Hybrid Planning Heuristics Based on Task Decomposition Graphs

Pascal Bercher and **Shawn Keen** and **Susanne Biundo**
Institute of Artificial Intelligence,
Ulm University, Germany,
firstName.lastName@uni-ulm.de

Abstract

Hybrid Planning combines Hierarchical Task Network (HTN) planning with concepts known from Partial-Order Causal-Link (POCL) planning. We introduce novel heuristics for Hybrid Planning that estimate the number of necessary modifications to turn a partial plan into a solution. These estimates are based on the task decomposition graph that contains all decompositions of the abstract tasks in the planning domain. Our empirical evaluation shows that the proposed heuristics can significantly improve planning performance.

Introduction

Hierarchical Task Network (HTN) planning relies on the concept of task decomposition (Erol, Hendler, and Nau 1994). While the goal in classical (non-hierarchical) planning is to find an action sequence that satisfies a goal description, the goal in hierarchical planning is to find an executable course of action that is a refinement of an initial partial plan. Partial plans may contain primitive and abstract tasks. While primitive tasks correspond to operators known from classical planning, abstract tasks represent complex activities and must therefore be refined (decomposed) into more concrete courses of action using so-called decomposition methods. The difficulty in solving hierarchical planning problems is to choose the "correct" decomposition method for an abstract task in a given partial plan.

To improve the performance of hierarchical planning systems, one can follow *domain-specific* approaches that encode a domain-specific search advice within the domain. SHOP2 (Nau et al. 2003) is one of the best-known hierarchical planning systems following that technique. Another approach is to design *domain-independent* search strategies (Marthi, Russell, and Wolfe 2008; Shivashankar et al. 2013; Elkawkagy et al. 2012). The hierarchical planning system GoDel (Shivashankar et al. 2013), for instance, uses landmarks known from classical planning (Porteous, Sebastia, and Hoffmann 2001) for search guidance. The hierarchical planning system by Elkawkagy et al. (2012) uses *hierarchical landmarks* to guide its search. These are abstract and primitive tasks, which occur on any refinement process from the initial partial plan to any solution plan. Such hierarchical landmarks can be extracted from a task decomposition graph (TDG), which represents the decomposition hierarchy of the planning problem at hand. They use these landmarks in order to decide whether one decomposition method is preferred over another. For the selection of a most-promising search node, however, conventional heuristics are used that are unaware of the underlying hierarchy.

In this paper, we introduce novel heuristics that *do* take hierarchical information into account. We use the Hybrid Planning paradigm (Kambhampati, Mali, and Srivastava 1998; Biundo and Schattenberg 2001) that fuses hierarchical planning with concepts known from Partial-Order Causal-Link (POCL) planning. We propose a novel variant of the *Hierarchical Decomposition Partial-Order Planner* HD-POP (Russell and Norvig 1994, edition 1, p. 374–375) suited for Hybrid Planning. The resulting system, PANDA, guides its search using informed heuristics. We propose such heuristics that estimate the number of modifications necessary to find a solution. To that end, the TDG is used to extract hierarchical landmarks and further information about the hierarchy. Our empirical evaluation shows that the proposed heuristics can significantly improve planning performance.

Hybrid Planning

Planning problems are given in terms of a *Hybrid Planning* formalization (Kambhampati, Mali, and Srivastava 1998; Biundo and Schattenberg 2001), which fuses concepts from Hierarchical Task Network (HTN) planning (Erol, Hendler, and Nau 1994) with concepts from Partial-Order Causal-Link (POCL) planning (McAllester and Rosenblitt 1991; Penberthy and Weld 1992).

In Hybrid Planning, there are two kinds of tasks: *primitive* and *abstract* tasks. Both primitive and abstract tasks $t(\bar{\tau})$ are tuples $\langle prec(\bar{\tau}), \mathit{eff}(\bar{\tau})\rangle$ consisting of a *precondition* and *effect* over the task parameters $\bar{\tau}$. Preconditions and effects are conjunctions of literals. As usual, states are sets of ground atoms. Tasks are called ground if all variables are bound to constants. Applicability of (sequences of) ground primitive tasks is defined as usual.

Partial plans are tuples $(PS, \prec, VC, CL)$ consisting of the following elements. The set of plan steps PS is a set of uniquely labeled tasks $l : t(\bar{\tau})$. The set $\prec$ of ordering constraints induces a partial order on the plan steps in PS. The set VC is a set of variable constraints that (non-)codesignate the task parameters with each other or with constants. CL is

a set of causal links. A causal link $l : t(\bar{\tau}) \rightarrow_{\varphi(\tau_i)} l' : t'(\bar{\tau}')$ denotes that the precondition literal $\varphi(\tau_i)$ of the plan step $l' : t'(\bar{\tau}')$ is supported by the same effect of the plan step $l : t(\bar{\tau})$. If there is no causal link to a precondition $\varphi(\tau_i)$ we call it an *open* precondition.

Partial plans may also contain abstract tasks. These cannot be executed directly. Instead, they need to be decomposed into more specific partial plans using so-called (decomposition) methods. A method $m = \langle t(\bar{\tau}), P \rangle$ maps an abstract task $t(\bar{\tau}) = \langle prec(\bar{\tau}), \mathit{eff}(\bar{\tau}) \rangle$ to a partial plan P that "implements" that task (Biundo and Schattenberg 2001). Thereby, causal links involving $t(\bar{\tau})$ can be passed down to one of its sub tasks within P when decomposing $t(\bar{\tau})$.

Now, a *Hybrid Planning Domain* $\mathcal{D}$ is given by the tuple $\langle T_a, T_p, M \rangle$ consisting of a finite set of abstract and primitive tasks T_a and T_p, respectively, and a set of methods M. A *Hybrid Planning Problem* is given by a domain and an initial partial plan P_{init}. As it is the case in standard POCL planning, P_{init} contains two special primitive tasks that encode an initial state and a goal description. The task t_0 has no precondition and the initial state as effect. The task t_∞ has the goal description as precondition and no effects.

A plan P_{sol} is a solution to a hybrid planning problem if and only if the following criteria are met:

1. P_{sol} is a refinement of P_{init} w.r.t. the decomposition of abstract tasks and insertion of ordering constraints, variable constraints, and causal links.
2. P_{sol} needs to be executable in the initial state. Thus,
   (a) all tasks are primitive and ground,
   (b) there are no open preconditions, and
   (c) there are no causal threats. That is, given a causal link $l : t(\bar{\tau}) \rightarrow_{\varphi(\tau_i)} l' : t'(\bar{\tau}')$, we call the task $l'' : t''(\bar{\tau}'')$ a threat to that link if and only if the ordering constraints allow it to be ordered between the tasks of the causal link and it has an effect $\neg\psi(\tau_i'')$ that can be unified with the protected condition $\varphi(\tau_i)$.

Criterion 1 relates any solution plan to the initial partial plan P_{init}. This is necessary, since P_{init} represents the actual planning problem. Note that one could also allow the insertion of tasks into a partial plan without being introduced via decomposition of an abstract task as one of the allowed refinement options. In this paper, however, we do not allow such insertions and thereby follow the typical HTN planning approach (Erol, Hendler, and Nau 1994). Other hierarchical planning approaches, such as the Hybrid Planning approach by Kambhampati et al. (1998) or HTN Planning with Task Insertion (Geier and Bercher 2011) do allow such insertions.

Criterion 2 is inherited from standard POCL planning. It ensures that any linearization of the tasks in a solution plan is executable in the initial state. Every linearization generates a state which satisfies the goal condition.

Planning Algorithm

We search for solutions by systematically refining the initial partial plan P_{init} until it meets all the solution criteria. To that end, we propose the following generic hybrid planning algorithm (cf. Alg. 1). The corresponding planning system PANDA (Planning and Acting in a Network Decomposition Architecture) is based on earlier work (Schattenberg 2009).

Algorithm 1: Hierarchical Refinement Planning

```
1 F ← {P_init}
2 while F ≠ ∅ do
3 |   P ← planSel(F)
4 |   F ← F \ {P}
5 |   if Flaws(P) = ∅ then return P
6 |   f ← flawSel(Flaws(P))
7 |   F ← F ∪ { modify(m, P) | m ∈ Mods(f, P) }
8 return fail
```

The algorithm employs search in the space of partial plans. It uses a search fringe F consisting of all created refinements of P_{init} that have not yet been chosen for expansion. First, it picks and removes a partial plan from the fringe F (line 3, 4). This choice is done by means of the *plan selection function* `planSel`. This function can be implemented in various ways and determines the actual search strategy. For instance, choosing always an "oldest" plan results in a *breadth first search*. More elaborated strategies like A^* or *greedy search* need heuristic functions to judge the quality or goal distance of partial plans, such as the ones we propose in this paper.

After a partial plan P has been chosen for refinement, we calculate all its *flaws* $\texttt{Flaws}(P)$. Flaws are syntactical representations of violations of solution criteria. For instance, every abstract task in P induces a so-called *abstract task flaw*, since its existence violates the executability required by solution criterion 2.(a). Further flaw classes are *open precondition flaws* and *causal threat flaws* according to solution criteria 2.(b) and 2.(c), respectively.

If P has no flaws, it is a solution and hence returned (line 5). If there are flaws, they need to be removed in a systematic manner. We follow the approach of the hierarchical planner *HD-POP* (Russell and Norvig 1994, edition 1, p. 374–375) and pick *one* of its flaws (line 6) to be resolved. To that end, the function $\texttt{Mods}(f, P)$ calculates all *modifications* that modify the partial plan P, such that the flaw f is addressed in the resulting refinement. Modifications specify which plan elements are to be added to or removed from a given partial plan (Schattenberg, Bidot, and Biundo 2007). The application of a modification m to the partial plan P by the function $\texttt{modify}(m, P)$ generates the corresponding successor plan. Abstract task flaws can only be resolved by applying a decomposition method for the respective task. Open precondition flaws can only be resolved by inserting a causal link or by decomposing an abstract task that might possibly introduce a task with a suitable effect. Finally, a causal threat flaw can be resolved by promotion, demotion, and separation, as well as by decomposition if one of the involved tasks is abstract.

The selection of a specific flaw does not constitute a backtracking point. This is due to the fact that every flaw needs to be resolved at some point and we generate *all* its possible successor plans (line 7). Hence, any partial plan can be

discarded if there is a flaw without modifications. Although the choice of a flaw does not influence correctness or completeness, it heavily influences planning performance. This choice point is one of the major differences to the hierarchical planner used by Elkawkagy et al. (2012). That planner orders all modifications and therefore comes without an explicit choice point for flaws.

After all successor plans of the selected partial plan and flaw have been inserted into the fringe, the loop starts over until a solution is generated or the fringe becomes empty. In case of an empty fringe, there is (provably) no solution and *fail* is returned (line 8).

Exploiting Task Decomposition Graphs

We now give some basic definitions based on TDGs that can be used to design heuristic functions.

Let $\mathcal{G} = \langle V_T, V_M, E \rangle$ be a TDG in accordance to Def. 5 given by Elkawkagy et al. (2012). Hence, $\mathcal{G}$ is a directed AND/OR graph with the following elements: V_T is a set of task vertices consisting of ground abstract and primitive tasks that can be obtained by decomposing the initial partial plan. V_M is a set of method vertices consisting of ground methods that decompose an abstract task within V_T. E is a set of edges connecting vertices from V_T with vertices from V_M and vice versa. More precisely: If $t(\bar{c}) \in V_T$ and $m = \langle t(\bar{c}), P \rangle \in V_M$, then $(t(\bar{c}), m) \in E$. Then, the method node m has an edge for every one of the ground plan steps in P to its respective tasks in V_T: If $t'(\bar{c}') \in V_T$, $t'(\bar{c}')$ being a task of the plan steps in P, then $(m, t'(\bar{c}')) \in E$. Fig. 1 shows a small example TDG.

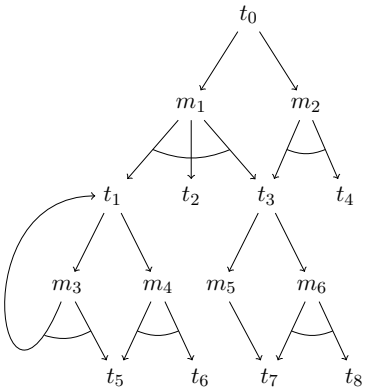


Figure 1: Example TDG depicted as AND/OR graph. The symbols t_0, t_1, t_3 and $t_2, t_4, \ldots, t_8$ represent ground abstract and ground primitive task vertices, respectively. The symbols m_1 through m_6 depict method vertices for the abstract tasks t_0, t_1, and t_3.

Although TDGs are finite, they can still be quite large because of the huge number of ground task and method instances that are possible in general. To be able to build TDGs despite their potentially large size, we follow a technique that allows to ignore certain irrelevant parts of the TDG (Elkawkagy, Schattenberg, and Biundo 2010).

Some of our TDG-based estimates are based on the concept of mandatory tasks. They have been introduced by Elkawkagy et al. (2012, Def. 6) and serve as approximation for landmarks (cf. Def. 4). Mandatory tasks are ground tasks that occur in all decomposition methods of the same abstract task. Elkawkagy et al. need these tasks as an intermediate step to calculate the set of optional tasks. In their algorithm, they use the number of these optional tasks to decide whether one decomposition method is preferred before another. We only require the mandatory tasks and use them to obtain an estimate for the modifications that need to be performed when decomposing abstract tasks. In contrast to Elkawkagy et al. (2012), we do not only count the tasks, but also use the number of their preconditions for our estimates.

To define mandatory tasks, we first need to define the *sub task set* $S(t(\bar{c}))$ of a ground abstract task $t(\bar{c})$, $\bar{c}$ denoting a sequence of constants the task's parameters are codesignated with. For each method m for a task $t(\bar{c})$, $S(t(\bar{c}))$ contains a set with all tasks in the partial plan referenced by m:

$$S(t(\bar{c})) = \{\{t'(\bar{c}') \mid (m, t'(\bar{c}')) \in E\} \mid (t(\bar{c}), m) \in E\}$$

For the TDG in Fig. 1, $S(t_0) = \{\{t_1, t_2, t_3\}, \{t_3, t_4\}\}$, $S(t_1) = \{\{t_1, t_5\}, \{t_5, t_6\}\}$, and $S(t_3) = \{\{t_7\}, \{t_7, t_8\}\}$.

The set of mandatory tasks $M(t(\bar{c}))$ of a ground abstract task $t(\bar{c})$ is then given by:

$$M(t(\bar{c})) = \bigcap_{s \in S(t(\bar{c}))} s$$

The set $M(t(\bar{c}))$ contains only tasks that inevitably will be inserted into a partial plan when decomposing $t(\bar{c})$. It therefore serves as a lower bound for estimating the modification effort for decomposing $t(\bar{c})$. In our example, we have $M(t_0) = \{t_3\}$, $M(t_1) = \{t_5\}$, and $M(t_3) = \{t_7\}$.

So far, we only consider the very next level of decomposition. Recursively incorporating the next levels for all tasks in $M(t(\bar{c}))$ results in higher estimates which are still lower bounds on the actual modification effort. The resulting set, the closure of $M(t(\bar{c}))$, denoted as $M^*(t(\bar{c}))$ is given by the following recursive equation having $M^*(t(\bar{c})) = M(t(\bar{c})) = \emptyset$ for a primitive task $t(\bar{c})$.

$$M^*(t(\bar{c})) = M(t(\bar{c})) \cup \bigcup_{t'(\bar{c}') \in M(t(\bar{c}))} M^*(t'(\bar{c}'))$$

Note that $M^*(t(\bar{c}))$ is finite even if the underlying TDG is cyclic. In our example, we have $M^*(t_0) = \{t_3, t_7\}$, $M^*(t_1) = M(t_1)$, and $M^*(t_3) = M(t_3)$.

The set $M^*(t(\bar{c}))$ can now be used to estimate the effort of decomposing $t(\bar{c})$. The easiest way is to take its cardinality, because for each task in that set, the planner needs to apply at least one modification (a task $t'(\bar{c}') \in M^*(t(\bar{c}))$ needs to be decomposed if $t'(\bar{c}')$ is abstract and – assuming primitive tasks have non-empty preconditions – at least one causal link must be inserted if $t'(\bar{c}')$ is primitive).

Definition 1 (Task Cardinality) *Let $t(\bar{c})$ be a ground abstract task. Then, the task cardinality of $t(\bar{c})$ is given by:*

$$TC(t(\bar{c})) := |M^*(t(\bar{c}))|$$

As argued before, $TC(t(\bar{c}))$ can be regarded as a reasonable estimate for the decomposition effort of $t(\bar{c})$. However, we can improve that estimate by taking their preconditions into account. We know that every precondition of every task needs to be supported with a causal link. Hence, we can assume that the number of required modifications is at least as large as the number of preconditions of the mandatory tasks.

Definition 2 (Precondition Cardinality) *Let $t(\bar{c})$ be a ground abstract task. Then, the precondition cardinality of $t(\bar{c})$ is given by:*

$$PC(t(\bar{c})) := \sum_{\substack{t'(\bar{c}') \in M^*(t(\bar{c})),\ t'(\bar{c}') \text{ primitive,} \\ \text{and } t'(\bar{c}') = \langle prec(\bar{c}'), eff(\bar{c}') \rangle}} |prec(\bar{c}')|$$

Note that we only incorporate the preconditions of the primitive tasks to avoid counting preconditions twice: since causal links to or from an abstract task are handed down to their subtasks when decomposing that task, it suffices to incorporate the preconditions of their primitive sub tasks.

So far, we have introduced two estimates for the modification effort of abstract tasks by focusing on the mandatory ones. Only focusing on these tasks might be too defensive, however. Consider a planning problem, where each abstract task has at least two (ground) decomposition methods, but only a few mandatory tasks. Our estimate does not consider the remaining tasks and could therefore still be improved. In our example, $M^*(t_0)$ ignores the tasks t_1, t_2, and t_4, although at least one of these is introduced when decomposing t_0 (they could be considered disjunctive landmarks). We hence investigate a third estimate that judges the minimal modification effort based on the entire TDG while combining the ideas of TC and PC.

Definition 3 (Minimal Modification Effort) *Let V be an arbitrary set of ground tasks. For a primitive task $t(\bar{c}) = \langle prec(\bar{c}), eff(\bar{c}) \rangle$, we set $h(t(\bar{c}), V) := |prec(\bar{c})|$.*

For an abstract task $t(\bar{c}) = \langle prec(\bar{c}), eff(\bar{c}) \rangle$ we set:

$$h(t(\bar{c}), V) := \begin{cases} 1 + |prec(\bar{c})| & \text{if } t(\bar{c}) \in V, \text{ or else:} \\ 1 + \min\limits_{s \in S(t(\bar{c}))} \sum\limits_{t'(\bar{c}') \in s} h(t'(\bar{c}'), \{t(\bar{c})\} \cup V) \end{cases}$$

Then, $MME(t(\bar{c})) := h(t(\bar{c}), \emptyset)$.

The basic idea of MME is to minimize the estimated modification effort per decomposition method represented by the different sets in $S(t(\bar{c}))$. The effort for a set of tasks within the same method is obtained by summing over the efforts for the single tasks. If such a task is primitive, we take the number of its preconditions as estimate, analogously to the estimate PC. If such a task is abstract, we account for its decomposition effort by 1 plus the effort of its "cheapest" decomposition method. Since the traversed TDG might be cyclic, we need to ensure termination. For that purpose, we use a set V of already visited tasks. If we discover an abstract task that was already decomposed and hence within V, we estimate its modification effort by 1 (for its decomposition) plus the number of its preconditions.

MME shows some similarities to the *add heuristic* (Bonet and Geffner 2001) that estimates the number of actions required to achieve an atom. The add heuristic, however, is a non-admissible heuristic w.r.t. the number of actions (due to the assumption of sub goal independence), while MME is admissible w.r.t. the number of modifications (while dominating both TC and PC).

Please note that not only the TDG can be calculated in a preprocessing step before the actual search, but also the functions we have presented so far (Def. 1 to 3). Thus, during search, no complicated calculations are necessary.

We still have to cope with the problem that the TDG consists entirely of ground task instances while the actual search process is done in a lifted fashion, where tasks are only partially ground. Given a partial plan $P = (PS, \prec, VC, CL)$ and one of its plan steps $ps = l : t(\bar{\tau})$, we define $Inst(ps)$ as a function that maps to a fixed, but arbitrary ground instance $t(\bar{c})$ of ps that is compatible with the variable constraints VC. During search, we then use the task $Inst(ps)$ when we want to retrieve an estimate for $t(\bar{\tau})$ from the TDG. An alternative would be to minimize or aggregate all (or some) of the possible ground instances of $t(\bar{\tau})$. We did not yet evaluate these possibilities, however.

Heuristic Functions

We want to estimate the number of necessary modifications to turn a partial plan P into a solution. In standard POCL planning, the heuristics for that purpose (Nguyen and Kambhampati 2001; Younes and Simmons 2003; Bercher, Geier, and Biundo 2013; Bercher et al. 2013) are based on the concept of task insertion, rather than task decomposition and hence not directly applicable in our setting. There are also several heuristics for the hybrid planning approach (Schattenberg, Bidot, and Biundo 2007; Schattenberg 2009). However, these heuristics incorporate the hierarchical aspects of abstract tasks not to their full extent, in particular, because they are not based upon a TDG.

A relatively simple hybrid and/or POCL heuristic is $h_{\#F}$ that returns the number of flaws. That heuristic might underestimate the number of required modifications, however. For example, estimating the number of modifications for an abstract task flaw by one ignores that decomposing the respective task introduces several new tasks thus raising new flaws. We hence estimate the number of required modifications by $h_{\#F}$ plus a value accounting for the hierarchy by inspecting the TDG w.r.t. the abstract tasks in a partial plan.

We now give heuristics for partial plans based on Def. 1 to 3, i.e., TC, PC, and MME. For all definitions, assume P to be a partial plan $(PS, \prec, VC, CL)$.

Definition 4 (Heuristic h_{TC+PC})

$$h_{TC+PC}(P) := \sum_{ps \in PS} TC(Inst(ps)) + PC(Inst(ps))$$

While both $TC(t(\bar{c}))$ and $PC(t(\bar{c}))$ guarantee not to overestimate the number of modifications for $t(\bar{c})$, h_{TC+PC} does not show this property. First, the arbitrary compatible instantiation of the plan step ps can be suboptimal. Second, TC, which is simply the cardinality of *all* the mandatory

tasks of $Inst(ps)$, contributes to the modification effort for each primitive task by one, assuming that for every primitive task, at least one causal link must be inserted. This effort, however, is already reflected in $PC(Inst(ps))$, as it counts the preconditions of the primitive tasks. We did not "fix" this possible overestimation, because we believe that the heuristics already underestimate the actual modification effort in practice (most importantly because both TC and PC restrict their estimates to the mandatory tasks).

The next heuristic, based on MME, incorporates the whole TDG for its estimates.

Definition 5 (Heuristic h_{MME})

$$h_{MME}(P) := \sum_{ps \in PS} MME(Inst(ps))$$

Evaluation

We evaluate the proposed TDG-based heuristics on four hybrid planning domains and compare them with standard search strategies that are not based on the TDG.

Planning Benchmarks

We use the same set of planning domains used by Elkawkagy et al. (2012) for their evaluation. We shortly review those domains and include information about the TDGs of the evaluated problem instances. We only build the *pruned* TDGs that incorporate reachability information of the respective problem instances. One of the TDG-related information is its maximal branching factor. Please note that this number is different from the branching factor of the explored search space, since not only decomposition methods need to be chosen, but also open precondition and causal threat flaws need to be resolved. In particular for inserting a causal link there might be several refinement options during search.

The first planning domain is based on the well-known **UM-Translog** (Andrews et al. 1995) domain for hierarchical planning. UM-Translog is a logistics domain, where goods of certain kinds need to be transported. It shows 48 primitive and 21 abstract tasks for which there are 51 methods. The domain does not contain recursive decompositions. In the conducted experiments, the respective TDGs contain 7 to 22 ground primitive tasks and 12 to 30 ground abstract tasks. The longest path within the TDG has length 11, while the average branching factor of the abstract tasks is at most 1.11 indicating that the reachability analysis ruled out most of the available decomposition options. The easiest problem instance can be solved by applying 25 modifications, while the hardest instance requires at least 76 modifications.

The **Satellite** domain is a hierarchical adaptation of a domain taken from the International Planning Competition (IPC) that features conducting orbital observations using a certain number of satellites and modes. It consists of 5 primitive and 3 abstract tasks with 8 methods in total. That domain does also not contain recursive decompositions. Despite the small number of tasks and the missing recursion, the domain is rather difficult due to the large number of reachable ground task instances. Those vary from 7 to 87 for primitive tasks and from 4 to 16 for abstract tasks. While the maximal path length of these TDGs is always 4, the average branching factor for decomposition ranges from 2.75 to 15.1. Depending on the problem instance, solutions require between 13 and 41 modifications.

The **SmartPhone** domain models a modern cell phone. The tasks are concerned with sending messages and creating contacts or appointments. It is defined over 87 primitive tasks, 50 abstract tasks, and 94 methods. The domain allows for recursion. The TDGs contain between 10 and 19 primitive ground tasks and 7 and 22 ground abstract tasks. The maximal acyclic path length ranges from 4 to 6 with an average branching factor ranging from 1.42 to 1.95. Solutions have a minimal depth of 18 to 54.

The last domain, **Woodworking**, is also based on a benchmark from the IPC and deals with cutting, planing, and finishing wood parts. The domain consists of 6 abstract tasks, 13 primitive tasks, and 14 methods. The domain does not have cyclic method definitions. The TDGs of the problem instances contain between 10 and 64 ground primitive tasks and 15 to 492 ground abstract tasks. The depth varies from 2 to 4, and the average branching factor ranges from 2.53 to 7.21. Solutions require between 22 and 53 modifications.

Search Strategies

For the evaluation, we use greedy search with varying heuristics. If some partial plans show the same heuristic value, ties are broken by chance.

For the flaw selection function, we always use Least-Cost Flaw-Repair (LCFR) (Joslin and Pollack 1994). LCFR minimizes the branching factor per search node by selecting a flaw that has minimal "repair cost", i.e., the least number of modifications. Ties between flaws are broken by chance.

Besides our new TDG-based heuristics, we have also included the *Number of Flaws* ($h_{\#F}$) of a partial plan and the *Number of Modifications* ($h_{\#M}$) for all flaws[1].

For every heuristic, we also evaluated a normalized version thereof. Let $P = (PS, \prec, VC, CL)$ be a partial plan; then $\|h(P)\|$ is defined by $\frac{h(P)}{|PS|}$. Taking the ratio of a heuristic (such as the number of flaws) to the number of plan steps prevents heuristic values for two consecutive partial plans from jumping too much. Consider two partial plans, P_1 and P_2, P_2 being the successor of P_1 due to the decomposition of an abstract task. In general, P_2 contains several new plan steps and therefore several new flaws. According to the heuristic $h_{\#F}$, for example, P_2 looks much worse than P_1 although the decomposition generating P_2 was inevitable. Normalizing tries to compensate that phenomenon.

In addition to greedy search using different heuristics, we also include several base line configurations. These include the uninformed *Breadth First Search* (BF) and *Depth First Search* (DF). Furthermore, the core ideas behind the well-known hierarchical planning systems UMCP (Erol, Hendler, and Nau 1994) and SHOP2 (Nau et al. 2003) can be captured by our hybrid planning framework (Schattenberg, Weigl, and Biundo 2005).

[1] Greedy search using $h_{\#M}$ corresponds to the search strategy used in the evaluation by Elkawkagy et al. (2012). They called this strategy *Fewer Modifications First* (fmf).

In case of UMCP, our emulation is very close to the original system. We have employed all three variants of that system described in Erol's dissertation (1996, Chapter 6.1.2.1). As proposed by Erol, we employ BF and DF as plan selection, as well as greedy search using a heuristic that always selects a partial plan with the smallest number of abstract tasks. UMCP always decomposes an abstract task before resolving open precondition flaws or causal threats.

To emulate SHOP2, we employ a depth first search with the flaw selection *earliest first*. That function always prefers a flaw associated with a plan element that is closest to the initial task, i.e., closest to the execution horizon. Note that our emulation of SHOP2 still shows some significant differences to the actual SHOP2 system. SHOP2 performs progression search in the space of states while our system is based on POCL techniques. Furthermore, the actual SHOP2 system uses domain-specific search space information that is encoded in the preconditions of methods while our approach uses domain-independent search strategies.

System Configuration

In all experiments – including the ones for SHOP2 and UMCP – we enable the TDG pruning technique described by Elkawkagy et al. (2010). The explored search space is reduced by omitting decomposition methods that are not supported by the pruned TDG.

We conducted the experiments on a system with Intel Xeon E5 2.5 Ghz processors. Because most of the problem instances are solved within only a few seconds, we report and focus on the number of expanded search nodes to obtain a more accurate measure of search effort. Note that the number of *created* search nodes is in general much larger. We limited the available CPU time per problem instance to 10 minutes and the available memory to 2 GB.

To obtain statistically significant values, we run each experiment 50 times and report the (rounded) mean number of expanded search nodes μ_s, its relative standard deviation σ/μ_s, and the (rounded) mean CPU time μ_t in seconds, including preprocessing. We report the number of successful runs in parentheses if some were not successful. The reported values are based only on these successful runs.

Experimental Results

The results for the evaluated domains are shown in Tab. 1 to 4. The best result is highlighted in bold and the second best in bold and italic.

UM-Translog In this domain, we did not find any significant differences between the compared configurations.

The evaluated UM-Translog problem instances[2] turn out to be very easy, since even the uninformed BF and DF strategies always found a solution very quickly. We attribute this result to the TDG pruning that eliminates most of the choices for different decomposition methods. Remember that the pruned TDG has a branching factor of at most 1.11 in this domain. Despite that observation, the experiments show that the evaluated UMCP and SHOP2 configurations performed much worse than the other strategies. Concerning the other configurations, results have to be interpreted with care, since the results do not differ significantly (cf. Tab. 1). We observe that in all but one problem instances, greedy search using $h_{\#F}+h_{TC+PC}$ and $h_{\#F}+h_{MME}$, respectively, was among the two best-performing configurations. Another interesting result is that both $h_{\#F} + h_{TC+PC}$ and $h_{\#F} + h_{MME}$ dominate the heuristic $h_{\#F}$ in *all* problem instances. We can hence conclude that adding the estimates based on the TDG improves the estimate that is entirely based on $h_{\#F}$.

[2]We evaluated 20 problem instances while we report only 8 of these in Tab. 1. The remaining instances turned out to be uninteresting for our evaluation, since all tested search strategies produce the same number of search nodes with a standard deviation of 0. Expanded search nodes vary from 25 to 55.

Satellite While the small problem instances can be solved almost instantly, the problem instances in which three observations have to be taken require more than 100.000 node expansions for many of the configurations. In this domain, all our TDG-based heuristics appear to be quite well informed. In 8 of 15 problem instances, $\|h_{\#F} + h_{TC+PC}\|$ and $\|h_{\#F} + h_{MME}\|$ are the two best-performing heuristics. We can also observe that these normalized versions of our heuristics expand less nodes than their non-normalized versions in all but a few cases. In 12 of the 15 evaluated problem instances, one of the proposed four heuristics is among the two best-performing configurations. Our heuristics are in general performing very well in this domain. Especially in comparison to SHOP2 and UMCP, a much better search efficiency can be observed for the proposed heuristics. In some problem instances, for example in 3–1–1, 3–2–1, and 3–3–1, the proposed heuristics expand several hundred thousand search nodes less than the SHOP2 and the BF and DF versions of UMCP. The heuristic version of UMCP, however, works very well in that domain, but is still dominated by our TDG-based heuristics in many cases.

Since the Satellite domain shows the largest deviation among the different search strategies and heuristics, we have included a plot (Fig. 2(a)) showing the number of solved instances over the number of search node expansions. Higher curves indicate that more solutions were found given the same number of expansions. As a baseline, we included BF, DF, and the SHOP2 configuration as well as UMCP-H, which is the best-performing variant of UMCP in that domain, and $h_{\#M}$. Concerning our proposed heuristics, we included $\|h_{\#F} + h_{MME}\|$ in the plot, as it is the one with the best results. The heuristic $\|h_{\#F} + h_{TC+PC}\|$ shows a similar behavior: its graph lies only slightly below that of $\|h_{\#F} + h_{MME}\|$. We also included the non-normalized version of that heuristic, $h_{\#F} + h_{MME}$. Again, the graphs of $h_{\#F} + h_{MME}$ and $h_{\#F} + h_{TC+PC}$ are almost identical.

The plot clearly reveals the superiority of the normalized versions compared to their non-normalized counterparts. Furthermore, the best configuration in that domain is the one based on $\|h_{\#F} + h_{MME}\|$. Also, UMCP-H and DF both perform very well in that domain.

Concerning CPU time, we also produced a plot corresponding to Fig. 2(a) where the x-axis shows the CPU time.

Table 1 to 4: **Strategy** lists the used system configuration in the first six cases. In the remaining cases, it specifies the heuristic used with greedy search and the flaw selection LCFR. **Problem** lists the used problem instances. The column μ_s reports the rounded mean number of expanded search nodes over 50 runs, σ/μ_s its relative standard deviation, and μ_t the rounded mean CPU time in seconds. The number of successful runs is reported in parentheses if it is not 50.

Table 1: Results for the **UM-Translog** domain.

Problem / Strategy	#06			#08			#09			#10			#11			#12			#13			#14		
	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t
BF	71	0.06	**0**	93	0.29	**0**	765	0.54	5	**34**	0.00	**0**	**33**	0.00	**0**	85	0.03	**1**	77	0.03	**1**	78	0.03	**1**
DF	62	0.14	**0**	**35**	0.00	**0**	**76**	0.02	***2***	**34**	0.00	**0**	**33**	0.00	**0**	***80***	0.05	**1**	***73***	0.04	**1**	***73***	0.05	**1**
SHOP2	21121	2.43	6	**35**	0.00	**0**	***80***	0.37	***2***	**34**	0.01	**0**	**33**	0.00	**0**	94	0.25	**1**	**71**	0.08	**1**	83	0.28	**1**
UMCP-BF	205	0.19	***1***	84	0.04	**0**	1548	0.08	8	36	0.02	**0**	**33**	0.02	**0**	174	0.14	3	266	0.13	6	168	0.16	3
UMCP-DF	119	0.57	***1***	**35**	0.01	**0**	542	0.74	4	***35***	0.03	**0**	**33**	0.02	**0**	133	0.33	***2***	168	0.47	4	126	0.32	***2***
UMCP-H	93	0.55	**0**	***38***	0.14	**0**	315	1.08	4	**34**	0.02	**0**	**33**	0.02	**0**	99	0.29	**1**	132	0.49	3	90	0.25	**1**
$h_{\#M}$	58	0.10	**0**	**35**	0.00	**0**	81	0.07	3	**34**	0.00	**0**	**33**	0.00	**0**	84	0.02	***2***	75	0.02	***2***	77	0.02	***2***
$\|h_{\#M}\|$	56	0.10	**0**	**35**	0.00	**0**	**76**	0.02	3	**34**	0.00	**0**	**33**	0.00	**0**	84	0.03	***2***	76	0.02	***2***	77	0.03	***2***
$h_{\#F}$	58	0.08	**0**	39	0.13	**0**	96	0.17	3	**34**	0.00	**0**	**33**	0.00	**0**	84	0.02	**1**	76	0.03	***2***	77	0.02	***2***
$\|h_{\#F}\|$	58	0.08	**0**	**35**	0.00	**0**	**76**	0.02	***2***	**34**	0.00	**0**	**33**	0.00	**0**	83	0.03	**1**	75	0.02	***2***	77	0.03	***2***
$h_{\#F} + h_{TC+PC}$	**54**	0.05	**0**	**35**	0.00	**0**	**76**	0.02	**1**	**34**	0.00	**0**	**33**	0.00	**0**	**78**	0.03	**1**	75	0.03	***2***	**71**	0.03	**1**
$\|h_{\#F} + h_{TC+PC}\|$	57	0.11	**0**	**35**	0.00	**0**	**76**	0.03	**1**	**34**	0.00	**0**	**33**	0.00	**0**	***80***	0.04	**1**	75	0.03	***2***	***73***	0.04	**1**
$h_{\#F} + h_{MME}$	***55***	0.07	**0**	**35**	0.00	**0**	**76**	0.02	**1**	**34**	0.00	**0**	**33**	0.00	**0**	**78**	0.04	**1**	75	0.03	***2***	**71**	0.04	**1**
$\|h_{\#F} + h_{MME}\|$	64	0.12	***1***	**35**	0.00	**0**	**76**	0.03	**1**	**34**	0.00	**0**	**33**	0.00	**0**	81	0.04	**1**	76	0.03	***2***	74	0.04	**1**

Table 2: Results for the **Satellite** domain. The caption *X–Y–Z* stands for X observations, Y satellites, and Z modes.

Problem / Strategy	1–1–1			1–2–1			2–1–1			2–1–2			2–2–1			2–2–2			3–1–1			3–1–2		
	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t
BF	21	0.18	**0**	44	0.10	**0**	292	0.22	***1***	4145	0.26	7	354	0.08	2	3012	0.31	8	10498	0.24	8	(7) 1670647	0.13	469
DF	16	0.20	**0**	24	0.46	**0**	72	0.68	**0**	521	0.57	***2***	57	0.70	**0**	442	0.74	***2***	1214	0.78	3	171006	1.16	31
SHOP2	15	0.13	**0**	25	0.55	**0**	399	1.38	***1***	376	0.76	**1**	238	2.10	***1***	653	0.90	***2***	246446	1.47	46	115096	1.35	28
UMCP-BF	16	0.11	**0**	39	0.11	**0**	673	0.20	***1***	1080	0.17	4	2021	0.25	5	2165	0.26	5	112686	0.18	26	368516	0.11	79
UMCP-DF	15	0.12	**0**	20	0.33	**0**	389	1.38	**0**	339	0.87	**1**	352	1.79	***1***	273	0.62	**1**	238745	1.26	33	35242	1.62	***12***
UMCP-H	15	0.12	**0**	24	0.28	**0**	75	0.39	**0**	344	0.66	**1**	89	0.36	**0**	443	0.47	***2***	413	1.53	***1***	**6374**	0.79	**7**
$h_{\#M}$	18	0.17	**0**	26	0.39	**0**	157	0.25	***1***	968	0.43	5	136	0.10	***1***	784	0.46	5	2557	0.20	6	29040	1.26	15
$\|h_{\#M}\|$	***14***	0.08	**0**	**14**	0.00	**0**	42	0.11	**0**	620	0.62	3	43	0.30	***1***	325	0.76	***2***	1561	0.46	5	127033	1.52	31
$h_{\#F}$	17	0.16	**0**	26	0.40	**0**	111	0.25	**0**	793	0.30	4	86	0.16	***1***	620	0.43	3	1418	0.22	5	***18236***	0.74	***12***
$\|h_{\#F}\|$	***14***	0.09	**0**	***15***	0.08	**0**	72	0.28	**0**	450	0.29	***2***	33	0.14	**0**	252	0.59	**1**	3316	0.35	7	248677	0.84	45
$h_{\#F} + h_{TC+PC}$	16	0.20	**0**	24	0.39	**0**	***41***	0.08	**0**	880	0.34	5	54	0.36	**0**	373	0.74	***2***	940	0.38	4	60745	1.80	18
$\|h_{\#F} + h_{TC+PC}\|$	**13**	0.00	**0**	17	0.40	**0**	**22**	0.03	**0**	**92**	0.30	**1**	***31***	0.30	**0**	**123**	1.07	**1**	***46***	0.42	**0**	21939	1.29	***12***
$h_{\#F} + h_{MME}$	16	0.19	**0**	25	0.10	**0**	42	0.10	**0**	739	0.42	4	44	0.45	**0**	333	0.83	***2***	997	0.33	4	62353	1.77	18
$\|h_{\#F} + h_{MME}\|$	**13**	0.00	**0**	21	0.42	**0**	**22**	0.03	**0**	***105***	0.37	**1**	**27**	0.38	**0**	***160***	0.94	**1**	**37**	0.34	**0**	24459	1.30	***12***

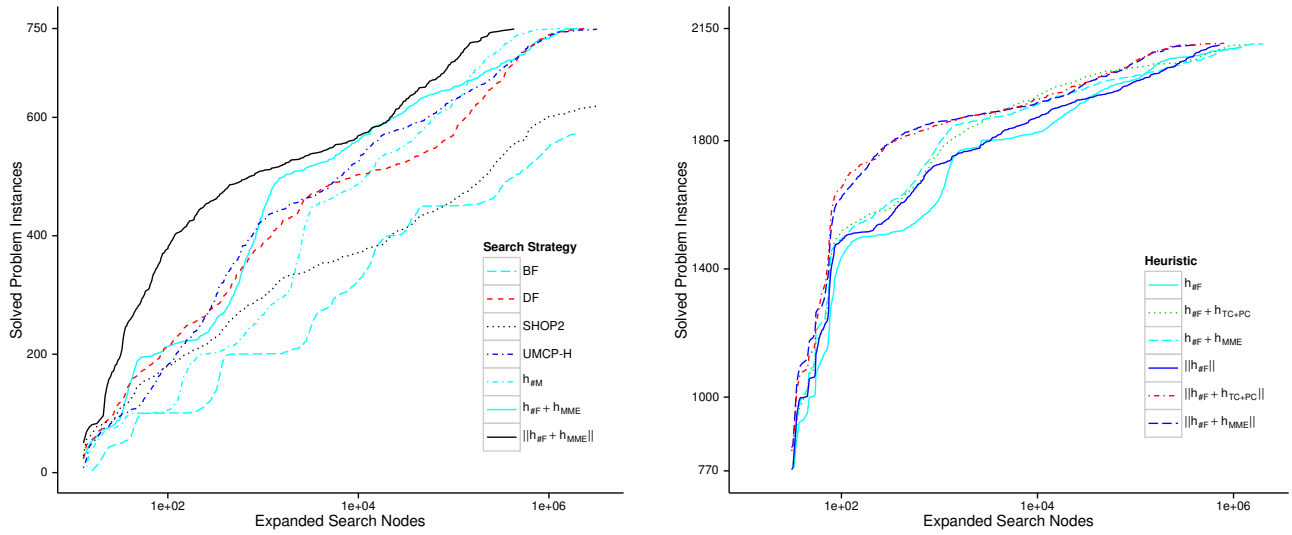
Problem / Strategy	3–1–3			3–2–1			3–2–2			3–2–3			3–3–1			3–3–2			3–3–3		
	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t
BF	—	—	—	15174	0.15	13	710661	0.34	140	—	—	—	36408	0.14	17	322284	0.32	69	(16) 1122746	0.25	338
DF	668416	0.77	109	940	0.99	4	106748	1.22	28	318515	0.96	61	1639	0.95	5	43456	1.18	16	239418	0.89	48
SHOP2	472177	0.88	99	(40) 333398	2.41	64	(37) 294693	1.37	77	(11) 532783	1.29	133	(39) 82194	1.87	22	(29) 339167	1.22	83	(13) 461074	1.66	101
UMCP-BF	605241	0.10	122	810422	0.21	17	—	—	—	—	—	—	(42) 1628608	0.17	374	—	—	—	—	—	—
UMCP-DF	***129381***	0.89	**32**	(41) 113375	1.94	20	(48) 287016	1.52	63	762774	1.13	151	566905	1.72	80	(49) 350331	1.51	72	581613	0.68	133
UMCP-H	**125524**	0.77	***33***	1101	2.03	***2***	19992	1.03	***11***	(49) 636852	1.03	133	1206	1.36	***3***	24799	0.99	12	513303	0.58	122
$h_{\#M}$	219016	0.64	42	2208	0.14	9	36058	0.97	19	205674	0.63	46	2692	0.10	9	25465	0.97	15	247083	1.07	52
$\|h_{\#M}\|$	354407	0.91	72	1149	0.34	6	88179	0.96	29	308752	0.95	66	617	0.16	4	***5996***	0.99	***10***	184650	2.34	41
$h_{\#F}$	(42) 468944	0.73	101	1133	0.09	6	19827	0.48	17	122587	0.96	34	1162	0.14	5	30125	0.57	18	163326	0.93	37
$\|h_{\#F}\|$	245524	0.75	46	691	0.42	5	112265	1.12	31	126640	1.12	31	599	0.33	4	20535	2.22	13	64105	2.30	**19**
$h_{\#F} + h_{TC+PC}$	599905	0.47	106	410	0.37	3	**5174**	1.54	**10**	***84298***	1.88	**26**	729	0.35	4	**3972**	0.86	11	***50111***	1.93	**19**
$\|h_{\#F} + h_{TC+PC}\|$	132639	0.81	44	***242***	0.56	***2***	14283	1.63	12	113607	1.37	36	***474***	0.90	***3***	9438	1.68	***10***	**47144**	1.40	***20***
$h_{\#F} + h_{MME}$	807016	0.46	142	601	0.74	3	***11600***	1.42	***11***	179804	1.22	44	699	0.34	4	7477	2.02	**8**	85274	1.45	26
$\|h_{\#F} + h_{MME}\|$	145929	0.64	46	**128**	1.19	**1**	14862	2.46	***11***	**59059**	1.10	***27***	**248**	1.28	**2**	14435	2.03	***10***	51977	1.28	22

Table 3: Results for the **SmartPhone** domain.

Problem / Strategy	#1			#2			#3		
	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t
BF	30	0.14	**0**	486980	0.24	103	—	—	—
DF	20	0.06	**0**	(12) 166	1.57	***1***	164	1.04	**1**
SHOP2	20	0.08	**0**	(8) **82**	0.24	**0**	60486	2.30	22
UMCP-BF	58	0.24	**0**	—	—	—	375530	0.04	55
UMCP-DF	***19***	0.08	**0**	(5) 2033	1.27	2	15863	1.45	6
UMCP-H	**18**	0.00	**0**	—	—	—	15964	1.21	7
$h_{\#M}$	**18**	0.00	**0**	—	—	—	27114	0.04	17
$\|h_{\#M}\|$	**18**	0.00	**0**	—	—	—	27111	0.04	17
$h_{\#F}$	***19***	0.05	**0**	—	—	—	1608	0.61	9
$\|h_{\#F}\|$	20	0.00	**0**	—	—	—	826	0.78	7
$h_{\#F} + h_{TC+PC}$	22	0.08	**0**	—	—	—	***112***	0.40	***2***
$\|h_{\#F} + h_{TC+PC}\|$	21	0.02	**0**	(4) ***120***	0.11	***1***	**67**	0.15	**1**
$h_{\#F} + h_{MME}$	**18**	0.00	**0**	(2) 139	0.13	***1***	164	0.30	***2***
$\|h_{\#F} + h_{MME}\|$	20	0.07	**0**	—	—	—	230	0.89	***2***

Table 4: Results for the **Woodworking** domain.

#1			#2			#3			#4			#5		
μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t
23	0.02	**0**	***48***	0.01	**0**	60	0.05	**0**	565	0.25	**1**	474	0.27	***1***
23	0.02	**0**	**47**	0.01	**0**	46	0.18	**0**	210	0.41	**1**	168	0.43	***1***
30	0.59	**0**	96	1.90	***1***	332	1.56	***2***	(29) 198536	1.78	86	(30) 152786	1.93	59
26	0.18	**0**	54	0.19	**0**	74	0.15	**0**	1642	0.31	***2***	1612	0.18	2
24	0.11	**0**	***48***	0.07	**0**	51	0.25	**0**	164	0.32	**1**	167	0.34	***1***
25	0.11	**0**	49	0.10	**0**	50	0.25	**0**	115	0.34	**1**	106	0.39	**0**
23	0.02	**0**	**47**	0.01	**0**	***37***	0.07	**0**	99	0.17	**1**	74	0.19	***1***
23	0.02	**0**	**47**	0.01	**0**	***37***	0.07	**0**	103	0.24	**1**	71	0.23	***1***
23	0.02	**0**	**47**	0.01	**0**	***37***	0.02	**0**	***89***	0.20	**1**	***67***	0.12	***1***
23	0.02	**0**	**47**	0.01	**0**	***37***	0.02	**0**	90	0.27	**1**	**62**	0.10	***1***
23	0.02	**0**	**47**	0.01	**0**	***37***	0.03	**0**	92	0.24	**1**	71	0.18	***1***
23	0.02	**0**	**47**	0.01	**0**	***37***	0.03	**0**	101	0.51	**1**	72	0.27	***1***
23	0.02	**0**	**47**	0.01	**0**	**36**	0.02	**0**	**82**	0.47	**1**	84	0.43	***1***
23	0.02	**0**	**47**	0.01	**0**	***37***	0.07	**0**	171	0.47	**1**	175	0.37	***1***



(a) Satellite Domain

(b) All Domains

Figure 2: These plots show the number of solved problem instances (each run 50 times) over the number of expanded search nodes. For Fig. (a) only results for the Satellite domain were used, while Fig. (b) shows results over all domains. For the latter, we have omitted the data points below 750 solved instances for visual clarity.

We did not include it in the paper due to space restrictions, but we can report that the two plots look very similar indicating that the reduced search space pays off and comes with no additional overhead w.r.t. computation time.

SmartPhone The SmartPhone domain is the hardest domain that we evaluated. The first problem instance looks very similar to the UM-Translog instances: while our heuristics are among the best-performing strategies, there is only very little difference between the configurations. Even BF and DF find solutions very quickly. In the third problem instance, our strategies also perform very well. They expand at most 229 search nodes on average, where the best version of UMCP expands more than 150.000 nodes. Surprisingly, DF performs very well with only 163 expanded search nodes on average. The second problem instance seems to be the hardest one. All configurations except BF and DF produced timeouts or exceeded the memory limit in almost all runs. The SHOP2 configuration was able to find a solution very quickly (81 on average), but only in 8 out of 50 runs. Two of our proposed heuristics also solve the problem very quickly, but only in 2 and 4 runs, respectively.

Woodworking In all but one problems, $h_{\#F} + h_{MME}$ is among the two best-performing heuristics. However, clear conclusions cannot be drawn, since planning performance does not significantly vary among the deployed strategies.

Summary In summary, we have seen that our TDG-based heuristics need to expand the smallest number of search nodes in order to find a solution. To investigate the difference between the proposed heuristics, we include a plot (Fig. 2(b)) that shows the number of solved problem instances given a number of expanded search nodes for all problem instances among all domains.

We can draw several conclusions when investigating the data. First, we can see that the heuristics that need to explore the largest part of the search space are the relatively simple heuristics $h_{\#F}$ and $\|h_{\#F}\|$. This is our most important finding, since we can see that taking the TDG into account actually improves these heuristics significantly. It comes to our surprise, however, that there are no significant differences between the heuristics $\|h_{\#F} + h_{MME}\|$ and $\|h_{\#F} + h_{TC+PC}\|$. Another interesting result is that the normalized version of a heuristic clearly performs better on average than the corresponding non-normalized one.

Conclusion

We have proposed novel heuristics for Hybrid Planning that estimate the necessary number of modifications for a partial plan to turn it into a solution. These heuristics are based on a task decomposition graph and hence capable of incorporating the hierarchical aspects of the underlying domain. We conducted experiments using a novel algorithm for Hybrid Planning. Our heuristics proved to be the most informed ones in the evaluated problem instances.

Acknowledgment

We want to thank Thomas Geier and Bernd Schattenberg for proofreading the paper. This work is done within the Transregional Collaborative Research Centre SFB/TRR 62 “Companion-Technology for Cognitive Technical Systems” funded by the German Research Foundation (DFG).

References

Andrews, S.; Kettler, B.; Erol, K.; and Hendler, J. A. 1995. UM translog: A planning domain for the development and benchmarking of planning systems. Technical Report CS-TR-3487, Department of Computer Science, Institute of Systems Research, University of Maryland.

Bercher, P.; Geier, T.; Richter, F.; and Biundo, S. 2013. On delete relaxation in partial-order causal-link planning. In *Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI 2013)*, 674–681.

Bercher, P.; Geier, T.; and Biundo, S. 2013. Using state-based planning heuristics for partial-order causal-link planning. In *Advances in Artificial Intelligence, Proceedings of the 36nd German Conference on Artificial Intelligence (KI 2013)*, 1–12. Springer.

Biundo, S., and Schattenberg, B. 2001. From abstract crisis to concrete relief – a preliminary report on combining state abstraction and HTN planning. In *Proceedings of the 6th European Conference on Planning (ECP 2001)*, 157–168.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129:5–33.

Elkawkagy, M.; Bercher, P.; Schattenberg, B.; and Biundo, S. 2012. Improving hierarchical planning performance by the use of landmarks. In *Proceedings of the 26th National Conference on Artificial Intelligence (AAAI 2012)*, 1763–1769. AAAI Press.

Elkawkagy, M.; Schattenberg, B.; and Biundo, S. 2010. Landmarks in hierarchical planning. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2010)*, volume 215, 229–234. IOS Press.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS 1994)*, 249–254. AAAI Press.

Erol, K. 1996. *Hierarchical Task Network Planning: Formalization, Analysis, and Implementation*. Ph.D. Dissertation, University of Maryland.

Geier, T., and Bercher, P. 2011. On the decidability of HTN planning with task insertion. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 1955–1961. AAAI Press.

Joslin, D., and Pollack, M. E. 1994. Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1994)*, 1004–1009. AAAI Press.

Kambhampati, S.; Mali, A.; and Srivastava, B. 1998. Hybrid planning for partially hierarchical domains. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI 1998)*, 882–888. AAAI Press.

Marthi, B.; Russell, S. J.; and Wolfe, J. 2008. Angelic hierarchical planning: Optimal and online algorithms. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 222–231. AAAI Press.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI 1991)*, 634–639. AAAI Press.

Nau, D. S.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research (JAIR)* 20:379–404.

Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, 459–466. Morgan Kaufmann.

Penberthy, J. S., and Weld, D. S. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the third International Conference on Knowledge Representation and Reasoning*, 103–114. Morgan Kaufmann.

Porteous, J.; Sebastia, L.; and Hoffmann, J. 2001. On the extraction, ordering, and usage of landmarks in planning. In *Proceedings of the 6th European Conference on Planning (ECP 2001)*, 37–48.

Russell, S., and Norvig, P. 1994. *Artificial Intelligence – A modern Approach*. Prentice-Hall, 1 edition.

Schattenberg, B.; Bidot, J.; and Biundo, S. 2007. On the construction and evaluation of flexible plan-refinement strategies. In *Advances in Artificial Intelligence, Proceedings of the 30th German Conference on Artificial Intelligence (KI 2007)*, 367–381. Springer.

Schattenberg, B.; Weigl, A.; and Biundo, S. 2005. Hybrid planning using flexible strategies. In *Advances in Artificial Intelligence, Proceedings of the 28th German Conference on Artificial Intelligence (KI 2005)*, 249–263. Springer.

Schattenberg, B. 2009. *Hybrid Planning & Scheduling*. Ph.D. Dissertation, University of Ulm, Germany.

Shivashankar, V.; Alford, R.; Kuter, U.; and Nau, D. 2013. The GoDeL planning system: A more perfect union of domain-independent and hierarchical planning. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2380–2386. AAAI Press.

Younes, H. L. S., and Simmons, R. G. 2003. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research (JAIR)* 20:405–430.