On Heuristics for Parsing-based Verification of Hierarchical Plans with a Goal Task

Simona Ondrčková and **Roman Barták**
Faculty of Mathematics and Physics
Charles University
Prague, Czech Republic
{ondrckova,bartak}@ktiml.mff.cuni.cz

Pascal Bercher
School of Computing
The Australian National University
Canberra, Australia
pascal.bercher@anu.edu.au

Gregor Behnke
Faculty of Engineering
University of Freiburg, Germany
ILLC, University of Amsterdam
The Netherlands
g.behnke@uva.nl

Abstract

Verification of hierarchical plans deals with the problem if a given action sequence is a valid hierarchical plan – the action sequence can be obtained by decomposing a particular (goal) task using given decomposition methods. The existing parsing-based verification approach greedily composes actions until it obtains the goal task. Greediness implies that this approach also generates tasks unrelated to the goal task. In this paper, we study the use of heuristics when creating new tasks. We also ask whether the prior knowledge of the goal task improves efficiency.

Introduction

In plan verification we must find out whether a given action sequence forms a correct plan according to a domain model. In classical planning this consists of verifying that each action is executable and that the goal condition is satisfied in the final state (Howey and Long 2003). In hierarchical planning we additionally require that a certain task decomposes into the given action sequence (the plan). A specific root task might be given as the goal task. In that case, for the plan to be valid, the goal task must decompose into the action sequence. Similarly to a goal condition in classical planning, there can be a goal description in hierarchical planning. The goal description is a list of proposition that must be true in the state after the last action was executed. The plan verification problem is NP-hard (Behnke, Höller, and Biundo 2015; Bercher et al. 2016).

There exist three approaches to hierarchical plan verification. One uses a translation of the verification problem into a Boolean satisfiability problem (Behnke, Höller, and Biundo 2017) (SAT approach). The second one translates the problem into a planning problem (Höller et al. 2022) (planning approach). The third one uses parsing (Barták et al. 2018; 2020; 2021b) (parsing approach).

Hierarchical domain models show similarities to a formal grammar (Höller et al. 2014; 2016; Barták and Maillard 2017). The plan verification problem is like checking if a word (action sequence) belongs to the language generated by the grammar (domain model). This can be done with parsing. The parsing approach is a bottom-up approach. It first builds tasks (layer 1 tasks), whose sub-tasks are actions. Then it builds new tasks that contain layer 1 tasks in their sub-tasks. Then it continues layer by layer until it gets a task that decomposes into all actions.

It is important to note that the parsing approach does not need information about the goal task. It can find any task that decomposes into the given sequence of actions. Therefore it can also be used in plan recognition (Vilain 1990) or plan correction (Barták et al. 2021a). However, that is not the standard definition of plan verification. In the standard definition, the verifier is provided with a goal task and sometimes also a goal description.

This paper focuses on improvements to the parsing approach. We want to utilize heuristics when creating a new task. We will create a version of the parsing approach that solves the standard definition of plan verification and utilizes the information about the goal. We shall refer to the information about goal task and goal description as *goal knowledge*. Then we will provide an empirical comparison with all the verification approaches.

HTN Plan Verification by Parsing

Hierarchical Task Network Planning (Erol, Hendler, and Nau 1996) is a planning framework that focuses on decomposition. Specifically, actions in the plan are obtained from higher-level (compound) tasks that decompose to sub-tasks until primitive tasks – actions – are obtained.

As explained in paper (Barták et al. 2021b) the parsing approach uses the STRIPS model of actions (Fikes and Nilsson 1971). Let P be a set of propositions that describe properties of a world state. Each world state is a set $S \subseteq P$ of propositions that are true. Propositions that are not in that state are false. Each action a is modeled by sets of propositions $(\texttt{pre}(a), \texttt{eff}^+(a), \texttt{eff}^-(a))$, where $\texttt{pre}(a), \texttt{eff}^+(a), \texttt{eff}^-(a) \subseteq P$. Preconditions $\texttt{pre}(a)$ are propositions that must be true before applying the action a. The positive $(\texttt{eff}^+(a))$ and negative effects $(\texttt{eff}^-(a))$ of action a cannot overlap $\texttt{eff}^+(a) \cap \texttt{eff}^-(a) = \emptyset$. Propositions $\texttt{eff}^+(a)$ will become true while propositions $\texttt{eff}^-(a)$ will become false after applying action a. Action a is *applicable* to state S if $\texttt{pre}(a) \subseteq S$. If action a is applicable to state S, the state right after applying action a is:

$$\gamma(S, a) = (S \setminus \texttt{eff}^-(a)) \cup \texttt{eff}^+(a).$$

Note that $\gamma(S, a)$ is undefined if action a is not applicable to state S. Let S_i be a state at position i. An action sequence $(a_1, \ldots, a_n)$ is *executable* with respect to a given initial state if each action is applicable to the state right before it.

Let T be a compound task and $(\{T_1, ..., T_k\}, C)$ be a task network, where C are its constraints. We describe the decomposition method as a rule that T decomposes to sub-tasks $T_1, ..., T_k$ under the constraints C:

$$T \rightarrow T_1, ..., T_k \ \ [C]$$

As the precedence constraints are stated explicitly in C, the order of sub-tasks on the right side of the rule is irrelevant.

Some tasks may not decompose into any sub-tasks (i.e., their decomposition methods don't mention tasks to decompose into). We demand them to have only before conditions (more on the conditions later). We call these tasks empty tasks. An example of an empty task is a task *get-to-already-there*. The task *get-to-already-there(truck,loc)* will have a before condition: *at(truck,loc)*. Let's assume we have a truck that needs to get to some location. However, if the truck is already there, it does not need to move.

Each task instance (T) in the decomposition has a start index indicated as *start(T)* and an end index indicated as *end(T)*. For an empty task the start index and end index is the position of the state in which its before conditions are satisfied. For a primitive task (action) the start index and end index are equal to the number of the action in the plan. For a task instance T that decomposes into other sub-tasks using the decomposition rule above ($T \rightarrow T_1, ..., T_k \ \ [C]$) the start and end index are as defined below:

$$start(T) = min(start(T_1), ..., start(T_k))$$

$$end(T) = max(end(T_1), ..., end(T_k)$$

Note that if there are multiple decomposition rules on how to decompose a task, we focus on the one that was used to create this specific task instance.

The constraints in C in the decomposition rule can be of three different types (first is an ordering constraint and the other two are decomposition constraints):

- $T_1 \prec T_2$: an ordering constraint meaning that in every plan, task T_1 is before task T_2. $end(T_1) < start(T_2)$.
- *before*(p, T): a precedence constraint meaning that in every plan, the proposition p holds in the state right before the first action obtained from task T.
- *between*(T_1, p, T_2): a prevailing constraint meaning that in every plan, the proposition p holds in all the states between the last action from the decomposition of task T_1 and the first action from the decomposition of task T_2.

The parsing approach begins by taking the initial state and creating the following state by applying the effects of the first action. It continues this way until all states are created. Then it checks whether the action sequence is executable with respect to the initial state. If not, the plan is invalid. If so, it continues by building the decomposition structure.

To build it, the parsing approach creates tasks. It begins by creating layer 1 tasks whose sub-tasks are actions. Once all layer 1 tasks are created it continues by creating layer 2 tasks, tasks whose sub-tasks are layer 1 tasks and actions. The approach continues this way layer by layer until it finds a task that decomposes into all actions (plan is valid) or until there are no more tasks to create (plan is invalid).

Every time the approach creates a task for a specific layer it makes sure that at least one sub-task is new (from the last layer). This ensures that we don't create the same task instances over and over again. When creating new tasks, the approach takes inspiration from the RETE algorithm (Forgy 1982). Each task remembers rules in which it is a sub-task. When a new instance of the task is created, the task will inform the rules. The moment all sub-tasks of a rule have an instance, the rule is "fired". This means it will try to create a new task instance based on these sub-tasks. A pseudocode and more detailed description of the parsing approach can be found in (Barták et al. 2021b).

Heuristics

Let us discuss first how we can use heuristics to increase the efficiency of the parsing approach. Let's look at a creation of task T (the sub-tasks are unordered):

$$T(x, y, z) \rightarrow A(x), B(y), C(x, y, z).$$

Such "lifted" tasks (with variables) represent a set of "groundings" (instantiating of variables x, y, z with given constants). Based on actions' preconditions and effects, grounding procedures can identify a subset of reachable groundings (Behnke et al. 2020). Assume the following groundings are available: $A(x1), A(x2)$ $B(y1), B(y2)$ $C(x3, y3, z1)$

The original parsing algorithm goes through the right side of the rule (sub-tasks) from left to right and it fills the main task T with the appropriate parameters. If the combination is not valid, then it removes this partial assignment from the list. If any partial assignment is still valid the program continues with the next sub-task. Let's look at an example. These are the partial assignments after task A:

$T1(x1, -, -)$ from $A(x1)$
$T2(x2, -, -)$ from $A(x2)$.

It takes two steps to create them. Then we combine them with task B. This takes 4 additional steps:

$T3(x1, y1, -)$ from $B(y1)$ and $T1$
$T4(x2, y1, -)$ from $B(y1)$ and $T2$
$T5(x1, y2, -)$ from $B(y2)$ and $T1$
$T6(x2, y2, -)$ from $B(y2)$ and $T2$

Then we try to combine each of these partial assignments with C only to discover that none of them are valid (4 more steps). In total we used 10 steps.

If we started with sub-task C, we would get this partial assignment: $T1(x3, y3, z1)$. Then we would try to combine it with sub-task A only to find no combination is valid. Since there are no partial assignments left we don't need to check with sub-task B. This only takes 3 steps (1 step for C and then 2 for A).

To test how this affects the efficiency of the program we have created four heuristics: *Most Parameters* (h0), *Least Parameters* (h1), *Original* (h2) and *Instances* (h3). The Most Parameters heuristic takes the sub-task with the most parameters first. The Least Parameters heuristic takes the sub-task

with the least parameters first. Original goes through sub-tasks from left to right. Finally, the Instances heuristic first takes the sub-task with the smallest number of instances.

In the example above we can see that both the Most Parameters heuristic and the Instances heuristic choose task C first (it only has one instance and it has most parameters) and then choose tasks A and B. The Original and Least Parameters heuristics go through tasks A and B first and finish with C last. So by using the Most Parameters or Instance heuristic we can save seven steps.

Note that each of these heuristics first takes the newest sub-task and then orders the other sub-tasks based on the heuristic. This is to ensure we don't create the same task multiple times. There are a few other technicalities that affect how we order the sub-tasks based on how new the task is. However, since these are all the same for each heuristic, we will not focus on them here.

Goal Knowledge

As discussed earlier, standard plan verification problems by definition provide a goal task and sometime goal description. Together, we refer to this as goal knowledge. The parsing approach by Barták et al. does however solve a different problem, namely answering the question of whether the given sequence can be obtained by *some* compound task. We thus extend their work to be applicable to standard plan verification problems where a goal task is provided in the input.

The parsing approach works in a bottom up manner – it creates all possible tasks from the ground up until it finds a task that decomposes into all actions. However, this means that it might create tasks that are unrelated to the given goal task. In this paper, we would like to utilize the information about the goal (goal knowledge) to reduce the number of tasks created. The goal task can be given as a single task that is already in the domain or as a new task that decomposes into multiple sub-tasks from the domain. To compensate for this we create a new task that decomposes into these sub-tasks. This new task is the goal task.

We utilize the goal knowledge to reduce the number of tasks created. First, we mark each lifted task that is reachable from the goal task. Then we mark recursively each sub-task of these sub-tasks. Once all reachable sub-tasks are marked, we go through all tasks again and remove those that are unreachable (i.e., not marked). This is done before any variable substitutions by constants are performed.

We also remove any method that would create an unreachable task. Let's assume we have tasks $T_1, ..., T_4$ and a plan consisting of two actions a_1 and a_2. Then let's assume we are given a goal as a new task: $G \rightarrow T_1, T_2$. These are the decomposition rules of all the other tasks:

$$T_1 \rightarrow T_3, T_2 \rightarrow a_1, T_3 \rightarrow a_2, T_4 \rightarrow a_2$$

After the reachability analysis tasks T_1, T_2 and T_3 are marked. Task T_4 isn't, so it's removed. The original parsing approach would create all 4 tasks as they decompose into the given actions (or other sub-tasks). However the new parsing approach will only create tasks T_1, T_2 and T_3 as T_4 is unreachable.

As we remove all unreachable tasks we prune the number of tasks the approach might create. This pruning should allow us to find the solution faster because we won't create any unreachable tasks. When it comes to empty tasks we still create them even if they are unreachable but we don't build anything on top of them.

Some domains are very versatile for example a planning domain Monroe (used in International Planning Competitions) has many top level tasks like: *shut-off-water-main* or *set-up-shelter*. If we know the goal task is *set-up-shelter*, we don't need the tasks for shutting water off. For these types of domains, the knowledge of goal task should be very useful. On the other hand, some domains only have one very obvious top level task. For example, the domain Transport almost always uses a goal task *deliver*, which decomposes into all the other tasks in the domain (*pick-up*, *drop*, *get-to*...). In this case the knowledge of the goal task does not help at all and only adds some time by doing the analysis. If many domains only have one obvious top task, the goal knowledge will not be much faster than the original parsing approach.

Some domains also provide a goal description. For this, we first create a timeline of all states and then we determine whether the state after the last action satisfies the goal description. If not we can stop and the plan is invalid.

Empirical Evaluation

We ran the experiments on an Xeon Gold 6242 CPU. For each instance, each verifier was given 10 minutes of runtime and 8 GB of RAM. The domains are split into four groups: totally ordered valid domains (to-val), totally ordered invalid domains (to-inval), partially ordered valid (po-val) and partially ordered invalid domains (po-inval).

We are using two versions of the parsing approach. One utilizes the goal knowledge (goal knowledge version) and one runs just like the original algorithm (finds any task that decomposes into all actions, not necessarily the goal task). First, we tested on domains with goal knowledge and with and without method preconditions (lines 1-8 in Table 1). Then we did two more experiments without goal knowledge and with and without method preconditions. Note that the domain might have still provided the goal but the parsing approach wouldn't have used it and instead found any task that decomposed into all actions.

There wasn't a big difference in the number of solved instances between the different heuristics for the parsing approach (first 4 lines and H0-H3 columns in Table 1). The best heuristic is the Instances heuristic (H3) which solved only 19 more instances than the Original heuristic (H2).

We believe this might be due to two reasons. First on domains that only have a few sub-tasks, the heuristics won't save many steps. This is because the number of steps tends to increase the more sub-tasks a task has. The second reason might be that while the heuristics save a certain number of steps, the steps are insignificant in the run of the whole algorithm. For a more in-depth analysis, we show in Tables 2 and 3 the number of verified valid total-order plans per domain. Both of these tables show the setting with goal knowledge. Table 2 shows the data for instances with method preconditions and Table 3 for instances without method precondi-

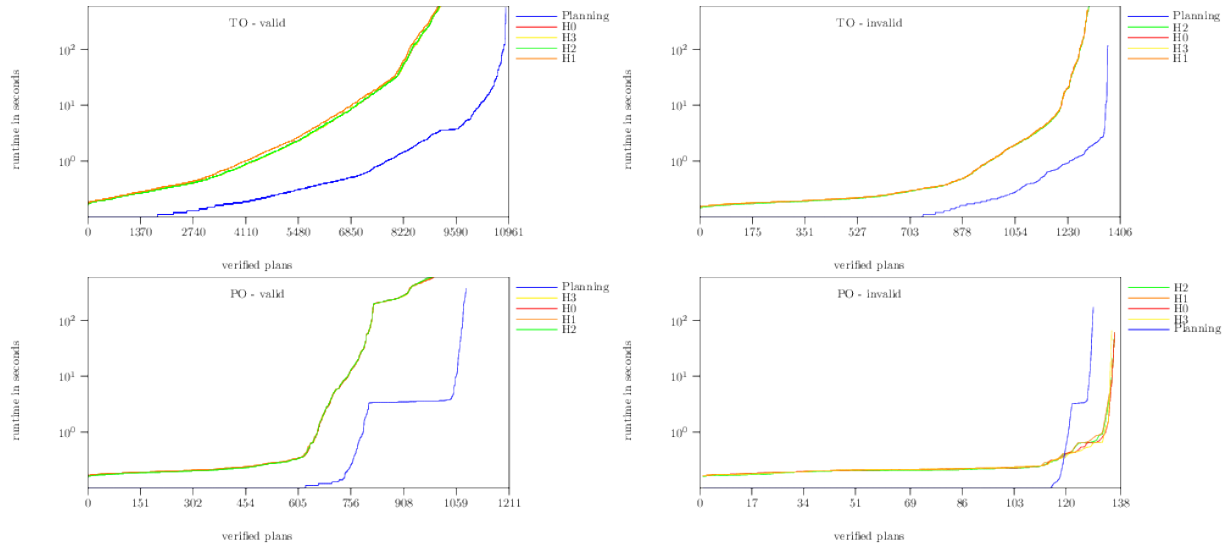

Figure 1: Comparison of verifiers with goal knowledge on domains with method preconditions.

tions. We focus only on the valid total-order plans as they form the largest part of the benchmark set. In most domains the heuristics don't influence the number of verified plans at all.

The exceptions (with method preconditions) are Factories-simple (H1 and H2 have 8 and 6 fewer instances), Monroe-Fully-Observable (H1 and H2 have about 25 fewer instances), and Transport (H2 and H3 have 4 fewer). Monroe-Partially-Observable is the domain on which the heuristics performed the best as every heuristics performed better than the original one (H2 has about 10-15 fewer). Both Monroe domains are generally hard to ground, have many tasks with many parameters and overall many more tasks than other domains and thus a much more complex decomposition structure. This allows our heuristics to optimise task instantiating a lot. For the set without method preconditions, the difference stems mostly from Barman-BDI. Longer Barman-BDI plans have a high number of objects (up to 55 shots, 50 ingredients, 50 cocktails). Also Barman-BDI contains methods with many parameters (up to 6) while tasks have fewer parameters (2 at most). All of this causes a higher influence of our heuristics.

In addition, we also provide information on the plan length of plan verified with H0, the shortest unverified plans as well as run-time statistics per domain including Pearson correlation of the run-time with the plan length. One notable item are the two Minecraft domains, where we fail to verify any plan. Here the Planning-based approach verifies most plans when given method preconditions, but non without. This provides us with an avenue of possible future work: better integrating method preconditions into parsing.

Let's now look at how goal knowledge affects the efficiency. The number of instances differs for runs with and without goal knowledge (lines 1-4 and 9-12 in Table 1). This is because some instances specify a goal description. If it is not satisfied, the runs with goal description knowledge will stop immediately after checking it and determine the instance invalid. However the runs without goal description knowledge just continue to try to find a task that decomposes into all actions. There are 73 instances in the *to* categories that turn from valid to invalid and 1 in the *po* categories. Once we subtract that, we can see that both versions (for H3) solve similar number of instances (less than 10 difference).

We have also compared the parsing verifier against other verification approaches. Firstly, we used a verifier that translates the verification problem into a SAT problem (Behnke, Höller, and Biundo 2017). We abbreviate it as SAT. This verifier cannot deal with method preconditions (before conditions for methods). Therefore we ran ours tests twice: once with and once without method preconditions (lines 1-4 and 5-8 in Table 1). Without here means that we have removed all method preconditions in the domain model. Note that as for the knowledge of the goal task, removing method preconditions can make previously invalid plans valid. Note that we did not use the original configuration of this verifier. Since the original grounding procedure used by the verifier is known to be inefficient (Wichlacz, Torralba, and Hoffmann 2019), we have replaced it with a newer and more efficient grounder (Behnke et al. 2020).

Secondly, we compare against a verification approach that uses planning itself (Höller et al. 2022). We denote it with "Planning Approach". This verifier compiles the plan to be verified and the planning domain into a new, modified planning problem. It supports method preconditions. Both these verifiers take the goal knowledge into account by default and cannot disable this. Thus a comparison with the parsing version that does not take the goal task into account is meaningless (noted as *no support* in Table 1).

The parsing approach solves more instances than both SAT and Planning approach on partially ordered invalid domains. On the other three sets of domains it is worse than the planning approach but better than the SAT approach. In

Benchmark	Instances	H0		H1		H2		H3		Planning Approach		SAT	
to-val	10961	9158	(83.55)	9117	(83.18)	9158	(83.55)	9177	(83.72)	**10881**	(**99.27**)	*no support*	
to-inval	1406	1299	(92.39)	1299	(92.39)	1301	(92.53)	1299	(92.39)	**1364**	(**97.01**)	*no support*	
po-val	1211	993	(82.00)	995	(82.16)	989	(81.67)	990	(81.75)	**1088**	(**89.84**)	*no support*	
po-inval	138	**136**	(**98.55**)	**136**	(**98.55**)	**136**	(**98.55**)	135	(97.83)	129	(93.48)	*no support*	
to-val-no-mprec	11264	7962	(70.69)	7811	(69.34)	7889	(70.04)	7958	(70.65)	**9658**	(**85.74**)	1036	(9.20)
to-inval-no-mprec	1103	915	(82.96)	915	(82.96)	915	(82.96)	915	(82.96)	**921**	(**83.50**)	684	(62.01)
po-val-no-mprec	1243	973	(78.28)	968	(77.88)	973	(78.28)	973	(78.28)	**1103**	(**88.74**)	897	(72.16)
po-inval-no-mprec	106	**106**	(**100.00**)	**106**	(**100.00**)	**106**	(**100.00**)	**106**	(**100.00**)	98	(92.45)	103	(97.17)
to-val-no-gs	11034	9230	(83.65)	9189	(83.28)	9221	(83.57)	**9253**	(**83.86**)	*no support*		*no support*	
to-inval-no-gs	1333	1223	(91.75)	1223	(91.75)	1223	(91.75)	**1224**	(**91.82**)	*no support*		*no support*	
po-val-no-gs	1212	999	(82.43)	998	(82.34)	**1001**	(**82.59**)	994	(82.01)	*no support*		*no support*	
po-inval-no-gs	137	**136**	(**99.27**)	**136**	(**99.27**)	**136**	(**99.27**)	135	(98.54)	*no support*		*no support*	
to-val-no-gs-no-mprec	11329	9525	(84.08)	9484	(83.71)	9516	(84.00)	**9548**	(**84.28**)	*no support*		*no support*	
to-inval-no-gs-no-mprec	1038	928	(89.40)	928	(89.40)	928	(89.40)	**929**	(**89.50**)	*no support*		*no support*	
po-val-no-gs-no-mprec	1244	1030	(82.80)	1029	(82.72)	**1032**	(**82.96**)	1024	(82.32)	*no support*		*no support*	
po-inval-no-gs-no-mprec	105	**105**	(**100.00**)	**105**	(**100.00**)	**105**	(**100.00**)	**105**	(**100.00**)	*no support*		*no support*	

Table 1: Table comparing runs of multiple approaches for plan verification.

Domain	#Plans	Verifier					Shortest	Plan Length			Runtime for Verified			Pearson
		H0	H1	H2	H3	Planning	unverified plan H0	Min–Max	Avg	Median	Min–Max	Avg	Median elation	Corr-
AssemblyHierarchical	193	135	135	135	135	**193**	34	4 – 256	31.1	14	0.4 – 571.411	35.7	0.968	0.618
Barman-BDI	423	397	372	397	396	**423**	273	10 – 1198	128.4	69	0.3 – 557.186	15.9	0.533	0.813
Blocksworld-GTOHP	158	124	124	124	118	**158**	466	21 – 6661	482.3	209.5	0.18 – 361.864	34.9	9.114	0.574
Blocksworld-HPDDL	172	166	166	166	165	**170**	3377	20 – 5732	461.1	163	1 – 501.249	13.9	0.2955	0.941
Childsnack	529	**529**	528	**529**	**529**	519	none	50 – 2500	119.8	80	0.4 – 500.954	2.6	0.385	0.764
Depots	455	427	427	427	427	**455**	524	15 – 971	129.1	92	0.2 – 450.339	20.1	0.428	0.934
Elevator-Learned-ECAI-16	2812	2790	2790	2789	2790	**2812**	1575	10 – 2165	225.1	200	0.2 – 527.518	10.1	19.045	0.894
Entertainment	159	**159**	**159**	**159**	**159**	**159**	none	24 – 128	71.7	64	0.3 – 2.391	1.0	0.562	0.361
Factories-simple	123	107	99	101	107	**123**	1636	15 – 2968	623.7	251	0.18 – 545.556	66.7	0.765	0.899
Freecell-Learned-ECAI-16	204	**204**	**204**	**204**	**204**	**204**	none	57 – 489	162.7	138.5	0.7 – 53.494	4.4	1.3245	0.889
Hiking	565	**565**	**565**	**565**	**565**	**565**	none	26 – 174	70.8	72	1 – 57.689	1.9	0.598	0.604
Logistics-Learned-ECAI-16	1108	1097	1097	1099	1097	**1108**	1033	27 – 2813	413.1	370	0.3 – 560.817	56.1	11.057	0.857
Minecraft-Player	75	0	0	0	0	**74**	35	35 – 278	51.9	44	–	–	0	–
Minecraft-Regular	766	0	0	0	0	**722**	35	35 – 9947	253.8	135	–	–	0	–
Monroe-Fully-Observable	248	154	150	174	177	**248**	16	3 – 96	41.5	39	0.68 – 575.484	318.1	262.294	0.318
Monroe-Partially-Observable	217	58	59	46	60	**217**	18	6 – 91	45.1	45	0.63 – 599.936	338.0	409.781	0.457
Multiarm-Blocksworld	443	376	376	376	376	**443**	41	20 – 543	182.1	124	0.2 – 307.697	4.7	0.3655	-0.092
Robot	117	90	90	90	90	**117**	433	2 – 1725	272.4	37	0.3 – 464.611	25.9	0.2045	0.899
Rover-GTOHP	509	268	268	269	269	**509**	127	16 – 2640	320.7	212	0.4 – 597.899	91.8	1.4755	0.515
Satellite-GTOHP	296	145	145	145	145	**296**	202	12 – 1584	379.1	270	0.18 – 313.948	56.3	9.461	0.743
Snake	183	170	170	170	171	**183**	20	2 – 162	20.6	16	0.49 – 374.823	49.8	124.93	0.672
Towers	17	**12**	**12**	**12**	**12**	**12**	8191	1 – 131071	15419.1	511	0.41 – 297.734	31.6	0.237	0.668
Transport	695	**691**	687	687	**691**	677	1504	8 – 5077	188.9	76	0.2 – 193.067	3.0	0.252	0.806
Woodworking	494	**494**	**494**	**494**	**494**	**494**	none	3 – 219	57.5	25	0.3 – 7.848	1.6	0.4805	0.994
	10961	9158	9117	9158	9177	**10881**	16	1 – 131071	239.2	119	1 – 599.936	26.6	0.975	0.095

Table 2: Statistics on the to-val dataset.

the runs with method preconditions, where we compared the planning and parsing approach the results remained the same (Figure 1 and lines 1-4 in Table 1). The planning approach solves more instances on the three domain groups.

Conclusion

In this paper, we set two goals. First, we wanted to use heuristics when creating a new task. We created three new heuristics and compared them with the original version. It has proven that the new heuristics did not increase the efficiency of the program in a significant way.

Our second goal was to utilize goal knowledge to obtain a parsing-based verifier that solves exactly the same verification problem as other verifiers. We have run multiple experiments with and without goal knowledge. The results have shown that the parsing approach is faster than the SAT verifier but slower than the planning verifier.

Acknowledgments

Research was supported by the joint Czech-German project registered under the number 21-13882J by the Czech Science Foundation (GAČR) and 452150823 by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation). S. Ondrčková is (partially) supported by SVV project number 260 575.

References

Barták, R., and Maillard, A. 2017. Attribute grammars with set attributes and global constraints as a unifying framework for planning domain models. In *Proc. of the 19th Int. Symposium on Principles and Practice of Declarative Programming (PPDP 2017)*, 39–48. ACM.

Barták, R.; Ondrčková, S.; Maillard, A.; Behnke, G.; and Bercher, P. 2020. A novel parsing-based approach for verification of hierarchical plans. In *Proc. of the 32nd Int. Conf. on Tools with AI (ICTAI 2020)*, 118–125. IEEE.

Barták, R.; Ondrčková, S.; Behnke, G.; and Bercher, P. 2021a. Correcting hierarchical plans by action deletion. In

Domain	#Plans	Verifier						Shortest unverified plan H0	Plan Length			Runtime for Verified			Pearson Corr-
		H0	H1	H2	H3	SAT	Planning		Min–Max	Avg	Median	Min–Max	Avg	Median elation	
AssemblyHierarchical	209	128	128	128	128	30	**148**	1	1 – 256	28.9	12	0.19 – 372.465	30.9	6.0255	0.866
Barman-BDI	423	333	195	276	333	80	**423**	163	10 – 1198	128.4	69	0.2 – 592.877	31.8	3.289	0.401
Blocksworld-GTOHP	158	75	75	75	75	2	**158**	164	21 – 6661	482.3	209.5	0.19 – 489.531	125.8	54.161	0.133
Blocksworld-HPDDL	172	118	118	118	118	0	**152**	290	20 – 5732	461.1	163	1.6 – 566.083	49.6	11.3115	0.508
Childsnack	529	**529**	528	**529**	**529**	92	516	none	50 – 2500	119.8	80	0.3 – 451.175	2.5	0.374	0.562
Depots	459	396	396	396	396	61	**459**	277	15 – 971	129.2	92	1.4 – 147.048	8.0	1.598	0.091
Elevator-Learned-ECAI-16	2877	2788	2788	2788	2788	65	**2877**	425	1 – 2165	220.1	195	0.2 – 523.926	34.1	9.0195	0.134
Entertainment	159	**159**	**159**	**159**	**159**	111	**159**	none	24 – 128	71.7	64	0.19 – 2.427	1.0	0.579	0.965
Factories-simple	123	**73**	67	67	**73**	9	58	425	15 – 2968	623.7	251	1.85 – 257.316	29.2	2.368	0.831
Freecell-Learned-ECAI-16	204	100	100	100	100	0	**204**	133	57 – 489	162.7	138.5	500.4 – 555.312	172.1	117.233	0.665
Gripper-new	10	**10**	**10**	**10**	**10**	4	**10**	none	11 – 107	65.6	71	0.24 – 0.332	0.3	0.242	–
Hiking	580	**557**	**557**	**557**	**557**	0	0	99	26 – 174	70.4	68	1.1 – 562.896	43.5	6.046	–
Logistics-Learned-ECAI-16	1128	487	487	487	487	20	**1128**	278	3 – 2813	405.9	363.5	1 – 575.366	105.8	19.477	-0.057
Minecraft-Player	75	0	0	0	0	0	0	35	35 – 278	51.9	44	–	–	0	–
Minecraft-Regular	766	0	0	0	0	0	0	35	35 – 9947	253.8	135	–	–	0	–
Monroe-Fully-Observable	248	165	161	161	162	0	**248**	16	3 – 96	41.5	39	0.76 – 599.535	368.5	296.851	0.485
Monroe-Partially-Observable	217	50	50	48	53	0	**217**	18	6 – 91	45.1	45	0.68 – 596.446	350.7	263.263	0.358
Multiarm-Blocksworld	443	261	261	261	261	9	**443**	41	20 – 543	182.1	124	0.2 – 551.058	50.7	5.438	0.523
Robot	117	82	82	82	82	20	**117**	198	2 – 1725	272.4	37	0.2 – 307.675	12.4	0.199	0.542
Rover-GTOHP	510	218	219	215	217	23	**510**	127	16 – 2640	320.2	212	1.8 – 599.276	98.2	1.39	0.333
Rover-PANDA	7	**7**	**7**	**7**	**7**	**7**	**7**	none	8 – 51	35.1	39	0.87 – 0.961	0.4	0.221	–
Rovers-Learned-ECAI-16	144	133	133	133	133	0	**144**	291	38 – 346	188.8	194	0.363 – 562.533	52.5	29.625	–
Satellite-GTOHP	296	111	112	114	108	9	**296**	70	12 – 1584	379.1	270	0.19 – 598.642	121.1	25.371	0.442
Snake	183	171	171	171	171	152	**183**	20	2 – 162	20.6	16	1.52 – 552.793	48.8	144.87	0.736
Towers	17	6	6	6	6	3	**12**	127	1 – 131071	15419.1	511	0.2 – 33.751	6.3	0.2815	0.523
Transport	698	**694**	690	690	**694**	66	677	1504	8 – 5077	188.2	76	0.2 – 188.539	3.0	0.248	0.590
UM-Translog	15	**15**	**15**	**15**	**15**	**15**	**15**	none	9 – 31	18.3	14	0.2 – 0.212	0.2	0.199	–
Woodworking	494	293	293	293	293	255	**494**	39	3 – 219	57.5	25	0.4 – 302.107	7.6	0.471	0.851
Zenotravel	3	**3**	**3**	**3**	**3**	**3**	**3**	none	9 – 18	12.0	9	0.216 – 0.258	0.2	0.223	–
	11264	7962	7811	7889	7958	1036	**9658**	1	1 – 131071	235.5	118	0.2 – 599.535	48.2	3.546	0.108

Table 3: Statistics on the to-val-no-mprec dataset.

Proc. of the 18th Int. Conf. on Principles of Knowledge Representation and Reasoning, 99–109.

Barták, R.; Ondrčková, S.; Behnke, G.; and Bercher, P. 2021b. On the verification of totally-ordered HTN plans. In *2021 IEEE 33rd Int. Conf. on Tools with Artificial Intelligence (ICTAI)*, 263–267. IEEE.

Barták, R.; Maillard, A.; and Cardoso, R. C. 2018. Validation of hierarchical plans via parsing of attribute grammars. In *Proc. of the 28th Int. Conf. on Automated Planning and Scheduling (ICAPS 2018)*, 11–19. AAAI Press.

Behnke, G.; Höller, D.; Schmid, A.; Bercher, P.; and Biundo, S. 2020. On succinct groundings of HTN planning problems. In *Proc. of the 34th AAAI Conf. on Artificial Intelligence (AAAI 2020)*, 9775–9784. AAAI Press.

Behnke, G.; Höller, D.; and Biundo, S. 2015. On the complexity of HTN plan verification and its implications for plan recognition. In *Proc. of the 25th Int. Conf. on Automated Planning and Scheduling (ICAPS 2015)*, 25–33. AAAI Press.

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 the 27th Int. Conf. on Automated Planning and Scheduling (ICAPS 2017)*, 20–28. AAAI Press.

Bercher, P.; Höller, D.; Behnke, G.; and Biundo, S. 2016. More than a name? on implications of preconditions and effects of compound HTN planning tasks. In *Proc. of the 22nd European Conf. on AI (ECAI 2016)*, 225–233. IOS Press.

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

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proc. of the 2nd Int. Joint Conf. on AI (IJCAI 1971)*, 608–620.

Forgy, C. L. 1982. Rete: A fast algorithm for the many pattern/many object pattern match problem. *AI* 19(1):17–37.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language classification of hierarchical planning problems. In *Proc. of the 21st European Conf. on AI (ECAI 2014)*, 447–452. IOS Press.

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 the 26th Int. Conf. on Automated Planning and Scheduling (ICAPS 2016)*, 158–165. AAAI Press.

Höller, D.; Wichlacz, J.; Bercher, P.; and Behnke, G. 2022. Compiling HTN plan verification problems into HTN planning problems. In *Proc. of the 32nd Int. Conf. on Automated Planning and Scheduling (ICAPS 2022)*. AAAI Press.

Howey, R., and Long, D. 2003. VAL's progress: The automatic validation tool for PDDL2.1 used in the int. planning competition. In *Proc. of the ICAPS'03 Workshop on the Competition: Impact, Organization, Evaluation, Benchmarks*.

Vilain, M. 1990. Getting serious about parsing plans: A grammatical analysis of plan recognition. In *Proc. of the 8th National Conf. on AI (AAAI 1990)*, 190–197. AAAI Press.

Wichlacz, J.; Torralba, A.; and Hoffmann, J. 2019. Construction-planning models in minecraft. In *Proc. of the 2nd ICAPS Workshop on Hierarchical Planning*, 1–5.