On the Impact of Grounding on HTN Plan Verification via Parsing

Simona Ondrčková[1][a], Roman Barták[1][b], Pascal Bercher[2][c] and Gregor Behnke[3][d]
[1]*Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic*
[2]*School of Computing, The Australian National University, Canberra, Australia*
[3]*Faculty of Engineering, ILLC, University of Amsterdam, Amsterdam, Netherlands*

Keywords: Hierarchical Planning, HTN (Hierarchical Task Network), Verification, Parsing, Grounding.

Abstract: The problem of hierarchical plan verification focuses on checking whether an action sequence is a valid hierarchical plan – the action sequence is executable and a goal task can be decomposed into it. The existing parsing-based verifier works on lifted domain models. In this paper we study whether grounding of the models could improve efficiency of the verifier. We also explore additional implementation improvements to increase the speed of the verifier.

1 INTRODUCTION

Planning focuses on generating and ordering actions into an action sequence (plan) in order to achieve a specific goal from a given initial state. Hierarchical planning is a form of planning that is similar to how people plan, as it splits difficult tasks into easier ones, until primitive tasks (directly executable actions) are obtained. Hierarchical planning is important in the areas of robotics (Kaelbling and Lozano-Pérez, 2011), machine learning (Mohr et al., 2018) or for providing personalised assistance, e.g., for Do-It-Yourself home improvement projects (Bercher et al., 2021).

Plan verification is the opposite process to planning. It checks whether a given action sequence is a valid plan – checks whether the actions are executable, and that goal conditions are present in the final state (Howey and Long, 2003). So far the steps are the same for classical and hierarchical planning. For hierarchical plans a check that a given goal task (network) decomposes into the action sequence is needed. Hierarchical plan verification is an NP-complete problem (Behnke et al., 2015; Bercher et al., 2022). Plan verification can be used for example in mixed initiative planning (Behnke et al., 2016).

There are currently three main approaches focusing on hierarchical plan verification: a compilation to boolean satisfiability problem (SAT) (Behnke et al., 2017), a compilation to Hierarchical Task Network (HTN) planning (Höller et al., 2022) and parsing-based approaches (Barták et al., 2018; Barták et al., 2020; Barták et al., 2021a). There is also a CYK-based (Cocke–Younger–Kasami algorithm) approach (Lin et al., 2023a), but it is limited to totally ordered hierarchical problems. There also exist approaches that correct a verified plan in case verification fails (Barták et al., 2021b; Lin et al., 2023b), but in this work we focus on verification, i.e., to verify plans as quickly as possible.

Hierarchical models share many similarities to formal grammars (Höller et al., 2014; Höller et al., 2016; Barták and Maillard, 2017), as compound tasks correspond to non-terminal symbols, primitive tasks (actions) correspond to terminal symbols, and decomposition methods correspond to production rules. The sole difference between formal grammars and HTN planning problems is that task networks specified in decomposition methods may be partially ordered (whereas right-hand sides of production rules are totally ordered), and that actions have preconditions and effects, whereas terminal symbols do not. The plan verification problem therefore corresponds to the language membership problem (checking whether a word is in the language produced by the formal grammar). This is solved via parsing in grammars – and so we deploy parsing as well in HTN plan verification.

The parsing-based approach is significantly faster than the SAT-based approach (Barták et al., 2020). However, overall the parsing-based approach is

[a] https://orcid.org/0000-0001-9081-4988
[b] https://orcid.org/0000-0002-6717-8175
[c] https://orcid.org/0000-0002-0795-4320
[d] https://orcid.org/0000-0002-1445-9934

Ondrčková, S., Barták, R., Bercher, P. and Behnke, G.
On the Impact of Grounding on HTN Plan Verification via Parsing.
DOI: 10.5220/0011618400003393
In *Proceedings of the 15th International Conference on Agents and Artificial Intelligence (ICAART 2023) - Volume 3*, pages 92-99
ISBN: 978-989-758-623-1; ISSN: 2184-433X

Table 1: Coverage of tested verifiers showing the number of instances solved for each verifier in each domain group.

	#inst	TI grounded with plan	TI grounded	TI lifted	Original	Planning
to-valid	10,961	10,106 (92.20)	9,763 (89.07)	9,156 (83.53)	9,173 (83.69)	**10,925** (99.67)
to-invalid	1,406	**1,390** (98.86)	1,378 (98.01)	1,301 (92.53)	1,302 (92.60)	1,364 (97.01)
po-valid	1,209	1,073 (88.75)	1,073 (88.75)	1,038 (85.86)	991 (81.97)	**1,110** (91.81)
po-invalid	138	**137** (99.28)	**137** (99.28)	136 (98.55)	136 (98.55)	129 (93.48)

Table 2: Per instance comparison between grounded (TI grounded with plan) and lifted (TI lifted) verifier.

	#inst	only grounded	only lifted	grounded faster	lifted faster
to-valid	10,961	1,407	458	4,063	**4,546**
to-invalid	1,406	104	15	**680**	593
po-valid	1,209	67	32	**567**	382
po-invalid	138	1	0	**68**	63

slower than the recently introduced planning-based approach (Ondrčková et al., 2022). The planning-based approach uses grounding (more details on this in Section 3), which is able to eliminate some of the tasks that the parsing-based verifier creates. We will analyze and test whether it could also help improve the efficiency of the parsing-based approach.

The goal of this paper is improving the parsing-based approach. We picked it because it can a handle a variety of inputs, conditions, and type of constraints. This will be described in more detail in the next section. Ideally we would like to bring it on par with the planning-based approach. In order to achieve this, we will be focusing on two main improvements:

1. Use of a grounding system.
2. Implementation improvements.

The paper is organized as follows. First, we will provide a definition of the HTN model and formulate the verification problem more precisely. We will also explain the parsing-based approach. Second, we will describe the grounding system we want to use. Third, we will explain the implementation improvements we do. Finally, we will provide an empirical evaluation of our improvements and compare them with the other approaches and we will analyse the results.

2 HTN PLAN VERIFICATION

Hierarchical task network (HTN) planning is planning that focuses on decomposition of tasks (Erol et al., 1996; Bercher et al., 2019). We rely on a STRIPS model (Fikes and Nilsson, 1971) of actions. The set of propositions describing a world state is called P. Each world state S is a set of propositions that are true in that state. Each action a has three sets of propositions $(\texttt{pre}(a), \texttt{eff}^+(a), \texttt{eff}^-(a))$, where $\texttt{pre}(a), \texttt{eff}^+(a), \texttt{eff}^-(a) \subseteq P$. The first set describes preconditions of the action. These must be true in the state right before the action. The other two sets describe the positive and negative effects of the actions. Negative effect represents what is no longer true after the action. If all preconditions of an action a are satisfied in state S (which immediately precedes the action), then action a is applicable to state S. The state immediately following action a will look like this: $\gamma(S,a) = (S \setminus \texttt{eff}^-(a)) \cup \texttt{eff}^+(a)$.

An action sequence is executable if all actions are applicable to the states right before them: $\texttt{pre}(a_i) \subseteq \gamma(\gamma(\ldots\gamma(S_0,a_1),\ldots),a_{i-1})$.

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

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

The order of sub-tasks is described explicitly in C, so the order of the tasks $T_1...T_k$ does not matter. There are three types of constraints:

- $T_1 \prec T_2$: an ordering constraint meaning that in every plan, task T_1 is before task T_2.
- $before(p,T)$: a precondition constraint meaning that in every plan, the proposition p holds in the state right before 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 task T_1 task T_2.

The hierarchical planning problem can be defined like this: Given a domain model (i.e., a description of tasks and their possible decompositions via decomposition methods) and problem instance (i.e., an initial state S_0 and goal task G), is there an executable action sequence (also known as plan) to which the goal task would be decomposable? This plan is the output. The hierarchical plan verification problem is defined like this: Assuming an action sequence, an initial state S_0, and goal task G, are the actions executable and does the goal task decompose into the action sequence?

Table 3: Per domain statistics for the to-valid dataset, showing number of solved instances per verifier and length of the shortest unverified plan for TI Grounded with plan verifier.

Domain	#Plans	Verifier					Shortest unverified plan for
		TI grounded with plan	TI grounded	TI lifted	Original	Planning	TI grounded with plan
AssemblyHierarchical	193	**193**	**193**	131	131	**193**	none
Barman-BDI	423	390	389	396	395	**421**	268
Blocksworld-GTOHP	158	126	126	118	118	**158**	196
Blocksworld-HPDDL	172	165	165	164	166	**170**	2880
Childsnack	529	**529**	509	**529**	**529**	528	none
Depots	455	**455**	**455**	426	426	**455**	none
Elevator-Learned-ECAI-16	2812	2750	2748	2782	2792	**2812**	341
Entertainment	159	**159**	**159**	**159**	**159**	**159**	none
Factories-simple	123	107	106	96	106	**122**	1636
Freecell-Learned-ECAI-16	204	**204**	**204**	**204**	**204**	**204**	none
Hiking	565	**565**	534	564	**565**	**565**	none
Logistics-Learned-ECAI-16	1108	723	719	1099	1099	**1108**	344
Minecraft-Player	75	74	24	0	0	**75**	278
Minecraft-Regular	766	568	546	0	0	**755**	107
Monroe-Fully-Observable	248	**248**	**248**	178	176	**248**	none
Monroe-Partially-Observable	217	**217**	**217**	67	64	**217**	none
Multiarm-Blocksworld	443	381	381	376	376	**443**	54
Robot	117	90	90	90	90	**117**	433
Rover-GTOHP	509	**509**	426	264	264	**509**	none
Satellite-GTOHP	296	**296**	237	145	145	**296**	none
Snake	183	**183**	**183**	171	171	**183**	none
Towers	17	11	11	**13**	12	12	4095
Transport	695	668	668	690	**691**	681	547
Woodworking	494	**494**	425	**494**	**494**	**494**	none

2.1 HTN Verification via Parsing

The parsing-based approach (Barták et al., 2018; Barták et al., 2020; Barták et al., 2021a) is a bottom-up approach. First, it checks for action executability. To do this, it calculates states between actions. Then it checks whether each action is applicable to the state before it. If a goal state is provided, the verifier checks whether all conditions are present in the final state.

Then the verifier starts building its first tasks. These are tasks that decompose into actions. We shall call them layer 1 tasks. Each time the verifier creates a task, it will also check whether the task is new. Then the verifier builds another layer of tasks on top of them. These layer 2 tasks decompose either into just layer 1 tasks or into a combination of layer 1 tasks and actions. The layers are then continuously built until a task that recursively decomposes into all actions is found. Let us name it task T. If a user has not provided a goal task then task T is the final task and the plan is valid. If a user provided a goal task, then task T is compared to the goal task and if they are the same, the plan remains valid. Otherwise, the approach will continue looking for other tasks and checking them against the goal task. If no found task is compatible then the plan is invalid.

A more detailed description of the approach and an exact algorithm can be found in the work by Barták et al. (Barták et al., 2021a). The reason why we choose the parsing approach is that it can cover many intricacies of the HTN networks. For example, it can handle instances with and without a given goal task as input (Ondrčková et al., 2022). The planning-based approach requires a goal task. Also to our knowledge the parsing-based approach is the only approach that can handle between conditions.

3 GROUNDER

The parsing-based approach to plan verification directly operates on the lifted HTN planning problem, i.e., on a problem where actions and methods still contain variables. At least for planning it is however often beneficial to not use a (complex) lifted representation, but a grounded one instead. For this, one instantiates all actions, methods, and predicates with all plausible parameter combinations. "Plausibility" is usually determined by an overapproximation to the actions that can appear in any plan, with the additional condition that this approximation can be efficiently computed. The most common technique is delete-relaxation – where we ignore the deleting effects $\texttt{eff}^-(a)$ of actions and determine all state features and actions reachable this way.

This simple mechanic was mainly developed for

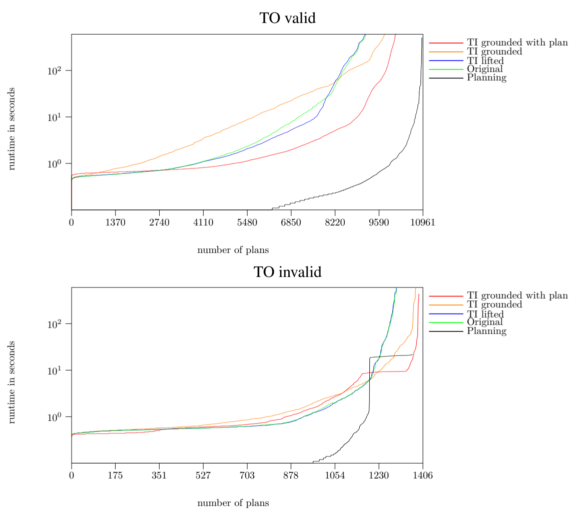

Figure 1: Runtime of verifiers on totally ordered domains.

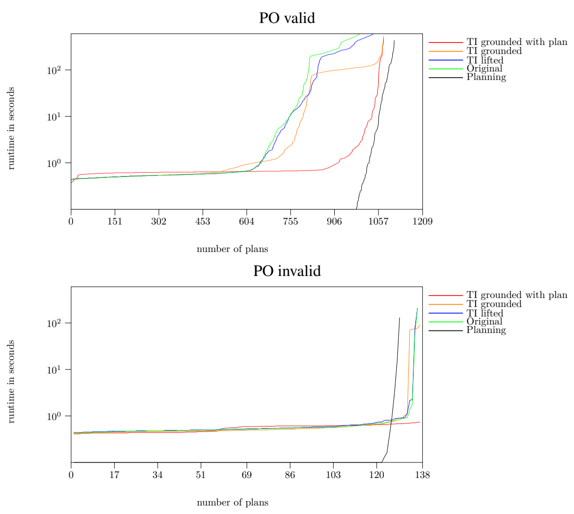

Figure 2: Runtime of verifiers on partially ordered domains.

classical, i.e., non-hierarchical planning. For HTN planning, Behnke et al. (Behnke et al., 2020) presented a technique for grounding. It comprises a four step process. Firstly, they overapproximate the methods, compound tasks, and actions that are reachable from the goal task G. This is done by a depth first search where they consider only the possible ranges of parameter values and not the values themselves. Secondly, they perform a standard delete-relaxed reachability analysis over the actions that were reachable in the first step. The set of reachable ground actions then forms the basis for the further grounding steps. Thirdly, they perform a bottom up composition analysis similar to parsing. For any method, if all its subtasks are ground reachable, then a ground instantiation of the method and its compound task is created. This process is repeated if new combinations of parameters for grounding are possible – similar as in parsing. Lastly, they perform a top-down search pass from G to remove all composed methods that are not reachable from the goal task. This may eliminate actions, methods, and compound tasks from the grounding. Steps two to four are repeated until convergence.

We expect that grounding the planning problem prior to plan verification is algorithmically beneficial. Since the grounding procedure has to perform fewer checks (it, e.g., ignores ordering constraints completely) it is much more efficient to execute. But grounding may produce more tasks and action instances than the parsing-based verifier – which takes more domain constraints into account and can thus perform a stricter pruning. On the other hand, the grounder's top down reachability analysis could prune a significant number of tasks. Therefore once computed grounding should enable the verifier to consider even fewer task instances than without it. With grounding, the parsing-based verifier is more efficient as it does not have to consider parameters. Given the once performed grounding, the parsing-based verifier can benefit from the information contained in the grounding, i.e., in the non-reachable tasks.

For plan verification, we can however do more than just grounding the planning problem. Before grounding, we already know what we want to use the grounding for: for verifying a plan. As such, any action that is in the grounding but not in the plan can safely be ignored – if the goal task would lead us to such actions, the resulting plan cannot possibly be the one we want to verify. To integrate this knowledge into grounding, we ignore all actions that are not part of the plan in step two of the grounding procedure.

4 IMPLEMENTATION IMPROVEMENTS

To discover how we can improve the parsing-based approach, we first further analyzed the results of the latest comparison with the other approaches provided by Ondrčková et al. (Ondrčková et al., 2022). We noticed a group of domains on which the parsing-based approach doesn't perform very well. For example on domains Minecraft-Regular and Minecraft-Player (Ondrčková et al., 2022) the parsing-based approach cannot solve a single instance within the time limit. The shortest plan in these domains has 35 actions. For the domain Rover-GTOHP the parsing-based approach was able to solve roughly only half of the instances in time, compared to the planning-based one. We did some additional testing to discover the reason behind this performance. One problem was checking whether a task is new after task creation. The check is necessary as it's possible that through multiple decompositions, we might find a task that has the same parameters and decomposes into the same actions.

The original parsing-based approach used an inefficient structure to store already created tasks. This meant that it took $O(n)$ steps (where n is the number of tasks) for each check. We have changed the structure to a hash-set structure which allows taking only $O(1)$ steps for each check. We then looked at other structures and analyzed whether it would also be beneficial to change them into a hash-set. We changed the structure containing conditions for each slot and a structure containing partially instantiated methods.

5 EMPIRICAL EVALUATION

In this section we will present a comparison between our improved verifiers, the original and the planning-based verifier. We will not be comparing our verifiers against the SAT verifier. This is because previous versions of the parsing-based approach were already shown to outperform it on the benchmark set we use (Barták et al., 2020).

We ran our experiments on Intel Xeon Gold 6130 with 8GB of RAM. We ran our verifiers on 13714 instances which were already used in previous evaluations of HTN plan verifiers (Barták et al., 2021a; Höller et al., 2022; Ondrčková et al., 2022). We split these instances into four groups based on their validity and ordering: *to-valid* (totally ordered domains with valid plans), *po-valid* (partially ordered domains with valid plans), *to-invalid* (totally ordered domains with invalid plans), and *po-invalid* (partially ordered domains with invalid plans). The instances for all four groups of plans stem mostly from the Planning Competition (IPC) 2020 and were complemented with plans generated by non-competing planners on the same domains[1]. They generated these instances by the planners in the IPC on the IPC domains[2] and on domains that were submitted but not selected for the IPC[3].

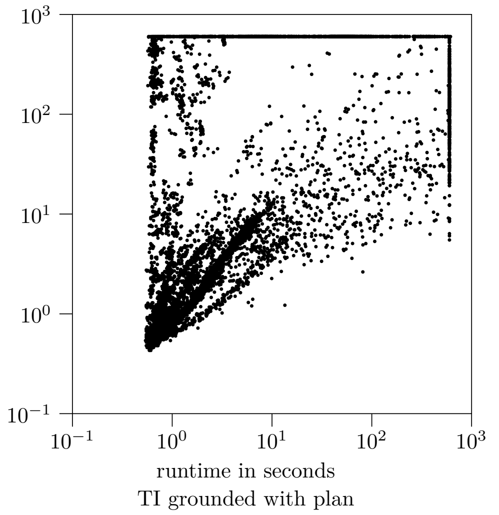

Figure 3: Runtimes of the parsing-based verifier on the grounded (TI grounded with plan, x-axis) vs the lifted (TI lifted, y-axis) instances.

We ran the experiments on five verifiers: *Original*, *TI lifted*, *TI grounded*, *TI grounded with plan* and *Planning*. *Original* is the original parsing-based verifier as presented in paper by Ondrčková et al. (Ondrčková et al., 2022). The *TI lifted* approach is the parsing-based approach extended by our implementation improvements. Note that the *TI grounded* approach is the *TI lifted* approach, but it is used on grounded domains. The *TI grounded* approach uses a grounder that does not take the plan to verify into account, while *TI grounded with plan* does. Finally *Planning* is the planning-based approach described in the paper by Höller et al. (Höller et al., 2022). The *Planning* approach uses grounding with knowledge of the plan. In order to ground the domain we use the grounder described in the Grounder section.

The *Planning* approach, the *TI grounded* approach and *TI grounded with plan* approach use grounded domains. The time needed to ground the domains is included in the total time of the verifiers. We ran the experiments on instances with a given goal task as the *Planning* approach needs a goal task as input. We used a time limit of 10 minutes. Table 1 shows the number of instances solved within this time limit for each verifier. Boldness in tables shows which approach solves most instances.

Let us first look at the performance of the *TI lifted* verifier. As we can see in Table 1 *TI lifted* solves around the same number of instances (less than 0.5% difference) as the *Original* for all groups except the *po-valid* group. In the *po-valid* group the approach solves additional 47 instances (around 3.9% of the instances). We can also see in Figure 2 that the *TI lifted* approach (blue line) is faster on the *po-valid* group than the original approach (green line). The speed is comparable in every other group.

Next, let us look at the performance of the *TI grounded* approach. First, we will compare it to the *Original* approach. The *TI grounded* consistently solves more instances than the original approach in

[1]https://github.com/panda-planner-dev/ipc-2020-plans
[2]https://github.com/panda-planner-dev/ipc2020-domains
[3]https://github.com/panda-planner-dev/domains

every single group. We believe, this is because the grounding prunes the number of tasks the parsing-based verifier has to create (more on this below). It solves additional 590 instances of *to-valid* instances, which is over 5.38%. When it comes to *to-invalid* it solves additional 76 instances, which is 5.4%. For *po-valid* instances it solves additional 82 instances (6.78%). Finally, for *po-invalid* it solves one extra instance (0.72%). This is mainly because the *Original* approach already performs very well on the *po-invalid* domains and can solve over 98% of instances.

In total the *TI grounded* approach solves 12,351 (90.06%) instances, while the Original solves 11,602 (84.6%) out of 13,714 instances. This is a difference of 749 instances (5.46% of all instances).

The grounding process introduces an overhead (compared to the original approach) that can be seen in Figures 1 and 2. We believe, the overhead is caused by the time needed to run the grounder and the size of the domain. The original domain has certain number of lifted methods and tasks. But for every single grounding of each method/task the grounder creates a new method/task. This can cause the number of methods and tasks to increase significantly. We believe that the main cause of the overhead is the size of the domain. This overhead is a especially prevalent on totally ordered valid domains. However, once the plans get big or difficult enough, the grounding benefits outweigh the overhead, and the *TI grounded* approach outperforms the *Original* approach.

Let us look at the *TI grounded with plan* approach. The *TI grounded with plan* approach outperforms (when looking at number of instances solved) the *TI grounded* approach on both totally ordered groups, while it performs the same in partially ordered groups.

The biggest difference is in the *to-valid* domains where it solves additional 343 instances (3.12% of the instances). In total the *TI grounded with plan* solves 12,706 instances (92.6%), compared to the *TI grounded*'s 12,351 and the *Original*'s 11,602.

As can be seen in Figures 1 and 2 *TI grounded with plan* approach also solves most instances faster than the *TI grounded* approach. We believe this is because the grounding with the plan knowledge makes the grounded domain smaller by removing methods that would decompose into actions that can be safely ignored (as explained in the Grounder section).

If we look at the speed of the *TI grounded with plan* and the *Original* approach, we can see that the *TI grounded with plan* is faster on *to-valid* domains. In every other group the *TI grounded with plan* approach performs similarly on easily solvable plans (for *to-invalid* domains slightly worse) but becomes significantly faster on difficult plans.

Next let us compare the *TI grounded with plan* and *TI lifted* approach. The *TI grounded with plan* approach outperforms the *TI lifted* approach (in number of instances solved) in every group. However we discovered that there exist domains that the *TI lifted* approach is able to solve in time but the *TI grounded with plan* approach is not. In Table 2 we can see the number of instances that only one of the two approaches solves and the number of instances in which one of them is faster. This suggests a possible area for future work, where we could try to analyze the domain before the verification begins to determine whether grounding would be beneficial. Then based on the results we would perform the grounding or not.

In Figure 3 we show the scatterplot of the runtime for *TI grounded with plan* and *TI lifted* approach. As shown in Table 2 the lifted approach solves more instances faster for the totally ordered valid domains, while the grounded approach outperforms it on partially ordered domains and invalid domains.

Let us now look at the performance on specific domains. Table 3 details all the domains for the *to-valid* group and the verifiers' performance on them. The table also shows the length of the shortest unverified plan with the *TI grounded with plan* approach. Table 4 provides information about length of the plan verified with the *TI grounded with plan* approach and run-time statistics per domain including Pearson correlation of the run-time with the plan length.

Most of the instances that the lifted approach solves and the grounded approach does not in *to-valid* group comes from Logistics-Learned-ECAI-16 domain. We believe this is because the grounding increases the size of the domain significantly.

Let us now look at the Minecraft-Player and Minecraft-Regular domains, where the *Original* approach wasn't able to solve a single instance in time. As can be seen in Table 3 the *TI grounded* approach is able to solve 24 instances and 546 instances which is 32% and 71% of the domains' instances and *TI grounded with plan* can solve 74 instances (98.6%) and 568 (74.1%) instances. In the Minecraft-Player domain we were able to go from no instances solved to essentially all instances solved (all but one). We believe that the reason for such increase in performance in these two domains is that the grounder's top down analysis helps prune significant amount of tasks.

To confirm our hypotheses we ran some additional tests on the Minecraft domain on a laptop with 32GB of memory. We took the smallest instance of the Minecraft-Regular domain and we performed a relaxed top down pruning pass and then a bottom up analysis of the grounder. Without it the instance is not groundable as the laptop runs out of memory.

Table 4: Per domain statistics for to-valid, with plan length information and runtimes for the TI grounded with plan approach.

Domain	Plan Length			Runtime for Verified (TI grounded with plan)			Pearson
	Min–Max	Avg	Median	Min–Max	Avg	Median	Correlation
AssemblyHierarchical	4 – 256	31.1	14	0.54 – 1.95	0.7	0.63	0.953
Barman-BDI	10 – 1198	128.4	69	0.58 – 396.2	12.0	0.78	0.131
Blocksworld-GTOHP	21 – 6661	482.3	209.5	0.55 – 284.46	13.5	1.335	0.358
Blocksworld-HPDDL	20 – 5732	461.1	163	0.57 – 543.35	17.7	1.22	0.853
Childsnack	50 – 2500	119.8	80	0.59 – 67.69	1.2	0.73	0.948
Depots	15 – 971	129.1	92	0.56 – 27.16	1.5	0.72	0.989
Elevator-Learned-ECAI-16	10 – 2165	225.1	200	0.05 – 585.06	6.4	1.83	0.804
Entertainment	24 – 128	71.7	64	0.62 – 1.1	0.8	0.77	0.688
Factories-simple	15 – 2968	623.7	251	0.59 – 375.41	42.8	1.63	0.017
Freecell-Learned-ECAI-16	57 – 489	162.7	138.5	0.86 – 7.8	1.6	1.305	0.901
Hiking	26 – 174	70.8	72	0.58 – 1.93	0.8	0.76	0.947
Logistics-Learned-ECAI-16	27 – 2813	413.1	370	0.6 – 586.09	47.6	2.87	0.405
Minecraft-Player	35 – 278	51.9	44	1.18 – 173.79	5.0	2.525	0.955
Minecraft-Regular	35 – 9947	253.8	135	0.87 – 585.05	74.2	43.315	0.222
Monroe-Fully-Observable	3 – 96	41.5	39	0.56 – 0.91	0.7	0.675	0.348
Monroe-Partially-Observable	6 – 91	45.1	45	0.6 – 0.83	0.7	0.68	0.458
Multiarm-Blocksworld	20 – 543	182.1	124	0.6 – 276.79	5.3	1.24	-0.187
Robot	2 – 1725	272.4	37	0.58 – 314.89	16.5	0.645	0.704
Rover-GTOHP	16 – 2640	320.7	212	0.57 – 79.46	3.5	1.42	0.921
Satellite-GTOHP	12 – 1584	379.1	270	0.59 – 448.76	29.8	2.165	0.708
Snake	2 – 162	20.6	16	0.55 – 1.83	0.7	0.64	0.918
Towers	1 – 131071	15419.1	511	0.59 – 268.75	28.9	0.72	0.900
Transport	8 – 5077	188.9	76	0.58 – 423.18	4.5	0.7	0.665
Woodworking	3 – 219	57.5	25	0.54 – 2.98	1.0	0.68	0.991

With the relaxed top-down reachability enabled, this resulted in 5,093 grounded tasks (includes both primitive and abstract tasks and instances of method preconditions) and 3,184 grounded methods. Then we ran the top down pruning. After this the domain has 300 grounded methods and 454 grounded tasks. So we were able to reduce the number of grounded tasks and methods by a factor of 10.

We did another test on a middle sized instance. Here the number of grounded methods after the first pass is 67,014 and the number of grounded tasks is 116,983. After the top down pruning we get 2,395 methods and 3,609 tasks. These results are for grounding without a plan knowledge. Finally, we ran one more test on the smallest instance using the grounder with plan knowledge. The number of grounded methods after the first pass is 987 and the number of tasks is 2,958. After the top down pruning we get 252 methods and 382 tasks. This supports our other hypotheses that the grounding with a plan outperforms the grounding without a plan knowledge, because the verifier must check fewer tasks/methods.

In Rover-GTOHP (to-valid) domain, the *Original* approach only solves 264 instances (51.8%), the new *TI grounded with plan* can solve all of them.

Figure 2 shows an interesting result. Four of the five approaches struggle with the 15 most difficult plans in po-invalid group. They increase in run-time significantly or in some cases are even unable to solve them (*Planning* approach – last 9). However the *TI grounded with plan* approach can solve all of them but one and its increase in run-time is negligible.

Finally, let us look at the *TI grounded with plan* and *Planning* approach. Table 1 shows that the *TI grounded with plan* solves more instances than the *Planning* approach on invalid but not on valid domains. This makes sense because the parsing-based approach works similarly to breadth-first search. It will create all possible tasks of a layer before it moves to the next one. So by the time it finds the goal task in layer *n*, it will create all possible tasks of lower layers. The *Planning* approach does not work this way so it might be able to find a solution quicker. However when it comes to invalid domains the *Planning* approach also has to try all the options. This suggest an idea for future work. We could adjust the parsing-based approach to use heuristics in order to create tasks that seem promising and then further build upon them instead of building all tasks of a particular layer.

6 CONCLUSIONS

We set a goal to improve the parsing-based approach with two sub-goals: create an improved version of the verifier and use grounding to increase performance. We succeeded and created an improved verifier TI-lifted and used it on lifted and grounded domains.

We empirically compared our new verifier, the original parsing-based verifier and the state-of-the-art planning-based verifier. The results have shown that the improved verifier on grounded domains outperforms the original parsing-based approach on all four tested groups. It also outperforms the planning-based approach on invalid, but it is worse on valid domains.

The experiments showed us where we could further improve the parsing-based approach. We now have two possible areas for future work. First, a possible check for whether grounding could be beneficial before running it. This would allow us to marry the benefits of lifted and grounded domains. Second, the use of heuristics to first create tasks that look the most promising instead of creating them in a manner similar to breadth-first search. This could allow us to avoid creating a significant number of tasks.

ACKNOWLEDGEMENTS

Research is supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215. S. Ondrčková is (partially) supported by SVV project number 260 575 and by the Charles University project GA UK number 280122.

REFERENCES

Barták, R. and Maillard, A. (2017). Attribute grammars with set attributes and global constraints as a unifying framework for planning domain models. In *PPDP 2017*, pages 39–48. ACM.

Barták, R., Maillard, A., and Cardoso, R. C. (2018). Validation of hierarchical plans via parsing of attribute grammars. In *ICAPS 2018*, pages 11–19. AAAI Press.

Barták, R., Ondrčková, S., Behnke, G., and Bercher, P. (2021a). On the verification of totally-ordered HTN plans. In *ICTAI 2021*, pages 263–267. IEEE.

Barták, R., Ondrčková, S., Behnke, G., and Bercher, P. (2021b). Correcting hierarchical plans by action deletion. In *KR 2021*, pages 99–109. IJCAI.

Barták, R., Ondrčková, S., Maillard, A., Behnke, G., and Bercher, P. (2020). A novel parsing-based approach for verification of hierarchical plans. In *ICTAI 2020*, pages 118–125. IEEE.

Behnke, G., Höller, D., Bercher, P., and Biundo, S. (2016). Change the plan – How hard can that be? In *ICAPS 2016*, pages 38–46. AAAI Press.

Behnke, G., Höller, D., and Biundo, S. (2015). On the complexity of HTN plan verification and its implications for plan recognition. In *ICAPS 2015*, pages 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 *ICAPS 2017*, pages 20–28. AAAI Press.

Behnke, G., Höller, D., Schmid, A., Bercher, P., and Biundo, S. (2020). On succinct groundings of HTN planning problems. In *AAAI 2020*, pages 9775–9784. AAAI Press.

Bercher, P., Alford, R., and Höller, D. (2019). A survey on hierarchical planning – one abstract idea, many concrete realizations. In *IJCAI 2019*, pages 6267–6275. IJCAI.

Bercher, P., Behnke, G., Kraus, M., Schiller, M., Manstetten, D., Dambier, M., Dorna, M., Minker, W., Glimm, B., and Biundo, S. (2021). Do it yourself, but not alone: *Companion*-technology for home improvement – bringing a planning-based interactive DIY assistant to life. *Künstliche Intelligenz – Special Issue on NLP and Semantics*, 35:367–375.

Bercher, P., Lin, S., and Alford, R. (2022). Tight bounds for hybrid planning. In *IJCAI-ECAI 2022*, pages 4597–4605. IJCAI.

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 *IJCAI 1971*, pages 608–620.

Höller, D., Behnke, G., Bercher, P., and Biundo, S. (2014). Language classification of hierarchical planning problems. In *ECAI 2014*, pages 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 *ICAPS 2016*, pages 158–165. AAAI Press.

Höller, D., Wichlacz, J., Bercher, P., and Behnke, G. (2022). Compiling HTN plan verification problems into HTN planning problems. In *ICAPS 2022*, pages 145–150. AAAI Press.

Howey, R. and Long, D. (2003). VAL's Progress: The Automatic Validation Tool for PDDL2.1 used in the International Planning Competition. In *ICAPS' Workshop on the Competition: Impact, Organization, Evaluation, Benchmarks 2003*.

Kaelbling, L. P. and Lozano-Pérez, T. (2011). Hierarchical task and motion planning in the now. In *IROS 2011*, pages 1470–1477. IEEE.

Lin, S., Behnke, G., Ondrčková, S., Barták, R., and Bercher, P. (2023a). On total-order HTN plan verification with method preconditions – an extension of the CYK parsing algorithm. In *AAAI 2023*. AAAI Press.

Lin, S., Grastien, A., and Bercher, P. (2023b). Towards automated modeling assistance: An efficient approach for repairing flawed planning domains. In *AAAI 2023*. AAAI Press.

Mohr, F., Wever, M., and Hüllermeier, E. (2018). ML-plan: Automated machine learning via hierarchical planning. *Machine Learning*, 107(8):1495–1515.

Ondrčková, S., Barták, R., Bercher, P., and Behnke, G. (2022). On heuristics for parsing-based verification of hierarchical plans with a goal task. In *FLAIRS 2022*.