Bringing Order to Chaos – A Compact Representation of Partial Order in SAT-based HTN Planning

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

Abstract

HTN planning provides an expressive formalism to model complex application domains. It has been widely used in real-world applications. However, the development of domain-independent planning techniques for such models is still lacking behind. The need to be informed about both state-transitions and the task hierarchy makes the realisation of search-based approaches difficult, especially with unrestricted *partial* ordering of tasks in HTN domains. Recently, a translation of HTN planning problems into propositional logic has shown promising empirical results. Such planners benefit from a unified representation of state and hierarchy, but until now require very large formulae to represent partial order. In this paper, we introduce a novel encoding of *HTN Planning as SAT*. In contrast to related work, most of the reasoning on ordering relations is not left to the SAT solver, but done beforehand. This results in much smaller formulae and, as shown in our evaluation, in a planner that outperforms previous SAT-based approaches as well as the state-of-the-art in search-based HTN planning.

Introduction

In many practical applications, Hierarchical Task Network (HTN) planning (Erol, Hendler, and Nau 1996) has proven to be a useful formalism for modelling planning problems (Nau et al. 2005; Straatman et al. 2013; Champandard, Verweij, and Straatman 2009; Dvorak et al. 2014; Bercher et al. 2015; Behnke et al. 2018). It allows for specifying a primitive action theory in conjunction with a hierarchical refinement structure. As plans have to be obtained using these refinement rules they form a second means to restrict the set of solutions. In fact, these restrictions are strictly more expressive than classical (even using ADL) planning (Erol, Hendler, and Nau 1996; Höller et al. 2014; Höller et al. 2016; Bercher et al. 2016).

The hierarchy can also be used to restrict the search space, leading to very efficient domain-*configurable* planning systems (like, e.g., SHOP2 (Nau et al. 2003)). However, this increases the modelling effort severely. Recent research has focussed on domain-*independent* HTN planning, which provides more freedom and flexibility to the domain modeller, as he does not have to take the way the planner will solve the problem into account. Progress has been made using heuristic search (Höller et al. 2018) and search space pruning (Dvorak et al. 2014). However, since heuristics need to be informed about both, the hierarchy and the state-transition system, the design of HTN heuristics is difficult. Empirically, translations of HTN planning problems into propositional logic have proven successful (Behnke, Höller, and Biundo 2018a; 2018b). They have a unified representation of state transition and hierarchy and benefit from ongoing research in SAT solving. Similarly, a translation into propositional logic has been proposed for the HTN plan verification problem (Behnke, Höller, and Biundo 2017).

The first encoding of hierarchical planning was proposed by Mali and Kambhampati (1998). Their formalism does not specify initial tasks to decompose, but freely inserts (abstract) tasks. Thus it does not comply with the established HTN formalism and its definition of a solution, but is equivalent to classical planning. Further, the encoding cannot handle recursion, i.e., even with an initial task, it would allow only for less expressive models. The second encoding has been restricted to totally ordered HTNs, but allows for recursion (Behnke, Höller, and Biundo 2018a). This decreases the expressivity severely (Höller et al. 2014). The encoding of the hierarchy has been based on the Path Decomposition Tree (PDT), which represents all possible decompositions up to a certain depth. It has been combined with the Kautz and Selman encoding (1996) to represent the state transition system. Behnke, Höller, and Biundo (2018b) introduced a canonical extension to partially ordered HTNs, representing ordering relations on top of the PDT. Thus the SAT solver is tasked with all reasoning about ordering relations.

In this paper we introduce a novel encoding of *Partially Ordered HTN Planning as SAT*. Most ordering-related reasoning is done *before* creating the propositional formula – alleviating it from the SAT solver. This leads to much more succinct encodings – theoretically $\mathcal{O}(n^3)$ instead of $\Theta(n^4)$ clauses, which practically often degenerates to a formula with $\mathcal{O}(n^2)$ clauses. We further combine our hierarchy representation with the more recent $\exists$-step encoding for classical planning (Rintanen, Heljanko, and Niemelä 2006). The resulting planner outperforms existing *SAT-based* HTN planners as well as the state-of-the-art in *search-based* domain-independent HTN planning.

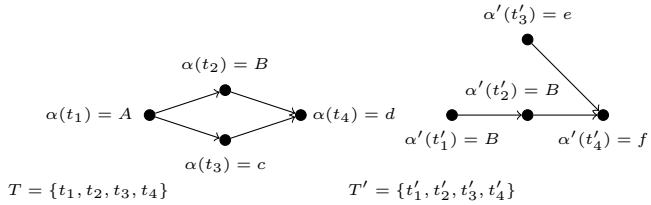

Figure 1: Two example task networks $tn = (T, \prec, \alpha)$ and $tn' = (T', \prec', \alpha')$ where the order is given by the depicted graphs. Compound task names are indicated with capital letters, while lower-case letters indicate primitive task names.

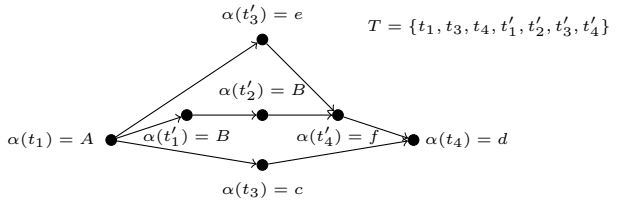

Figure 2: The task network $tn^* = (T^*, \prec^*, \alpha^*)$ resulting from applying the method (B, tn') to t_2 in tn of Fig. 1.

Preliminaries

We use the HTN formalism of Geier and Bercher (2011). Task networks represent partially ordered sets of tasks.

Definition 1 (Task Network)**.** *A* task network tn *over a set of task names X is a tuple $(T, \prec, \alpha)$, where*
- *T is a finite, possibly empty, set of tasks*
- *$\prec \subseteq T \times T$ is a strict partial order on T*
- *$\alpha : T \to X$ labels every task with a task name*

TN_X denotes the set of all task networks over task names X. Two task networks $tn = (T, \prec, \alpha)$ and $tn' = (T', \prec', \alpha')$ are *isomorphic*, written $tn \cong tn'$, iff a bijection $\sigma : T \to T'$ exists, s.t. $\forall t, t' \in T$ it holds that $(t, t') \in \prec$ iff $(\sigma(t), \sigma(t')) \in \prec'$ and $\alpha(t) = \alpha'(\sigma(t))$. Two examples for task networks are depicted in Fig. 1. The second example also demonstrates the necessity of a separate label set T, as the task name B occurs twice in tn'.

Definition 2 (HTN Planning Problem)**.** *An* HTN planning problem *is a 6-tuple $\mathcal{P} = (L, C, O, M, c_I, s_I)$, with*
- *L, a finite set of* proposition symbols
- *C, a finite set of* compound task names
- *O, a finite set of* primitive task names *with $C \cap O = \emptyset$*
- *$M \subseteq C \times TN_{C \cup O}$, a finite set of* decomposition methods
- *$c_I \in C$, the* initial task name
- *$s_I \in 2^L$, the* initial state

The state transition semantics of primitive task names $o \in O$ is that of classical planning, given in terms of a precondition-, an add-, and a delete-list: $prec(o) \in 2^L$, $add(o) \in 2^L$, and $del(o) \in 2^L$.

A solution in HTN planning is obtained by starting with the initial task and repeatedly applying *decomposition methods* until all tasks in the current task network are primitive. The notion of decomposition is defined as follows. An example can be seen in Fig. 2.

Definition 3 (Decomposition)**.** *A method $m = (c, tn_m) \in M$ decomposes a task network $tn_1 = (T_1, \prec_1, \alpha_1)$ into a task network tn_2 by replacing the task t, written $tn_1 \xrightarrow[t,m]{} tn_2$, if and only if $t \in T_1$, $\alpha_1(t) = c$, and $\exists tn' = (T', \prec', \alpha')$ with $tn' \cong tn_m$ and $T' \cap T_1 = \emptyset$, where*

$$tn_2 = (T'', \prec_X \cap (T'' \times T''), (\alpha_1 \cup \alpha' \setminus \{(t, c)\})) \text{ with}$$

$$T'' = (T_1 \setminus \{t\}) \cup T'$$

$$\prec_X = \{(t_1, t_2) \in T_1 \times T' \text{ with } (t_1, t) \in \prec_1\} \cup \{(t_1, t_2) \in T' \times T_1 \text{ with } (t, t_2) \in \prec_1\} \cup \prec_1 \cup \prec'$$

We write $tn_1 \to_D^* tn_2$ if tn_1 can be decomposed into tn_2 using an arbitrary number of decompositions.

The solutions to a planning problem are defined as follows.

Definition 4 (Solution)**.** *A task network tn_S is a solution to a planning problem $\mathcal{P}$, if and only if*
(1) $\exists$ a linearisation of the tasks of tn_S, executable in s_I
(2) $(\{id_1\}, \emptyset, \{(id_1, c_I)\}) \to_D^* tn_S$
$\mathfrak{S}(\mathcal{P})$ denotes the sets of all solutions of $\mathcal{P}$, respectively.

Decomposition Trees (DTs) are witnesses showing that a task sequence π is a solution to the planning problem (Geier and Bercher 2011). They describe how π can be obtained from the initial abstract task via decomposition. An example of a DT can be seen as a subgraph in Fig. 3.

Definition 5 (Decomposition Tree)**.** *Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ be an HTN problem. A decomposition tree T is a 5-tuple $T = (V, E, \prec, \alpha, \beta)$, where*
1. *(V, E) is a directed tree with a root-node r.*
2. *$\prec \subseteq V \times V$ is a strict partial order on V and is inherited along the tree, i.e., if $a \prec b$, then $a' \prec b$ and $a \prec b'$ for any children a' of a and b' of b.*
3. *$\alpha : V \to C \cup O$ assigns each inner node an abstract task and each leaf a primitive task and $\alpha(r) = c_I$.*
4. *$\beta : V \to M$ assigns each inner node a method.*
5. *for all inner nodes $v \in V$ with $\beta(v) = (c, tn)$, $tn = (T_{tn}, \prec_{tn}, \alpha_{tn})$, and children $D = \{c_1, \ldots, c_n\}$, it holds that $c = \alpha(v)$. Further, a bijection $\phi : D \to T_{tn}$ must exist with $\alpha(c_i) = \alpha_{tn}(\phi(c_i))$ for all c_i, and $c_i \prec c_j$ iff $\phi(c_i) \prec_{tn} \phi(c_j)$.*

$\prec$ may not contain orderings apart from those induced by 2. or 5. The yield $yield(T)$ of T is the task network induced by the leafs of T, i.e. V, α, and $\prec$ restricted to these leafs.

Geier and Bercher (2011) showed the following theorem:

Theorem 1. *Given a planning problem $\mathcal{P}$, then for every task sequence π the following holds:*
There exists a decomposition tree (DT) T s.t. π is a linearisation of $yield(T)$ if and only if $\pi \in \mathfrak{S}(\mathcal{P})$.

This means that instead of finding a solution to the planning problem $\mathcal{P}$, we can equivalently try to find a DT whose yield has an executable linearisation.

Path Decomposition Trees and SAT

The new representation of partial order of HTN planning problems and its SAT encoding will be combined with a previous encoding of HTN decomposition (Behnke, Höller, and

Biundo 2018b). As such, we start by reviewing this encoding, denoted as SAT-tree. It is based on two key ideas:

1. Bound the maximum depth K of decomposition (and iterate to achieve completeness)
2. Compute a tree that contains all possible DTs of depth $\leq K$ as its subtrees

This tree – the Path Decomposition Tree (PDT) – is the basis for a compact encoding of all possible decompositions into a single SAT formula. Since every solution to a planning problem corresponds to a DT (with executable yield), a solution can be expressed equivalently by selecting a subtree T of the PDT and checking that T is a DT. Decision variables represent the selected subtree T while the requirements for a DT (see Def. 5) are formulated as a propositional formula.

The translation starts by constructing the PDT for a depth bound K. The PDT is a tree that contains all DTs of depth $\leq K$ as subtrees – with the additional condition that the root of these subtrees is the root of the PDT. Behnke, Höller, and Biundo ignored the order contained in methods completely when constructing the PDT and check the ordering afterwards in the formula. Ignoring order eases the construction, but ignores important information in the domain, as we will show in this paper. We denote with $\mathfrak{L}(V, E)$ the set of leafs of a tree (V, E).

Definition 6. *Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ be a planning problem and K a height bound. A PDT P_K of height K is a triple $P_K = (V, E, \alpha)$ where*

1. *(V, E) is a tree of height $\leq K$ with the root node r.*
2. *$\alpha : V \to 2^{C \cup O}$ assigns each node a set of possible tasks.*
3. *$\alpha(r) = \{c_I\}$*
4. *for all inner nodes $v \in V$, for each abstract task $c \in \alpha(v) \cap C$, and for each method $(c, tn) \in M$ with $tn = (T_{tn}, \prec_{tn}, \alpha_{tn})$, there exists a subset $D^v_{(c,tn)} = \{v_1, \ldots, v_{|T_{tn}|}\}$ of v's children, such that a bijection $\phi^v_{(c,tn)} : D^v_{(c,tn)} \to T_{tn}$ exists with $\alpha_{tn}(\phi^v_{(c,tn)}(d)) \in \alpha(d)$ for all $d \in D^v_{(c,tn)}$*
5. *$\forall v \in \mathfrak{L}(V, E)$: either $\alpha(v) \subseteq O$ or the height of v is K.*

We denote with $\mathfrak{L}(P_K) = \mathfrak{L}(V, E)$ the leafs of the PDT.

The labelling function α provides for every node v the set of tasks with which this node can be labelled in a DT. Requirement 4 of Def. 6 enforces the mechanism of decomposition. Whenever we can label a node v with an abstract task c there might be a DT that contains v labelled with c. If so, the node v must have children in the DT representing the tasks obtained by applying a method to c. The PDT enforces that for every applicable method children of v exist that can serve as the children of v with the correct labels in a DT. An example for a PDT and a DT as its subtree can be found in Fig. 3.

The PDT for a given planning problem $\mathcal{P}$ and depth bound K is *not* uniquely defined. A concrete PDT can be constructed by specifying both $D^v_{(c,tn)}$ – the children of each node representing the subtasks of a method (c, tn) applied to v, and $\phi^v_{(c,tn)}$ – the function deciding which child is used to represent which task in the decomposition. Based on them, a PDT can be constructed by expanding its current leafs until the necessary depth has been reached. The encoding of PDTs into propositional logic is correct for every PDT, i.e., one should choose a PDT leading to an easily decidable formula. The procedure that, given the label set $\alpha(v)$ of a node, computes all $D^v_{(c,tn)}$ and $\phi^v_{(c,tn)}$ for applicable methods (c, tn) is called *child arrangement*. Behnke, Höller, and Biundo (2018b) used a greedy child arrangement ignoring the order in methods to compute their PDTs with the goal of minimising the number of label-sets $\alpha(v)$ for all children.

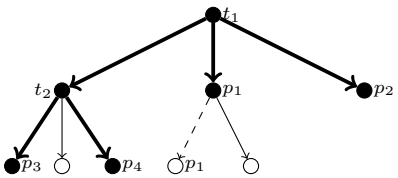

Figure 3: An example PDT, a DT as its subgraph (nodes filled), and the extension for primitive tasks (dashed line).

Based on a PDT, one can construct a propositional formula that is satisfiable if and only if a DT with height $\leq K$ and executable yield exists (Behnke, Höller, and Biundo 2018a; 2018b). Their construction comprises two parts: one representing the DT as a subtree of the computed PDT and one ensuring that the yield of the represented DT is executable (see conditions 1 and 2 of Def. 4). The first formula uses only two types of decision variables

- t^v – v is part of the DT and labelled with t, i.e., $\alpha(v) = t$.
- m^v – method m was applied to node v, i.e., $\beta(v) = m$

Their formula ensures that a satisfying valuation forms a DT. To ease testing executability it further ensures that whenever a primitive task is assigned to an inner node of the PDT, it is inherited to one of its children. Thus the tasks of the yield of the represented DT are exactly the tasks assigned to leafs of the PDT. An example for this can be seen in Fig.3. The primitive p_1 is assigned to an inner node and is inherited to its left child.

The second part of the formula ensures that the yield of the DT has an executable linearisation. This is done by choosing such a linearisation and checking its executability using the classical planning formula of Kautz and Selman (1996). This formula expresses a plan as a sequence of timesteps t. Note that in the version of the encoding used by Behnke, Höller, and Biundo at most a single action can be executed at each timestep. A task t being executed at timestep i is represented by the variable $t@i$. The number of timesteps is chosen as the number of leafs of the PDT, which we denote with $L = |\mathfrak{L}(P_K)|$. The formula by Behnke, Höller, and Biundo (2018b) matches the tasks assigned to the leafs $\mathfrak{L}$ of the PDT to these timesteps. Matching a leaf node v to timestep i is represented by the variable $\overline{vi}$. To ensure that this matching creates a valid linearisation, the following conditions are asserted in the propositional formula:

1. every leaf and every timestep is matched at most once
2. a leaf is matched if and only if it is assigned a task
3. the task assigned to a leaf must appear at the timestep that the leaf is matched to
4. tasks are only present at a matched timesteps

5. the chosen linearisation is consistent with the order induced on the leafs by the applied decomposition methods

For condition 1 any of the known encodings of the at-most-one constraint can be used. In our experiments, we have used the sequential encoding (Sinz 2005). We will use the notation $\mathbb{M}(V)$ to refer to a formula that expresses that at most one of the decision variables V is true at a time. The first constraint can be ensured by the following formula:

$$F_1 = \bigwedge_{i=1}^{L} \mathbb{M}(\{\overline{vi} \mid v \in \mathfrak{L}(P_K)\}) \wedge \bigwedge_{v \in \mathfrak{L}(P_K)} \mathbb{M}(\{\overline{vi} \mid 1 \leq i \leq L\})$$

Next, they define the decision variables $\mathtt{a}^v$, which are true if the leaf node v contains any action, i.e., is **a**ctive.

$$F^* = \bigwedge_{v \in \mathfrak{L}(P_K)} \left[\left(\neg \mathtt{a}^v \rightarrow \bigwedge_{o \in \alpha(v)} \neg o^v \right) \wedge \left(\mathtt{a}^v \rightarrow \bigvee_{o \in \alpha(v)} o^v \right) \right]$$

For conditions 2, 3, and 4, they use the following formulae:

$$F_2 = \bigwedge_{v \in \mathfrak{L}(P_K)} \left[\left(\neg \mathtt{a}^v \rightarrow \bigwedge_{1 \leq i \leq L} \neg \overline{vi} \right) \wedge \left(\mathtt{a}^v \rightarrow \bigvee_{1 \leq i \leq L} \overline{vi} \right) \right]$$

$$F_3 = \bigwedge_{v \in \mathfrak{L}(P_K)} \bigwedge_{t \in \alpha(v)} \bigwedge_{1 \leq i \leq L} t^v \wedge \overline{vi} \rightarrow t@i$$

$$F_4 = \bigwedge_{1 \leq i \leq L} \left[\left(\bigwedge_{v \in \mathfrak{L}(P_K)} \neg \overline{vi} \right) \rightarrow \left(\bigwedge_{t \in O} \neg t@i \right) \right]$$

Checking condition 5 is the main difficulty of the encoding. Since the order of all methods was ignored when constructing the PDT, it has to be traced inside the formula. Behnke, Höller, and Biundo (2018b) added variables $\mathtt{b}^v_{v'}$ indicating that the leaf v must occur before the leaf v' in the linearisation. These variables can be maintained by using the m^v decision variables. Based on them, the following formula checks for every pair of matched leafs and timesteps whether their relative ordering is forbidden by a $\mathtt{b}^v_{v'}$ variable.

$$F_5 = \bigwedge_{1 \leq i \leq L} \bigwedge_{i < i' \leq L} \bigwedge_{v \in \mathfrak{L}(P_K)} \bigwedge_{v' \in \mathfrak{L}(P_K)} (\overline{vi} \wedge \overline{v'i'}) \rightarrow \neg \mathtt{b}^{v'}_{v}$$

Behnke, Höller, and Biundo (2018b) proved that this encoding is correct and complete for any given PDT. Their encoding proved efficient in an empirical evaluation and outperformed existing HTN planning techniques for satisficing planning.

Despite its success, it has two major disadvantages. First, the size of the formula is $\Theta(n^4)$ in the size of the leafs of the PDT, i.e., the plan length, due to F_5. Second, the encoding does not take advantage of modern encoding techniques for classical planning, which were shown to outperform the encoding by Kautz and Selman (1996). We show how to overcome these issues and thereby improve the performance of SAT-based HTN planning even further.

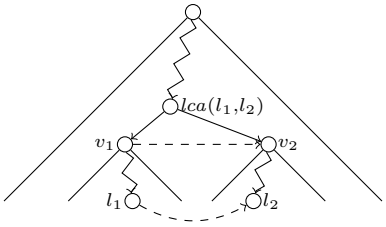

Figure 4: Order between two leafs of a PDT and their implication for order inside a method.

Solution Order Graphs

The original encoding has $O(n^4)$ clauses due to the formula F_5, which performs reasoning about the order between the leafs of the PDT *inside* the formula. We generate the PDT while simultaneously performing reasoning on order *before* creating the formula thus easing reasoning for the SAT solver.

Consider two leafs $l_1, l_2 \in \mathfrak{L}(P_K)$ of the PDT (see Fig. 4). The ordering between them is determined by the method applied to their least common ancestor $lca(l_1, l_2)$ in the PDT. More precisely, the order between l_1 and l_2 is the one between their ancestors v_1 and v_2, which are direct children of $lca(l_1, l_2)$, as determined by the applied method. In the SAT-tree encoding, this can be any ordering ($v_1 \prec v_2$, $v_2 \prec v_1$, or no order between them), depending on the chosen decomposition method. We ensure by construction of the PDT that irrespective of the method chosen for $lca(l_1, l_2)$, the order between v_1 and v_2 is always the same (provided that both are part of the selected DT). If so, the order between all leafs below v_1 and all leafs below v_2 will be fixed and the same as the one between v_1 and v_2. Since the assignment of method's subtasks to children in the PDT is done by the child arrangement, we require the following property for it, that ensures the described property.

Definition 7 (Order-Consistent Child Arrangement)**.** *Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ be a planning problem. Let v be an inner node of a PDT $P_K = (V, E, \alpha)$ and $D^v_{(c,tn)}$ and $\phi^v_{(c,tn)} : D^v_{(c,tn)} \rightarrow T(tn)$ its child arrangement function. They induce the following order $\prec^*_v$ on v's children:*

$$v_1 \prec^*_v v_2 \Leftarrow \exists c \in \alpha(v) \exists (c, (T, \prec, \alpha)) \in M \exists t_1, t_2 \in T : \\ t_1 \prec t_2 \wedge \phi^v_{(c,tn)}(v_1) = t_1 \wedge \phi^v_{(c,tn)}(v_2) = t_2$$

The child arrangement is order-consistent iff $\prec^*$ is acyclic and for all methods $(c, (T, \prec, \alpha))$ with $c \in \alpha(v)$:

$$\forall t_1, t_2 \in T : t_1 \prec t_2 \Leftrightarrow {\phi^v_{(c,tn)}}^{-1}(t_1) \prec^*_v {\phi^v_{(c,tn)}}^{-1}(t_2)$$

The order of all applicable methods for an inner node v induces an ordering of all its children, which can be interpreted as a directed graph. In the above definition we require that this graph is acyclic, i.e., it is a valid partial order. Consider as an example an inner node for which two methods are applicable resulting in the task networks tn and tn' shown in Fig. 1. Due to the ordering of both task networks, it is impossible to find an order-consistent child arrangement with only four children while five are sufficient. The minimally

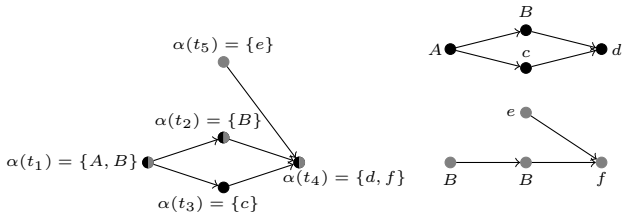

Figure 5: An order-consistent child arrangement for an inner node of a PDT with two applicable decomposition methods, resulting in tn and tn'. The mapping of the two task networks to the children (i.e. $\phi^{v}_{(c,tn)}$) is indicated with patterns.

possible child arrangement is depicted in Fig. 5, where we show the five required children, their label sets $\alpha(v)$, and the partial order induced onto them by the child arrangement.

If we construct the PDT using an order-consistent child arrangement, the order of leafs, i.e., tasks in a solution, will be uniquely determined. This order is compactly described in a graph called the Solution Order Graph (SOG) $\mathcal{S}(P_K)$. The SOG represents exactly those ordering constraints that need to be checked inside the propositional formula. These constraints are furthermore independent from the chosen decomposition methods and can thus be checked statically.

Definition 8 (Solution Order Graph)**.** *Given a PDT $P_K = (V, E, \alpha)$ that was constructed using an order-consistent child arrangement. Its Solution Order Graph $\mathcal{S}(P_K) = (\mathfrak{L}(V, E), \prec_L)$ is given by*

$$\begin{aligned}\prec_L = \{(l_1, l_2) \mid \exists v_1, v_2 \text{ children of } lca(l_1, l_2),\\ v_1 \prec^*_{lca(l_1,l_2)} v_2,\\ v_1 \text{ ancestor of } l_1, \text{ and } v_2 \text{ ancestor of } l_2\}\end{aligned}$$

By construction we know that if two leafs l_1, l_2 have tasks assigned to them, i.e., are part of the selected DT, their order in the yield of the DT will be the one between l_1 and l_2 in the SOG.

Theorem 2. *Given a PDT $P_K = (V, E, \alpha)$ that was constructed using an order-consistent child arrangement. Let l_1, l_2 be two leafs of a selected DT T as a subtree of P_K. The order of l_1, l_2 in $yield(T)$ will be the same as in $\mathcal{S}(P_K)$.*

Proof. Let P_K be a path decomposition tree and T be a decomposition tree that is a rooted subtree of P_K. Let further be l_1 and l_2 two leafs of T. Primitive tasks assigned by the DT T to any inner node are inherited in P_K towards the leafs. This can equivalently be seen as extending the DT T by repeating leafs containing these tasks. Thus we can w.l.o.g. assume that l_1 and l_2 are also leafs of P_K.

Next, we show that any order between l_1 and l_2 was introduced by the method applied to their least common ancestor in the DT T. Consider any method m applied to a node v in T that is not an ancestor of l_1 nor of l_2. Let tn be the task network to which m is applied[1]. According to Def. 3, m can only change ordering constraints that were related to the task it decomposes, any other in tn remain unchanged. Any changed order will relate l_1 and l_2 (or their ancestors) to a direct child of v, i.e., cannot introduce order between l_1 and l_2 (or its ancestors). Newly introduced orderings will only influence descendants of v, i.e., neither l_1 nor l_2. As such, m cannot introduce order between l_1 and l_2.

Consider any node v that is an ancestor of l_1 (l_2) but not of l_2 (l_1, respectively). Let v' be the ancestor of l_2 in the task network tn and v^* the child of v that is an ancestor of l_1. Any order between v and v' is replaced by the same order between v^* and v' while no new ordering constraints can be added between them. As such, this method cannot introduce the ordering between l_1 and l_2, but must only maintain it.

Any method applied to an ancestor v of l_1 and l_2 above $lca(l_1, l_2)$ cannot distinguish between l_1 and l_2 as they are still represented by a single task at this point. As such, it cannot introduce any order between them, nor can any other method apart from the one applied to $lca(l_1, l_2)$.

The order introduced by the method applied to $lca(l_1, l_2)$ on the respective ancestors v_1 and v_2 that are children of v, is inherited towards the leafs of the tree by any method application, as shown above. As such, if an ordering was introduced here, it is still present between l_1 and l_2. In conclusion, the order between l_1 and l_2 is the same as between their respective ancestors v_1 and v_2 that are children of v.

What remains to show is that the order between l_1 and l_2 as induced by the method applied to $lca(l_1, l_2)$ is the same as in the SOG $\mathcal{S}(P_K)$. The child arrangement used to construct P_K induces an order $\prec^*_{lca(l_1,l_2)}$ on the children of $lca(l_1, l_2)$. According to Def. 8 the order between v_1 and v_2 under $\prec^*_{lca(l_1,l_2)}$ is the same as between l_1 and l_2 in $\mathcal{S}(P_K)$.

Let α_T be the assignment of nodes of the DT T to their tasks. Let $m = (c, tn)$ be the method applied to the task $c = \alpha_T(lca(l_1, l_2))$. Let t_1 and t_2 be the tasks with $\phi_m^{lca(l_1,l_2)^{-1}}(t_1) = v_1$ and $\phi_m^{lca(l_1,l_2)^{-1}}(t_2) = v_2$, i.e., those that are mapped by the child arrangement ϕ to the children v_1 and v_2 that are ancestors of l_1 and l_2. The order between l_1 and l_2 should be the same as between t_1 and t_2. Since we are considering a DT T as a subtree of a PDT P_K, the method m was taken into account when constructing the PDT ($\alpha_T(lca(l_1, l_2)) \in \alpha_{P_K}(lca(l_1, l_2))$). As the child arrangement used to construct the PDT P_K is order-consistent, we can use Def. 7 to conclude that t_1 and t_2 have the same order as v_1 and v_2 and thus l_1 and l_2. □

Constructing Child Arrangements

In the previous section we have only stated requirements to the child arrangement s.t. we are able to compute a SOG in conjunction with a PDT. We have not given a method to actually compute such a child arrangement. Unfortunately the child arrangement is again not uniquely defined. By adding new and separate children for every applicable method, we could easily create an order-consistent child arrangement. Such an arrangement would, however, not be very efficient as the resulting PDT would be extremely large. It is, however, not clear how an optimal child arrangement looks like. It is uncertain whether it is better to have more leafs with

[1] A decomposition tree T can equivalently be seen as sequences of task networks, where each is created from the next by decomposition. The first task network is the one containing only the initial task and the last one is $yield(T)$.

smaller α sets or fewer leafs with larger α sets – which is the decision that the child arrangement makes. We presume that having fewer leafs is advantageous, as it will minimise the size of the resulting propositional formula.

To further analyse the child arrangement, we first transform it into a more abstract graph problem. Given an inner node v of a PDT, which is labelled with $\alpha(v)$, all methods $m_i = (c, tn_i)$ for $c \in \alpha(v)$ are potentially applicable. We assign to each such task network $tn_i = (T_i, \prec_i, \alpha_i)$ a graph $G(m_i) = (T_i, \prec_i)$ representing its order. These graphs are acyclic and transitively closed. We are now looking for a single graph $G^* = (V^*, E^*)$ which is transitively closed, s.t. all $G(m_i)$ are induced subgraphs of G^*. Given such a graph and the respective mappings $\phi_i : T_i \to V^*$, we can easily construct the child arrangement from it. The set of children will be V^*, $D^v_{m_i} = \{\phi_i(t) \mid t \in T_i\}$, and $\phi^v_{(c,tn)} = \phi_i^{-1}$. Since all $G(m_i)$ are *induced* subgraphs, we will fulfil the main property of Def. 7, and as G^* is transitively closed, the resulting order $\prec^*_v$ will be acyclic. We show that minimising the size of V^*, i.e., minimising the number of children for an inner node, is $\mathbb{NP}$-complete.

Definition 9 (TRANSITIVE INDUCED SUBGRAPH)**.** *Let $G_i = (V_i, E_i)$ be a family of n transitively closed DAGs and $K \in \mathbb{N}$.* TRANS-IND-SUBGRAPH *is to decide whether a graph G with at most K vertices exists, s.t. every G_i is an induced subgraph of G.*

Theorem 3. TRANS-IND-SUBGRAPH *is $\mathbb{NP}$-complete.*

Proof. Membership: Guess a graph G with $k = \min\{\sum_i |V_i|, K\}$ vertices[2]. Since G can have at most $\mathcal{O}(k^2)$ edges, this can be done in quadratic time. Checking whether G is transitively closed requires cubic time. Next, we loop over all G_i and guess for each an injective function $\mu : V_i \to V$ and check whether G_i is an induced subgraph of G under μ. This loop needs cubic time, as it runs linearly often and requires a quadratic check (one per edge) per graph G_i.

Hardness: We reduce from the subgraph isomorphism problem (Garey and Johnson 1979, GT48). Let $I_1 = (V_1^I, E_1^I)$ and $I_2 = (V_2^I, E_2^I)$ be two undirected graphs with $|V_1^I| \leq |V_2^I|$. W.l.o.g. we assume that neither I_1 nor I_2 contain isolated vertices (i.e. those without a connected edge), else they could be removed. We have to decide whether I_1 is isomorphic to a subtraph of I_2. We construct two graphs $G_1 = (V_1^G, E_1^G)$ and $G_2 = (V_2^G, E_2^G)$ with

- $V_i^G = V_i^I \cup E_i^I$ and
- $E_i^G = \{(v, e) \mid v \in V_i^I, e \in E_i^I, \text{ and } v \in e\}$

Clearly, G_1 and G_2 are transitively closed. Then I_1 is isomorphic to a subgraph of I_2, iff TRANS-IND-SUBGRAPH is true for the family G_1, G_2 and $K = |V_2^G|$.

$\Rightarrow$: Let I_2' be a subgraph of I_2 that is isomorphic to I_1 under the isomorphism $\phi : I_1 \to I_2'$. We select $G = G_2$ (i.e., G is transitively closed) and show that G_1 is isomorphic to an induced subgraph of G_2. This subgraph is $G_2' = (V_2^{G'}, E_2^{G'})$ and is defined as

- $V_2^{G'} = \{\phi(v) \mid v \in V_1^I\} \cup \{(\phi(u), \phi(v)) \mid \{u, v\} \in E_1^I\}$
- $E_2^{G'} = \{(v, e) \mid v, e \in V_2^{G'} \text{ and } v \in e\}$

We choose $\mu(v) = \begin{cases} \phi(v) & \text{if } v \in V_1^I \\ (\phi(u_1), \phi(u_2)) & \text{if } v = \{u_1, u_2\} \in E_2^I \end{cases}$ as the isomorphism from G_1 to G_2', showing that TRANS-IND-SUBGRAPH is true.

$\Leftarrow$: Let $G = (V, E)$ be the transitively closed induced supergraph of both G_1 and G_2 with $|V| \leq |V_2^G|$. Thus, $G_2 \cong G$, and w.l.o.g. $G_2 = G$ and we have an injective homomorphism $\mu : V_1^G \to V_2^G$. We can choose the bijection $\phi = \{(v_1, v_1') \mid v_1 \in V_1^I \text{ and } \mu(v_1) = v_1'\}$. We have to show that the domain of ϕ is V_2^I, i.e., that for every $v \in V_1^I$, μ maps it to a vertex in V_2^I and not in E_2^I (which are also vertices of G_2). I_1 does not contain isolated vertices, v has at least one outgoing edge in V_1^G. Since μ is a homomorphism, $\mu(v)$ must also have an outgoing edge. As only vertices $v' \in V_2^I$ have outgoing edges in G_2, $\mu(v) \in V_2^I$. Thus ϕ maps I_1 to a subgraph of I_2. We lastly, have to show that ϕ is also an isomorphism. If $e = \{v_1, v_2\} \in E_1^I$, then $(v_1, e), (v_2, e) \in E_1^G$ and thus $(\mu(v_1), \mu(e)), (\mu(v_2), \mu(e)) \in E_2^G$, as μ is a homomorphism. By construction of G_2, we have $\{v_1, v_2\} \in E_2^I$. The inverse holds if $e \notin E_1^I$, as μ is a homomorphism. □

[2]We cannot guess a graph of size K, as K is encoded logarithmically, i.e., the size of G would be exponential in the input.

Due to this result, we have – for the time being – opted to compute the supergraph G^* in a greedy fashion. Given all $G(m_i)$, we start with $G^* = G(m_1)$ and try to merge all other graphs $G(m_i)$ with $i \geq 1$ into G^*. For that purpose we maintain both the actual graph G^* and a list containing all forbidden edges, i.e., those that cannot be inserted into G^* or else the already processed graphs would not be induced subgraphs any more. In the beginning this list contains all edges not contained in $G(m_1)$. We do the merging of a new $G(m_i)$ node-by-node. We choose repeatedly a non-merged node v^i of $G(m_i)$ and check all nodes v^* in G^* whether merging v^i with v^* would violate ordering constraints. For that we have to check the edges to all vertices v^j already merged with their counterpart v_j^*: if the edge (v_j^*, v^*) exists in G^* and (v^i, v^j) not in $G(m_i)$ or (v^i, v^j) does in $G(m_i)$, but is forbidden in G^*, then merging is not allowed. If there are multiple possible merge candidates, we choose randomly. If there is none, we add a new vertex to G^*. In both cases we update the edge-set and the list of forbidden edges accordingly. After we have merged all $G(m_i)$, we use the mechanism described at the beginning of this section to compute the child arrangement.

Exploiting SOGs

Having computed the PDT P_K and extracted $\mathcal{S}(P_K)$ using the above described greedy child arrangement, we can exploit the structural information it exposes. Let α_{P_K} be the labelling function of P_K. For a SOG $\mathcal{S}(P_K)$ and a leaf l, we write $N^+_{\mathcal{S}(P_K)}(l)$ to denote the direct successors of l in the transitive reduction of $\mathcal{S}(P_K)$, i.e., the version of $\mathcal{S}(P_K)$ in which all transitive edges have been removed.

We propose a new encoding for constraint number 5, i.e., F_5, whose original version consists of $\Theta(n^4)$ clauses (Behnke, Höller, and Biundo 2018b). It asserted for all possible pairs of matchings of leafs l_1, l_2 to timesteps

$i < j$ that if l_1 is matched to i and l_2 to j, the order of the leafs in $yield(T)$ for the represented DT T does not imply $l_2 \prec l_1$, which would be violated by this pair of matchings. Instead we check the following condition: If a leaf l is matched to a timestep i, successors of l in $\mathcal{S}(P_K)$ can only be matched to positions *after* i, or equivalently, it is forbidden to match them to a position before i. Checking this condition is sufficient: if a leaf l_1 is matched to a timestep i, then any leaf l_2 with $l_1 \prec l_2$ cannot be matched to a timestep j with $j < i$, which is what the old F_5 enforced. For determining the set of leafs occurring after any given leaf l_1, we can use the SOG $\mathcal{S}(P_K)$ due to Thm. 2. We construct a replacement for F_5 by modelling this reduced condition as follows: we first introduce new decision variables.

- f_i^l – matching the leaf l to timestep i is forbidden

We then split the replacement for F_5 into four parts.

$$F_5 = \bigwedge_{l \in \mathfrak{L}(P_K)} \bigwedge_{1 \leq i \leq L} f_1(l,i) \wedge f_2(l,i) \wedge f_3(l,i) \wedge f_4(l,i)$$

The first asserts that if leaf l is matched to timestep i, no direct successor of l in $\mathcal{S}(P_K)$ can be matched to its direct predecessor $i-1$. f_2 and f_3 will extend this transitively.

$$f_1(l,i) = \text{if } i = 1 \text{ then } true \text{ else } \bigwedge_{l' \in N^+_{\mathcal{S}(P_K)}(l)} \overline{li} \rightarrow f^{l'}_{i-1}$$

f_2 extends "forbiddenness" from a single leaf l transitively to all its successors in $\mathcal{S}(P_K)$.

$$f_2(l,i) = \bigwedge_{l' \in N^+_{\mathcal{S}(P_K)}(l)} f_i^l \rightarrow f_i^{l'}$$

f_3 extends "forbiddenness" from a timestep i to all its predecessors. Thereby the f_i^l variables model the condition above.

$$f_3(l,i) = \text{if } i = 1 \text{ then } true \text{ else } f_i^l \rightarrow f_{i-1}^l$$

f_4, lastly, enforces that the restrictions modelled by f_i^l are actually respected by he matching variables $\overline{li}$, thus achieving a correct implementation of the above condition.

$$f_4(l,i) = f_i^l \rightarrow \neg\overline{li}$$

We call this encoding SAT-F. The new formula has only $\mathcal{O}(n^2\Delta^+(S))$ many clauses – where $\Delta^+(S)$ is the maximum out-degree of any node in the transitive reduction of S. In the worst case, $\Delta^+(S)$ can be n, i.e., in the worst case the encoding has $\mathcal{O}(n^3)$ clauses. However, in practice $\Delta^+(S)$ is often small and the sum of all direct successors is relatively small, i.e., constant, making the encoding $\mathcal{O}(n^2)$ in practice. To conclude the presentation of the encoding, we formally show that the SAT-F encoding is correct and complete.

Theorem 4. *The SAT-F encoding generated based on a P_K constructed by an order-consistent child arrangement for depth K has a satisfiable valuation iff the planning problem $\mathcal{P}$ has a solution with a decomposition tree of height $\leq K$.*

Proof. Let $\mathcal{F}$ be the propositional formula generated using the SAT-F encoding based on a P_K constructed by an order-consistent child arrangement for depth K. It consists of four conjunctive parts: the unaltered decompositional formula $\mathcal{F}_D$ by Behnke, Höller, and Biundo (2018a), the unaltered order formulae F_1, F_2, F_3, and F_4 by Behnke, Höller, and Biundo (2018b), the encoding for primitive executability by Kautz and Selman (1996), and our formula F_5. Note that the first two formulae were constructed using a PDT P_K computed by our new child arrangement.

Completeness: Let $\pi \in \mathfrak{S}(\mathcal{P})$ be a solution to $\mathcal{P}$ with a decomposition tree T of height $\leq K$. Since we have constructed the PDT P_K using a valid child arrangement, i.e., so that P_K is actually a PDT according to Def. 6, $\mathcal{F}_D$ has a satisfying valuation that represents the decomposition tree T and assigns the tasks in $yield(T)$ to the leafs of P_K (Behnke, Höller, and Biundo 2018a, Thm. 3). Further, there is a valuation of $F_1, \ldots, F_4$ and the formula for primitive executability that represents a mapping of the leafs of P_K to timesteps s.t. the resulting sequence of tasks is executable (Kautz and Selman 1996; Behnke, Höller, and Biundo 2018b, Thm. 3). Note that this theorem already guarantees that the assignment of leafs to timesteps is a valid linearisation of $yield(T)$. What remains to show is that there is a satisfying valuation of F_5. Let for every leaf l of the represented DT T be $\tau(l)$ the timestep it is mapped to. We then set f_i^l to true for all l and $i < \tau(l)$. This valuation satisfies each conjunct of F_5.

- For every leaf l matched to a timestep $\tau(i)$ (i.e. $\overline{l\tau(i)}$ is true) consider any direct successor l' in the SOG $\mathcal{S}(P_K)$. By Thm. 2 we know that l' is ordered after l in $yield(T)$, thus it is matched to a timestep $> \tau(i)$. Thus $f^{l'}_{\tau(l)-1}$ is true, fulfilling f_1.
- If f_i^l is true, l is matched to a timestep after i. By Thm. 2 we know that any successor l' of l in $\mathcal{S}(P_K)$ occurs after l in $yield(T)$ and has thus be matched to a timestep $> \tau(l) > i$. Thus $f_i^{l'}$ is true, fulfilling f_1.
- By choice of our valuation f_3 holds.
- If f_i^l is true, l is matched to a timestep $> i$, i.e., not i, making $\overline{li}$ false.

Correctness: Let β be a satisfying valuation of the propositional formula. We know that β represents for $\mathcal{F}_D$ a decomposition tree T and assigns $yield(T)$ to the leafs of P_K (Behnke, Höller, and Biundo 2018a, Thm. 3). Likewise, we know that β represents for $F_1, \ldots, F_4$ a matching of $yield(T)$ to timesteps and that $yield(T)$ is executable in the chosen order (Kautz and Selman 1996; Behnke, Höller, and Biundo 2018b, Thm. 3).

What remains to show is that the chosen matching is a valid linearisation of $yield(T)$. Assume that this is not the case. Then two leafs l_1 and l_2 in $yield(T)$ exist that are ordered $l_1 \prec l_2$ in $yield(T)$, but are matched to timesteps $i > j$. As such $\overline{l_1 i}$ and $\overline{l_2 j}$ are true. Since $l_1 \prec l_2$ in $yield(T)$ there must be a path in $\mathcal{S}(P_K)$ from l_1 to l_2 (Thm. 2). Let $p = (l_1, l_1^*, \ldots, l_n^*, l_2)$ be this path. Since β satisfies f_1 and $\overline{l_1 i}$ is true, $f^{l_1^*}_{i-1}$ must be true, as l_1^* is a direct successor of l_1 in $\mathcal{S}(P_K)$. Since β satisfies f_2 we can conclude via induction that all $f^{l_i^*}_{i-1}$ are true and finally that $f^{l_2}_{i-1}$ is true. Since β satisfies f_3 we can conclude via induction that all $f_k^{l_2}$ with

$k \leq i-1$ are true. Especially $f_j^{l_2}$ will thus be true. Since β satisfies f_4, $\overline{l_2 j}$ must be false, which is a contradiction. □

Allowing Parallelism

With the presented encoding we have significantly lowered the number of clauses. Both SAT-tree and SAT-F are, however, still using a fairly old propositional encoding for checking executability of the chosen linearisation, namely that of Kautz and Selman (1996). The high efficiency of modern SAT-based classical planners is to a large extent based on them allowing for parallel action execution. This is represented in the encoding by multiple $t@i$ atoms being true for one timestep i. The state-of-the-art in these encodings is the ∃-step encoding by Rintanen, Heljanko, and Niemelä (2006). It uses the same general structure as the Kautz and Selman formula. As such we can simply replace the Kautz and Selman formula with the ∃-step formula.

The formula that matches the leafs of the PDT to timesteps cannot match two leafs to the same timestep, i.e. forbids any parallelism. We can remove this constraint by removing the first conjunct from the F_1 formula. It asserts that for every timestep i at most one $\overline{li}$ atom can be true. After the removing these clauses, the atoms $\overline{li}$ still represent a matching that respects all ordering constraints with the sole change that multiple leafs can be matched to the same timestep.

This however leads to an incorrect encoding. It is now allowed to match two leafs l_1 and l_2 to the same timestep i while l_1 and l_2 are labelled with the same task t. As a result the solution will contain the task t only once, despite the HTN domain having forced us to execute it twice.

To forbid this situation, we define new variables $\overline{vti}$ stating that leaf l is matched to timestep i and contains task t:

$$\bigwedge_{v \in \mathfrak{L}(P_K)} \bigwedge_{t \in \alpha(v)} \bigwedge_{i=1}^{l} \overline{vi} \wedge t^v \rightarrow \overline{vti}$$

We then replace the first conjunct of F_1 with the following formula, ensuring correctness of the matching under parallelism.

$$\bigwedge_{i=1}^{l} \bigwedge_{t \in O} \mathbb{M}(\{\overline{vti} \mid v \in \mathfrak{L}(P_K)\})$$

This replacement can be applied to SAT-tree and SAT-F, which will lead to the encodings SAT-tree ∃ and SAT-F ∃.

Evaluation

We have conducted an empirical evaluation of our planner to show that it performs favourably compared to other HTN planning systems. The code of our planner will be published.

Domains. Due to the absence of a standardised set of benchmark domains, we have compared our planner against the state-of-the-art in SAT based HTN planning on the domains used in their evaluations. All of these domains are freely available for download.

Planners. Each planner was given 10 minutes runtime and 4 GB RAM per instance on an Intel Xeon E5-2660. We have compared our new encodings against previous ones, as well as against the following state-of-the-art HTN planners:

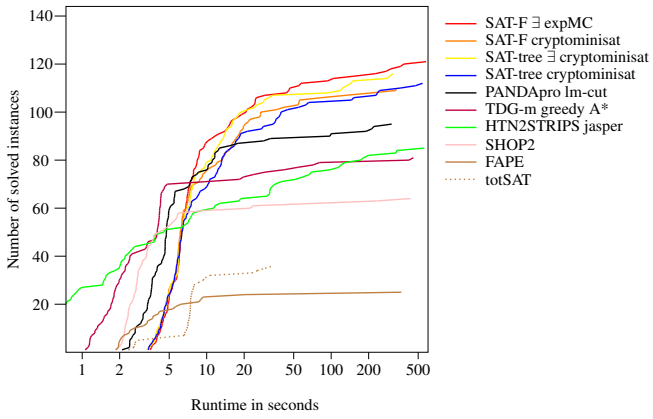

Figure 6: Runtime vs number of solved instances per planner

- SHOP2 (Nau et al. 2003),
- FAPE (Dvorak et al. 2014),
- PANDA with the TDG_m and TDG_c heuristics (Bercher et al. 2017) using greedy A*,
- HTN2STRIPS (Alford et al. 2016),
- PANDApro (Höller et al. 2018) using greedy A* and the ADD, FF and lm-cut heuristics,
- totSAT (Behnke, Höller, and Biundo 2018a).

FAPE – according to the description in its paper – does not support recursive domains. Thus, we ran it only on the domains SATELLITE, WOODWORKING, and ROVER, which are the non-recursive domains in our evaluation. Similarly, totSAT is only applicable to domains where all methods are totally-ordered. As such, it was run only on those 36 instances which are totally-ordered. The HTN2STRIPS planner translates an HTN planning problem into a sequence of classical planning problems, which it passes to a classical planner. We have tested the original planner from the paper, jasper (Xie, Müller, and Holte 2014), as well as the best planners from the agile and satisficing tracks of IPC 2018: Fast Downward Stone Soup (Seipp and Röger 2018), saarplan (Fickert et al. 2018), and LAPKT-BFWS-Preference (Frances et al. 2018). Lastly, we have also included the best known SAT-based classical planner MpC (Rintanen 2014), since using both the HTN2STRIPS translation and Madagascars translation, i.e., the ∃-step encoding, in a row would also constitute a propositional encoding for HTN planning.

We have included both the previous encoding for partially-ordered domains by Behnke, Höller, and Biundo (2018b), as well as their encoding for totally ordered domains (Behnke, Höller, and Biundo 2018a). Note that the latter encoding cannot be used for partially ordered domains. Thus it was only executed on the domains UM-TRANSLOG and ENTERTAINMENT, as well as on those five instance of SATELLITE which don't contain partial order. Further note, that in their evaluation partially-ordered instances were manually changed to be totally ordered.

We have compared all four encodings – SAT-tree, SAT-F, SAT-tree ∃, and SAT-F ∃ – with the same set of solvers. For all encodings, we have used the same scheme for con-

	#instances	SAT-F ∃ expMC	SAT-F ∃ MapleLCM	SAT-F ∃ CaDiCaL	SAT-F ∃ cryptominisat	SAT-F expMC	SAT-F MapleLCM	SAT-F CaDiCaL	SAT-F cryptominisat	SAT-tree ∃ expMC	SAT-tree ∃ MapleLCM	SAT-tree ∃ CaDiCaL	SAT-tree ∃ cryptominisat	SAT-tree expMC	SAT-tree MapleLCM	SAT-tree CaDiCaL	SAT-tree cryptominisat	PANDApro lm-cut	PANDApro FF	PANDApro ADD	TDG-m greedy A*	TDG-c greedy A*	HTN2STRIPS jasper	HTN2STRIPS FD-SS 2018	HTN2STRIPS Saarplan	HTN2STRIPS LAPKT-BFWS	HTN2STRIPS MpC	SHOP2	FAPE	totSAT
UM-TRANSLOG	22	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	19	17	17	17	6	**22**	-	19/19
SATELLITE	25	**25**	**25**	**25**	**25**	24	24	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	24	23	**25**	21	23	19	14	12	0	22	22	5/5
WOODWORKING	11	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	10	9	9	8	10	5	5	5	5	4	8	0	-
SMARTPHONE	7	**7**	**7**	**7**	**7**	**7**	**7**	6	**7**	6	6	**7**	**7**	6	6	6	**7**	5	5	5	5	5	6	6	5	5	4	4	-	-
PCP	17	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	11	**12**	11	**12**	11	**12**	9	10	11	9	8	3	3	3	3	0	0	-	-
ENTERTAINMENT	12	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	11	11	**12**	9	9	5	5	5	4	4	5	-	**12**/12
ROVER	20	10	**11**	9	8	5	6	4	4	4	4	4	6	4	4	4	5	4	3	4	2	2	5	5	4	4	4	3	3	-
TRANSPORT	30	**22**	20	20	20	15	14	15	17	**22**	20	19	21	15	15	15	18	9	11	7	1	1	19	17	13	13	3	0	-	-
total	144	**121**	120	118	117	108	108	107	110	114	112	111	116	106	107	106	112	95	95	93	81	78	85	77	66	63	25	64	25/56	36/36

Table 1: Number of solved instances per planner per domain. Maxima are indicated in bold.

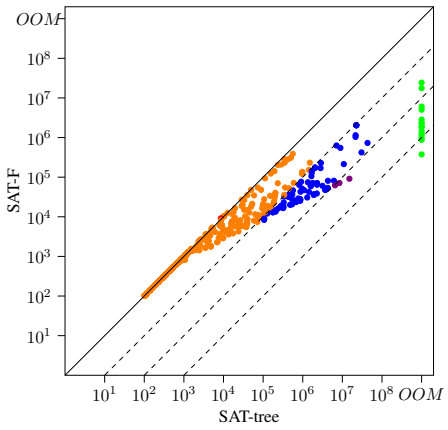

Figure 7: Number of clauses in instances compared per encoding, axes are scaled logarithmically. Colours indicate orders of magnitude. OOM = Out-Of-Memory

structing and evaluating the formulae. We always start with a depth bound of $K = 1$, run the solver until either SAT or UNSAT is returned, and in case of the latter increase K by 1. We have tested the best performing solvers of the SAT Competition 2018. Amongst them, expMC (Chowdhury, Müller, and You 2018), cryptominisat5.5 (Soos 2018), CaDiCaL (Biere 2018), and MapleLCMDistChronoBT (Ryvchin and Nadel 2018) have shown to be the best performing ones and have thus been included in this evaluation.

Results. We only present information about coverage, runtime, and relative sizes of encodings in the main paper.

In Figure 7 we present a scatter plot showing the number of clauses needed to encode each problem for every attempted depth bound K for the SAT-tree and SAT-F encodings. Note that even though we have reduced the size of the encoding from $\mathcal{O}(n^4)$ to $\mathcal{O}(n^3)$ in theory, an empirical evaluation is still warranted. In both cases, the size of the encoding depends on the number of leafs of the constructed PDT. Since we require an order-consistent child arrangement when constructing our PDTs, they might have more leafs than those constructed with the original method by Behnke, Höller, and Biundo (2018a). This however can – counter-intuitively – also lead to a decrease in the number of leafs, due to a "better" selection of label sets for the children. On the evaluated domains the number of leafs does not differ significantly between the two methods. Out of 269 PDTs constructed using the two methods, 68 differed in size. The highest increase in size was 9.5%, the highest decrease 14.2%. In absolute terms, the highest increase was an additional 16 leafs and the highest decrease was 5 leafs. We can see that SAT-F has by construction fewer clauses and that the difference can reach up to two orders of magnitude. Further, there are several instances where the SAT-F formula could be constructed (and solved) while the SAT-tree encoding caused an out-of-memory. E.g. this is the case in 14 out of 20 instances of the rover domain.

In Tab. 1 we show the number of instances solved per planner and domain. Fig. 6 shows the solved instances depending on runtime, where we show for each propositional encoding only the best performing SAT solver and for the HTN2STRIPS encoding only the best performing classical planner, jasper. Unfortunately, the reduction in number of clauses between the SAT-tree and SAT-F encodings alone does not improve the performance, as we hoped for. Depending on the solver, the coverage either rises by 1-2 instances or falls by 2. However, combining the ∃-step encoding with SAT-tree improves coverage by between 4 and 8 instances. Interestingly, if both improvements are combined, coverage increases by 5 to 15 over the base encoding, by 1 to 8 over the SAT-tree ∃, and by 7 to 13 over the SAT-F encoding.

The presented encodings outperform all other HTN planners. This is true for planners based on heuristic search (PANDA and PANDApro), search space pruning (FAPE), and translation into classical planning (HTN2STRIPS). As a side note, we also see that the performance of recent classical planners on translated HTN problems has degraded compared to the four year old jasper. The poor performance of MpC is due to an interaction between its disabling graph and the HTN2STRIPS encoding forbidding all parallelism.

Conclusion

In this paper we introduced a new method for constructing Path Decomposition Trees such that we can extract a fixed order of their leafs. This order is expressed in the Solution Order Graphs (SOG). Based on it, we have introduced a novel propositional encoding SAT-F ∃ for HTN planning

that is much more compact than previous ones. Our planner using this new encoding outperforms all state-of-the-art HTN planners. We think that the SOG is an interesting starting point for further investigations into the solutions of HTN planning problems and could allow for further exploitation of domain structure while planning.

Acknowledgments

This work was done within the technology transfer project "Do it yourself, but not alone: Companion-Technology for DIY support" of the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG). The industrial project partner is the Corporate Research Sector of the Robert Bosch GmbH.

References

Alford, R.; Behnke, G.; Höller, D.; Bercher, P.; Biundo, S.; and Aha, D. W. 2016. Bound to plan: Exploiting classical heuristics via automatic translations of tail-recursive HTN problems. In *Proc. of ICAPS*.

Behnke, G.; Schiller, M.; Kraus, M.; Bercher, P.; Schmautz, M.; Dorna, M.; Minker, W.; Glimm, B.; and Biundo, S. 2018. Instructing novice users on how to use tools in DIY projects. In *Proc. of IJCAI-ECAI*.

Behnke, G.; Höller, D.; and Biundo, S. 2017. This is a solution! (... but is it though?) – verifying solutions of hierarchical planning problems. In *Proc. of ICAPS*.

Behnke, G.; Höller, D.; and Biundo, S. 2018a. totSAT – Totally-ordered hierarchical planning through SAT. In *Proc. of AAAI*.

Behnke, G.; Höller, D.; and Biundo, S. 2018b. Tracking branches in trees – A propositional encoding for solving partially-ordered HTN planning problems. In *Proc. of ICTAI*.

Bercher, P.; Richter, F.; Hörnle, T.; Geier, T.; Höller, D.; Behnke, G.; Nothdurft, F.; Honold, F.; Minker, W.; Weber, M.; and Biundo, S. 2015. A planning-based assistance system for setting up a home theater. In *Proc. of AAAI*.

Bercher, P.; Höller, D.; Behnke, G.; and Biundo, S. 2016. More than a name? On implications of preconditions and effects of compound HTN planning tasks. In *Proc. of ECAI*.

Bercher, P.; Behnke, G.; Höller, D.; and Biundo, S. 2017. An admissible HTN planning heuristic. In *Proc. of IJCAI*.

Biere, A. 2018. CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT entering the SAT competition 2018. In *Proc. of SAT Competition 2018*.

Champandard, A.; Verweij, T.; and Straatman, R. 2009. The AI for Killzone 2's multiplayer bots. In *Proc. of GDC*.

Chowdhury, M. S.; Müller, M.; and You, J.-H. 2018. Description of expsat solvers. In *Proc. of SAT Competition 2018*.

Dvorak, F.; Bit-Monnot, A.; Ingrand, F.; and Ghallab, M. 2014. A flexible ANML actor and planner in robotics. In *Proc. of PlanRob*.

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

Fickert, M.; Gnad, D.; Speicher, P.; and Hoffmann, J. 2018. Saarplan: Combining saarland's greatest planning techniques. In *IPC2018 – Classical Tracks*.

Frances, G.; Geffner, H.; Lipovetzky, N.; and Ramirez, M. 2018. Best-first width search in the IPC 2018: Complete, simulated, and polynomial variants. In *IPC2018 – Classical Tracks*.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability. A Guide to the Theory of NP-Completeness*.

Geier, T., and Bercher, P. 2011. On the decidability of HTN planning with task insertion. In *Proc. IJCAI*.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language classification of hierarchical planning problems. In *Proc. of ECAI*.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2016. Assessing the expressivity of planning formalisms through the comparison to formal languages. In *Proc. of ICAPS*.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, B. 2018. A generic method to guide HTN progression search with classical heuristics. In *Proc. of ICAPS*.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. of AAAI*.

Mali, A., and Kambhampati, S. 1998. Encoding HTN planning in propositional logic. In *Proc. of AIPS*.

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

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Wu, D.; Yaman, F.; Muñoz-Avila, H.; and Murdock, J. 2005. Applications of SHOP and SHOP2. *Intelligent Systems, IEEE* 20:34–41.

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 170(12-13):1031–1080.

Rintanen, J. 2014. Madagascar: Scalable planning with SAT. In *The 2014 IPC*.

Ryvchin, V., and Nadel, A. 2018. Maple_LCM_Dist_ChronoBT: Featuring chronological backtracking. In *Proc. of SAT Competition 2018*.

Seipp, J., and Röger, G. 2018. Fast downward stone soup 2018. In *IPC2018 – Classical Tracks*.

Sinz, C. 2005. Towards an optimal CNF encoding of boolean cardinality constraints. In *Proc. of CP 2005*.

Soos, M. 2018. The CryptoMiniSat 5.5 set of solvers at the SAT competition 2018. In *Proc. of SAT Competition 2018*.

Straatman, R.; Verweij, T.; Champandard, A.; Morcus, R.; and Kleve, H. 2013. *Game AI Pro: Collected Wisdom of Game AI Professional*. chapter Hierarchical AI for Multiplayer Bots in Killzone 3.

Xie, F.; Müller, M.; and Holte, R. 2014. Jasper: The art of exploration in greedy best first search. In *The 2014 IPC*.