Exploiting Solution Order Graphs and Path Decomposition Trees for More Efficient HTN Plan Verification via SAT Solving

Songtuan Lin[1], Gregor Behnke[2], Pascal Bercher[1]

[1] School of Computing, The Australian National University, Canberra, Australia
[2] ILLC, University of Amsterdam, Amsterdam, The Netherlands
{songtuan.lin, pascal.bercher}@anu.edu.au, g.behnke@uva.nl

Abstract

The task of plan verification is to decide whether a given plan is a solution to a planning problem. In this paper, we study the plan verification problem in the context of Hierarchical Task Network (HTN) planning. Concretely, we will develop a new SAT-based approach via exploiting the data structures *solution order graphs* and *path decomposition trees* employed by the state-of-the-art SAT-based HTN planner which transforms an HTN plan verification problem into a SAT formula. Additionally, for the purpose of completeness, we will also reimplement the old SAT-based plan verifier within an outdated planning system called PANDA$_3$ and integrate it into the new version called PANDA$_\pi$.

Introduction

Plan verification is the task of deciding whether a given plan is a solution to a planning problem. Research over the plan verification problem has drawn increasing attention in the last few years for its potential usages in numerous applications, e.g., in mixed initiative planning (see the work by Behnke, Höller, and Biundo (2017) for more details) and in International Planning Competition where a plan verifier is used to validate plans produced by participated planners.

We consider the plan verification problem in the context of HTN planning, which is a hierarchical approach to plan in which so-called abstract tasks are kept being refined until primitive ones (i.e., actions) are obtained. The HTN plan verification problem has been proved to be NP-complete (Behnke, Höller, and Biundo 2015; Bercher et al. 2016), and there exist three HTN plan verification approaches, namely, the SAT-based approach (Behnke, Höller, and Biundo 2017), the parsing-based approach (Barták, Maillard, and Cardoso 2018; Barták et al. 2020, 2021), and the planning-based approach (Höller et al. 2022) which transform a plan verification problem into a SAT problem, a language parsing problem, and an HTN planning problem, respectively. In this paper, we will develop a new SAT-based plan verification approach exploiting two data structures employed in the state-of-the-art SAT-based HTN planner (Behnke, Höller, and Biundo 2018, 2019a).

Specifically, we will adapt *solution order graphs* (SOGs) and *path decomposition trees* (PDTs) which are two data structures for formatting and storing refinement processes in an HTN planning problem. The core aspect of HTN plan verification, which is similar to solving an HTN planning problem, is to find a refinement process except that in plan verification, we demand that the refinement process must result in the given plan. These two data structures have been shown to be efficient in solving an HTN planning problem, but they are not exploited by the old SAT-based approach. Hence, we will adapt these data structures in our new SAT-based approach to see whether the performance can be improved. In order to distinguish our new SAT-based approach from the existing one, we call it the SOG-based approach and the existing one the DT-based approach where the term 'DT' refers to decomposition trees which we will introduce later on, and it is the core of the existing SAT approach. We emphasize 'SOG' because the majority of this paper aims to explain how to exploit SOGs in plan verification.

Apart from developing the new SOG-based plan verification approach, we will also reimplement the (old) SAT-based verifier. The old verifier is a part of an outdated HTN planning system called PANDA$_3$, which is written in JAVA and is now deprecated. Recently, a new version of PANDA called PANDA$_\pi$ has been developed which is written in C++. Thus, we would also like to rewrite the old verifier in C++ and integrate it into PANDA$_\pi$ for the purpose of completeness.

HTN Formalism

Before explaining the SOG-based approach, we first introduce the HTN formalism employed in the paper, which is an adoption of the one by Bercher, Alford, and Höller (2019). We start with the concept of task networks.

Definition 1. A task network tn is a tuple $(T, \prec, \alpha)$ where T is a set of task identifiers, $\prec \subseteq T \times T$ specifies the partial order defined over T, and α is a function that maps a task identifier to a task name.

Two task networks $tn = (T, \prec, \alpha)$ and $tn' = (T', \prec', \alpha')$ are said to be isomorphic, written $tn \cong tn'$, *iff* there exists a one-to-one mapping $\varphi : T \to T'$ such that for all $t \in T$, $\alpha(t) = \alpha'(\varphi(t))$, and for all $t_1, t_2 \in T$, if $(t_1, t_2) \in \prec$, $(\varphi(t_1), \varphi(t_2)) \in \prec'$.

Given a task network tn, the notations $T(tn)$, $\prec(tn)$, and $\alpha(tn)$ refer to the task identifier set, the partial order, and the identifier-name mapping function of tn, respectively. For convenience, we also define a restriction operation.

Definition 2. Let D and V be two arbitrary sets, $R \subseteq D \times D$ be a relation, $f : D \to V$ be a function and tn be a task network. The restrictions of R and f to some set X are defined by

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

Task names are further categorized as being primitive and compound. A primitive task name p, also called an action, is mapped to its precondition, add, and delete list by a function δ written $\delta(p) = (prec, add, del)$, where add and del are called the effects of p. On the other hand, a compound task name c can be refined (decomposed) into a task network tn by some method $m = (c, tn)$.

Definition 3. Let $tn = (T, \prec, \alpha)$ be a task network, $t \in T$ be a task identifier, c be a compound task name with $(t, c) \in \alpha$, and $m = (c, tn_m)$ be a method. We say m decomposes tn into another task network $tn' = (T', \prec', \alpha')$, written $tn \to_m tn'$, if and only if there exists a task network $tn'_m = (T_m, \prec_m, \alpha_m)$ with $tn'_m \cong tn_m$ such that

- $T' = (T \backslash \{t\}) \cup T_m$.
- $\prec' = (\prec \cup \prec_m \cup \prec_X)|_{T'}$ with $\prec_X = \{(t_1, t_2) \mid (t_1, t) \in \prec, t_2 \in T_m\} \cup \{(t_2, t_1) \mid (t, t_1) \in \prec, t_2 \in T_m\}$.
- $\alpha' = (\alpha \backslash \{(t, c)\}) \cup \alpha_m$.

An HTN planning problem is then defined as follows.

Definition 4. An HTN planning problem P is defined as a tuple (D, c_I, s_I) where D is called the domain of P. The domain D is a tuple (F, N_p, N_c, δ, M) in which F is a finite set of facts (i.e., propositions), N_p is a finite set of primitive task names, N_c is a finite set of compound task names with $N_c \cap N_p = \emptyset$, $\delta : N_p \to 2^F \times 2^F \times 2^F$ maps primitive task names to their preconditions and effects, and M is a set of (decomposition) methods. $c_I \in N_c$ is the initial task, which can be viewed as a task network consisting of solely one compound task, and $s_I \in 2^F$ is the initial state.

As mentioned in the introduction, the core aspect of solving an HTN planning problem (and solving a plan verification problem) is to find the decompositions which lead to a solution. Hence, before presenting the precise definition of a solution to an HTN planning problem, we would like to introduce the concept of *decomposition trees* (Geier and Bercher 2011) which capture such decomposition processes in an HTN planning problem.

Definition 5. Given a planning problem P, a decomposition tree $g = (V, E, \prec_g, \alpha_g, \beta_g)$ with respect to P is a set of labeled directed trees where V and E are the sets of vertices and edges respectively, $\prec_g$ is a partial order defined over V, $\alpha_g : V \to N_p \cup N_c$ labels a vertex with a task name, and β_g maps a vertex $v \in V$ to a method $(c, tn) \in M$.

A decomposition tree is *valid iff* for each $t \in T(tn_I)$, there exists a root vertex $r \in V$ labeled with $\alpha(tn_I)(t)$, and for each $v \in V$ with $\beta_g(v) = m$, $m = (c, tn)$, and $c \in N_c$, the following holds.

1) $\alpha_g(v) = c$.
2) tn is isomorphic to the task network induced by the children of v denoted as $ch(v)$, i.e.,
$$tn \cong (ch(v), \prec_g|_{ch(v)}, \alpha_g|_{ch(v)})$$
3) For any child v_c of v and any $v' \in V$, if $(v', v) \in \prec_g$, $(v', v_c) \in \prec_g$, and if $(v, v') \in \prec_g$, $(v_c, v') \in \prec_g$.
4) There are no other ordering constraints in $\prec_g$ except those demanded by 2) and 3).

The *yield* of a decomposition tree g, written $yield(g)$, is the task network $(T, \prec, \alpha)$ such that T is the set of all leafs of g, i.e., the set of all vertices which have no children, $(t_1, t_2) \in \prec$ *iff* $(t_1, t_2) \in \prec_g$ for any $t_1, t_2 \in T$, and $\alpha(t) = \alpha_g(t)$ for all $t \in T$.

Lastly, the solution criteria for HTN planning problems are defined as follows.

Definition 6. Let P be an HTN planning problem. A solution to P is a task network tn such that all tasks in it are primitive, there exists a valid decomposition tree g with respect to P such that $yield(g) = tn$, and it possesses a linearization of the tasks that is executable in the initial state.

Note that a solution to an HTN planning problem is a *partially ordered* task network, which is *not* a plan we refer to in practice, i.e., a plan is normally referred to as a *sequence* of actions (primitive tasks). Hence, we formally define the plan verification problem for HTN planning as follows.

Definition 7. Given a plan $\pi = \langle p_1 \cdots p_n \rangle$ ($n \in \mathbb{N}$) which is a sequence of primitive tasks and an HTN planning problem P, the plan verification problem for HTN planning is to decide whether there is a task network $tn = (T, \prec, \alpha)$ such that it is a solution to P, $|T| = n$, and it possesses a linearization $\overline{tn} = \langle t_1 \cdots t_n \rangle$ such that it is executable in the initial state of P, and for each $1 \leq i \leq n$, $\alpha(t_i) = p_i$.

Implementation of the SOG-based Approach

Having presented the HTN formalism, we now move on to introduce our SOG-based plan verification approach. We begin with introducing the data structures *path decomposition trees* (PDTs) and *solution order graphs* (SOGs) by Behnke, Höller, and Biundo (2019a).

Informally speaking, a PDT with depth K ($K \in \mathbb{N}$) stores *all* possible decomposition trees with depth at most K in an HTN planning problem.

Definition 8. A path decomposition tree $\mathcal{T}_K$ of a certain depth K ($K \in \mathbb{N}$) with respect to an HTN planning problem P is a labelled directed tree (V, E, γ) of depth K in which V is the set of vertices, E is the set of edges, $\gamma : V \to 2^{N_c \cup N_p}$ mapping each vertex to a set of task names, $\gamma(r) = \{c_I\}$ with $r \in V$ being the root of the tree (i.e., the vertex without ancestors), and for every inner vertex $v \in V$ which is neither the root nor a leaf (i.e., a vertex without children), it holds that for each $c \in \gamma(v) \cap N_c$ and every $m \in M$ with $m = (c, tn)$ and $tn = (T, \prec, \alpha)$, there exists a subset $S = \{v'_1, \cdots v'_{|T|}\}$ of v's children such that there exists a bijective mapping β_m from S to T which is also called a child arrangement function of v', and for every $v' \in S$, $\alpha(\beta_m(v')) \in \gamma(v')$.

We use $\mathcal{L}(\mathcal{T}_K)$ to refer to the set of all leafs of $\mathcal{T}_K$. From the definition, one might recognize that $\mathcal{L}(\mathcal{T}_K)$ stores the yields of all decomposition trees of depth smaller or equal to K. Hence, the idea of solving a plan verification problem in terms of PDTs is straightforward, that is, after constructing a PDT with a certain depth K, we check whether we can *select* a decomposition tree from it whose yield possesses a linearization which is identical to the plan. Particularly, Behnke, Höller, and Biundo (2017) have shown that for any instance of the plan verification problem, we can calculate the upper bound K such that the given plan is a solution *iff* it can be obtained from a decomposition tree of depth smaller or equal to K.

The decision procedure can be captured by a SAT formula. Particularly, the SAT clauses for constructing a PDT with a certain depth and selecting a decomposition tree from the PDT have already been given by Behnke, Höller, and Biundo (2019a), which are still exploited in the context of plan verification. Consequently, in this paper, we focus on constructing the clauses expressing the constraint that the yield of the selected decomposition tree must possess a linearization that is identical to the plan to be verified.

To this end, we shall also introduce the data structure solution order graphs (SOGs) by Behnke, Höller, and Biundo (2019a), which can significantly reduce the number of clauses and state variables required in constructing SAT formulae.

Definition 9. The solution order graph $\mathcal{S}(\mathcal{T}_K) = (\hat{V}, \hat{E})$ of a PDT $\mathcal{T}_K$ of a certain depth K is a directed graph in which $\hat{V} = \mathcal{L}(\mathcal{T}_K)$ is the set of vertices, and an edge $(v_1, v_2) \in \hat{E}$ *iff* for the least common ancestor v of v_1 and v_2, every method $m = (c, tn)$ with $c \in \gamma(v) \cap N_c$ and $tn = (T, \prec, \alpha)$, and the child arrangement function β_m, there exist two children $\hat{v}_1, \hat{v}_2$ of v such that $(\beta_m(\hat{v}_1), \beta_m(\hat{v}_2)) \in \prec$.

Intuitively, the SOG of a PDT $\mathcal{T}_K$ contains the yields of all possible decomposition trees of depth smaller or equal to K, and for each such yield $(T, \prec, \alpha)$, $(t_1, t_2) \in \prec$ for some $t_1, t_2 \in T$ *iff* there is an edge from v_1 to v_2 where v_1, v_2 are two vertices corresponding to t_1 and t_2, respectively.

Having presented the definitions of SOGs and PDTs, we now introduce the SAT clauses encoding that the yield of a decomposition tree selected from a SOG must possess a linearization that is identical to a given plan. For this, we can assume that we already have the SOG $\mathcal{S}(\mathcal{T}_K)$ of a PDT $\mathcal{T}_K$ in hand, because the remaining parts, i.e., constructing the PDT, extracting the SOG, selecting the yield of a decomposition tree from the SOG, and the SAT clauses for encoding these processes, have all been described in the work by Behnke, Höller, and Biundo (2019a).

Further, since the selection of decomposition trees has already been encoded, we can simplify our goal as constructing SAT clauses to encode that for a SOG, there must be a subset of the vertex set such that there is a total order of this subset respecting the edges (i.e., we can view each edge as an ordering constraint), and this chain forms the given plan via selecting a task name for each vertex from its label set, i.e., the set of all possible task names assigned to the vertex.

Given a plan $\pi = \langle p_1 \cdots p_n \rangle$ and a SOG $\mathcal{S}(\mathcal{T}_K) = (\hat{V}, \hat{E})$ of a PDT $\mathcal{T}_K = (V, E, \gamma)$, we start by introducing the SAT variables used to construct the clauses. For each $v \in \hat{V}$ and every *task name* $t \in \gamma(v)$, we construct a state variable v_t indicating whether t in v is selected. For every $1 \leq i \leq n$ and $v \in \hat{V}$ with $p_i \in \gamma(v)$, the variable m_v^i indicates whether p_i is mapped to the vertex v, and the variable f_v^i indicates whether mapping p_j to v is *forbidden* for *every* $1 \leq j \leq i$. In other words, if f_v^i is set to *true*, then any p_j with $1 \leq j \leq i$ cannot be mapped to v. For convenience, for each p_i, $1 \leq i \leq n$, we use $\mathcal{V}(p_i)$ to refer to the set of all vertices v such that $p_i \in \gamma(v)$. Conversely, for every $v \in \hat{V}$, $\mathcal{V}^{-1}(v)$ refers to the set of all *integers* i with $p_i \in \gamma(v)$. For every $v \in \hat{V}$, a_v indicates whether the vertex v is activated. The activation of a vertex here is associated with the selection of a decomposition tree, i.e., if a vertex is activated, then it must be a leaf of the selected decomposition tree, see the work by Behnke, Höller, and Biundo (2019a) for more details.

We first construct the clauses $\mathcal{F}_1$ to enforce the constraint that for every task p_i $(1 \leq i \leq n)$, if it is mapped to a vertex v with $v \in \mathcal{V}(p_i)$, then p_i in $\gamma(v)$ must be selected.

$$\mathcal{F}_1 = \bigwedge_{1 \leq i \leq n} \bigwedge_{v \in \mathcal{V}(p_i)} m_v^i \rightarrow v_{p_i}$$

Next, we construct the clauses to enforce that if a task p_i $(1 \leq i \leq n)$ is forbidden to be mapped to a vertex $v \in \hat{V}$, then the mapping cannot happen.

$$\mathcal{F}_2 = \bigwedge_{1 \leq i \leq n} \bigwedge_{v \in \mathcal{V}(p_i)} f_v^i \rightarrow \neg m_v^i$$

Further, we shall encode the transition of forbiddenness.

$$\mathcal{F}_3 = \bigwedge_{2 \leq i \leq n} \bigwedge_{v \in \mathcal{V}(p_i)} f_v^i \rightarrow f_v^{i-1}$$

Another important constraint we have to deal with is that the mapping between the plan and the SOG must respect the edges (i.e., ordering constraints). For every $v \in \hat{V}$, we use $\mathcal{V}^+(v)$ to refer to the set of all predecessors of v, i.e., the set of all vertices that are reachable from v. This constraint is then expressed as follows.

$$\mathcal{F}_4 = \bigwedge_{2 \leq i \leq n} \bigwedge_{v \in \mathcal{V}(p_i)} \bigwedge_{v' \in \mathcal{V}^+(v)} m_v^i \rightarrow f_{v'}^{i-1}$$

Informally, the formula $\mathcal{F}_3$ enforce that for any primitive task p_i with $2 \leq i \leq n$, if it is mapped to a vertex v in the SOG, then any predecessor of v is not allowed to be mapped to p_{i-1} for the purpose of respecting ordering constraints.

The next formula encodes the constraint that every action in the plan must be mapped to *at least one* vertex in the SOG.

$$\mathcal{F}_5 = \bigwedge_{1 \leq i \leq n} \Big(\bigvee_{v \in \mathcal{V}(p_i)} m_v^i \Big)$$

Simultaneously, every action in the plan is allowed to be mapped to *at most one* vertex in the SOG. To encode this constraint, we adopt the encoding by Sinz (2005) which enforces that, given a set X of SAT variables, at most one of them can be set to *true*. To ease the notation, we use $\mathbb{M}(X)$ to refer to the encoding. Hence, the constraint in our context is expressed as follows.

	Transport	Woodworking	UM-Translog	Satellite	Monroe-Partially-Observable	PCP	Monroe-Fully-Observable
Total Instances	188	137	52	246	103	26	129
SOG-based	188 (100.00%)	137 (100.00%)	52 (100.00%)	246 (100.00%)	102 (99.03%)	26 (100.00%)	128 (99.22%)
DT-based	138 (73.40%)	95 (69.34%)	52 (100.00%)	246 (100.00%)	0 (0.00%)	25 (96.15%)	0 (0.00%)

Table 1: The number of solved instances in each domain. The header shows the name of each domain. The first row indicates the total number of instances in each domain. The last two rows indicate the number of solved instances by the two approaches.

$$\mathcal{F}_6 = \bigwedge_{1 \le i \le n} \mathbb{M}(\{m_v^i \mid v \in \mathcal{V}(p_i)\})$$

Lastly, for every vertex in the SOG, if it is activated, then exactly one action in the plan is mapped to it.

$$\mathcal{F}_7 = \bigwedge_{v \in \hat{V}} a_v \to \Big((\bigvee_{i \in \mathcal{V}^{-1}(v)} m_v^i) \land \mathbb{M}(\{m_v^i \mid i \in \mathcal{V}^{-1}(v)\}) \Big)$$

The formula encoding that there exists a yield of a decomposition tree in the SOG whose linearization is identical to the given plan is thus the conjunction of the previous clauses.

Apart from implementing this SOG-based approach, we also reimplement the DT-based approach (Behnke, Höller, and Biundo 2017). Since the reimplementation is simply a translation from the original JAVA code to the C++ code, we omit the discussion of the technique details here. For more information, we refer to the original work by Behnke, Höller, and Biundo (2017).

Empirical Evaluation

We now compare the performance of our SOG-based approach with the reimplemented DT-based one. We used the benchmark set from the IPC 2020 on HTN Planning[1]. The benchmark set contains 1067 instances from 9 domains. However, two domains in the benchmark set, i.e., 'Rover' and 'Barman-BDI', feature so-called *method preconditions*, which are not yet supported by both SOG-based and DT-based (re)implemented in the paper. Consequently, our experiments are run on the remaining 7 domains which include 881 plan instances in total. For each instance, we gave it 10 minutes timeout and 8GB memory limit. All input planning problems were first grounded by the PANDA$_\pi$ grounder (Behnke et al. 2020).

The SOG-based approach successfully solved 879 instances and only failed in two (i.e., the two instances ran out of the timeout). The two failed instances are from the domains 'Monroe-Fully-Observable' and 'Monroe-Partially-Observable', respectively. In contrast, the reimplementation of the DT-based approach only solved 556 instances. Particularly, it failed in *all* instances from the domains 'Monroe-Fully-Observable' and 'Monroe-Partially-Observable', because the planning problems from these two domains contain too many methods and tasks which results in exceeding the memory limit. Tab. 1 shows the number of instances solved by these two approaches in *each* domain.

[1]https://github.com/panda-planner-dev/ipc-2020-plans/tree/master/po-plans

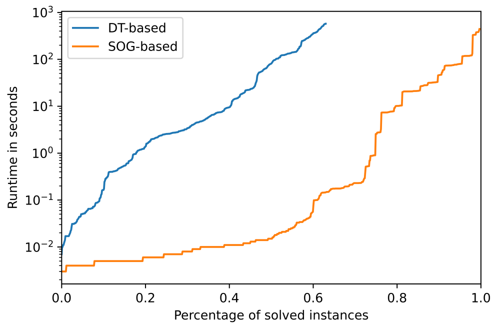

Figure 1: The runtimes against the percentages of solved instances by the SOG-based and the DT-based approach.

Further, Fig. 1 depicts the runtimes against the percentages of solved instances by the two approaches. From the figure, we can see that the new approach is significantly better than the old one. More concretely, over 70% of instances can be solved in one second by the new approach, whereas the number is only about 20% for the old approach.

Future Work and Discussion

In future work, we are going to implement an optimization in both approaches which discards the primitive tasks in a given planning problem which are *not* in a plan to be verified together with all compound tasks that can *only* be decomposed into them. The optimization is employed in the old SAT approach implemented in JAVA but not yet delivered by our reimplementation. One might recognize that the performance of our reimplementation is slightly worse than the old one in JAVA (the evaluation results are shown in the work by Höller et al. (2022)), and we assume that the underperformance is caused by the lack of the optimization.

Additionally, the optimization is also implemented in the planning-based approach (Höller et al. 2022) which transforms a plan verification problem into a planning problem. The planning-based approach happens to produce a SAT formula that is similar to the one produced by our SOG-based approach while the SAT *planner* is exploited. This is however not surprising because both our SOG-based plan verification approach and the SAT-based planner rely on PDTs and SOGs. Thus far, the performance of the planning-based approach with the SAT planner is also slightly better than our SOG-based approach which can solve all instances in the 7 domains on which we ran the empirical evaluation, and we believe that the reason for this is also the optimization.

More importantly, there are two functionalities that are yet delivered in both approaches implemented in the paper. The first is to support method preconditions. Although many HTN planning formalisms presented in literature do not feature method preconditions, they occur quite often in practice, e.g., in almost all totally ordered (TO) HTN planning domain from the IPC 2020 on HTN planning. Hence, for the purpose of providing an independent plan verifier for IPC, supporting method preconditions is mandatory.

The second functionality we want to implement is calculating a tight bound for the maximal depth of a decomposition tree. As mentioned earlier, a plan is a solution to a planning problem if and only if there exists a decomposition tree of depth smaller or equal to a certain bound. Thus far, the two approaches implemented in the paper calculate such a bound in a loose way, i.e., the obtained bound is significantly larger than the optimal one. As a consequence, our implementations will be less efficient when verifying a plan that is *not* a solution, because in such a case, both approaches need to construct a PDT (DT) of depth up to the bound. Hence, in the future work, we will adapt the approach by Behnke, Höller, and Biundo (2019b) for calculating an optimal bound for a decomposition tree.

As a preliminary work, this paper only did the comparison between the two (re)implemented approaches. In future work, we would like to give a complete comparison between our approaches and other existing ones, e.g., the parsing-based one (Barták, Maillard, and Cardoso 2018; Barták et al. 2020, 2021) and the planning-based one (Höller et al. 2022). On top of that, we would also investigate some theoretical properties of our approaches, e.g., the size of a SAT formula obtained.

Conclusion

In this paper, we developed a new SAT-based HTN plan verification approach which we call SOG-based approach and reimplemented an existing one (which we call DT-based approach). The empirical results show that the SOG-based one is significantly better than the reimplemented one. However, due to the lack of certain optimization techniques, the reimplementation of the DT-based approach slightly underperforms the original one in JAVA, and the SOG-based approach is also defeated by the state-of-the-art planning-based plan verification approach. Hence, we will further improve the (re)implementations in our future work.

References

Barták, R.; Maillard, A.; and Cardoso, R. C. 2018. Validation of Hierarchical Plans via Parsing of Attribute Grammars. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling, ICAPS 2018*, 11–19. AAAI.

Barták, R.; Ondrcková, S.; Behnke, G.; and Bercher, P. 2021. On the Verification of Totally-Ordered HTN Plans. In *Proceedings of the 33rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2021*, 263–267. IEEE.

Barták, R.; Ondrcková, S.; Maillard, A.; Behnke, G.; and Bercher, P. 2020. A Novel Parsing-based Approach for Verification of Hierarchical Plans. In *Proceedings of the 32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020*, 118–125. IEEE.

Behnke, G.; Höller, D.; and Biundo, S. 2015. On the Complexity of HTN Plan Verification and Its Implications for Plan Recognition. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling, ICAPS 2015*, 25–33. AAAI.

Behnke, G.; Höller, D.; and Biundo, S. 2017. This Is a Solution! (... But Is It Though?) - Verifying Solutions of Hierarchical Planning Problems. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling, ICAPS 2017*, 20–28. AAAI.

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

Behnke, G.; Höller, D.; and Biundo, S. 2019a. Bringing Order to Chaos - A Compact Representation of Partial Order in SAT-Based HTN Planning. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence, AAAI 2019*, 7520–7529. AAAI.

Behnke, G.; Höller, D.; and Biundo, S. 2019b. Finding Optimal Solutions in HTN Planning - A SAT-based Approach. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI 2019*, 5500–5508. IJCAI.

Behnke, G.; Höller, D.; Schmid, A.; Bercher, P.; and Biundo, S. 2020. On Succinct Groundings of HTN Planning Problems. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence, AAAI 2020*, 9775–9784. AAAI.

Bercher, P.; Alford, R.; and Höller, D. 2019. A Survey on Hierarchical Planning - One Abstract Idea, Many Concrete Realizations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI 2019*, 6267–6275. IJCAI.

Bercher, P.; Höller, D.; Behnke, G.; and Biundo, S. 2016. More than a Name? On Implications of Preconditions and Effects of Compound HTN Planning Tasks. In *Proceedings of the 22nd European Conference on Artificial Intelligence, ECAI 2016*, volume 285, 225–233. IOS.

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

Höller, D.; Wichlacz, J.; Bercher, P.; and Behnke, G. 2022. Compiling HTN Plan Verification Problems into HTN Planning Problems. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling, ICAPS 2022*. AAAI.

Sinz, C. 2005. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *Proceedings of the 11th Principles and Practice of Constraint Programming, CP 2005*, 827–831. Springer.