment of the corresponding ILP model. According to Höller, Bercher, and Behnke (2020) 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 subset 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 recognizes 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. (2014) that acyclic and delete-relaxed (t.o.) 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 (2023a) to show NP-hardness of acyclic and regular, delete- and precondition-relaxed problems to unique tasks so that we can conclude that our studied problem is also already NP-hard (and complete).

For our situation now, it needs to be evaluated empirically whether the transformation to unique tasks pays off but a preliminary evaluation showed no positive effect. The heuristic can be used nevertheless, it just relaxes the problem a bit more in case of non-unique tasks as discussed above.

Admissibility

We saw that for every solution of a TOHTN planning problem there exist a valid assignment of the ILP model, so the heuristic is safe. The objective function of the ILP minimizes the number of primitive actions and methods that need to be applied. So it estimates the distance to the goal and not the length of a minimal plan. However that can easily be changed by setting the objective function to just minimizing the number of primitive tasks. Then the heuristic value is bounded from above by the length of an optimal plan, which makes it admissible. The artificial actions encoding the initial state and goal should be excluded in that function.

Corollary 1. *The heuristic is admissible, goal aware, and safe with the objective function* $min \sum_{a \in A} CA_a$.

Evaluation

We evaluated our proposed heuristic in satisficing planning, where one tries to find a solution as fast as possible within

a given time limit, as well as optimal planning. Therefore, we integrated the heuristic into the progression-based version of the PANDA$_\pi$ system[3][4] (Höller et al. 2020). We used the currently best-performing configuration according to the last IPC in 2023, which is GBFS (and A* for optimal planning, respectively) with loop detection (Höller and Behnke 2021) and dead-end analysis with look-aheads and early refinements (Dealer) (Olz and Bercher 2023b; Olz, Höller, and Bercher 2023). For completeness reasons we also included the results without the latter, though below we focus our report on results with Dealer since those yield overall better results.

We compared our heuristic, called TOILP, against the ILP-based heuristic DOF by Höller, Bercher, and Behnke (2020) and the currently best-performing heuristics. For satisficing planning, this is the Relaxed Composition (RC) heuristic (Höller et al. 2020) in combination with the classical Add heuristic (Bonet and Geffner 2001) as RC(Add), and for optimal planning, RC combined with LM-cut (Helmert and Domshlak 2009) as RC(lmc), which is the only other existing admissible HTN planning heuristic.

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 in the benchmark set of the IPC 2020[5]. Each planning problem was granted one core, a maximum of 8 GiB RAM, and a time limit of 1800 seconds.

Satisficing Planning

In Table 1 we report results for satisficing planning: 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.

The RC(Add) heuristic outperforms both other heuristics over all in terms of solved instances (744 versus 496 and 415, respectively) and IPC score (15.32 versus 8.21 and 7.02, respectively). When the DOF heuristic was published it was on par with the the RC(Add) heuristic on the benchmark set of that time. It is interesting to see that this picture changed completely with the new domains. There is not a single domain in which the DOF solves more instances than RC(Add). The TOILP heuristic can at least outperform RC(Add) in two domains, namely Depots and Transport, with 2 and 3 more solved problems, respectively.

Our main intention was to improve the DOF heuristic so we compare it with the TOILP next. We observe that TOILP could solve 19.5% more problems and the IPC score was improved by around 17%. Looking at individual domains, we can see that in the Logistics-Learned, Multiarm-Blocksworld, and Towers domains, TOILP solved around twice as many problems as DOF. In the Hiking domain, TOILP solved 10 problems where DOF was only able to solve 1 problem. So we can conclude that, indeed, overall the TOILP performs better than the DOF heuristic on total-order domains.

We evaluated the satisficing setting for reasons of completeness. Given that non-admissible heuristics often outperform admissible ones here, it is not too surprising that TOILP does not emerge as the best overall.

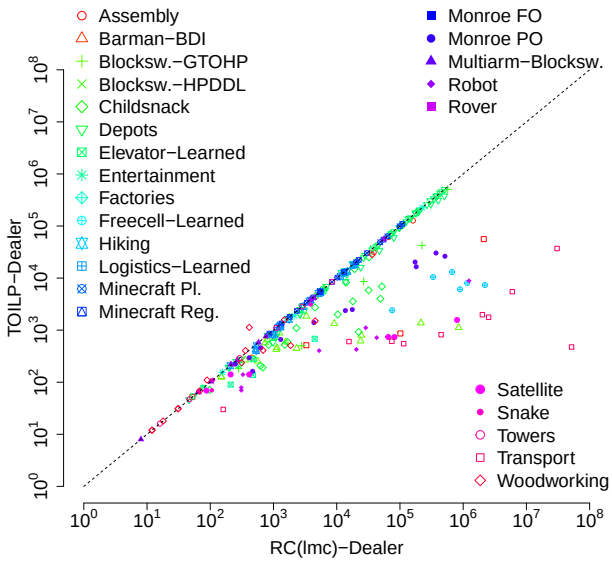

Figure 2: Number of generated search nodes for optimal planning. Be aware of the log scale.

Optimal Planning

Given the limited number of admissible HTN planning heuristics available to date the TOILP heuristic has potential in optimal planning. As indicated in Table 2, the performance gap between the RC heuristic and ILP-based heuristics is indeed narrower in optimal planning than in satisficing planning. Although the RC(lmc) heuristic still surpasses the others in terms of coverage, with scores of 360 compared to 302 and 261, and in IPC scores, with 6.93 versus 5.56 and 4.71, respectively, the TOILP heuristic demonstrates superior performance in several domains. Notably, in 13 out of 24 domains, TOILP matched or exceeded the performance of RC(lmc). The Childsnack domain stands out, where the RC heuristic failed to solve any problems, whereas TOILP and DOF solved 5.

The RC(lmc) heuristic runs in polynomial time, while the ILP-based heuristics are capable of encoding and solving NP-hard problems. From a theoretical perspective, this suggests that the ILP-based heuristics are more informed at the cost of increased computation time. While with greedy search such additional runtime often does not pay off since we just want to find any solution, in optimal planning the enhanced information can prove valuable in narrowing down the search space. This is supported by the data in Figure 2, which shows that TOILP generally requires fewer search nodes than RC(lmc) for most problems, with significant dif-

[3]http://panda.hierarchical-task.net

[4]https://github.com/ipc2023-htn/PandaDealer

[5]https://ipc2020.hierarchical-task.net/

domain		RC(Add)-Dealer		RC(Add)		TOILP-Dealer		TOILP		DOF-Dealer		DOF	
		Coverage	IPC	Coverage	IPC	Coverage	IPC	Coverage	IPC	Coverage	IPC	Coverage	IPC
Assembly	30	**30**	**0.89**	**30**	0.88	29	0.42	25	0.36	24	0.33	20	0.28
Barman-BDI	20	**16**	**0.75**	**16**	0.68	12	0.36	10	0.30	14	0.38	11	0.29
Blocksw.-GTOHP	30	**30**	0.86	**30**	**0.94**	25	0.75	25	0.71	27	0.76	25	0.68
Blocksw.-HPDDL	30	**30**	**0.76**	26	0.68	21	0.39	20	0.28	20	0.33	19	0.26
Childsnack	30	**23**	0.64	**23**	**0.65**	18	0.24	18	0.24	17	0.19	17	0.19
Depots	30	22	**0.73**	22	**0.73**	**24**	0.70	**24**	0.67	22	0.52	21	0.48
Elevator-Learned	147	**147**	**0.75**	**147**	0.60	124	0.43	108	0.35	107	0.35	95	0.30
Entertainment	12	**12**	0.94	**12**	**0.95**	**12**	0.87	**12**	0.86	**12**	0.81	**12**	0.80
Factories	20	**11**	**0.37**	8	0.32	6	0.22	5	0.17	6	0.20	4	0.15
Freecell-Learned	60	16	**0.10**	**18**	0.06	0	0.00	0	0.00	0	0.00	0	0.00
Hiking	30	**25**	0.67	**25**	**0.68**	10	0.16	10	0.22	1	0.01	1	0.01
Logistics-Learned	80	**80**	**0.79**	48	0.45	47	0.33	46	0.26	22	0.24	22	0.20
Minecraft Pl.	20	**4**	**0.08**	**4**	**0.08**	0	0.00	0	0.00	0	0.00	0	0.00
Minecraft Reg.	59	**42**	0.58	**42**	**0.59**	40	0.47	40	0.46	40	0.43	40	0.42
Monroe FO	20	**20**	**0.50**	**20**	**0.50**	0	0.00	0	0.00	0	0.00	0	0.00
Monroe PO	20	**13**	**0.28**	11	0.24	0	0.00	0	0.00	0	0.00	0	0.00
Multiarm-Blocksw.	74	**74**	**0.89**	**74**	0.80	35	0.18	14	0.10	16	0.11	12	0.08
Robot	20	**20**	**0.93**	**20**	0.91	**20**	0.75	11	0.53	**20**	0.76	**20**	0.74
Rover	30	27	0.57	**29**	**0.65**	9	0.24	9	0.24	9	0.23	9	0.23
Satellite	20	**19**	0.66	**19**	**0.68**	10	0.35	10	0.35	10	0.31	10	0.31
Snake	20	**20**	**0.91**	**20**	0.90	4	0.12	3	0.09	3	0.07	3	0.02
Towers	20	**13**	**0.49**	**13**	0.46	5	0.19	6	0.20	3	0.12	3	0.11
Transport	40	22	0.54	25	**0.58**	**28**	**0.58**	**28**	0.57	25	0.45	25	0.42
Woodworking	30	**28**	**0.66**	27	0.64	17	0.45	16	0.39	17	0.40	17	0.39
Overall	892	**744**	**15.32**	709	14.63	496	8.21	440	7.34	415	7.02	386	6.33
Normalized Coverage		**19.50**		18.85		12.23		11.03		10.89		10.24	

Table 1: Coverage and IPC score for satisficing planning

ferences observed in several cases. This indicates that the TOILP heuristic provides more precise heuristic values, although the computation time remains slightly too high, resulting in overall superior performance by RC(lmc).

Despite having considerably fewer variables and constraints than the DOF heuristic, the TOILP heuristic's performance, in terms of calculated search nodes, is comparable to that of DOF, as illustrated in Figure 3, only a few problems need more search nodes. This suggests that TOILP's unique constraint ensuring executability, coupled with the precalculated task ordering, delivers results of similar quality to those of the DOF constraints but faster.

Future Work

We already discussed some future work, which we briefly recap now together with further ideas:

- The constraints for strongly connected components, as proposed by Höller, Bercher, and Behnke (2020) in their ILP model, could be incorporated to better handle cyclic domains.
- We can introduce new tasks so that every task appears just once over all methods, which makes the calculation of achiever actions more precise. However, it also increases the model, which might slow down the ILP solver significantly.

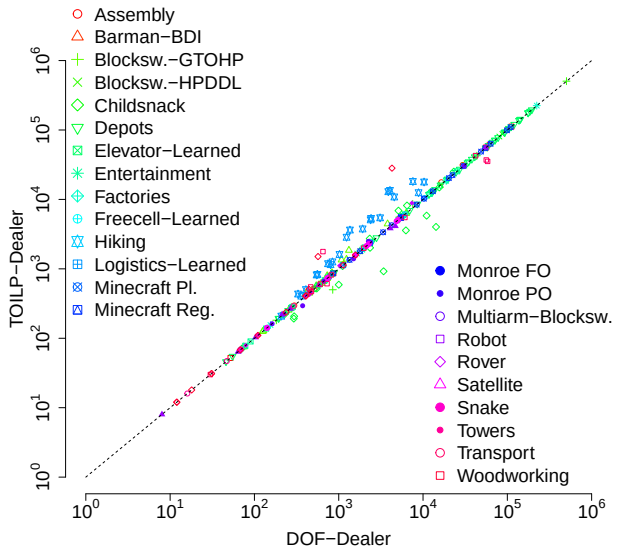

Figure 3: Number of generated search nodes for optimal planning. Be aware of the log scale.

domain		RC(lmc)-Dealer		RC(lmc)		TOILP-Dealer		TOILP		DOF-Dealer		DOF	
		Coverage	IPC	Coverage	IPC	Coverage	IPC	Coverage	IPC	Coverage	IPC	Coverage	IPC
Assembly	30	4	0.11	4	0.10	**5**	**0.12**	4	0.10	**5**	0.10	4	0.09
Barman-BDI	20	**10**	**0.32**	**10**	0.29	6	0.14	5	0.11	5	0.11	5	0.08
Blocksw.-GTOHP	30	**26**	0.68	23	0.61	25	**0.73**	25	0.66	24	0.71	23	0.62
Blocksw.-HPDDL	30	**5**	**0.12**	**5**	0.11	**5**	0.11	4	0.09	**5**	0.10	4	0.09
Childsnack	30	0	0.00	0	0.00	**5**	**0.05**	4	0.03	**5**	0.03	3	0.01
Depots	30	18	**0.55**	18	0.52	**19**	0.51	**19**	0.48	18	0.38	18	0.33
Elevator-Learned	147	92	0.33	**112**	**0.39**	67	0.24	64	0.22	55	0.20	53	0.19
Entertainment	12	5	0.42	5	0.42	**9**	**0.58**	**9**	0.57	8	0.53	8	0.52
Factories	20	**6**	**0.23**	5	0.20	5	0.20	4	0.15	5	0.18	4	0.14
Freecell-Learned	60	**0**	**0.00**	**0**	**0.00**	**0**	**0.00**	**0**	**0.00**	**0**	**0.00**	**0**	**0.00**
Hiking	30	6	0.05	6	0.05	**12**	**0.14**	5	0.04	3	0.03	2	0.02
Logistics-Learned	80	**27**	**0.25**	**27**	**0.25**	22	0.20	22	0.18	22	0.20	22	0.17
Minecraft Pl.	20	**2**	**0.03**	1	0.02	1	0.01	0	0.00	1	0.00	0	0.00
Minecraft Reg.	59	33	0.33	33	0.33	**38**	**0.35**	**38**	**0.35**	33	0.27	33	0.27
Monroe FO	20	**19**	**0.37**	12	0.23	0	0.00	0	0.00	0	0.00	0	0.00
Monroe PO	20	**10**	**0.15**	7	0.14	0	0.00	0	0.00	0	0.00	0	0.00
Multiarm-Blocksw.	74	12	**0.13**	12	0.12	**13**	0.11	8	0.08	9	0.08	8	0.07
Robot	20	**11**	**0.55**	**11**	**0.55**	**11**	0.51	**11**	0.50	**11**	0.51	**11**	0.51
Rover	30	**8**	**0.22**	**8**	0.21	**8**	0.21	**8**	0.21	**8**	0.20	**8**	0.20
Satellite	20	6	0.21	6	0.21	**10**	**0.34**	**10**	0.33	**10**	0.29	**10**	0.28
Snake	20	**20**	**0.76**	**20**	0.68	4	0.13	4	0.10	3	0.07	3	0.02
Towers	20	**13**	**0.48**	12	0.45	6	0.19	6	0.19	3	0.10	3	0.10
Transport	40	10	0.15	9	0.14	**15**	**0.28**	14	0.27	13	0.24	13	0.19
Woodworking	30	**17**	**0.51**	16	0.50	16	0.40	13	0.34	15	0.37	15	0.35
Overall	892	360	**6.93**	**362**	6.51	302	5.56	277	5.00	261	4.71	250	4.25
Normalized Coverage		**10.00**		9.33		7.99		7.30		6.99		6.66	

Table 2: Coverage and IPC score for optimal planning

- Currently we only calculate the set of achiever tasks once in the beginning. However, one could update the sets according to the reachable tasks of the current search node. Since some methods may not be reachable anymore, the sets can become smaller and more accurate. However, it needs to be evaluated whether this additional computation time pays off.
- The ILP (Integer Linear Program) can be relaxed to a LP (Linear Program), where the variables can be assigned real numbers instead of integers. LPs are solvable in polynomial time but the solution might not correspond to a plan anymore. Since the objective value of an LP is always less or equal to the one of the corresponding ILP the heuristic is still admissible. It needs to be evaluated whether the speedup of calculation time can compensate the loss of information. In the original work by Höller, Bercher, and Behnke (2020) the ILP version was slightly better.

Conclusion

We proposed a novel ILP-based HTN planning heuristic tailored to total-order domains. The ordering information is calculated in advance and integrated into the ILP model so that the number of constraints can be significantly reduced compared to an existing ILP heuristic, which ignores the ordering completely. Empirical results indicate that the new heuristic outperforms the original one clearly, dominating it in terms of solved instances (coverage) and IPC score on every existing total-order domain. When comparing our NP-complete heuristic against the currently best performing admissible HTN heuristic that can be computed in polynomial time, RC(lmc), results are mixed. When looking at the overall sum of solved instances and IPC score, RC(lmc) performs better than the one proposed. However, neither clearly dominates the other as there are approximately as many domains on which RC(lmc) performs better than than the proposed one as there are domains on which the proposed one is better. This shows the higher informedness of our proposed heuristic pays out in several domains, but it too costly in several others. It will be interesting to see how the proposed heuristic performs with further progress on the research question on how to compute inferred effects of compound tasks. The informed of the heuristic depends on the amount of effects computed, so the heuristic will *automatically* become more informed when more preconditions and 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

Alford, R.; Bercher, P.; and Aha, D. W. 2015. Tight Bounds for HTN Planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 7–15. AAAI Press.

Alford, R.; Shivashankar, V.; Kuter, U.; and Nau, D. 2014. On the Feasibility of Planning Graph Style Heuristics for HTN Planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 2–10. AAAI Press.

Behnke, G.; Höller, D.; and Biundo, S. 2018. totSAT – Totally-Ordered Hierarchical Planning through SAT. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018)*, 6110–6118. AAAI Press.

Bercher, P. 2021. A Closer Look at Causal Links: Complexity Results for Delete-Relaxation in Partial Order Causal Link (POCL) Planning. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS 2021)*, 36–45. AAAI Press.

Bercher, P.; Behnke, G.; Höller, D.; and Biundo, S. 2017. An Admissible HTN Planning Heuristic. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*, 480–488. IJCAI.

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

Geier, T.; and Bercher, P. 2011. On the Decidability of HTN Planning with Task Insertion. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 1955–1961. AAAI Press.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

Höller, D. 2023a. The PANDA λ System for HTN Planning in the 2023 IPC. In *Proceedings of the 11th International Planning Competition: Planner Abstracts – Hierarchical Task Network (HTN) Planning Track, IPC 2023*.

Höller, D. 2023b. The PANDA Progression System for HTN Planning in the 2023 IPC. In *Proceedings of the 11th International Planning Competition: Planner Abstracts – Hierarchical Task Network (HTN) Planning Track, IPC 2023*.

Höller, D.; and Behnke, G. 2021. Loop Detection in the PANDA Planning System. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS 2021)*, 168–173. AAAI Press.

Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2020. HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI 2020)*, 9883–9891. AAAI Press.

Höller, D.; and Bercher, P. 2021. Landmark Generation in HTN Planning. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI 2021)*, 11826–11834. AAAI Press.

Höller, D.; Bercher, P.; and Behnke, G. 2020. Delete- and Ordering-Relaxation Heuristics for HTN Planning. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020)*, 4076–4083. IJCAI.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2018. A Generic Method to Guide HTN Progression Search with Classical Heuristics. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS 2018)*, 114–122. AAAI Press.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2020. HTN Planning as Heuristic Progression Search. *Journal of Artificial Intelligence Research (JAIR)*, 67: 835–880.

Imai, T.; and Fukunaga, A. 2015. On a Practical, Integer-Linear Programming Model for Delete-Free Tasks and its Use as a Heuristic for Cost-Optimal Planning. *Journal of Artificial Intelligence Research (JAIR)*, 54: 631–677.

McAllester, D. A.; and Rosenblitt, D. 1991. Systematic Nonlinear Planning. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI 1991)*, 634–639. AAAI Press.

Olz, C.; and Bercher, P. 2023a. Can They Come Together? A Computational Complexity Analysis of Conjunctive Possible Effects of Compound HTN Planning Tasks. In *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS 2023)*, 314–323. AAAI Press.

Olz, C.; and Bercher, P. 2023b. A Look-Ahead Technique for Search-Based HTN Planning: Reducing the Branching Factor by Identifying Inevitable Task Refinements. In *Proceedings of the 16th International Symposium on Combinatorial Search (SoCS 2023)*, 65–73. AAAI Press.

Olz, C.; Biundo, S.; and Bercher, P. 2021. Revealing Hidden Preconditions and Effects of Compound HTN Planning Tasks – A Complexity Analysis. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI 2021)*, 11903–11912. AAAI Press.

Olz, C.; Höller, D.; and Bercher, P. 2023. The PANDADealer System for Totally Ordered HTN Planning in the 2023 IPC. In *Proceedings of the 11th International Planning Competition: Planner Abstracts – Hierarchical Task Network (HTN) Planning Track (IPC)*.