Proceedings of the 4th ICAPS Workshop on Hierarchical Planning (HPlan 2021)

Domain Analysis: A Preprocessing Method that Reduces the Size of the Search Tree in Hybrid Planning

Michael Staud
StaudSoft UG (haftungsbeschränkt), Ulm, Germany
michael.staud@staudsoft.com

Abstract

We introduce a new method that can reduce the size of the search tree in hybrid planning. Hybrid planning fuses task insertion HTN planning with POCL planning. As planning is a computationally difficult problem, the size of the search tree can grow exponentially to the size of the problem.

We create so-called plan templates in a preprocessing step of the domain. They can be used to replace an abstract task in a partial plan. This task then no longer needs to be decomposed.

We provide empirical evidence in favor of this approach and show that the use of plan templates can drastically reduce the size of the search tree in hybrid planners. We use a PANDA-like planner as a testbed and publicly available planning domains to verify our claims.

1 Introduction

Planning is an important branch of artificial intelligence and is widely used in practice. We focus on planning algorithms that create action sequences to achieve a given goal in a deterministic and fully observable world. One approach is planning algorithms that create a search tree to find a solution to a problem (Ghallab, Nau, and Traverso 2004). This is used in both classical planning and *Hierarchical Task Network* (HTN) planning. In the latter, the nodes of the tree contain partial plans with plan steps. In classical (non-hierarchical) planning, the nodes store states. Unfortunately, HTN planning is not flexible enough to be used with our approach. We therefore use *Hybrid planning* (Schattenberg and Biundo 2006; Biundo and Schattenberg 2001), which extends HTN planning with partial order causal link (POCL) techniques. It allows the insertion of new plan steps during the planning process (task insertion).

A hybrid planning domain contains primitive and abstract tasks. Primitive tasks correspond to operators known from classical planning. Abstract tasks represent complex courses of actions. They must be decomposed into more concrete tasks using *decomposition methods*. In this process, an instance of an abstract task is replaced by a partial plan, which was stored in the decomposition method. This process is repeated until there are only plan steps with primitive tasks. Each decomposition creates a new choice point in the search tree. This process may introduce new abstract tasks into the partial plan that must be decomposed again. This in turn creates new choice points in the search tree.

Our method is a *domain analysis* technique that preprocesses a planning domain D (see Section 2). The goal is to speed up the planning process. We create new decomposition methods. They decompose an abstract task into a single (newly generated) primitive task. Our innovation is that such a decomposition method does not introduce new plan steps with abstract tasks. The planning algorithm itself is not changed. Before the execution of the planning algorithm, the new domain $D_S \supseteq D$ is created, which contains the new methods and primitive tasks. After the planning algorithm has found a solution P_{Ds}, the plan steps with the newly added primitive tasks are removed from it. They are replaced by partial plans $P_{R_i}(\overline{\tau})$ (called *replacement plans*) that were generated during the preprocessing step. We call this the replacement process (see Section 4). After that, the solution P_{Ds} contains only primitive tasks found in the original domain D. The replacement process uses neither search nor further planning. A *plan template* is defined as the combination of a method, a primitive task and a replacement plan.
Our contribution is an algorithm that generates replacement plans and placeholder tasks in a preprocessing step, which are then used during planning to reduce the size of the search tree. After a plan is found, the placeholder tasks are replaced by the replacement plans.

In the following section, we first introduce hybrid planning. In the next section, we show how to create a plan template with hybrid planning. Then, we describe the post-processing step that is performed after the planning process. At this point, the replacement plans are inserted into the solution. We then present our results, which show the effectiveness of our approach.

2 Hybrid Planning

Hybrid planning combines the concepts of Hierarchical Task Network (HTN) planning and Partial-Order Causal-Link (POCL) planning (Bercher, Alford, and Höller 2019; Biundo and Schattenberg 2001). In our paper, hybrid planning is with task insertion (TIHTN = HTN planning with task insertion). The following definitions are taken in part from Bercher, Keen, and Biundo (2014). Let V be the set of all variables and C be the set of all constants.

A *hybrid planning domain* is a tuple $D = (T_a, T_p, M)$. T_a is the set of abstract tasks, T_p is the set of primitive tasks,

and M is the set of all decomposition methods. All sets are finite.

Both primitive and abstract tasks are tuples $t(\overline{\tau}) = \langle prec_t(\overline{\tau}), eff_t(\overline{\tau})\rangle$ consisting of a precondition and an effect. Each task has parameters $\overline{\tau}$. The preconditions and effects are conjunctions of literals and unequal variable constraints over the task parameters $\overline{\tau} = (\overline{\tau}_1, \ldots, \overline{\tau}_n)$, $\overline{\tau}_i \in C \cup V, i \in [1 \ldots n]$. A literal is an atom or its negation. An atom is a predicate applied to a tuple of terms. A task is grounded if all its variables are bound to constants. A unequal constraint can be between two variables or between a variable and a constant. The preconditions and effects of an abstract task have the semantics of Definition 7 from Bercher et al. (2016). If it is clear from the context, we will omit the parameters of a task.

Partial plans are tuples $P = (PS, \prec, VC, CL)$ consisting of plan steps PS that are uniquely labeled tasks $l : t(\overline{\tau})$. The set $\prec$ contains *ordering constraints* of the form $(l, l') \in PS \times PS$ that induce a partial order on the planning steps in PS. The set VC contains the *unequal variable constraints* and the set CL contains the *causal links*. The CSP in VC must be solvable. A causal link $l : t(\overline{\tau}) \rightarrow_{\phi(\overline{\tau}')} l' : t'(\overline{\tau}')$ (shortened: $l \rightarrow_\phi l'$) links a precondition literal $\phi(\overline{\tau}')$ of the plan step $l' : t'(\overline{\tau}')$ to a unifiable effect of $l : t(\overline{\tau})$. If there is another plan step $t''(\overline{\tau}'')$ in the plan that has an effect $v(\overline{\tau}'')$ that is unifiable with $\neg\phi(\overline{\tau}')$, and if the ordering constraints allow l'' to be ordered between l and l', we call this a *causal threat*. A precondition without a causal link is called *open*.

A partial plan P may contain abstract tasks. These must be decomposed during the planning process using *decomposition methods*. A method $m = \langle t(\overline{\tau}_m), P_m\rangle$ is a tuple that maps an abstract task $t(\overline{\tau}_m)$ to a partial plan $P_m = (PS_m, \prec_m, VC_m, CL_m)$ that "implements" the task (Bercher et al. 2016, Def. 7). For the following definition, we need the function $unique(P_m, \overline{\tau}) = P^*_m$, that replaces each variable and label with a new name that does not appear in any other partial plan. If a variable is used in the parameters $\overline{\tau}$, it is not changed.

Definition 1 (Task Decomposition)**.** Let $l : t_a(\overline{\tau}_t)$ be a plan step with an abstract task t_a to be decomposed with $m = \langle t_a(\overline{\tau}_m), P_m\rangle$. And let $P'_m = (PS'_m, \prec'_m, VC'_m, CL'_m) = unique(P_m, \overline{\tau}_m)[\overline{\tau}_{m,1}/\overline{\tau}_{t,1}, \ldots, \overline{\tau}_{m,n}/\overline{\tau}_{t,n}]$ be the partial plan to be inserted. The set $\prec_X = \{(l_1, l_2) \in PS \times PS'_m | (l_1, l) \in \prec\} \cup \{(l_1, l_2) \in PS'_m \times PS | (l, l_2) \in \prec\}$ defines the new additional ordering constraints. The set CL_X contains the new causal links. The function $fE(P, \phi)$ returns a task from the partial plan P that has an effect that is unifiable with ϕ. And the function $fP(P, \phi)$ returns a set of tasks, each of which has a precondition that is unifiable with ϕ. The two functions always find a return value, since our domain must follow Definition 7 from Bercher et al. (2016).

$$CL_X = \bigcup_{l' \rightarrow_\phi l'' \in CL} \begin{cases} fE(P'_m, \phi) \rightarrow_\phi l'' & l' = l \\ \bigcup_{l^* \in fP(P'_m, \phi)} l' \rightarrow_\phi l^* & l'' = l \\ l' \rightarrow_\phi l'' & \text{else} \end{cases}$$

The new partial plan has the form $P' = (PS', \prec', VC', CL')$. Where $PS' = (PS \setminus \{l\}) \cup PS'_m$, $\prec' = \prec \cup \prec'_m \cup \prec_X$, $VC' = VC \cup VC'_m$ and $CL' = CL_X \cup CL'_m$ holds.

Task insertion is defined as the insertion of a new plan step with a primitive task $l : t(\overline{\tau}')$.

Definition 2 (Planning Process)**.** A partial plan can be refined by decomposing an abstract task, task insertion, adding a variable constraint, adding an ordering constraint or adding a causal link.

A hybrid planning problem is a combination of a domain D and an initial plan P_{init}. Let L be the set of all conjunctions of grounded literals and L_+ be the set of all conjunctions of positive grounded literals. As in standard POCL planning, we encode the initial state $I_s \in L_+$ and the goal description $G_s \in L$ as two special primitive tasks $t_0(\overline{\tau}) = \langle \varnothing, I_s\rangle$, $t_\infty(\overline{\tau}) = \langle G_s, \varnothing\rangle$ in the initial plan P_{init}.

Definition 3 (Solution)**.** A plan P_{sol} is a solution if there are no open preconditions, when all tasks are primitive and grounded and when there are no causal threats (Bercher, Keen, and Biundo 2014). This implies that all linearization of P_{sol} can be executed.

3 Plan Template Generation

In the preprocessing step, a *plan template* is created for an abstract task $t_a(\overline{\tau}_a) \in T_a$. This is done using a modified hybrid planner. The template is a tuple $P_T = \langle P_R(\overline{\tau}), m_T(\overline{\tau}), p_T(\overline{\tau})\rangle$ over the parameters $\overline{\tau}$ ($\overline{\tau} \supseteq \overline{\tau}_a$). We need to generate:

- a *replacement plan* $P_R(\overline{\tau})$, which is a partial plan
- a primitive task $p_T(\overline{\tau})$
- and a method $m_T(\overline{\tau}) = \langle t_a(\overline{\tau}_a), P'\rangle$ that allows the planner to select the primitive task. It holds $P' = (\{l' : p_T(\overline{\tau})\}, \emptyset, \emptyset, \emptyset)$.

The creation process is then the following:

Task Selection: Select $t_a(\overline{\tau}_a)$ (see Section 3.2).

Problem Creation: The replacement plan $P_R(\overline{\tau})$ is the solution to the following hybrid planning problem in the original domain D: The initial plan P_{init} contains the abstract task $t_a(\overline{\tau}_a)$ and the special tasks $t_0(\overline{\tau}_0)$ and $t_\infty(\overline{\tau}_\infty)$. The initial state I_s is equal to the preconditions of $t_a(\overline{\tau}_a)$ and the goal description G_s is equal to the effects of $t_a(\overline{\tau}_a)$ (if it has effects, otherwise the goal description is empty). The plan steps are then $l_0 : t_0$, $l_a : t_a$, $l_\infty : t_\infty$. Thus, $P_{init} = \{\{l_0, l_a, k_\infty\}, \{(l_0, l_a), (l_a, l_\infty)\}, \varnothing, \varnothing\}$. Note that these tasks need not be grounded. The solution is also not grounded. A replacement plan $P_R(\overline{\tau})$ can and should contain unbounded variables for maximum flexibility. According to our empirical tests, if the abstract task $t_a(\overline{\tau}_a)$ has effects, this improves the quality of the plan template.

Replacement Plan Creation: We use a modified hybrid planner. The *first modification* of the planning process (see Definition 2) is that we introduce a new (additional) refinement type when we encounter an open precondition. Let $r \in prec_t(\overline{\tau})$ be an open precondition of the plan step $l : t(\overline{\tau})$. Then we add a new effect r to the task $t_0(\overline{\tau}_0)$ and create a causal link $l_0 \rightarrow_r l$. Note that the parameters $\overline{\tau}$ from t that occur in r are added as new parameters to $\overline{\tau}_0$.

The *second change* is that the planning problem is now an optimization problem that attempts to minimize an objective function f_{t_a}, described in Section 3.1. A solution P_{sol} must satisfy the solution criteria from Definition 3 and it must be

minimal with respect to f_{t_a}. This is necessary to produce a useful plan template (see Section 5).

The solution $P_{sol} = (PS_{sol}, \prec_{sol}, VC_{sol}, CL_{sol})$ is then the replacement plan $P_R(\overline{\tau})$. The parameters $\overline{\tau}$ contain all variables from P_{sol}.

Primitive Task Creation: From a solution P_{sol}, a primitive task $p_T(\overline{\tau}) = \langle prec_{p_T}(\overline{\tau}), \mathit{eff}_{p_{p_T}}(\overline{\tau})\rangle$ is generated. The precondition $prec_{p_T}(\overline{\tau})$ of this task contains the effects of the primitive task $t_0(\overline{\tau})$ in P_{sol}. The unequal variable constraints VC_{sol} in P_{sol} are also added as a precondition. Thus, $prec_{p_T}(\overline{\tau}) = \mathit{eff}_{t_0}(\overline{\tau}) \cup VC_{sol}$ holds.

The effects $\mathit{eff}_{p_T}(\overline{\tau})$ of the task consist of all effects that would be in the goal state if we were to execute P_{sol}. Thus, we do not add effects that are undone during the execution of the partial plan (since it is a solution, this is true in any linearization). Thus, it holds:

$$\mathit{eff}_T(\overline{\tau}) = \bigcup_{\substack{r \in \mathit{eff}_{p_{T'}}(\overline{\tau'}) \\ l' : p_{T'} \in PS_{sol}}} \begin{cases} r & \text{if no other effect } \neg r' \\ & \text{of a plan step } l'' \text{ exists} \\ & \text{with } (l', l'') \in \prec_{sol} \\ \varnothing & \text{else} \end{cases}$$

Unless all effects and variable constraints are added, it would not be possible to replace the primitive task $p_T(\overline{\tau})$ with the replacement plan $P_R(\overline{\tau})$ without using search (see Section 4).

Method Creation: The decomposition method $m_T(\overline{\tau})$ is created according to its definition. The method $m_T(\overline{\tau})$ and the primitive task $p_T(\overline{\tau})$ are added to the original planning domain D.

3.1 Objective Function

The objective function evaluates a partial plan P. Let $P = (PS, \prec, VC, CL)$ be the current partial plan at template generation. And let $t_0(\overline{\tau}) = \langle prec_{t_0}(\overline{\tau}), \mathit{eff}_{t_0}(\overline{\tau})\rangle \in PS$ be the initial task. The abstract task for which the template is generated is called $t_a(\overline{\tau})$. We define the following auxiliary functions:

- $ch(t_a(\overline{\tau}), a)$: Returns 1 if the literal's predicate occurs as an effect in any decomposition of $t_a(\overline{\tau})$. Otherwise, it returns 0.
- $st_{ct}(A)$: Given a set of atoms A, this function checks whether the state constraints according to Gerevini and Schubert (1998) are violated. It returns ∞ if a violation is found. Otherwise, it returns 0.
- $P(a \in S(t_a))$: Probability that the predicate of a is available as a precondition for t_a in a randomly selected problem in the domain D. This is approximated by solving example domains. We count in all these solutions which predicates occur in causal links. In doing so, we consider only the links that go from a plan step that was not generated by a decomposition of $t_a(\overline{\tau})$ to a plan step that was generated by a decomposition of $t_a(\overline{\tau})$. From these counts, the probability of occurrence of each predicate is calculated.

We tested two different objective functions $f_{t_a}(P_{opt}, t_a)$:

$f_1(P, t_a) = |prec_{t_0}(\overline{\tau})| + |PS|$

$$f_2^x(P, t_a) = |PS| + \sum_{a \in prec_{t_0}(\overline{\tau})} + ch(t_a(\overline{\tau}), a)x + st_{ct}(prec_{t_0}(\overline{\tau}))$$

Regarding f_2^x we used the two variants f_2^1 and $f_2^{P(a \in S(t_a))}$. Often there are multiple solutions where an objective function is minimal or near-minimal. If there are example problems, we evaluate the plan templates with them by solving them. We then select the plan template that maximizes the reduction of the search space. The others are *discarded* (see Table 1, `rover` domain).

3.2 Selection of Abstract Tasks

To determine which abstract task is suitable for generating plan templates, we use the task decomposition graph (Bercher, Keen, and Biundo 2014). For each task, we compute the set of mandatory M_{t_a} and optional tasks O_{t_a}. The mandatory tasks occur in every decomposition of a given abstract task t_a. The optional tasks occur in at least one decomposition. Thus, the amount of mandatory tasks $|M_{t_a}|$ relative to the optional tasks gives us an indication of how much the decompositions of an abstract task t_a will differ from each other. In our experiments, the higher the ratio $|M_{t_a}|/|O_{t_a}|$, the more successful the generation of a plan template. Thus, we select the abstract task with the highest ratio.

4 Replacement Process

The replacement process is a post-processing step performed after the solution $P_{Ds} = (PS_{Ds}, \prec_{Ds}, VC_{Ds}, CL_{Ds})$ has been generated in D_s. The goal is then to generate a solution $P_{Ds'} = (PS_{Ds'}, \prec_{Ds'}, VC_{Ds'}, CL_{Ds'})$ that is valid in the original domain D. For each plan step $l : p_T(\overline{\tau}_t)$, which uses a primitive task from a plan template $P_T(\overline{\tau}) = \langle P_R(\overline{\tau}), m_T(\overline{\tau}), p_T(\overline{\tau})\rangle$, $p_T(\overline{\tau}) = \langle prec_{p_T}(\overline{\tau}), \mathit{eff}_{p_T}(\overline{\tau})\rangle$ we perform the following steps:

Decomposition: The plan step $l : p_T(\overline{\tau})$ is treated as an abstract task and decomposed using the method $m' = \langle p_T(\overline{\tau}), P_R(\overline{\tau})\rangle$. Let $P'_R = (PS'_R, \prec'_R, VC'_R, CL'_R) = \mathit{unique}(P_R, \overline{\tau})[\overline{\tau}_1/\overline{\tau}_{t,1}, \ldots, \overline{\tau}_n/\overline{\tau}_{t,n}]$ (see Definition 1).

Relinking: After the decomposition, the plan steps $l_0, l_\infty \in PS'_R$ are removed and their causal links are relinked. Let $l_0 \to_\phi l'$ be a casual link from l_0. And let $l'' \to_\phi l$ be a causal link to $l : p_T$. Then these two links are replaced by $l'' \to_\phi l'$. Similarly, a causal link of the form $l' \to_\phi l_\infty$ is treated. Let $l \to_\phi l''$ be a causal link with $l : p_T$. Then these two links are replaced again by $l' \to_\phi l''$. Note that because of the way p_T is defined, relinking is always possible. The new partial plan after the decomposition is then $P_{Ds'}$.

Removing Causal Threats: We remove causal threats introduced by the decomposition step. Let $\neg\phi$ be an effect of $l' \in PS_{Ds'}$ that threatens a causal link $l'' \to_\phi l^*$. Let $k_p(l)$ be a function that returns 1 if $l \in PS'_R$ and 0 otherwise.

It cannot be $k_p(l') = k_p(l'') = k_p(l^*)$ because both the replacement plan P_R and the original plan P_{Ds} were solutions (and thus without causal threats). It is also not possible that $k_p(l'') \neq k_p(l^*)$ holds, because then the causal link would have been created during the *relinking* step. And since we have only connected pre-existing causal links together, there can therefore be no causal threat. It follows that

satellite2	1o1s1m	2o1s2m	4o2s3m	3o2s2m
WO/W/I	66/17/3.88	11371/340/33.44	42140/29959/1.40	130791/1873/69.83
	1o2s1m	2o1s1m	2o2s2m	3o1s3
WO/W/I	101/27/3.74	449/109/4.11	4472/119/37.57	> 300000/22312/> 12.44
	3o1s1m	3o2s1m	3o2s3m	3o3s1
WO/W/I	5472/1209/4.52	15679/2773/5.64	> 300000/5483/54.71	29570/12945/2.28
	3o3s2m	3o3s3m	3o1s2m	
WO/W/I	> 300000/2380/> 126.05	> 300000/986/> 304.25	136927/3368/40.65	
woodworking	00	01	02	03
WO/W/I	4370/1738/2.51	67/67/1.00	79/79/1.00	374/224/1.67
	04			
WO/W/I	1930/1797/1.07			
transport	pfile01	pfile02	pfile03	
WO/W/I	497/57/9.20	237244/63242/3.75	252274/127007/1.98	
rover	pfile1	pfile2	pfile3	pfile4
WO/W/I	2194/1006/2.18	130/3347/none	19603/8454/2.31	636/405/1.57
WD/WD2	4330/4786	612/86	21745/22290	1234/552

Table 1: *Satellite Domain:* Problems of the satellite domain solved without and with plan templates. It holds XoYsZm = `Xobs-Ysat-Zmod.hddl` and WO/W/I = Solved without plan template / Solved with plan template / The improvement factor.
Woodworking Domain: Problems of the woodworking domain. In this domain, an abstract task is on average decomposed into two primitive tasks without plan template. Thus, there is not much room for improvement when the plan template is used.
Transport Domain: Problems of the transport domain.
Rover Domain: Problems of the rover domain solved without and with plan templates. The rover domain contains abstract tasks without effects. However, due to the structure of the rover domain, a plan template could still be generated. We also added measurements for 2 plan templates that were discarded because they were not as effective as the selected template. Note that in the `pfile02` example, the selected template performs very poorly and a discarded template reduces the number of search nodes. It holds WD/WD2 = With Discarded Template / With Discarded Template 2.

domain	satellite2	rover	transport	woodworking
States	182	31476	73	66
Generated	1	4	2	4
Used	1	1	2	1

Table 2: Number of search nodes needed to generate a plan template. Note that adding multiple plan templates to a domain can reduce performance because it increases the branching factor during the planning process. In the transport domain, we used 2 templates, which increases the number of states in the first example `pfile01`.

$k_p(l'') = k_p(l^*)$ and $k_p(l') \neq k_p(l'')$.

If $k_p(l') = 0$, then $k_p(l'') = k_p(l^*) = 1$. Then we can add (l', l'') or (l^*, l') to the ordering constraints $\prec_{Ds'}$. This is always possible because $\prec_{Ds}$ cannot contain both (l'', l') and (l', l^*) after the *decomposition* step as defined by Definition 1. All ordering constraints (l_1, l_2) where $k_p(l_1) \neq k_p(l_2)$ holds were added during this step.

If $k_p(l') = 1$, it follows that $k_p(l'') = k_p(l^*) = 0$. If adding (l', l'') or (l^*, l') to the ordering constraints $\prec_{Ds'}$ is not possible, we must relink the causal link $l'' \rightarrow_\phi l^*$. We know that it must hold $\neg\phi \notin \mathit{eff}_{p_T}(\overline{\tau})$ because:

- all effects of a plan step in P_R, except those removed by another task, are added to $\mathit{eff}_{p_T}(\overline{\tau})$.
- if $\neg\phi \in \mathit{eff}_{p_T}(\overline{\tau})$ then there would be no causal threat because P_{Ds} is a solution.

It follows $\neg\phi \notin \mathit{eff}_{p_T}(\overline{\tau})$ and according to Section 3 (Primitive Task Creation) there must be at least one more task $l^{**} \in PS'_R$ with effect ϕ and $(l', l^{**}) \in \prec'_R$. We then add $l^{**} \rightarrow_\phi l^*$ and delete the old causal link.

Suppose the new link is again threatened by another plan step l^{***}, $k_p(l^{***}) = 1$ with an effect $\neg\phi$. Then the link is rewired again. This is repeated until the link is no longer threatened. And there cannot be a plan step l^{***} with $k_p(l^{***}) = 0$ that threatens the link, because in that case it would also threaten the original link, which is a contradiction because P_{Ds} is a solution.

Theorem 1. The replacement process does not introduce new flaws into the plan. The resulting plan is a solution.

Proof Sketch: We will show that the plan satisfies the solution criteria after the replacement process:

- *No Open Preconditions*: In the replacement plan $P_R(\overline{\tau})$ all preconditions are linked. And all other links that are broken during the *relinking* step are reconnected.
- *No Causal Threats*: All causal threats were repaired in the last step of the replacement process.
- *Grounded*: Each plan step is grounded because all parameters of $P_T(\overline{\tau})$ are constants. The parameters also do not violate any variable constraints in $P'_R(\overline{\tau})$. This is the case because we have added all the variable constraints of $P_R(\overline{\tau})$ to the preconditions of the primitive task $p_T(\overline{\tau})$.

□

5 Results

We evaluated four sample domains (University Ulm 2019) with plan templates generated with the $f_2^{P(a \in S(t_a))}$ function (see Table 1). In each test domain, the algorithm selects n abstract tasks for which a plan template is generated according to Section 3.2. The number n of plan templates is determined by the user. We then compared the size of the search tree when using the original domain D and when using the domain D_{new} with the plan template. We counted the search nodes in all our tests. These are shown in the tables.

We used a self-developed partial order planner with support for abstract tasks and TIHTN+POCL planning using the hybrid heuristic $h_{\#F} + h_{TC}$ from Bercher, Keen, and Biundo (2014). The heuristic works with grounded abstract tasks, so we find a random grounding to apply this heuristic to a lifted task. The planner is sound and complete. As a flaw selection heuristic, we select the flaw that has the least number of distinct resolutions. This results in a low branching factor.

The `satellite2` domain showed the largest performance gain. Here, the planner mostly (in 86% of the cases) chose the template method during the planning process, which reduced the number of states by a factor of up to 300. In the `transport` domain, only one plan template was used. In the `rover` example, only one template was also used. Since the rover example contains 9 different abstract tasks, this reduced the usability of our plan template, which only replaced one abstract task.

In all examples, we also tried to use more templates, but this resulted in an increase in the number of search nodes. Without *task insertion* the templates were not used by the planner because in most of the cases the preconditions of the primitive task p_T could not be fulfilled.

We also tested the different evaluation functions from Section 3.1. The f_1 function did not provide any usable templates. Without evaluating the initial conditions added during the planning process, the planning algorithm does not know in which direction to optimize.

The functions f_2^1 and $f_2^{P(a \in S(t_a))}$ provided usable results. But only $f_2^{P(a \in S(t_a))}$ was successful in all sample domains (see Table 2). The f_2^1 function did not produce a usable plan template in the `transport` domain. When we used $f_2^{P(a \in S(t_a))}$, we did not use the same problems for the evaluation of the plan templates and performance measurements.

The performance gain is also supported by research about plan merging. Since we solve a part of the plan independently, the research results of Korf (1987) apply. He showed that if we can solve n subgoals separately, this divides the base and exponent of the complexity function by n. Thus, theoretically, our approach can lead to an exponential reduction in the size of the search tree.

6 Related Work

Macros are also a technique to encapsulate sequences of operators. Like plan templates, they have preconditions and effects like operators. This idea originated in the 1970s (Fikes and Nilsson 1971). They are generated from plans in a preprocessing step. These plans are solutions of example problems. In contrast to our approach, they do not use HTN domains and therefore cannot use the additional information they contain (abstract tasks, methods) (Chrpa, Vallati, and McCluskey 2015).

7 Conclusions

We presented a new domain analysis method that can reduce the size of the search tree. We showed that a drastic reduction in the size of the search tree is empirically possible. This is even more effective when example problems are available to guide the search when generating the plan templates. This is because it improves the quality of the plan templates.

This method can be used wherever many problems need to be solved in the same domain. It can quickly generate plan templates. Further research is needed to determine which domain features make plan template generation successful.

References

Bercher, P.; Alford, R.; and Höller, D. 2019. A Survey on Hierarchical Planning-One Abstract Idea, Many Concrete Realizations. In *IJCAI 2019*, 6267–6275. IJCAI Organization.

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 *ECAI 2016*, 225–233. IOS Press.

Bercher, P.; Keen, S.; and Biundo, S. 2014. Hybrid Planning Heuristics Based on Task Decomposition Graphs. In *SoCS 2014*, 35–43. AAAI Press.

Biundo, S.; and Schattenberg, B. 2001. From Abstract Crisis to Concrete Relief – A Preliminary Report on Combining State Abstraction and HTN Planning. In *ECP 2001*, 157–168. Springer.

Chrpa, L.; Vallati, M.; and McCluskey, T. L. 2015. On the Online Generation of Effective Macro-Operators. IJCAI 15, 1544–1550. AAAI Press.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. IJCAI 71, 608–620. Morgan Kaufmann.

Gerevini, A.; and Schubert, L. 1998. Inferring State Constraints for Domain-Independent Planning. AAAI 98/IAAI 98, 905–912. AAAI Press.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Elsevier.

Korf, R. E. 1987. Planning as Search: A Quantitative Approach. *AI 87* 33(1): 65–88.

Schattenberg, B.; and Biundo, S. 2006. A Unifying Framework for Hybrid Planning and Scheduling. In *KI 06*, 361–373. Springer.

University Ulm. 2019. PANDA Planning Domains and Problems. Https://www.uni-ulm.de/en/in/ki/research/-software/panda/, Accessed: 2020-10-01.