The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

totSAT — Totally-Ordered Hierarchical Planning through SAT

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

Abstract

In this paper, we propose a novel SAT-based planning approach for hierarchical planning by introducing the SAT-based planner totSAT for the class of totally-ordered HTN planning problems. We use the same general approach as SAT planning for classical planning does: bound the problem, translate the problem into a formula, and if the formula is not satisfiable, increase the bound. In HTN planning, a suitable bound is the maximum depth of decomposition. We show how totally-ordered HTN planning problems can be translated into a SAT formula, given this bound. Furthermore, we have conducted an extensive empirical evaluation to compare our new planner against state-of-the-art HTN planners. It shows that our technique outperforms any of these systems.

Introduction

Hierarchical planning provides a natural and expressive, but nevertheless easily usable means for planning. Its most common formalism is Hierarchical Task Network (HTN) planning (Erol, Hendler, and Nau 1996), which extends classical STRIPS-based planning in two ways. It introduces first the concept of abstract tasks – tasks which cannot be executed directly, and second decomposition methods, which relate abstract tasks to primitive STRIPS-like actions by providing a course of (primitive and/or abstract) tasks that needs to be performed to achieve the abstract task. The various decomposition methods for an abstract task define the allowed portfolio of ways to achieve that task. They can, e.g., be gained from domain experts, whose knowledge can be encoded and exploited this way. This makes HTN planning particularly useful for planning in complex real-world domains. Its top-down and decomposition-based reasoning also bears similarities with human thought processes (Byrne 1977).

Although HTN planning is often used in real-world applications (Nau et al. 2005; Champandard, Verweij, and Straatman 2009; Hartanto and Hertzberg 2008; Morisset and Ghallab 2002; Dvorak et al. 2014; Bercher et al. 2015), there has so far only been little research in developing fast, domain-independent HTN planners. Systems are either guided by instructional information, hard-coded in the domain model, use blind search, or heuristic search in the plan space, relying on quite simple heuristics (Bercher et al. 2017). Recently, a new HTN planning technique based on a bounded translation into classical planning as been proposed (Alford et al. 2016a). Most search processes of HTN planners consider both parts of the planning problem – the hierarchy and the primitive action model – separately and do not analyse interactions between them. This is especially true for heuristics, which have – so far – either been based solely on the hierarchy or solely on the primitive actions. The work by Alford et al. (2016a) provides an integration of both worlds. The planner FAPE (Dvorak et al. 2014) does this too (Bit-Monnot, Smith, and Do 2016), but restricts the decomposition hierarchy to be non-recursive, which greatly reduces the expressiveness of HTN planning.

In this paper, we present a new technique for HTN planning based on a translation of the planning problem into a propositional satisfiability problem. It enables a unified view on both parts of the problem and leads to a significantly better performance compared to previous approaches. We restrict ourselves to the class of totally-ordered HTN planning, thereby easing the SAT formula's construction. This restriction is justified by various observations. First, we retain most of the expressive power of HTN planning and are still more expressive than STRIPS-based planning (Höller et al. 2014; Höller et al. 2016). Second, HTN domains often have a high degree of total order (Alford et al. 2016a), so that planners have even been specifically designed for this problem class (Nau et al. 1999; Alford, Kuter, and Nau 2009). Third, restricting models to be totally-ordered enables the creation of complete planning algorithms, as partially ordered HTN planning is undecidable (Erol, Hendler, and Nau 1996), while totally-ordered planning is $\mathbb{EXPTIME}$-complete (Alford, Bercher, and Aha 2015). Finally, studying the structurally simpler class of totally-ordered HTN planning problems is a suitable first step towards creating a SAT-based planner for general HTN planning problems.

The remainder of the paper is organised as follows. Next, we introduce the totally-ordered HTN formalism and survey related approaches. We then describe our planning algorithm totSAT and present its empirical evaluation.

HTN Planning

In this paper, we use an adaptation of the formulation of HTN planning of Geier and Bercher (2011), in which we omit everything related to partial order. Hierarchical plan-

ning distinguishes between two types of tasks: primitive and abstract. Primitive tasks a are identical to standard STRIPS actions: they can be executed directly and are specified by $prec$, add, and del lists with the usual state transition semantics. In contrast, abstract tasks must be refined through methods into primitive actions before they can be executed. To describe the allowed refinements, HTN planning problems use task networks, which describe a partially ordered multiset of – both primitive and abstract – tasks. In the totally-ordered case a task network is merely a sequence of tasks.

Definition 1. *Given a set of tasks T, a task network tn is any element of T^*. We define $tn(i)$ to be the ith task of tn.*

A totally-ordered HTN planning problem is defined as:

Definition 2. *A totally-ordered HTN planning problem is a 6-tuple $\mathcal{P} = (L, C, O, M, c_I, s_I)$, s.t.*

- *L is a finite set of proposition symbols*
- *C is a finite set of abstract (or **c**ompound) tasks*
- *O is a finite set of primitive actions (or **o**perators)*
- *$M \subseteq C \times (C \cup O)^*$ is a set of decomposition methods*
- *$c_I \in C$ is the initial abstract task*
- *$s_I \subseteq L$ is the initial state.*

We define $M(t) = \{(t, tn) \mid (t, tn) \in M\}$ to be the set of methods for the abstract task t and $M(T) = \bigcup_{t \in T} M(t)$. Decomposition methods are essentially rules that describe how an abstract task can be replaced by a sequence of (primitive or abstract) tasks, defined as follows.

Definition 3. *Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ be a planning problem, $tn = \omega_1 c \omega_2$ a task network where $c \in C$, and $m = (c, \varphi) \in M$ a decomposition method. Then the application of m to tn results in the task network $tn' = \omega_1 \varphi \omega_2$, written $tn \rightarrow_D^m tn'$. Likewise we write $tn \rightarrow_D tn'$ if any such method exists, and $tn \rightarrow_D^* tn'$ if any sequence of decompositions exists, which transforms tn into tn'.*

We define a solution to an HTN planning problem as:

Definition 4. *Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ be a planning problem. A task network $tn \in O^*$ is a solution to $\mathcal{P}$ if and only if $c_I \rightarrow_D^* tn$ and tn is executable in s_I. The set of all such task networks is defined as $\mathfrak{S}(\mathcal{P})$.*

In this paper, we consider the problem of satisficing planning, i.e., the decision problem to decide whether a task network $tn \in \mathfrak{S}(\mathcal{P})$ exists. We want to emphasise that the HTN part of the planning problem is much more than a mere heuristic guide to solve some (in its core) classical planning problem – it is a restriction on the allowed plans. This additional way to restrict the model makes HTN planning both more expressive than classical planning and computationally more difficult. Our formalism differs from the one used by the HTN planning system SHOP (Nau et al. 1999) in that we do not allow for methods to have preconditions. This is not a restriction, as SHOP-style domains can be compiled into our formalism by introducing tasks with each method's precondition, which are ordered before all other tasks in their respective methods. Three of our evaluation domains contain method preconditions, which are compiled by the planner.

The reduced formalism bears a striking similarity to context-free grammars (CFGs). In fact, the set of solutions $\mathfrak{S}(\mathcal{P})$ can be described by a CFG and for all CFGs there is a planning problem generating the same language (Höller et al. 2014). In theory, the task of planning boils down to finding a word generated by a context-free grammar, which can be done in linear time. The planning problem is a compact representation of the corresponding grammar, as it defines the solutions as the intersection of two properties: being a valid decomposition and being executable. Constructing the grammar explicitly would lead to an exponential blow-up, which makes any such approach to planning useless.

Related Work

The idea of translating hierarchical planning problems into logical formulae is not entirely new. Mali and Kambhampati (1998) have proposed a translation of HTN planning problems into a SAT formula. Their notion of HTN planning significantly differs from the (by now) established HTN formalism, making their formula much simpler and structurally different from ours. First, their formalism allows inserting additional tasks and does not have an initial task, which makes it equivalent to STRIPS planning. Second, they allow task sharing (Alford et al. 2016b), which leads both to significantly shorter or even new solutions. Consider a planning problem with a single abstract task A which decomposes into two instances of the primitive task a, which can only be executed once. Using standard HTN semantics this problem has no solution, while with task sharing it does have one of length one. Lastly, their technique is also far less powerful than ours, as it cannot handle recursive domains, which greatly reduces the expressive power of hierarchical planning. Such domains can always be translated into an equivalent STRIPS planning problem, which is not the case for general or totally-ordered domains (Höller et al. 2014).

Dix, Kuter, and Nau (2003) have proposed an encoding of totally-ordered HTN planning into answer set programming (ASP), based on the mechanics of SHOP. Their empirical evaluation shows that the translated domain performs significantly worse than the standard SHOP algorithm (up to a factor of 1.000). We assume that this is due to the high expressive power needed for their models – it requires numbers. Also, ASP is commonly solved by a translation into SAT, thus we deem a direct encoding to be more efficient. We have not included their approach in our evaluation as no code for their translation is publicly available and (as stated) its performance is worse than SHOP, which we did include.

In recent work, we presented a SAT-based technique for verifying HTN plans (Behnke, Höller, and Biundo 2017), a task which was shown to be $\mathbb{NP}$-complete (Behnke, Höller, and Biundo 2015). In plan verification, the objective is, given a sequence of actions, to determine whether this sequence can be obtained via decomposition from the initial abstract task. In theory, our proposed encoding could also be used as the basis for a planner, but its size and complexity make it impractical for that purpose. Also, the encoding we present here differs significantly from the verification encoding in that it represents decomposition in a tree-like structure rather than as a process iterating over sequences of actions.

As noted, the problem of finding a plan for a totally-ordered HTN planning problem is equivalent to finding a word of a CFL given in a highly compressed form. For this test, it would however be necessary to construct the respective grammar G explicitly, which would require to construct the state space explicitly – which is not feasible. Due to the specific nature of our input, there has been no work in the formal languages community – as far as we know. There has been work relating to the usage of SAT techniques when analysing properties of CFLs, like universality (Axelsson, Heljanko, and Lange 2008), but their techniques are not applicable here, as they need the expanded CFG as their input.

Height-bounded Decomposition Trees

Our translation of HTN planning problems into SAT formulae is based on the notion of decomposition trees (DTs), which were originally introduced by Geier and Bercher (2011). A DT is a witness that a task network is a solution to a given planning problem. We adapt their definition and simplify it for the case of totally-ordered domains as follows:

Definition 5. *Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ be an HTN planning problem. A valid totally-ordered decomposition tree T is a 4-tuple $T = (V, E, \alpha, \beta)$, where*

- *V are the nodes of a directed tree, whose edges are given by the function $E : V \to \overline{V}$[1], mapping each node in the tree to a ordered list of vertices – its children. Let r be the root-node of this tree.*
- *$\alpha : V \to C \cup O$ assigns each inner node an abstract task and each leaf a primitive task.*
- *$\beta : V \to M$ assigns each inner node a method.*
- *$\alpha(r) = c_I$*
- *for all inner nodes $v \in V$ with $\beta(v) = (c, tn)$ and children $E(v) = c_1, \ldots, c_n$, it holds that $c = \alpha(v)$ and $\alpha(c_1) \ldots \alpha(c_n) = tn$.*

The function E implicitly defines an order of all nodes of the tree. Two nodes a and b are ordered $a \prec b$, if any only if for the lowest common ancestor c of a and b in the tree, the child a' of c which is the ancestor of a occurs in $E(c)$ before the respective ancestor b' of b.

The solution represented by a DT T are the tasks assigned to its leafs in the order implicitly induced by the function E. Geier and Bercher (2011) showed that for every solution $tn \in \mathfrak{S}(\mathcal{P})$ a DT exists, whose leafs are assigned the tasks of tn in the correct order and vice versa. Thus, instead of generating a formula that describes all solutions, we can generate one that describes all possible DTs. This, however, is not possible in general, since the HTN can contain recursive methods, requiring to describe infinitely many DTs in one formula. We take the same approach as SAT-planning does for classical planning: Restrict the solution by some bound and increase the bound until a solution has been found. We propose to use the height of the DTs as this bound – in contrast to the plan length in classical planning[2]. I.e., given a height-bound K, we generate a SAT formula that describes all possible decomposition trees whose height is at most K.

Here the question arises, which values should be tried for K? As the lower bound, we use the minimal height necessary to derive a task network solely containing primitive tasks from c_I. Formally this height is the smallest K s.t. there exists an DT with height K whose leafs are assigned only primitive tasks. This bound can be computed inductively: For any primitive task it is 0, for every method it is the maximum of any contained task, and for any abstract task it is 1 plus the minimum over its decomposition methods. For cycles in the decomposition hierarchy, we can iterate until convergence. The following theorem provides an upper bound, showing that a planning procedure based on iteratively increasing the allowed height will terminate in finite time. Similar to classical planning, where there is also such an upper bound ($2^{|L|}$), the determined bound is quite high, but plays no practical role, as solutions are usually found before that bound is reached.

Theorem 1. *Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ be an HTN planning problem. If $\mathfrak{S}(\mathcal{P}) \neq \emptyset$, then $\mathfrak{S}(\mathcal{P})$ also contains a solution whose decomposition height is at most $|C| \cdot \left(2^{|L|}\right)^2$.*

Proof. Assume that $\mathfrak{S}(\mathcal{P})$ is not empty, let tn be a solution with the minimal decomposition height, and $K > |C| \cdot \left(2^{|L|}\right)^2$ be this height. Then the decomposition tree of tn contains a path of length K. Label every node a in the tree with the states immediately before and after executing the primitive tasks resulting from a (i.e. the leafs below a). Since this path is longer than $|C| \cdot \left(2^{|L|}\right)^2$, at least two nodes will be labelled with the same abstract task and states. We can now remove the part of the tree between these two occurrences (including one of the two duplicated nodes). The resulting tree still represents a task network, which is a solution to $\mathcal{P}$, as it only contains valid decompositions and the resulting task network will be executable. However the height of this path in the tree has decreased. If we repeat this process for all paths with length K, the resulting tree will represent a solution with a decomposition height $< K$. □

Before we start to construct the formula itself, we first need a suitable abstraction of DTs, which can describe sets of DTs – in our case all DTs up to a given height. We have developed such an abstraction called Path Decomposition Trees (PDTs). A PDT P is required to have every DT of height $\leq K$ as a sub-tree. Figure 1 show an example for a PDT and a DT as its subgraph. The nodes of the PDT belonging to the DT are labelled with the respective task $\alpha(v)$. Our formula will describe such a subgraph DT and then checks that its leafs contain primitive tasks, which are executable in the order induced by the DT. Conveniently, this will be the same order as those nodes have in the PDT, which is marked with dashed lines in Figure 1.

[1] Let $\overline{X}$ be the set of all sequences over X.

[2] Bounding the height of a DT implicitly also bounds the plan length, but it is much simpler to construct a SAT-formula bounding the height, e.g., due to methods with empty task networks.

We start by detailing the formal criteria we impose on a PDT and show a theoretical property ensuring the completeness of our SAT-translation. There are several possible PDTs for every planning problem and height K that fulfil these criteria. As such, we introduce a broader framework upon which further research could be based. As a second step, we describe a way to generate PDTs for a given planning problem, leading to a single PDT for every problem/height combination. We then construct our SAT formula based on this single PDT.

Definition 6. *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*

- *V are the nodes of a directed tree of height $\leq K$, whose edges are given by the function $E : V \to \overline{V}$, and which has a single root node r.*
- *$\alpha : V \to 2^{C \cup O}$ assigns each node a set of possible tasks.*
- *$c_I \in \alpha(r)$*
- *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 sub-sequence $v_1, \ldots, v_{|tn|}$ of the children $E(v)$, such that $tn(i) \in \alpha(v_i)$ for all $i \in \{1, \ldots, |tn|\}$*
- *all leaf nodes are either assigned only primitive tasks, or are at height K*

A DT $T = (V_T, E_T, \alpha_T, \beta_T)$ is contained in P_K iff there is a sub-tree $G' = (V', E')$ of (V, E), s.t. $r \in V'$, which is isomorphic to (V_T, E_T) with the isomorphism $\psi : V' \to V_T$ s.t. $\forall v' \in V' : \alpha_T(\psi(v')) \in \alpha(v')$.

Theorem 2. *Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ be a planning problem and K a height bound. Then every DT T which has a height of at most K is contained in a PDT P_K.*

Proof. Let $T = (V_T, E_T, \prec_T, \alpha_T, \beta_T)$ be a DT of height at most K and $P_K = (V, E, \prec, \alpha)$ be a PDT, then we have to find a sub-tree G' of P_K which is isomorphic to T and satisfies the conditions from the definition. We can construct this sub-tree recursively, starting with the roots of P_K and T. For the root node r_T of T we select the root r of P_K to be in G' and set the isomorphism respectively. $\beta(r_T)$ assigns a decomposition method $m = (\alpha_T(r_T), tn)$ to r_T. Since P_K is a PDT and $\alpha_T(r_T)$ is contained in $\alpha(r)$, i.e., m is applicable to a task in $\alpha(r)$, we know that there is a sequence of children $(v_1, \ldots, v_{|tn|})$ of r in P_K s.t. for each v_i the label set $\alpha(v_i)$ contains the necessary label $tn(i)$. Let $v_1^T, \ldots, v_{|tn|}^T$ be that sequence of children of r_T in T. As T is a DT, we have $\alpha_T(v_i^T) = tn(i)$ and thus $\alpha_T(v_i^T) \in \alpha(v_i)$. Thus we can extend G' by the c_i and the isomorphism by the pairs (v_i^T, v_i). Lastly, we continue constructing the sub-tree with each (v_i^T, v_i) pair, as if they were the roots of the trees, until both are assigned a primitive task. The sub-tree G' then fulfils the criteria from the definition by construction. □

The definition of the PDT leaves several choice-points, e.g., how to select the sequence of children for a method or how to select the label sets for the nodes. When constructing our SAT formula, we are interested in a small PDT, i.e., one with the least amount of nodes and/or a PDT where the label set of each node is as small as possible. Since both parameters are relevant for the "size" of the PDT, defining a formal optimisation problem is complicated. Also, performing this optimisation might be infeasible as we can't construct all DTs in beforehand.

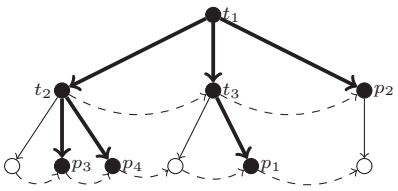

Figure 1: A PDT. A subgraph-DT is marked black.

Given a planning problem $\mathcal{P}$ and a heigh bound K, we can construct a PDT P_K using only a single function σ. σ essentially determines for a node v labelled with a set of tasks $\mathcal{T} = \alpha(v)$ which children it should have and how they should be labelled in the PDT. Formally σ has the signature $2^{C \cup O} \to \mathbb{N} \times (M \to \overline{\mathbb{N}})$, and returns for every possible label set of a node of P_K, the number of its children, and for every applicable method the positions of its subtasks. We call such a function a *child arrangement function*. The label set of the ith child must contain a task t, if there is a method for a task in $\mathcal{T}$, whose jth task is t, s.t. σ will assign the jth task in the method to the child i. More formally, the set of labels induced by σ for the ith child is:

$$\Sigma(i, \sigma, \mathcal{T}) = \{t \mid \sigma(\mathcal{T}) = (n, \mu), t \in \mathcal{T}, m = (t, tn) \in M,$$
$$\mu(m) = (c_1, \ldots, c_n), \exists j \in \{1, \ldots, n\} : c_j = i, tn(j) = t\}$$

Lastly, we can define the PDT induced by a child arrangement σ. To ease the definition, we here assume a slightly extended HTN formalism, in which we do not have an initial task, but a set of initial tasks $\mathcal{T}$, from which any can be chosen arbitrarily. In this construction, we use σ to construct the PDT locally and then recursively expand the tree until the height-limit K has been reached.

Definition 7. *Let $\mathcal{P} = (L, C, O, M, \mathcal{T}, s_I)$ be a planning problem, K be a height bound, and σ be a child arrangement function. Then the PDT P_K^σ of height K induced by σ is defined as follows:*

If $K = 0$, then $P_K^\sigma = (\circ, \emptyset, \{(\circ, \mathcal{T})\})$. Else let c be the number of children determined by $\sigma(\mathcal{T})$. Let for every child i be $P_i' = P_{K-1}^\sigma = (V_i, E_i, \alpha_i)$ the PDT of height $K - 1$ induced by σ, where the set of initial tasks is $\Sigma(i, \sigma, \mathcal{T})$. Assume that the vertex-sets of all P_i' are disjunct (else rename them appropriately). Let r_i be the root vertex of each P_i'. Then $P_K^\sigma = (V, E, \alpha)$ where

- $V = \{\circ\} \cup \bigcup_{i=1}^{c} V_i$
- $E = \{(\circ, (r_1, \ldots, r_c))\} \cup \bigcup_{i=1}^{c} E_i$
- $\alpha = \{(\circ, \mathcal{T})\} \cup \bigcup_{i=1}^{c} \alpha_i$

Clearly, the constructed tree is a valid PDT.

Proposition 1. *Given a planning problem $\mathcal{P}$ and a child arrangement function σ, P_K^σ is a valid PDT of height K.*

The last thing that remains to define is the child arrangement function σ. In our planner, we have used a simple scheme, were we have set the number of children always to the size of the largest applicable method. For each method its subtasks are assigned greedily to the children. We process the tasks in a method in the order occurring in that method and assign each task to one of the child nodes. When processing a task, we determine the first child node that this task can be assigned to (the one after the node that the previous task was assigned to). If that child is already labelled with the task, we use that child, else we try to use the next child – if enough children are left.

Translating PDTs into SAT

Based on the PDT P_K^σ we have described in the previous section, we now construct a SAT formula $\mathcal{F}(\mathcal{P}, K)$ that is satisfiable if and only if there is a DT of height $\leq K$ whose leafs form an executable sequence of actions. As shown in Theorem 2, such a DT is essentially a sub-tree G' of P_K^σ. In addition, its nodes must be labelled with tasks allowed by P_K^σ and its nodes must represent correct decompositions. Lastly, to be a solution the primitive tasks assigned to the leafs of G' must form an executable action sequence. The formula describes such a sub-tree G' of P_K^σ and will thus closely resemble the structure of the PDT P_K^σ. To ease the construction of the clauses expressing executability, we will enforce that the leafs of G' are also leafs of P_K^σ. We can achieve this by adding (as appropriate) new nodes to the leafs of G', which simply repeat the primitive task of the original leaf.

Initially, we separate the formula into two parts: one $\mathcal{F}_D(\mathcal{P})$ describing the sub-tree G' of the PDT, and one $\mathcal{F}_E(\mathcal{P})$ describing that the resulting primitive task network must be executable. The formula will contain subformulae enforcing that at most one of a set of atoms can be true. As the naive quadratic exclusion encoding produces to many clauses, we have employed a binary encoding for all our at-most-one constraints, which only results in $n \log n$ clauses while introducing $\log n$ additional variables (Sinz 2005). We will denote this construct with $\mathbb{M}(L)$ for a set of literals L. Further, we will not describe the formula in disjunctive normal form due to readability[3].

Every satisfying valuation of $\mathcal{F}_D(\mathcal{P})$ has to describe a single sub-tree G' contained in P_K^σ that is a valid DT. We use the structure of the PDT and define for every node v in the PDT the formula necessary to represent a (potential) sub-tree DT rooted at that node. Each node in P_K^σ is uniquely identified by the path $\pi = (p_1, p_2, \ldots, p_i)$ from the root to that node, where the path $\pi \circ i$ identifies the ith child of the node identified by π. From here on, we use the path of a node to denote that node, and write $K(\pi)$ for the height of π in P_K^σ.

The function $f(\pi)$ constructs said formula for each node π of the PDT P_K^σ. The formula contains the local conditions necessary for the described sub-tree to be valid DT and the formulae necessary for all its children. While constructing the formula, we will use two types of decision variables with the following meaning:

- t^π – stating that in G' the node π of P_K^σ is labelled with the task t, i.e., that $\alpha_{G'}(\pi) = t$.
- m^π – stating that in G' the method m is chosen to decompose the task $\alpha_{G'}(\pi)$, i.e., that $\beta_{G'}(\pi) = m$

If for some node π all t^π atoms are false, it is not part of the described DT G'. Essentially, the sub-tree G' will be defined by the nodes of P_K^σ that are assigned a task.

If $\alpha(\pi)$ only contains primitive tasks or $K(\pi)$ equals K, then $f(\pi)$ returns the formula $\mathbb{M}(\{t^\pi \mid t \in \alpha(\pi) \cap O\})$, i.e., if π is part of G', then it must be labelled with one primitive task in $\alpha(\pi)$. These nodes are the leafs of P_K^σ. If the mentioned condition is not true, we are dealing with an actual inner node of the PDT. If π is not part of G', then no task is assigned to it and thus all of its children cannot have a task assigned to them. Else, if the assigned task is abstract, a suitable decomposition is chosen, or if it is primitive, then the primitive task is inherited to the first child. Here we split $f(\pi)$ into six sub-formulae as follows:

$$\begin{aligned} f(\pi) = \mathbb{M}(\{t^\pi \mid t \in \alpha(\pi)\}) \wedge\ & \mathit{selectMethod}(\pi) \\ \wedge\ & \mathit{applyMethod}(\pi) \wedge \mathit{inheritPrimitive}(\pi) \\ \wedge\ & \mathit{nonePresent}(\pi) \wedge \mathit{subTree}(\pi) \end{aligned}$$

The first formula again ensures that π receives – if any – a unique task, as required for a DT. *selectMethod* ensures that a decomposition method is chosen if and only if the node π is assigned an abstract task, and that a single applicable method for that task is chosen. This also ensures that if a primitive task is chosen, no method can be applied.

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

The next formula *applyMethod* states the results of an applied method. We have to assign each task in a method's task network to one of π's children $\pi \circ i$. This assignment is determined by the child arrangement function σ, which we have described in the previous section. To ease notation, we denote the two values returned by $\sigma(\alpha(\pi))$ as c and μ (the number of children to produce and the task assignment function, respectively) and use the function $\Sigma(i, \sigma, \mathcal{T})$ from the previous chapter, which returns all tasks that can be assigned to the ith child. The formula itself forces that if a method is selected for a node, then the correct tasks are assigned to its children and that all children in P_K^σ that are not assigned a task cannot contain a task, i.e., are not part of G'.

$$\mathit{applyMethod}(\pi) = \bigwedge_{m=(t,tn) \in M(\alpha(\pi))} \left[m^\pi \rightarrow \left(\bigwedge_{i=1}^{|tn|} tn_i^{\pi \circ \mu(m)_i} \wedge \bigwedge_{i \in \{1,\ldots,c\} \setminus \mu(m)} \bigwedge_{k \in \Sigma(j,\sigma,\alpha(\pi))} \neg k^{\pi \circ j}\right)\right]$$

The next two formulae take care of the cases where the node π is either assigned a primitive action or none at all. In the

[3]The formula can easily be transformed into DNF, which is required by most SAT solvers.

former case it is passed on to one of the children in the tree, in the latter none of the children can be assigned a task.

$$inheritPrimitive(\pi) = \bigwedge_{p \in \alpha(\pi) \cap O} \left[p^\pi \rightarrow \left(p^{\pi \circ 1} \wedge \bigwedge_{i=2}^{c} \bigwedge_{k \in \Sigma(j,\sigma,\alpha(\pi))} \neg k^{\pi \circ j} \right) \right]$$

$$nonePresent(\pi) = \left(\bigwedge_{t \in \alpha(\pi)} t^\pi \right) \rightarrow \left(\bigwedge_{i=1}^{c} \bigwedge_{t \in \Sigma(i,\sigma,\alpha(\pi))} \neg t^{\pi \circ i} \right)$$

The formula $subTree(\pi) = \bigwedge_{i=1}^{c} f(\pi \circ i)$ is responsible for the recursive construction of the tree. As the full decomposition formula $\mathcal{F}_D(\mathcal{P})$, we choose $f(()) \wedge c_I^{()}$, i.e., the decomposition formula for the root and the assertion that the initial task is contained in G'.

The second part of $\mathcal{F}_E(P)$ expresses that the tasks assigned to the leafs of G' form an executable sequence of actions. We use an adaptation of the formula introduced by Kautz and Selman (1996) for this purpose. The difference is that we have to restrict the set of possible actions to the set $\alpha(\pi)$ for each task, and to connect the chosen action with the decomposition formula by using the variables t^π. Let $\Pi = (\pi_1, \ldots, \pi_n)$ be the leafs of the PDT in their order induced by P_K^σ. We introduce for every proposition symbol $l \in L$ the decision variable l^i for $0 \leq i \leq n$, stating that l is true after executing the action assigned to π_i. Then $\mathcal{F}_E(P)$ is defined as $\bigwedge_{l \in s_I} l^0 \wedge \bigwedge_{l \in L \setminus s_I} \neg l^0 \wedge \bigwedge_{i=0}^{n-1} (action(i) \wedge maintain(i))$ where

$$action(i) = \bigwedge_{t \in \alpha(\pi_{i+1}) \cap O} t^\pi \rightarrow \left(\bigwedge_{l \in prec(t)} l^i \wedge \bigwedge_{l \in add(t)} l^{i+1} \wedge \bigwedge_{l \in del(t)} \neg l^{i+1} \right)$$

$$maintain(i) = \bigwedge_{l \in L} \left[(\neg l^i \wedge l^{i+1}) \rightarrow \bigvee_{t \in \alpha(\pi_{i+1}) \cap O \text{ with } l \in add(t)} t^\pi \right]$$

As a last step in this chapter, we provide a proof sketch ensuring that the constructed formula is satisfiable iff the respective planning problem is solvable.

Theorem 3. *Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ be a planning problem and K a height bound. Then $\mathcal{F}(\mathcal{P}, K)$ is satisfiable if and only if $\mathcal{P}$ has a solution with a decomposition tree of height $\leq K$.*

Proof. $\Rightarrow$: Let γ be a satisfying valuation of $\mathcal{F}(\mathcal{P}, K)$. We can construct a DT $G' = (V, E, \alpha, \beta)$ as follows:

- $V = \{\pi \mid \exists t \in O \cup C : t^\pi$ is true and for the parent π' of π no $o^{\pi'}$ for $o \in O$ is true $\}$
- $E(\pi) = (\pi \circ i \mid \exists t \in O \cup C : t^{\pi \circ i}$ is true and $\pi \in V)$, sorted by i
- $\alpha(\pi) = t$ for the one atom t^π that is true – there is at most one due to the $\mathbb{M}$ constraints in the formula
- $\beta(\pi) = m$ for the one atom m^π that is true – *selectMethod* requires such an atom for all inner nodes and there is at most one due to the $\mathbb{M}$ constraint.

By the assertion of the atom $c_I^{()}$, we have $\alpha(r) = c_I$ for the root r of G'. The only thing left to show for G' is that for each inner node π, its children are labelled according to the method $\beta(\pi)$. This is ensured by the constraints of the formula *applyMethod*. Since only the nodes contained in V are assigned tasks, the sequence of primitive actions in $\mathcal{F}_E(\mathcal{P})$ is the same as the leafs of G'. As $\mathcal{F}_E(\mathcal{P})$ is satisfied, the sequence of primitive actions induced by G' is executable.

$\Leftarrow$: Let s be a solution of $\mathcal{P}$ and let T be a decomposition tree for s whose height is at most K. By Theorem 2 there exists a sub-tree G' of P_K^σ that is isomorphic to T and fulfils the criterion in Definition 6. Using G', we can construct a satisfying valuation of $\mathcal{F}(\mathcal{P}, K)$ as follows:

- t^π is true iff $\pi \in G'$ and the node v of T that is isomorphic to π is labelled with t, i.e., $\alpha_T(v) = t$, or if t is primitive and a predecessor π' of π whose isomorphic partner v in G' is labelled with t such that π is reachable from π only through first children.
- m^π is true iff $\pi \in G'$ and the node v of T that is isomorphic to π is decomposed using m, i.e., $\beta_T(v) = m$.
- l^i if l is true after executing i actions of s.

At most a single t^π and m^π is true for all nodes π. If t^π is true for some abstract task t, then a suitable method atom m^π is also true, as π must be an inner node of G' and π has a correct β-label. Also, if t is primitive no method is selected, as π is a leaf of G'. Since G' is a valid DT, the constraints in *applyMethod* are also satisfied (by the fourth point of a DT's definition). By the or-clause of the first bullet point, we also ensure that *inheritPrimitive* is satisfied. As G' is a tree *nonePresent* is satisfied. Lastly, since s is a solution, it executable and so $\mathcal{F}_E(\mathcal{P})$ is satisfied. □

Evaluation

To show that our HTN planner totSAT performs well in practice, we have conducted the following empirical evaluation. Our implementation of totSAT uses the parser and preprocessor of the planning system PANDA (Bercher, Keen, and Biundo 2014). We will release the code of totSAT publicly.

Domains. We have taken four commonly used HTN benchmark domains (UM-Translog, Woodworking, Satellite, and SmartPhone) from Bercher, Keen, and Biundo (2014). Since we have seen that our planner solves all these instances rather quickly, we have also added three new combinatorially more difficult domains.

ENTERTAINMENT is based on the non-hierarchical Assembly domain (Bercher et al. 2014), which describes setting up HiFi and video devices that can be connected with different types of cables. The goal is to connect the devices s.t. all available signals are transmitted to the appropriate signal sinks (e.g. TVs). The original domain is rather restrictive, e.g., cables can only be inserted if one of the connected devices already has a signal and unplugging cables is not possible. Our domain uses the additional expressivity of HTN planning to lift these two restrictions.

ROVER is the HTN-structure developed for SHOP together with the problem instances of the IPC3 domain ROVER.

	min	max	average	media
Variables	61	331132	18335	1627
Literals per clause	1.993	12.82	2.5	2.2
Horn clauses	83.8%	98.7%	91.4%	91.9%
Assertional clauses	0.0005%	5.63%	0.6%	0.2%

Table 1: Statistical data on all SAT formulae generated during planning. Percentages are given as average percentages over all formulae. Horn clauses are those containing exactly one positive literal. Assertional clauses are those containing only one literal.

	min	max	avg	min %	max %	avg %
SHOP vs optimal	0	5	+0.37	0%	+41%	3.2%
HTN2STRIPS vs optimal	0	1	+0.4	0%	+10%	+0.3%
TDG_c vs optimal	0	0	0	0%	0%	0%
TDG_m vs optimal	0	3	+0.14	0%	+25%	1.4%
totSAT vs optimal	0	9	0.58	0%	+90%	5.4%
totSAT vs SHOP	-17	20	+0.79	-53.8%	+90%	+0.79%
totSAT vs HTN2STRIPS	-2	+58	+5.48	-9.1 %	+287%	+36.9%

Table 2: Relative sizes of solutions. Given are statistics for the difference in number of actions. The first part of the table shows the comparison against known optimal solutions, computed using Dijkstra's Algorithm (Dijkstra 1959). The second part of the table shows the relative comparison of planners solving instances the optimal plan is unknown.

TRANSPORT is a domain describing a standard deliver-with-trucks scenario. There are several trucks (which do not need fuel) to deliver packages from their start location to a destination in a road network.

For UM-TRANSLOG, WOODWORKING, SATELLITE, and SMARTPHONE, we have added ordering constraints to ensure that the domains are totally-ordered. For all these domains solvability was retained by this change. ENTERTAINMENT, ROVER and TRANSPORT were constructed s.t. the domains are already totally-ordered. We will release these domains publicly.

Planners. Each planner was given 10 minutes runtime and 4 GB of RAM per instance on an Intel Xeon E5-2660.

We compare totSAT against two types of planning approaches: Those specifically designed for totally-ordered domains and those capable of handling unrestricted HTN planning problems. The first category consists of the planning strategies SHOP (Nau et al. 1999) and HTN2STRIPS (Alford et al. 2016a). We have compared us against both the original implementation of SHOP2 and the re-implementation of SHOP recently added to PANDA. Note that SHOP was originally developed for totally-ordered domains and SHOP2 is its extension to partial order, i.e., using SHOP2 on totally-ordered instances will lead to the behaviour described in the original SHOP-paper. HTN2STRIPS is based on a translation of HTN planing problems into classical planning (Alford et al. 2016a). This planner needs – similar to totSAT – a bound K for the translation, for which both a lower and an upper bound can be computed (Alford et al. 2016a). Given a time-bound t, the procedure of the planner is: run the classical planner (we used jasper (Xie, Müller, and Holte 2014), as did the original paper) on bound K with a timelimit of t minutes. If no solution was found, increment K and repeat. If the planner reached K's upper bound, we let it run until the total timelimit. We tried all possible values for t and found that the coverage of the planner is highest for $t = 3$min.

The second group of planners consists of general HTN planning techniques. All of them are based on plan-space search – in contrast to SHOP (using progression) and HTN2STRIPS. We have used the two currently best know heuristics for plan-based search in HTN planning (TDG_c and TDG_m, with and without recomputing (PR) (Bercher et al. 2017)) implemented in PANDA using Greedy-A* with a weight of 2. We also included the strategies DFS, BFS, and Dijkstra provided by PANDA, as well as its emulation of the UMCP planner (Erol, Hendler, and Nau 1994). Lastly, we also included the planner FAPE (Dvorak et al. 2014). However it does not accept all instances as input, as it cannot handle recursion in the domain. As such, we ran it only on the domains SATELLITE, WOODWORKING, and ROVER.

We have tested four SAT solvers for totSAT. Three of them are top performers at the SAT Competition 2016: cryptominisat5 (Soos 2016), MapleCOMSPS (Liang et al. 2016), and Riss6 (Manthey, Stephan, and Werner 2016). The fourth solver is standard unmodified minisat. Similar to HTN2STRIPS, totSAT has to try several values for the bound K in order to find a solution to the planning problem. The first value of K used by totSAT is the lowest height needed to reach a primitive task network. We then construct $\mathcal{F}(\mathcal{P}, K)$ and pass it to the SAT-solver. We have always set the timeout of the solver to the remaining runtime. Learnt clauses are not reused if the bound K is increased.

Discussion. In Table 1 we present statistical data on the generated SAT formulae. We can see that in all instances, the size of the formula is moderately large (never above $4 \cdot 10^5$). Also, most clauses of the formulae are horn – which are algorithmically easier to handle for a SAT solver. From this data we assume that the formulae are structurally relatively simple, which is wilful by construction. We think that our encoding actually exposes structural information about the domain in a way that is useful to the solvers.

Table 3 shows the overall coverage of all evaluated planners. We can see that totSAT solves – depending on the SAT solver – 123 to 125 of the 127 benchmark instances. For all domains except transport, totSAT solves all instances, while only two transport instances (#29 and #30) cannot be solved in time. Even using the simple SAT-solver minisat, only thee instance remain unsolved. totSAT would solve both these instance with 30 minute timelimit. Further we can see that the planners which are specifically designed for totally-ordered domains have a higher coverage than general HTN planner. This is expected, as they can exploit this special property during search. However, it is extremely surprising that the SHOP-strategy – which is a blind depth-first progression-search – outperforms the best known heuristic HTN plan-

	#instances	totSAT cms	totSAT Maple	totSAT Riss6	totSAT minisat	SHOP-PANDA	SHOP original	HTN2STRIPS	TDG-c	TDG-m	BFS	DFS	Dijkstra	FAPE	UMCP-BF	UMCP-DF	UMCP-H
UM-TRANSLOG	22	**22**	**22**	**22**	**22**	**22**	**22**	18	**22**	**22**	**22**	**22**	**22**	0	**22**	**22**	**22**
SATELLITE	25	**25**	**25**	**25**	**25**	**25**	**25**	23	24	**25**	24	**25**	**25**	**25**	24	18	20
WOODWORKING	11	**11**	**11**	**11**	**11**	**11**	10	5	6	9	6	9	9	0	6	6	6
SMARTPHONE	7	**7**	**7**	**7**	**7**	5	4	6	4	4	4	5	6	0	4	4	4
ENTERTAINMENT	12	**12**	**12**	**12**	**12**	9	5	5	5	5	5	9	9	0	5	5	6
ROVER	20	**20**	**20**	**20**	**20**	18	16	5	0	3	0	2	2	15	0	0	0
TRANSPORT	30	**28**	27	26	27	2	0	20	1	0	1	2	1	0	1	0	1
total	127	**125**	124	123	124	92	82	82	62	68	62	74	74	40	62	55	59

Table 3: Number of solved instances per planner per domain. Maxima are indicated in bold. cms = cryptominisat5

ners significantly. UM-Translog and Satellite are solved by practically every planner, i.e., we might suggest to remove these two from future benchmarks.

When looking at the new domains transport and rover we can clearly see the different advantages of SHOP and HTN2STRIPS on these domains. While SHOP is able to exploit the structure of the HTN in the rover domain, it cannot solve the combinatoric problem in transport. In contrast, HTN2STRIPS can solve transport easier, but struggles to utilise the information encoded in the rover domain. totSAT can solve both domains, showing that it combines the ability to solve combinatorially hard problems with the ability to exploit the HTN hierarchy. Especially the general HTN planners perform worse on these domains, showing the need for the development of better informed heuristics for HTN planning. This becomes even more obvious if we compare these heuristics with blind BFS – it outperforms them all.

Figure 2 shows the number of solved instances as a function of time. The choice of the SAT-solver seems to be unimportant, as all three show almost the same behaviour. We also see that totSAT solves more instances than all other approaches starting roughly after 4 seconds of runtime. Over all instances, totSAT calls the SAT-solver 3.71 (media 4) times, with a maximum of 8 calls before either a solution is found or the timeout is reached.

Lastly, although we did not aim for optimal planning, we have also evaluated the relatives quality of solutions in terms of their length. The results are shown in Table 2. totSAT tends to produce longer solutions, as it is allowed by the formula to assign tasks to all leafs. However, we think that it would be possible to create an optimal planner based on our encoding.

Conclusion

We have presented a new planning technique for HTN planning – SAT-based HTN planning. It is based on a translation of decomposition height bounded totally-ordered HTN planning problems into a SAT formula. This way, a planning problem can be solved by iteratively increasing this bound. We have evaluated it against state-of-the-art HTN planning systems and have shown that it outperforms all of them. The presented technique solves totally-ordered HTN planning problems – which are occurring naturally in practice. In view of our results, extending totSAT to partially ordered domains is the natural next step to further improve the performance of HTN planning. Furthermore, the presented SAT-encoding seems to be well suited as the basis for allowing additional constraints during planning. This occurs often in so-called mixed-initiative planning systems where users influence the plan generation process (Nothdurft et al. 2015). This integration can, e.g., be achieved by enabling the planner to process certain requests to change a current plan (Behnke et al. 2016), which could be transformed into additional SAT formulae.

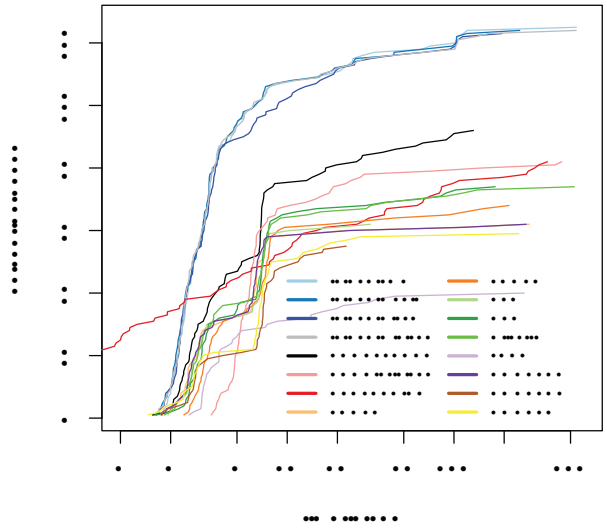

Figure 2: Runtime vs number of solved instances per planner

Acknowledgments

This work was done within the Transregional Collaborative Research Centre SFB/TRR 62 “Companion-Technology for Cognitive Technical Systems” funded by the German Research Foundation (DFG).

References

Alford, R.; Bercher, P.; and Aha, D. W. 2015. Tight bounds for HTN planning. In *Proc. of ICAPS 2015*.

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

Alford, R.; Shivashankar, V.; Roberts, M.; Frank, J.; and Aha, D. W. 2016b. Hierarchical planning: relating task and goal decomposition with task sharing. In *Proc. of IJCAI 2016*.

Alford, R.; Kuter, U.; and Nau, D. S. 2009. Translating HTNs to PDDL: A small amount of domain knowledge can go a long way. In *Proc. of IJCAI*.

Axelsson, R.; Heljanko, K.; and Lange, M. 2008. Analyzing context-free grammars using an incremental sat solver. In *Proc. of ICALP 2008*.

Behnke, G.; Höller, D.; Bercher, P.; and Biundo, S. 2016. Change the plan - how hard can that be? In *Proc. of ICAPS 2016*.

Behnke, G.; Höller, D.; and Biundo, S. 2015. On the complexity of HTN plan verification and its implications for plan recognition. In *Proc. of ICAPS 2015*.

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 2017*.

Bercher, P.; Biundo, S.; Geier, T.; Hörnle, T.; Nothdurft, F.; Richter, F.; and Schattenberg, B. 2014. Plan, repair, execute, explain - how planning helps to assemble your home theater. In *Proc. of ICAPS 2014*.

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 2015*.

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

Bercher, P.; Keen, S.; and Biundo, S. 2014. Hybrid planning heuristics based on task decomposition graphs. In *Proc. of SoCS 2014*.

Bit-Monnot, A.; Smith, D. E.; and Do, M. 2016. Delete-free reachability analysis for temporal and hierarchical planning. In *Proc. of HSDIP 2016*.

Byrne, R. 1977. Planning meals: Problem solving on a real data-base. *Cognition* 5:287–332.

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

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1(1):269–271.

Dix, J.; Kuter, U.; and Nau, D. 2003. Planning in answer set programming using ordered task decomposition. In *Proc. of KI 2003*.

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

Erol, K.; Hendler, J. A.; and Nau, D. S. 1994. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Proc. of AIPS 1994*.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence* 18(1).

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

Hartanto, R., and Hertzberg, J. 2008. Fusing DL reasoning with HTN planning. In *Proc. of KI 2008*.

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

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 2016*.

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

Liang, J. H.; Oh, C.; Ganesh, V.; Czarnecki, K.; and Poupart, P. 2016. MapleCOMSPS, MapleCOMSPS_LRB, MapleCOMSPS_CHB. In *Proc. of SAT Competition 2016*.

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

Manthey, N.; Stephan, A.; and Werner, E. 2016. Riss 6 solver and derivatives. In *Proc. of SAT Competition 2016*.

Morisset, B., and Ghallab, M. 2002. Learning how to combine sensory-motor modalities for a robust behavior. In *Advances in Plan-Based Control of Robotic Agents*.

Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In *Proc. of IJCAI 1999*.

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

Nothdurft, F.; Behnke, G.; Bercher, P.; Biundo, S.; and Minker, W. 2015. The interplay of user-centered dialog systems and AI planning. In *Proc. of SIGDIAL 2015*, 344–353. Association for Computational Linguistics.

Sinz, C. 2005. Towards an optimal CNF encoding of boolean cardinality constraints. In *Proc. of CP 2005*, volume 3709.

Soos, M. 2016. The CryptoMiniSat 5 set of solvers at SAT Competition 2016. In *Proc. of SAT Competition 2016*.

Xie, F.; Müller, M.; and Holte, R. 2014. Jasper: The art of exploration in greedy best first search. In *The 8th Int. Planning Competition*, 39–42.