Tracking Branches in Trees – A Propositional Encoding for Solving Partially-Ordered HTN Planning Problems

Gregor Behnke
Institute of Artificial Intelligence
Ulm University
gregor.behnke@uni-ulm.de

Daniel Höller
Institute of Artificial Intelligence
Ulm University
daniel.hoeller@uni-ulm.de

Susanne Biundo
Institute of Artificial Intelligence
Ulm University
susanne.biundo@uni-ulm.de

***Abstract*—Planning via SAT has proven to be an efficient and versatile planning technique. Its declarative nature allows for an easy integration of additional constraints and can harness the progress made in the SAT community without the need to adapt the planner. However, there has been only little attention to SAT planning for hierarchical domains. To ease encoding, existing approaches for HTN planning require additional assumptions, like non-recursiveness or totally-ordered methods. Both limit the expressiveness of HTN planning severely. We propose the first propositional encodings which are able to solve general, i.e., partially-ordered, HTN planning problems, based on a previous encoding for totally-ordered problems. The empirical evaluation of our encoding shows that it outperforms existing HTN planners significantly.**

***Index Terms*—planning, hierarchical planning, SAT**

I. Introduction

Hierarchical Task Network (HTN) planning [1] is a versatile planning formalism, which has been used in many practical applications [2]–[5]. It extends classical planning by introducing abstract tasks in addition to primitive (classical) actions. They represent portfolios of more complex courses of action which – if executed – achieve the abstract task. Decomposition methods map abstract tasks to partially-ordered sets of other tasks (that might be primitive or abstract) – and by that express the connection between higher- and lower-levels of action abstraction. Decomposition is continued until all tasks are primitive and these actions can be executed in the initial state. This decompositional structure is a powerful way to describe the set of possible solutions, making HTN planning more expressive than classical planning [1], [6], [7]. To solve HTN planning problems, fast and domain-independent planning systems are required that are informed about both – hierarchy and state. But as of now, the research in this area lacks behind that in classical planning. Most current HTN planners are based on heuristic search, as in classical planning. In classical planning, SAT-based planning has also proven to be highly efficient and has advantages compared to planning via heuristic search. Most notably, SAT-based planners benefit from future progress in SAT research without the need to adapt the planner – simply replacing the solver is sufficient. Also propositional encodings are easily extendable, e.g., to add further constraints like goals formulated in LTL. Lastly propositional logic seems to be a suitable means to solve HTN planning problems, as verifying solutions was shown to be $\mathbb{NP}$-complete [8] and solving it via SAT has proven to be efficient [9].

In HTN planning, there has been little research on SAT-based techniques. Most importantly, there is no SAT-based HTN planner capable of handling all HTN planning problems. There are only two restricted encodings, one by Mali and Kambhampati [10] – which (among other restrictions) cannot handle recursion, and one by Behnke, Höller, and Biundo [11] – which cannot handle partial order in methods, but can handle recursion. Both restrictions limit the expressiveness of HTN planning severely [1], [6] and limit the domain-modeller's freedom unnecessarily. We present the first encoding that can handle all propositional HTN planning problems.

We will show how the encoding of Behnke, Höller, and Biundo [11] can be adapted such that it can also be applied to partially ordered domains. Since in that case, any ordering information in the encoding is lost, we propose a mechanism for representing the ordering constraints contained in the domain by additional decision variables. Since the order between two primitive tasks can only originate from a single method, this encoding is fairly compact.

Our empirical evaluation compares our encoding against state-of-the-art HTN planners. Here, we have considered combinatorial HTN planning problems, and not those where the HTN is hand-coded to help the planner find a solution. Our SAT-planner outperforms existing HTN planning techniques on these domains, some of them significantly.

First we introduce HTN planning formally and discuss related work. Then, we review the concept of totally-ordered Path Decomposition Trees and the SAT formula based on them. In section five, we introduce the concept of partially-ordered Path Decomposition Trees and present our SAT formula that can be used for planning in such domains. In the following chapter we describe the evaluation we conducted.

II. PRELIMINARIES

We use the HTN formalism of Geier and Bercher [12], where plans (partially ordered sets of task) are represented by task networks.

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 the task names X. We write $T(tn) = T$, $\prec(tn) = \prec$ and $\alpha(tn) = \alpha$ for a task network $tn = (T, \prec, \alpha)$. 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 $\alpha(t) = \alpha'(\sigma(t))$. Next we define the restriction notation.

Definition 2 (Restriction)**.** *Let* $R \subseteq D \times D$ *be a relation,* $f : D \to V$ *a function and* tn *be a task network. Then:*

$$R|_X = R \cap (X \times X) \qquad f|_X = f \cap (X \times V)$$
$$tn|_X = (T(tn) \cap X, \prec(tn)|_X, \alpha(tn)|_X)$$

An HTN planning problem is defined as follows.

Definition 3 (Planning Problem)**.** *A* planning problem *is a 6-tuple* $\mathcal{P} = (L, C, O, \gamma, M, c_I, s_I)$*, 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* $C \cap O = \emptyset$
- $\gamma : O \to 2^L \times 2^L \times 2^L$*, defining the preconditions and effects of each primitive task*
- $M \subseteq C \times TN_{C \cup O}$*, 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 an precondition-, an add-, and a delete-list:* $\gamma(o) = (prec(o), add(o), del(o))$*. A primitive task is applicable in a state* $s \subseteq L$ *iff* $prec(o) \subseteq s$ *and its application results in the state* $\delta(s, o) = (s \setminus del(o)) \cup add(o)$*. A sequence of primitive tasks* $o_1, \ldots, o_m$ *is applicable in a state* s_0 *iff there exist states* $s_1, \ldots, s_n$*, each* o_i *is applicable in* s_{i-1}*, and* $\delta(s_{i-1}, o_i) = s_i$*. We define* $M(c) = \{(c, tn) \mid (c, tn) \in M\}$ *to be the methods applicable to c.*

To obtain a solution in HTN planning, one starts with the initial compound task and repeatedly applies *decomposition methods* to compound tasks until all tasks in the current task network are primitive.

Definition 4 (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*

$$\begin{aligned} tn_2 &= (T'', \prec_1 \cup \prec' \cup \prec_X, \alpha_1 \cup \alpha')|_{T''} \text{ with} \\ T'' &= (T_1 \setminus \{t\}) \cup T' \\ \prec_X &= \{(t_1, t_2) \in T_1 \times T' \text{ with } (t_1, t) \in \prec_1\} \cup \\ &\quad\ \{(t_1, t_2) \in T' \times T_1 \text{ with } (t, t_2) \in \prec_1\} \end{aligned}$$

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

Using the previous definition we can describe the set of solutions to a planning problem $\mathcal{P}$.

Definition 5 (Solution)**.** *A task network* tn_S *is a solution to a planning problem* $\mathcal{P}$*, if and only if*

(1) there is a linearisation $t_1, \ldots, t_n$ *of* $T(tn_S)$ *according to* $\prec(tn_S)$*,*
(2) $\alpha(tn_S)(t_1), \ldots, \alpha(tn_S)(t_n)$ *is executable in* s_I*, and*
(3) $(\{1\}, \emptyset, \{(1, c_I)\}) \to_D^* tn_S$*,*

$\mathfrak{S}(\mathcal{P})$ *denotes the sets of all solutions of* $\mathcal{P}$*, respectively.*

Note that this definition of HTN planning problems excludes some of the features in the original formulation by Erol, Hendler, and Nau [1]. His formalisation allows for constraints to be present in task network, namely before, after, and between constraints. The constraint type used most often, are before constraints, which correspond to SHOP(2)'s method preconditions. Our planner can handle them by compiling them into additional actions, as does SHOP2. So far, we don't support other constraint types.

To show that a task sequence π is a solution to a planning problem, we use *Decomposition Trees* (DTs) as witnesses [12]. They describe how π can be obtained from the initial abstract task via decomposition.

Definition 6. *Let* $\mathcal{P} = (L, C, O, M, c_I, s_I)$ *be an HTN planning problem. A valid 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.*
4) $\beta : V \to M$ *assigns each inner node a method.*
5) $\alpha(r) = c_I$
6) *for all inner nodes* $v \in V$ *with* $\beta(v) = (c, tn)$ *and children* $ch(v) = \{c_1, \ldots, c_n\}$*, it holds that* $c = \alpha(v)$*. Further, a bijection* $\phi : ch(v) \to T(tn)$ *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 those induced by 2. or 6. 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 [12] showed the following theorem:

Theorem 1. *Given a planning problem* $\mathcal{P}$*, then for every task sequence* π *the following holds:*

There exists a valid decomposition tree 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 is executable – the approach we use in this paper.

III. Related Work

Past research has already investigated possible translations of HTN planning problems into logic.

A. HTNs and Logic

Notably, Mali and Kambhampati [10] proposed a SAT-translation for HTNs. Their HTN formalism differs significantly from the established HTN formalism, making their encoding simpler and different from ours. They allow inserting tasks into task networks apart from decomposition and do not specify an initial task. Furthermore their encoding is also restricted to non-recursive domains. Such domains can be translated into an equivalent STRIPS planning problem, which is not the case for general domains [6]. Dix, Kuter, and Nau [13] have proposed an encoding of totally-ordered HTN planning into answer set programming, mimicking the search of SHOP. Their evaluation shows that the translated domain performs significantly worse than the SHOP algorithm (up to a factor of 1.000).

B. PDT-based encoding

Since our work is based on the encoding presented by Behnke, Höller, and Biundo [11], we start by reviewing this encoding in detail. Their idea was to restrict the maximum depth of decomposition. The planner start with some small bound K and constructs a SAT formula satisfiable if a solution with depth $\leq K$ exists. If not, K is increased and the process is repeated. To construct this formula, they used a compact representation of all possible decompositions with depth $\leq K$ – the Path Decomposition Tree PDT P. A satisfying valuation of the SAT formula then represents a decomposition tree T that is a subgraph of P. They however studied PDTs and the resulting formula only in the context of totally-ordered HTN planning, which is as we have argued in the introduction far less expressive and versatile than full partially-ordered HTN planning. Also we want to note, that almost all current HTN planning systems are constructed for partially-ordered domains, as most domains used in practice are partially ordered.

A PDT is a compact representation of all possible decompositions of the initial abstract task up to a given depth-bound K. Every such decomposition is represented by a decomposition tree (see Def. 6). The PDT is then a graph P such that it contains every possible decomposition tree as one of its subgraphs P'. To ensure a "common structure" we also require that the root of P' is the root of P. Next we give the formal definition of totally-ordered Path Decomposition Trees. To ease notation, we denote with $\mathfrak{L}(T = (V, W))$ the set of all leafs of a tree T.

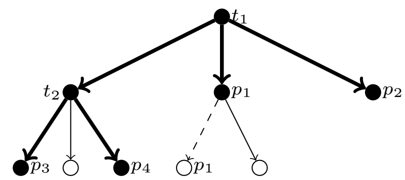

Fig. 1. An example PDT, a DT as its subgraph (nodes filled), and the extension for primitive tasks (dashed line). The nodes of the DT are each annotated with the task (t_i for abstract and p_i for primitives ones) that they are be labelled with in the DT. The node labelled p_2 does not have children even though it is not at the "lowest" level due to the fact that it can only be labelled with primitive tasks (p_2 in our example), while the node labelled with p_1 can potentially also be labelled with an abstract task. For this consider e.g. the methods $t_1 \mapsto t_2, p_1, p_2$ and $t_1 \mapsto t_2, t_3, p_2$. Note that there is one non-filled node that is also labelled with a task. This is an encoding trick to ensure that the leafs of the DT are also leafs of the PDT – primitive tasks are simply "inherited" by one of their children in the PDT.

Definition 7. *Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ be a planning problem and K a height bound. A Path Decomposition Tree P_K of height K is a triple $P_K = (V, E, \alpha)$ where*

1) *V are the nodes of a tree of height $\leq K$, with edges given by function $E : V \to V^*$, and which has the root node r.*
2) *$\alpha : V \to 2^{C \cup O}$ assigns each node a set of possible tasks.*
3) *$c_I \in \alpha(r)$*
4) *for all inner nodes $v \in V$, for each abstract task $c \in \alpha(v) \cap C$ that can be assigned to that node, and for each method $(c, tn) \in M(c)$, there exists a subsequence $v_1, \ldots, v_{|T(tn)|}$ of the children $E(v)$, such that $tn_i \in \alpha(v_i)$ for all $i \in \{1, \ldots, |T(tn)|\}$, where tn_i is the i^{th} element of the sequence of task names of tn.*
5) *$\forall v \in \mathfrak{L}(V, E)$: either $\alpha(v) \subseteq O$ or the height of v is K.*

This definition assumes that the tasks in a method's task network are totally-ordered and thus can be projected directly to a totally-ordered sequence of children. As a result, the leafs of the PDT are also totally-ordered (according to the order implied by their common ancestors). Behnke, Höller, and Biundo [11] provide an algorithm constructing a PDT P_K^σ given a so-called child-arrangement function σ. Based on it, they describe a SAT-formula $\mathcal{F}_D(\mathcal{P}, K)$ that is satisfiable if and only if there exists a subgraph G' of the PDT P_K^σ that forms a valid decomposition tree. A satisficing valuation of $\mathcal{F}_D(\mathcal{P}, K)$ represents such a DT G' – expressed by two types decision variables:

- t^v – v is part of G' and is labelled with t, i.e., $\alpha(v) = t$.
- m^v – the method m was applied to the node v of G', i.e., $\beta(v) = m$

Their encoding propagates primitive tasks occurring at any node v downwards through the first child of v in the PDT. This ensured that $yield(G')$ is represented by the leafs of P_K^σ that have a task assigned to them – else inner nodes of P_K^σ may belong to the yield. In addition to $\mathcal{F}_D(\mathcal{P}, K)$, Behnke, Höller, and Biundo used a second formula $\mathcal{F}_E(\mathcal{P}, K)$ ensuring executability of the tasks assigned to the leafs of G'.

For the formula $\mathcal{F}_D(\mathcal{P}, K)$ – and for other formulae thereafter, we use the functor $\mathbb{M}(V)$, which given a set of decision variables V, outputs a formula that is satisfiable if and only if at most one of them [14]. $\mathcal{F}_D(\mathcal{P}, K)$ consist solely of local constraint, i.e., one sub-formula is generated per node of the PDT. The formula to be generated for a node v of the PDT $P_K^\sigma = (V, E, \alpha)$ is either $\mathbb{M}(\{t^v \mid t \in \alpha(v) \cap O\}) \wedge_{c \in C} \neg c^v$ if $v \in \mathcal{L}(P_K^\sigma)$, i.e., if v is a leaf, or else the following formula:

$$f(v) = \mathbb{M}(\{t^v \mid t \in \alpha(v)\}) \wedge \mathit{selectMethod}(v) \\ \wedge \mathit{applyMethod}(v) \wedge \mathit{inheritPrimitive}(v) \\ \wedge \mathit{nonePresent}(v)$$

It first asserts that every node in the decomposition tree can be labelled with at most one task. The next four sub-formulae encode the further restrictions a decomposition tree must fulfil. $\mathit{selectMethod}$ ensures that an applicable method is chosen and that only one is chosen, provided v is labelled with an abstract task.

$$\mathit{selectedMethod}(v) = \mathbb{M}(\{m^v \mid M(\alpha(v) \cap C)\}) \wedge \\ \left[\bigwedge_{t \in \alpha(v) \cap C} \left(t^v \rightarrow \bigvee_{m \in M(t)} m^v\right)\right] \wedge \left[\bigwedge_{m \in M(\alpha(t) \cap C)} (m^v \rightarrow t^v)\right]$$

$\mathit{applyMethod}$ forces that whenever a method is selected, the tasks in its task network are assigned to the children of v. Let for a method $m = (c, tn)$ be $v_1, \dots, v_{|T(tn)|}$ the subsequence given in Def. 6. Let further denote $t_{tn,i}$ the ith task of the (totally-ordered) task network tn.

$$\mathit{applyMethod}(v) = \bigwedge_{m=(t,tn) \in M(\alpha(v))} \Bigg[m^v \rightarrow \\ \left(\bigwedge_{i=1}^{|tn|} t_{tn,i}^{v_i} \wedge \bigwedge_{v_i \in E(v) \setminus \{v_1, \dots, v_{|tn|}\}} \bigwedge_{t_* \in C \cup O} \neg {t_*}^{v_i} \right) \Bigg]$$

These clauses also propagate the total order between the subtasks $v_1, \dots, v_{|tn|}$. $\mathit{inheritPrimitive}$ and $\mathit{nonePresent}$ take care of the border cases, where v is either assigned a primitive task, or none at all. Let here be v_1 the first node in $E(v)$.

$$\mathit{inheritPrimitive}(v) = \\ \bigwedge_{p \in \alpha(v) \cap O} \left[p^v \rightarrow \left(p^{v_1} \wedge \bigwedge_{v_i \in E(v) \setminus \{v_1\}} \bigwedge_{k \in C \cup O} \neg k^{v_i} \right) \right]$$

$$\mathit{nonePresent}(v) = \left(\bigwedge_{t \in \alpha(v)} \neg t^v \right) \rightarrow \left(\bigwedge_{v_i \in E(v)} \bigwedge_{t \in C \cup O} \neg t^{v_i} \right)$$

The full decomposition formula $\mathcal{F}_D(\mathcal{P})$ is then simply $\bigwedge_{v \in V} f(v)$.

IV. Partially-Ordered Decomposition

We can extend this encoding, allowing us to track the partial order induced by the methods. As a first step, we have to ignore the fact that the PDT represents any ordering constraint. For that purpose, we introduce unordered PDTs, which differ only slightly from PDTs. Unordered PDTs – as their names suggests – don't have an ordering on the children of a node. Based on this, the main difference lies in 4. of the definition. For PDTs every node and applicable method, the subtasks of that method must from a subsequence of the nodes children, while for an unordered PDT it suffices that they are a subset.

Definition 8. *Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ be a planning problem and K a height bound. An unordered 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) *$\alpha : V \rightarrow 2^{C \cup O}$ assigns each node a set of possible tasks.*
3) *$c_I \in \alpha(r)$*
4) *for all inner nodes $v \in V$, for each abstract task $c \in \alpha(v) \cap C$ that can be assigned to v, and for each method $(c, tn) \in M(c)$, there exists a subset $D = \{v_1, \dots, v_{|T(tn)|}\}$ of v's children, such that a bijection $\phi^v_{(c,tn)} : D \rightarrow T(tn)$ exists with $\alpha(tn)(\phi^v_{(c,tn)}(d)) \in \alpha(d)$ for all $d \in D$*
5) *$\forall v \in \mathfrak{L}(V, E)$: either $\alpha(v) \subseteq O$ or the height of v is K.*

As uPDTs are a structural relaxation of PDTs, we can use the same generation procedure based on a child-arrangement function σ – simply by ignoring that methods are partially ordered – we use some topological ordering of the methods for generating P_K^σ instead. Based on the generated uPDT, we can also use the same formula $\mathcal{F}_D(\mathcal{P}, K)$ describing decomposition. To capture the partial order we add new decision variables for bookkeeping:

- b_w^v – for nodes v and w that have the same parent, i.e., are siblings. If b_w^v is true, the order $v \prec w$ is contained in the method applied to the parent of v and w.

These variables are sufficient to infer the order between all elements of $\mathit{yield}(G')$. This is due to how order is inherited in a decomposition tree. Essentially, the order between two nodes v and v' can only stem from the method applied to their last common ancestor in G'. The structure is illustrated in Figure 2. For two leafs v and v' of the tree, let $\mathfrak{A}(v, v')$ be the last common ancestor of v and v'. Further be $\mathfrak{C}(a, v)$, be the child c of a, s.t. the leaf v is below c. Then v stems from $\mathfrak{C}(\mathfrak{A}(v, v'), v)$, while v' from $\mathfrak{C}(\mathfrak{A}(v, v'), v')$. Then the formal property is the following:

Theorem 2. *Let $T = (V, E, \prec, \alpha, \beta)$ be a decomposition tree. Let $v, v' \in \mathfrak{L}(V, E)$ be two leafs of T, $c = \mathfrak{A}(v, v')$ be the last common ancestor of v and v'. Then the order between v and v' is the same as between $v_c = \mathfrak{C}(c, v)$ and $v'_c = \mathfrak{C}(c, v')$ induced by the method applied to c.*

Proof. Suppose there is an order between v_c and v'_c. Then by 2. of Def. 6, this order must also be present between v and v'.
Suppose there is no order between v_c and v'_c. Then the direct children of v_c and v'_c that are ancestors of v and v' respectively cannot contain any order, too. By definition, any order between them must either be introduced by methods or by 2. of Def. 6. Clearly, no decomposition methods could have introduced the

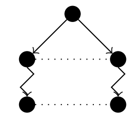

Fig. 2. An illustration where order originates from in a decomposition tree.

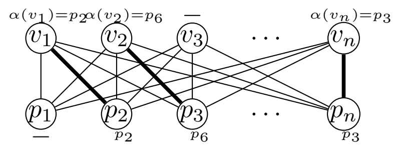

Fig. 3. Matching structure between leafs of P^σ_K, and positions in the primitive sequence.

ordering since the tasks don't have a common parent. Also since v_c and v'_c have no order between them 2. of Def. 6 is not applicable. By induction, we can conclude that there is not order between v and v'. $\square$

To keep track of the ordering constraints, we have to add for every decision variable m^v clauses that enforce that the correct b^v_w variables are set true. We therefore add for every m^v the following clauses to $F_D(\mathcal{P}, K)$, where $m = (c, tn)$, $\{v_1, \ldots, v_n\}$ are the nodes of P^σ_K to which the tasks of tn are mapped, and $\{t_1, \ldots, t_n\}$ be those tasks.

$$\bigwedge_{i=1}^{n} \bigwedge_{j \in \{1,\ldots,n\} \text{ s.t. } (t_i, t_j) \in \prec(tn)} (m^v \rightarrow b^{v_i}_{v_j})$$

These clauses enforce that the b^v_ws represent a superset of the ordering constraints induced by the applied methods.

To complete the encoding we need a formula $\mathcal{F}_E(\mathcal{P}, K)$ that is satisfiable if and only if $yield(G')$ is executable. Let $l = |\mathfrak{L}(P^\sigma_K)|$ be the number of leafs of P^σ_K. We separate this formula into two parts: representing a linearisation of $yield(G')$ and checking that this linearisation is executable. A linearisation of $yield(G')$ is a mapping of the leafs of G' to a sequence of *positions*. We can use l as an upper bound to the number of positions – and we have always used this value in our encoding. Also we denote these positions as $1, \ldots, l$. This mapping is essentially a bipartite matching that must not contradict the ordering constraints. Figure 3 illustrates these structures.

We have to generate a SAT formula that represents such a matching and is only satisfiable iff the matching is valid (i.e. an actual matching and it respects the order). We omit a formal proof of correctness, as we deem the encoding straight-forward enough to be considered correct by construction. We introduce two new decision variables:

- c^v_i – leaf v connected with position i
- a^v – leaf v contains a task (i.e. is a leaf of G' and has to be matched)

Based on these variables, we can formulate the restrictions a valid matching must fulfil. First, every leaf or position may be matched only once.

$$F_1 = \bigwedge_{i=1}^{l} \mathbb{M}(\{c^v_i \mid v \in \mathfrak{L}(P^\sigma_K)\}) \wedge \bigwedge_{v \in \mathfrak{L}(P^\sigma_K)} \mathbb{M}(\{c^v_i \mid 1 \leq i \leq l\})$$

Next, we define the a^v atoms, that are true exactly if the leaf v of P^σ_K contains an action. We use them as intermediate variables to decrease the overall size of the formula.

$$F_2 = \bigwedge_{v \in \mathfrak{L}(P^\sigma_K)} \left[\left(\neg a^v \rightarrow \bigwedge_{o \in \alpha(v)} \neg o^v \right) \wedge \left(a^v \rightarrow \bigvee_{o \in \alpha(v)} o^v \right) \right]$$

Next, a leaf of P^σ_K that contains a task has to be matched – else it would be allowed to disregard it when checking the executability of $yield(G')$.

$$F_3 = \bigwedge_{v \in \mathfrak{L}(P^\sigma_K)} \left[\left(\neg a^v \rightarrow \bigwedge_{1 \leq i \leq l} \neg c^v_i \right) \wedge \left(a^v \rightarrow \bigvee_{1 \leq i \leq l} c^v_i \right) \right]$$

If all these formulae are fulfilled, the atoms c^v_i represent a matching between all leafs of G' and the positions. As a next step, we have to ensure that this matching does not violate any ordering constraint induced by the chosen decomposition methods. To do that, we have to exclude the possibility that there are two positions $i < i'$ where the tasks they are matched with must occur in the opposite order. F_4 forbids the mentioned situation.

$$F_4 = \bigwedge_{i=1}^{l} \bigwedge_{i'=i+1}^{l} \bigwedge_{v,v' \in \mathfrak{L}(P^\sigma_K)} \left((c^v_i \wedge c^{v'}_{i'}) \rightarrow \neg b^{\mathfrak{C}(\mathfrak{A}(v,v'),v')}_{\mathfrak{C}(\mathfrak{A}(v,v'),v)} \right)$$

The second constraint states that the chosen linearisation of the tasks at the leafs of G' must be executable in the initial state. To express executability, we use the encoding proposed by Kautz and Selman [15]. For every proposition symbol $p \in L$, we introduce a decision variable p^i for $0 \leq i \leq L$. p^i is true if p is true after executing the i^{th} action. Further, we introduce decision variables t^i for every primitive task $t \in O$, stating that t is executed at timestep i. Then the formula $\mathcal{F}_{LE}$ is defined as follows:

$$\mathcal{F}_{LE} = \bigwedge_{p \in s_I} p^0 \wedge \bigwedge_{p \in L \setminus s_I} \neg p^0 \wedge \bigwedge_{i=0}^{l-1} (\mathcal{F}_A(i) \wedge \mathcal{F}_M(i)) \wedge \bigwedge_{i=1}^{l} \mathbb{M}(\{t^i \mid t \in O\})$$

$$\mathcal{F}_A(i) = \bigwedge_{t \in O} t^{i+1} \rightarrow \left(\bigwedge_{p \in prec(t)} p^i \wedge \bigwedge_{p \in add(t)} p^{i+1} \wedge \bigwedge_{p \in del(t)} \neg p^{i+1} \right)$$

$$\mathcal{F}_M(i) = \bigwedge_{p \in L} \left[(\neg p^i \wedge p^{i+1}) \rightarrow \bigvee_{t \in O \text{ with } p \in add(t)} t^{i+1} \right]$$

So far, we have only checked that the matching is valid and that the sequence of actions assigned to the positions is executable, but not that the matching influences the tasks assigned to positions. I.e., we have to add two more formulae that express that if a position is not matched to any leaf, then it also cannot contain a task, and that if it is matched it has to contain exactly the same task as the leaf does.

$$F_5 = \bigwedge_{1\leq i\leq l} \left[\left(\bigwedge_{v\in\mathfrak{L}(P_K)} \neg c_i^v\right) \rightarrow \left(\bigwedge_{t\in O} \neg t^i\right)\right]$$
$$F_6 = \bigwedge_{v\in\mathfrak{L}(P_K)} \bigwedge_{t\in\alpha(v)} \bigwedge_{1\leq i\leq l} t^v \wedge c_i^v \rightarrow t^i$$

To sum up, the full formula expressing executability is:

$$\mathcal{F}_E(\mathcal{P},K) = F_1 \wedge F_2 \wedge F_3 \wedge F_4 \wedge F_5 \wedge F_6 \wedge \mathcal{F}_{LE}$$

We know that the satisfying valuations of $\mathcal{F}_D(\mathcal{P},K)$ represent exactly all decomposition trees of $\mathcal{P}$ with an height $\leq K$ [11]. Based on this, the correctness and completeness of our encoding can be shown.

Theorem 3. *$\mathcal{F}_E(\mathcal{P},K) \wedge \mathcal{F}_D(\mathcal{P},K)$ is satisfiable iff $\mathcal{P}$ has a solution with decomposition height $\leq K$.*

Proof. $\Rightarrow$: Let ν be a satisfying valuation of $\mathcal{F}_E(\mathcal{P},K) \wedge \mathcal{F}_D(\mathcal{P},K)$. Then ν represents a decomposition tree, since $\mathcal{F}_D(\mathcal{P},K)$ is satisfied [11]. Thus the tasks assigned to the leafs of the Path Decomposition Tree encoded by $\mathcal{F}_D(\mathcal{P},K)$ from the yield Y of a Decomposition Tree. Also the sequence of actions S represented by the t^i is executable, due to $\mathcal{F}_{LE}$. What remains to show, is that this sequence is a linearisation of the yield Y. Due to $F_1 \wedge F_2 \wedge F_3$ the c_i^v represent a matching of Y to S and due to $F_5 \wedge F_6$ matched elements of Y and S contain the same task. Lastly, due to Theorem 2, the order between two tasks in Y depends solely on the method applied to their last common ancestor. Due to the clauses introducing the b_w^v variables, at least those orderings induced by the decomposition tree are true. Allowing for more order is not a problem, since ν already represents a linearisation. Lastly, F_4 ensures that the order encoded by the b_w^v is respected.

$\Leftarrow$: Let $T = (V, E, \prec, \alpha, \beta)$ be a decomposition tree whose yield is executable. Then a valuation ν exists that satisfies $\mathcal{F}_D(\mathcal{P},K)$ [11] and represents T. Let $v_1, \dots, v_n$ be the leafs of the PDT who have a task assigned to them in ν. Let further be $i_1, \dots, i_n$ the indices of these tasks in the executable linearisation of the yield of T. We then set $c_{i_j}^{v_j}$ true for all $j \in \{1, \dots, n\}$. We also set the $\alpha(v_j)^{i_j}$ and the appropriate p^i true. Also we set b_w^v true as appropriate, which cannot violate the clauses of F_4, as the respective order must also be present in the yield of T. This valuation satisfies $\mathcal{F}_E(\mathcal{P},K) \wedge \mathcal{F}_D(\mathcal{P},K)$. □

V. 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 is available at WWW.UNI-ULM.DE/IN/KI/PANDA/. Since most planning problems are given lifted, we use a combination of the planning graph and task decomposition graphs [16] to ground them.

Domains. Since there is no standard set of benchmark domains for HTN planning, we have compared our planner on instances used by previous evaluations [16], [17]. These evaluations also provide further detail on the domains and their properties. As such, we want to note that the domains were not in any way designed to be amendable to our translation. All of the domains are publicly available at WWW.UNI-ULM.DE/IN/KI/PANDA/. The benchmarking set is composed of the following domains:

- UM-TRANSLOG, WOODWORKING, SATELLITE, and SMARTPHONE are the benchmark domains of Bercher, Keen, and Biundo [18].
- ENTERTAINMENT describes setting-up HiFi devices.
- ROVER is the domain used by Höller et al. [17]. It is based on the problem instances of the IPC3 domain ROVER combined with an HTN-structure similar to the one developed for SHOP.
- TRANSPORT describes a deliver-with-trucks scenario. There are several trucks (which do not need fuel) to deliver packages from their start location to a destination.
- PCP is an encoding of Post's Correspondence Problem. Since HTN planning is undecidable, we felt it proper to show that an HTN planner is able to solve undecidable problems (like PCP) when encoded in an HTN domain.

The domains ENTERTAINMENT, ROVER, and TRANSPORT contain method preconditions, which we compile away into additional actions preceeding all other actions.

Behnke, Höller, and Biundo [11] used domains with the same names, except for PCP, in their evaluation. Their planner, however, was amendable only to totally-ordered instances. Since most instances are naturally partially-ordered, they had to alter them. Behnke, Höller, and Biundo [11] have manually added additional ordering constraints to each partially-ordered method. Adding ordering constraints to HTN domains can make them unsolvable (see e.g. PCP, which cannot contain a solution when totally ordered). The additional orderings were chosen such that at least one solution was retained. We also want to note that adding these orderings makes some of the domains much easier to solve. For example, in transport, interleaving using the partial order is required to find optimal solutions. If the domain is totally-ordered, one package has to be delivered before another package could be picked up.

In our evaluation we have used the original, partially-ordered versions of all domains. Note also that all planners were given the same input.

Planners. Each planner was given 10 minutes runtime and 4 GB RAM per instance on an Intel Xeon E5-2660. We have compared all state-of-the-art HTN planning systems: SHOP2 [19], FAPE [5], UMCP [20], PANDA with the TDG_m and TDG_c heuristics [16] using greedy A*, PANDApro using the FF heuristic [17], HTN2STRIPS [21], and totSAT [11]. FAPE – according to the description in its paper – does not support recursive domains. Thus, we ran it only on the

TABLE I

STATISTICS ON DOMAIN PROPERTIES

Domain	$\|L\|$		$\|O\|$		$\|C\|$		$\|M\|$	
	min	max	min	max	min	max	min	max
PCP	6	9	8	14	4	46	10	34
ENTERTAINMENT	10	146	16	455	10	170	20	541
UM-TRANSLOG	9	25	7	22	2	27	2	28
SATELLITE	6	37	7	123	3	25	10	214
WOODWORKING	10	101	7	739	4	443	9	2002
SMARTPHONE	10	103	8	231	3	66	4	360
ROVER	21	511	73	4257	14	285	49	3279
TRANSPORT	11	364	13	1968	11	802	21	3158

TABLE II

STATISTICS ON GENERATED FORMULAE

Domain	$\|\mathcal{L}(P_K)\|$		K		#clause		#plansteps	
	min	max	min	max	min	max	min	max
PCP	12	70	4	9	14.012	12.091.312	10	42
ENTERTAINMENT	8	78	4	6	416	42.028	7	42
UM-TRANSLOG	7	40	3	4	218	281.642	7	26
SATELLITE	5	40	3	5	183	1.375.308	5	20
WOODWORKING	3	25	3	7	531	689.552	3	19
SMARTPHONE	7	78	3	5	3.332	18.878.346	5	77
ROVER	53	61	5	5	4.048.432	7.045.922	27	36
TRANSPORT	8	48	4	6	3.980	5.176.067	8	42

domains SATELLITE, WOODWORKING, and ROVER, which are the non-recursive ones in our evaluation. Similarly, as totSAT can only handle totally-ordered instance, we have run it only on those instances from our benchmark set that are totally ordered. Lastly, we have tested HTN2STRIPS with two different classical planners. We have used both jasper (which was originally used by Alford et al. [21]) as well as Madagascar [22], the currently best known SAT planner. We chose to do so, to compare our propositional encoding with the theoretically only so-far known propositional encoding for partially-ordered HTNs: first using the HTN2STRIPS translation and then the ∃-step encoding [23] for the resulting planning problem. To complete our evaluation, we have also used the best planners from the agile and satisficing tracks of IPC 2018: Fast Downward Stone Soup [24], saarplan [25], and LAPKT-BFWS-Preference [26].

We have not included the planner GTOHP [27] in our evaluation. Like totSAT, it takes only totally-ordered instances as its input. Since totSAT already solves all totally ordered instances of the domain set, GTOHP cannot have a better performance.

For our planner, we have evaluated three SAT solvers, each a top performers at the SAT Competition 2016. These were: cryptominisat5 [28], MapleCOMSPS [29], and Riss6 [30]. As our planner performs the translation using a bound K, we usually have to try several values for K. We started with $K = 1$ and increased by 1 if the formula was unsolvable. This iterative procedure allows us to handle any recursion in the domains, as we gradually unroll it. We construct the formula completely anew for each K and do not use the ability SAT solvers to incrementally solve formulae. We however suspect that applying a technique similar to the known technique for classical planning [31] will increase the planner's performance.

Results. In Tab. III we show the number of solved instances per planner within the given time and memory limits. Fig. 4 shows the solved instances depending on runtime. First, our SAT-encoding, no matter the solver, solves more instances than any other planner. Second, our planner is on par in every domain with the best solver for that domain, or solves significantly more instances than other planners.

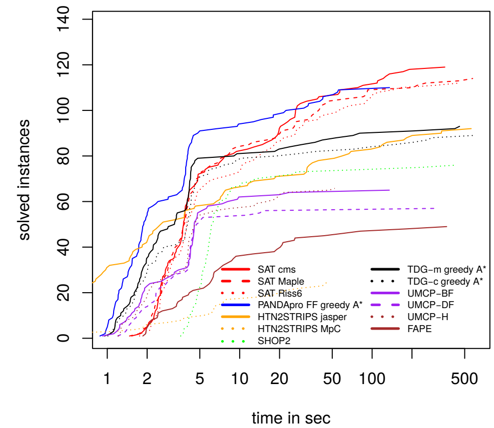

Fig. 4. Runtime vs number of solved instances per planner

TABLE III

NUMBER OF SOLVED INSTANCES PER PLANNER PER DOMAIN. MAXIMA ARE INDICATED IN BOLD. CMS = CRYPTOMINISAT5

	#instances	SAT cms	SAT Maple	SAT Riss6	PANDApro FF	HTN2STRIPS jasper	HTN2STRIPS FD-SS 2018	HTN2STRIPS Saarplan	HTN2STRIPS LAPKT-BFWS	HTN2STRIPS MpC	SHOP2	TDG-m	TDG-c	UMCP-BF	UMCP-DF	UMCP-H	FAPE	totSAT [AAAI18]
PCP	17	**11**	**11**	10	10	3	17	17	17	3	0	9	8	0	0	0	-	-
ENTERTAINMENT	12	**12**	**12**	**12**	11	5	19	14	12	4	5	9	9	5	5	6	-	12 / 12
UM-TRANSLOG	22	**22**	**22**	**22**	**22**	19	5	5	5	7	**22**	**22**	**22**	**22**	**22**	**22**	-	19 / 19
SATELLITE	25	**25**	24	23	**25**	23	6	5	5	8	22	**25**	21	18	20	23	22	5 / 5
WOODWORKING	11	**11**	**11**	**11**	10	5	3	3	3	4	8	8	10	6	6	6	0	-
SMARTPHONE	7	**7**	6	6	5	6	5	5	4	5	4	5	5	4	4	4	-	-
ROVER	20	4	4	4	3	**5**	5	4	4	4	3	2	2	0	0	0	3	-
TRANSPORT	30	16	14	13	13	**19**	17	13	13	3	0	1	1	1	0	0	-	-
total	144	**108**	104	101	99	85	77	66	63	38	64	81	78	56	57	61	25	36

We want to point out our performance in the domains TRANSPORT and PCP. In TRANSPORT we only solve 3 instances less than HTN2STRIPS, while all other planners solve at most a single instance. In PCP, we solve significantly more instances than HTN2STRIPS. This is notable, as both domains contain difficult combinatorially problem. This is especially notable, since HTN2STRIPS internally uses a state-of-the-art classical planner (jasper, [32]). However, there still seems to be room for improvement, as no planner seem to be well equipped to exploit the hierarchy in the ROVER domain.

We can see that the HTN2STRIPS encoding does not seem to work well in combination with state-of-the-art classical planners as all of the top-performing planners from IPC 2018 performed worse than jasper in its four year old version.

The original totSAT for totally ordered domains has poor coverage, based on the fact that most domains of the benchmark set are partially-ordered. Lastly, we can observe that using Madagascar in combination with the HTN2STRIPS encoding seems to perform extremely poorly. In most instances Madagascar is aborted after only a few seconds as it reached the memory limit. This is probably due to the large number of groundings for the operators in the HTN2STRIPS encoding representing methods, which is a known problem of the encoding. We have re-run Madagascar with a memory limit of 20 GB instead of 4 GB and have only seen an increase by 4 solved instances. Also, the per-instance runtime when compared to jasper is fairly poor. We suppose that this is due to the way the encoding works. Modern SAT-based planning draws its efficiency mainly from the ability to execute several operators in parallel. This is not possible in the encoded domain as the next-predicates ensure that all simultaneously applicable actions form a clique in the disabling graph, i.e., cannot be executed parallel in the propositional encoding.

VI. Conclusion

We have presented the first encoding for SAT-based HTN planning that can solve all propositional HTN planning problems. To that end, we have utilised a previous encoding that was only usable for totally-ordered planning, which restricts the freedom of the domain modeller unnecessarily, and extended it to partial order. Lastly, we have shown that our new planner outperforms state-of-the-art HTN planners. This planner has already been used in practice, namely in an assistant teaching users how to use electronic tools in Do-It-Yourself projects [33], as well as for generating corpora for plan and goal recognition [34]. The presented planner is also well suited for altering existing plans in accordance with a user's instruction, which is a computationally hard problem [35]. We plan to answer these instructions via translation into LTL, for which efficient propositional encodings exist [36].

Acknowledgments

This work was partly done within the technology transfer project "Do it yourself, but not alone: Companion-Technology for DIY support" of the SFB/TRR 62 funded by the German Research Foundation (DFG).

References

[1] K. Erol, J. Hendler, and D. Nau, "Complexity results for HTN planning," *Annals of Mathematics and AI*, vol. 18, no. 1, pp. 69–93, 1996.

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

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

[4] A. Champandard, T. Verweij, and R. Straatman, "The AI for Killzone 2's multiplayer bots," in *Proc. of GDC*, 2009.

[5] F. Dvorak, A. Bit-Monnot, F. Ingrand, and M. Ghallab, "A flexible ANML actor and planner in robotics," in *Proc. of PlanRob*, 2014, pp. 12–19.

[6] D. Höller, G. Behnke, P. Bercher, and S. Biundo, "Language classification of hierarchical planning problems," in *Proc. of ECAI*, vol. 263. IOS Press, 2014, pp. 447–452.

[7] D. Höller, G. Behnke, P. Bercher, and S. Biundo, "Assessing the expressivity of planning formalisms through the comparison to formal languages," in *Proc. of ICAPS*. AAAI Press, 2016, pp. 158–165.

[8] G. Behnke, D. Höller, and S. Biundo, "On the complexity of HTN plan verification and its implications for plan recognition," in *Proc. of ICAPS*. AAAI Press, 2015, pp. 25–33.

[9] G. Behnke, D. Höller, and S. Biundo, "This is a solution! (... but is it though?) – verifying solutions of hierarchical planning problems," in *Proc. of ICAPS*. AAAI Press, 2017, pp. 20–28.

[10] A. Mali and S. Kambhampati, "Encoding HTN planning in propositional logic," in *Proc. of AIPS*. AAAI, 1998, pp. 190–198.

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

[12] T. Geier and P. Bercher, "On the decidability of HTN planning with task insertion," in *Proc. IJCAI*. AAAI Press, 2011, pp. 1955–1961.

[13] J. Dix, U. Kuter, and D. Nau, "Planning in answer set programming using ordered task decomposition," in *Proc. of KI*. Springer, 2003, pp. 490–504.

[14] C. Sinz, "Towards an optimal CNF encoding of boolean cardinality constraints," in *Proc. of CP 2005*, vol. 3709. Springer, 2005, pp. 827–831.

[15] H. Kautz and B. Selman, "Pushing the envelope: Planning, propositional logic, and stochastic search," in *Proc. of AAAI*, 1996, pp. 1194–1201.

[16] P. Bercher, G. Behnke, D. Höller, and S. Biundo, "An admissible HTN planning heuristic," in *Proc. of IJCAI*. IJCAI, 2017, pp. 480–488.

[17] D. Höller, P. Bercher, G. Behnke, and B. Biundo, "A generic method to guide HTN progression search with classical heuristics," in *Proc. of ICAPS*. AAAI Press, 2018.

[18] P. Bercher, S. Keen, and S. Biundo, "Hybrid planning heuristics based on task decomposition graphs," in *Proc. of SoCS*. AAAI Press, 2014, pp. 35–43.

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

[20] K. Erol, J. Hendler, and D. Nau, "UMCP: A sound and complete procedure for hierarchical task-network planning," in *Proc. of AIPS*. AAAI Press, 1994, pp. 249–254.

[21] R. Alford, G. Behnke, D. Höller, P. Bercher, S. Biundo, and D. W. Aha, "Bound to plan: Exploiting classical heuristics via automatic translations of tail-recursive HTN problems," in *Proc. of ICAPS*. AAAI Press, 2016, pp. 20–28.

[22] J. Rintanen, "Madagascar: Scalable planning with SAT," in *The 2014 IPC*, 2014, pp. 66–70.

[23] J. Rintanen, K. Heljanko, and I. Niemelä, "Planning as satisfiability: parallel plans and algorithms for plan search," *Artificial Intelligence*, vol. 170, no. 12-13, pp. 1031–1080, 2006.

[24] J. Seipp and G. Röger, "Fast downward stone soup 2018," in *IPC2018 – Classical Tracks*, 2018, pp. 72–74.

[25] M. Fickert, D. Gnad, P. Speicher, and J. Hoffmann, "Saarplan: Combining saarland's greatest planning techniques," in *IPC2018 – Classical Tracks*, 2018, pp. 10–15.

[26] G. Frances, H. Geffner, N. Lipovetzky, and M. Ramirez, "Best-first width search in the IPC 2018: Complete, simulated, and polynomial variants," in *IPC2018 – Classical Tracks*, 2018, pp. 22–26.

[27] A. Ramoul, D. Pellier, H. Fiorino, and S. Pesty, "Grounding of htn planning domain," *International Journal on Artificial Intelligence Tools*, vol. 26, no. 5, pp. 1–24, 2017.

[28] M. Soos, "The CryptoMiniSat 5 set of solvers at SAT Competition 2016," in *Proc. of SAT Competition 2016*. University of Helsinki, 2016, p. 28.

[29] J. H. Liang, C. Oh, V. Ganesh, K. Czarnecki, and P. Poupart, "MapleCOMSPS, MapleCOMSPS_LRB, MapleCOMSPS_CHB," in *Proc. of SAT Competition 2016*. University of Helsinki, 2016, pp. 52–53.

[30] N. Manthey, A. Stephan, and E. Werner, "Riss 6 solver and derivatives," in *Proc. of SAT Competition 2016*. University of Helsinki, 2016, pp. 56–57.

[31] S. Gocht and T. Balyo, "Accelerating sat based planning with incremental sat solving," in *Proc. of ICAPS*. AAAI Press, 2017, pp. 135–139.

[32] F. Xie, M. Müller, and R. Holte, "Jasper: The art of exploration in greedy best first search," in *The 2014 IPC*, 2014, pp. 39–42.

[33] G. Behnke, M. Schiller, M. Kraus, P. Bercher, M. Schmautz, M. Dorna, W. Minker, B. Glimm, and S. Biundo, "Instructing novice users on how to use tools in DIY projects," in *Proc. of IJCAI-ECAI*. IJCAI, 2018, pp. 5805–5807.

[34] D. Höller, G. Behnke, P. Bercher, and S. Biundo, "Plan and goal recognition as htn planning," in *Proc. of ICTAI*, 2018.

[35] G. Behnke, D. Höller, P. Bercher, and S. Biundo, "Change the plan - how hard can that be?" in *Proc. of ICAPS*. AAAI Press, 2016, pp. 38–46.

[36] G. Behnke and S. Biundo, "X and more parallelism: Integrating ltl-next into sat-based planning with trajectory constraints while allowing for even more parallelism," *Inteligencia Artificial*, vol. 21, no. 62, pp. 75–90, 2018.