Landmarks in Hierarchical Planning

Mohamed Elkawkagy and Bernd Schattenberg and Susanne Biundo

Abstract. In this paper we introduce a novel landmark technique for hierarchical planning. Landmarks are abstract tasks that are mandatory. They have to be performed by any solution plan. Our technique relies on a landmark extraction procedure that pre-processes a given planning problem by systematically analyzing the ways in which relevant abstract tasks can be decomposed. We show how the landmark information is used to guide hierarchical planning and present some experimental results that give evidence for the considerable performance increase gained through our technique.

1 Introduction

In recent years, the exploitation of knowledge gained by preprocessing a planning domain and/or problem description has proven to be an effective means to reduce planning effort. Various preprocessing procedures, like effect relaxation [2], abstractions [8], and landmarks [15], have been proposed for classical planning, where they serve to compute strong search heuristics. As opposed to this, pruning the search space of a hierarchical planner by pre-processing the underlying HTN-based domain description has not been considered so far.

Hierarchical Task Network (HTN) planning is based on the concepts of tasks and methods [4, 13]. Abstract tasks represent compound activities like making a business trip or transporting certain goods to a specific location. Primitive tasks correspond to classical planning operators. Hierarchical domain models hold a number of methods for each abstract task. Each method provides a task network, also called partial plan, which specifies a pre-defined (abstract) solution of the corresponding abstract task. Planning problems are (initial) task networks. They are solved by incrementally decomposing the abstract tasks until the network contains only primitive tasks and is consistent w.r.t. to their ordering and causal structure. The decomposition of an abstract task by an appropriate method replaces the abstract task by the partial plan specified by the respective method.

In this paper, we present a novel landmark technique to increase the performance of a hierarchical planner. In hierarchical planning, landmarks are mandatory abstract or primitive tasks, i.e. tasks that have to be performed by any solution plan. For an initial task network that states a current planning problem, a pre-processing procedure computes the corresponding landmarks. It does so by systematically inspecting the methods that are eligible to decompose the relevant abstract tasks. Beginning with the (landmark) tasks of the initial network, the procedure follows the way down the decomposition hierarchy until no further abstract tasks qualify as landmarks. As for primitive landmarks, a reachability test is accomplished; a failure indicates that the method which introduced the primitive landmark is no longer eligible. This information is propagated back, up the decomposition hierarchy and serves to identify all methods that will never lead to a solution of the current planning problem. Being able to prune useless regions of the search space this way, a hierarchical planner performs significantly better than it does without exploiting the landmark information.

While the use of landmark tasks is a novelty in hierarchical planning, landmarks are a familiar concept in classical state-based planning. There, landmarks are facts that have to hold in some intermediate state of every plan that solves the problem. The concept was introduced in [15] and further developed in [22] and [10], where landmarks and orderings between them are extracted from a planning graph of the relaxed planning problem. Other strands of research arranged landmarks into groups of intermediate goals to be achieved [20] and extended the landmark concept to so-called disjunctive landmarks [7, 14]. A disjunctive landmark is a set of literals any of which has to be satisfied in the course of a valid plan. A generalization of disjunctive landmarks resulted in the notion of so-called (disjunctive) action landmarks [12, 16, 21]. They represent landmark facts by actions that are appropriate to achieve them. Most recent approaches use landmark information to compute heuristic functions for a forward searching planner [12, 16] and investigate their relations to critical-path-, relaxation-, and abstraction-heuristics [9].

In summary, it turned out that the use of landmark information significantly improves the performance of classical state-based planners.

Before introducing the landmark extraction procedure for hierarchical planning in Section 3, we will briefly review the underlying framework in Section 2. Afterwards, Section 4 shows how landmark information is exploited during planning. Section 5 presents experimental results from a set of benchmark problems of the UM-Translog and Satellite domains, which give evidence for the considerable performance increase gained through our technique. The paper ends with some concluding remarks in Section 6.

2 Formal Framework

Our approach relies on a domain-independent hybrid planning framework [1]. Hybrid planning [11] combines hierarchical task network planning along the lines of [4] with concepts of partial-order-causal-link (POCL) planning. The resulting systems integrate task decomposition with explicit causal reasoning. Therefore, they are able to use predefined standard solutions like in pure HTN planning and thus benefit from the landmark technique we will introduce below; they can also develop (parts of) a plan from scratch or modify a default solution in cases where the initial state deviates from the presumed standard. It is this flexibility that makes hybrid planning particularly well suited for real-world applications [3, 5].

In our framework, a task network or partial plan \( P = (S, \prec, V, C) \) consists of a set of plan steps \( S \), i.e. (partially) instantiated task
schemata, a set of ordering constraints $\prec$ that impose a partial order on the plan steps, and a set of variable constraints $V$. $C$ is a set of causal links. A causal link $s_i \rightarrow s_j$ indicates that the precondition $\varphi$ of plan step $s_j$ is an effect of plan step $s_i$ and is supported this way. A domain model $D = (T, M)$ includes a set of task schemata and a set of decomposition methods. A task schema $t(\tau) = (\text{prec}(t(\tau)), \text{add}(t(\tau)), \text{del}(t(\tau)))$ specifies the precondition as well as the positive and negative effects of a task. Preconditions and effects are sets of literals and $\tau = \tau_1, \ldots, \tau_n$ are the task parameters. Both primitive and abstract tasks show preconditions and effects. This enables the use of POCL planning operations even on abstract levels and allows for the generation of abstract solutions [1]. This option is not considered in this paper, however. A method $m = \langle t, P \rangle$ relates an abstract task $t$ to a partial plan $P$, which represents (an abstract) solution or “implementation” of the task. In general, a number of different methods are provided for each abstract task. Please note that no application conditions are associated with the methods, as opposed to typical HTN-style planning. A planning program $\Pi = (D, S_0, S_g, P_{init})$ includes a domain model $D$, an initial state $S_0$, and a goal state $S_g$. $P_{init}$ represents an initial partial plan.

Based on these strictly declarative specifications of planning domains and problems, hybrid planning is performed by refining an initial partial plan $P_{init}$ stepwise until a partial plan $P = \langle S, \prec, V, C \rangle$ is obtained that satisfies the following solution criteria: (1) each precondition of a plan step in $P$ is supported by a causal link in $C$; (2) the ordering and variable constraints are consistent; (3) none of the causal links in $C$ is threatened, i.e. for each causal link $s_i \rightarrow s_j$ the ordering constraints ensure that no plan step $s_k$ with effect $\neg \varphi$ can be ordered between plan steps $s_i$ and $s_j$; (4) all plan steps in $S$ are primitive tasks. Refinement steps include the decomposition of abstract tasks by appropriate methods, the insertion of causal links to support open preconditions of plan steps as well as the insertion of plan steps, ordering constraints, and variable constraints.

### 3 Landmark Extraction

For a given planning problem $\Pi = (D, S_0, S_g, P_{init})$, landmarks are the abstract tasks that occur in any sequence of decompositions leading from the initial task network $P_{init}$ to a solution plan. Landmark extraction is done using a so-called task decomposition tree (TDT) of $\Pi$. Figure 1 depicts such a tree schematically. The TDT of $\Pi$ is an AND/OR tree that represents all possible ways to decompose the abstract tasks of $P_{init}$ by methods in $D$ until a primitive level is reached or a task is encountered that is already included in an upper level of the TDT. Each level of a TDT consists of two parts, a task and a method level. The root node on Level 0 is an artificial method node that represents the initial partial plan $P_{init}$. Method nodes are AND nodes. The children of a method node are the tasks that occur in the partial plan of the respective method. The children of the root are the tasks of $P_{init}$. Method edges connect method nodes on Level $i$ to task nodes on Level $i + 1$. Task nodes are OR nodes. The children of a task node are the methods that can be used to decompose the respective task. Primitive tasks are leaves of the TDT. A TDT is built by forward chaining from the abstract tasks in the initial task network until all nodes of the fringe are leaf nodes.

In order to determine the landmarks of a planning problem $\Pi$ we need to identify those tasks which all decomposition methods of a certain abstract task have in common. To this end, we define the Common Task Set of two methods.

**Definition 1 (Common Task Set $\tilde{\cap}$).** For two methods $m_i = \langle t_i, S_i, \prec_i, V_i, C_i \rangle$ and $m_j = \langle t_j, S_j, \prec_j, V_j, C_j \rangle$ of a task $t$, the Common Task Set $\cap$ of $m_i$ and $m_j$ is defined as

$$m_i \cap m_j = S_i \cap S_j$$

In a similar way, the sets of tasks in which two methods differ are given as follows.

**Definition 2 (Remaining Task Sets $\tilde{\cup}$).** Given two methods $m_i = \langle t_i, S_i, \prec_i, V_i, C_i \rangle$ and $m_j = \langle t_j, S_j, \prec_j, V_j, C_j \rangle$ of a task $t$, the Remaining Task Sets $\cup$ of $m_i$ and $m_j$ are

$$m_i \cup m_j = \{ S_i \setminus (m_i \cap m_j), \{ S_j \setminus (m_i \cap m_j) \}$$

A landmark table records for each abstract landmark task $t$ a set of subtasks $I(t)$ as well as a set of sets of subtasks $O(t)$ as depicted in Table 1. The intersection $I(t)$ contains those subtasks which occur on every possible path of decompositions that transforms $t$ into a primitive plan. The options $O(t)$ represent sets of those subtasks that optionally occur when decomposing the respective landmark task towards a solution plan. Every such set is indexed by the name of the method which contains these subtasks.

<table>
<thead>
<tr>
<th>Task $t_i$</th>
<th>Intersection $I(t)$</th>
<th>Options $O(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{init}$</td>
<td>${ t_{11}, t_{12}, \ldots }$</td>
<td>${ { t_{13}, t_{14}, t_{15}, t_{16}, \ldots }, }$</td>
</tr>
<tr>
<td>Task $t_n$</td>
<td>${ t_{n1}, t_{n2}, \ldots }$</td>
<td>${ { t_{n3}, t_{n4}, t_{n5}, t_{n6}, \ldots }, }$</td>
</tr>
</tbody>
</table>

Now we are ready to present the landmark extraction algorithm (Algorithm 1). It takes a task decomposition tree, a current tree level and a landmark table as input and computes a final landmark table. For a given planning problem the task decomposition tree is computed and the algorithm is called with an empty landmark table and tree level 1. It runs recursively through all levels of the task decomposition tree in order to identify landmarks, insert them in the table, and prune useless branches from the tree, until the maximum level has been reached.

For each abstract task $t$ of task level $i$ that has not yet been entered into the landmark table all methods $M = \{ m_1, m_2, \ldots, m_n \}$ of method level $i$ that decompose $t$ (TDT($t$)) are collected (lines 6-8). The Common Task Set $I(t)$ of all methods in $M$ is computed according to Definition 1. Please note that if there is only one method $m_0$ that can decompose $t$, then $I(t)$ is just the set of plan steps of the partial plan provided by $m_0$. In the next step the Remaining Task Sets $O(t)$ are obtained by processing the methods in $M$ according to Definition 2. Afterwards, each task $tst$ of a task set $T$ in $O(t)$ is investigated (lines 9-14). If $tst$ is primitive and unreachable, then all sub-trees with roots $\alpha \in T$ are pruned from the task decomposition.
tree and the option \( T \) is removed from \( O(t) \). The reason is that those decompositions will never lead to a solution of the abstract task \( t \) under consideration. The reachability test estimates the achievability of the preconditions of \( t^st \). Like in [6], it is based on the type structure of the domain model of the planning problem and detects whether some preconditions of a primitive task can never be satisfied.

**Algorithm 1: Landmark Extraction** (\( TDT, i, LT \))

Initialize: \( LT := \emptyset, i := 1 \)

Input: \( TDT \): Task Decomposition Tree,

\( i \): Index of the current level in \( TDT \), \( LT \): LandmarkTable

Output: a LandmarkTable

1. begin
2. if \( i \geq \text{maxlevel}(TDT) \) then
3. return \( LT \)
4. else
5. foreach abstract task \( t \) in task level \( i \) with \( t \notin LT \) do
6. \{ \( m_1, m_2, \ldots, m_n \) \} \leftarrow Methods(\( TDT_i(t) \))
7. \( I(t) \leftarrow \bigcap_{m=1}^{n} m_i \)
8. \( O(t) \leftarrow \bigcup_{m=1}^{n} m_i \)
9. foreach set \( T \in O(t) \) do
10. foreach task \( t \in T \) do
11. if \( t \) is a primitive task with \( t \) is unreachable then
12. \( TDT \leftarrow \text{Remove} \{TDT, t\} \) \; \forall \text{ tasks } t \in T \)
13. \( O(t) \leftarrow O(t) \setminus T \)
14. continue with next set \( T \) from \( O(t) \).
15. \( LT \leftarrow \text{Append} \{(LT_i(t), O(t))\} \)
16. return Landmark Extraction(\( TDT, i + 1, LT \))
17. end

Finally, the current landmark table \( LT \) is updated by inserting the current abstract task \( t \) and the related sets \( I(t) \) and \( O(t) \), respectively. Then the landmark extraction algorithm is called recursively with the (modified) task decomposition tree and updated landmark table to inspect the next level of the tree.

In order to illustrate our algorithm, let us consider a simple example from the UM-Translog domain. Assume a package \( P_1 \) is at location \( L_1 \) in the initial state and we would like to transport it to a customer location \( L_3 \) in the same city by using truck \( T_1 \), which initially is located at \( L_3 \). Figure 2 shows part of the task decomposition tree of this example.

The **Landmark Extraction** algorithm detects that the first level in the \( TDT \) has only one abstract task \( t = \text{transport}(P_1, L_1, L_3) \) and that there is only one method, \( P_{L_2a.de} \), that can decompose the task into a partial task, which has subtasks \( \text{pickup}(P_1), \text{carry}(P_1, L_1, L_3), \) and \( \text{deliver}(P_1) \). \( I(t) \) becomes \{\{\text{collect fees}(P_1)\}\}, \{\text{have permit}(P_1)\}, \{\text{collect insurance}(P_1)\}\} and \( O(t) = \emptyset \). The current abstract task and sets \( I(t) \) and \( O(t) \) are entered as the first row of the landmark table as shown in Table 2.

Then the **Landmark Extraction** algorithm takes the (unchanged) \( TDT \) and the modified landmark table to investigate the next tree level. The abstract tasks to be inspected on this level are \( \text{pickup}(P_1), \text{carry}(P_1, L_1, L_3), \) and \( \text{deliver}(P_1) \). Suppose, we choose the task \( t = \text{pickup}(P_1) \) first. As shown in Figure 2 the task decomposition tree accounts for three methods to decompose this task: **Pickup hazardous, Pickup normal**, and **Pickup valuable**. By computing the Common Task Set and Remaining Task Sets we get \( I(t) = \{\text{collect fees}(P_1)\} \) and \( O(t) = \{\{\text{have permit}(P_1)\}, \{\text{collect insurance}(P_1)\}\} \). Please note that all empty sets are omitted. At this point, reachability has to be tested for each primitive task in each set of \( O(t) \). Assume that the primitive task \( \text{have permit}(P_1) \) is reachable, whereas \( \text{collect insurance}(P_1) \) is unreachable. The task set which contains \( \{\text{collect insurance}(P_1)\} \) has therefore to be omitted from \( O(t) \).

In the second iteration the abstract task \( t = \text{carry}(P_1, L_1, L_3) \) is considered. The methods **Carry normal** and **Carry via hub** are available to decompose this task. We obtain \( I(t) = \emptyset, O(t) = \{\{\text{collect fees}(P_1)\}\} \). Suppose the primitive task \( \text{go through centers} \) is unreachable. The sub-tree with root \( \text{carry via hub} \) has then to be removed from the \( TDT \) and the set which contains the unreachable task \( \text{go through centers} \) is removed from \( O(t) \). The current abstract task \( t = \text{transport}(P_1, L_1, L_3) \) together with \( I(t) \) and the modified \( O(t) \) are added to the landmark table.

**Table 2: Landmark table of the transportation task**

<table>
<thead>
<tr>
<th>Task</th>
<th>Interaction(t)</th>
<th>Options(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>transport((P_1, L_1, L_3))</td>
<td>{\text{pickup}(P_1), \text{carry}(P_1, L_1, L_3), \text{deliver}(P_1)}</td>
<td>-</td>
</tr>
<tr>
<td>pickup((P_1))</td>
<td>{\text{collect fees}(P_1)}</td>
<td>{\text{have permit}(P_1)}, {\text{carry via hub, go through centers}}</td>
</tr>
<tr>
<td>carry((P_1, L_1, L_3))</td>
<td>-</td>
<td>{\text{collect fees}(P_1)}</td>
</tr>
</tbody>
</table>

In the second iteration the abstract task \( t = \text{transport}(P_1, L_1, L_3) \) is considered. The methods **Carry normal** and **Carry via hub** are available to decompose this task. We obtain \( I(t) = \emptyset, O(t) = \{\{\text{collect fees}(P_1)\}\} \). Suppose the primitive task \( \text{go through centers} \) is unreachable. The sub-tree with root \( \text{carry via hub} \) has then to be removed from the \( TDT \) and the set which contains the unreachable task \( \text{go through centers} \) is removed from \( O(t) \). The current abstract task \( t = \text{transport}(P_1, L_1, L_3) \) together with \( I(t) \) and the modified \( O(t) \) are added to the landmark table.

**Table 3: Search space reduction in the UM-Translog Domain**

<table>
<thead>
<tr>
<th>Problems</th>
<th>Regular Truck Problems</th>
<th>Various Truck-Type Problems</th>
<th>Airplane Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>before pre-processing</td>
<td>21</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>after pre-processing</td>
<td>51</td>
<td>32</td>
<td>31</td>
</tr>
</tbody>
</table>

Table 3 shows the reduction of the domain model for typical examples from the UM-Translog domain in terms of the number of abstract tasks (AbT) and methods (Met). It indicates that in this domain the landmark technique achieves a reduction of the number of abstract tasks that ranges between 33% and 42%, while the reduction of the number of methods varied between 27% and 41%.

**4 Landmark Exploitation**

Our planning approach makes use of an explicit representation of plan-refinement operators, the so-called plan modifications. Given a partial plan \( P = \{S, \prec, V, C\} \) and a domain model \( D \), a plan modification is defined as \( n = (E^\oplus, E^\ominus) \), where \( E^\oplus \) and \( E^\ominus \) are disjoint sets of elementary additions and deletions of plan elements over \( P \) and \( D \). Consequently, all elements in \( E^\oplus \) are elements of \( S, \prec, V \) or \( C \), while \( E^\ominus \) consists of new plan elements. This generic definition makes all changes a modification imposes on a plan explicit. With that, a planning strategy is able to compare the available refinement options qualitatively and quantitatively and can hence choose opportunistic among them. Applying a modification \( n = (E^\oplus, E^\ominus) \) to a plan \( P \) returns \( P' \) that is obtained from \( P \) by adding all elements in \( E^\oplus \) and removing those of \( E^\ominus \). Hybrid planning distinguishes various classes of plan modifications including task expansion and task insertion. For each class \( H \), our system provides a corresponding modification generation module \( f^\text{mod}_H \).

For a partial plan \( P \) that is not yet a solution, so-called flaws make every violation of the solution criteria mentioned in Sec. 2 explicit. We distinguish various flaw classes including abstract tasks, unsupplied preconditions, and inconsistencies in the constraint sets. As for the generation of plan modifications, we employ a flaw detection module \( f^\text{det}_F \) for each flaw class \( F \).
Furthermore, we make use of a modification trigger function $\alpha$ that relates each flaw class to those modification classes that are suitable for generating refinements that solve the respective flaws.

Algorithm 2: Plan($P_1 \ldots P_n, \Pi$)

Require: Sets of flaw detection and modification generation modules $\ Reset$.

Input: $P_1 \ldots P_n$: Sequence of Plans, $\Pi = \{D, S_0, S_p, P_{init}\}$: Planning Problem

Output: Plan or failure

1 begin
2 if $n = 0$ then
3 return failure
4 $P_{current} \leftarrow P_1$; Fringe $\leftarrow P_2 \ldots P_n$; $F \leftarrow \emptyset$
5 forall $f^det \in \ Det$ do
6 $F \leftarrow F \cup f^det(P_{current}, \Pi)$
7 if $F = \emptyset$ then
8 return $P_{current}$
9 $M \leftarrow \emptyset$
10 forall $F_x \subseteq F$ with $F_x \neq \emptyset$ do
11 forall $f^mod \in M$ with $f^mod \subseteq \alpha(F_x)$ do
13 $M \leftarrow M \cup f^mod(P_{current}, F, D)$
14 if $f$ was un-addressed then
15 $P_{next} \leftarrow f_{planSel}($Fringe$)$
16 return Plan($P_{next} \circ (\text{Fringe} - P_{next})$, $\Pi$)
17 forall $n \in f_{modSel}(P_{current}, F, M)$ do
18 Fringe $\leftarrow apply(n, P_{current}) \circ \text{Fringe}$
19 $P_{next} \leftarrow f_{planSel}($Fringe$)$
20 return Plan($P_{next} \circ (\text{Fringe} - P_{next})$, $\Pi$)
21 end

Based on these definitions, Algorithm 2 sketches a generic hybrid planning algorithm. The procedure is initially called with the partial plan $P_{init}$ of a planning problem $\Pi$ as a unary list of plans and with the problem itself. This list of plans represents the current plan development options in the fringe of the search space. An empty fringe ($n = 0$) means that no more plan refinements are available. Lines 5-8 call the detection functions to collect the flaws in the current plan $P_{current}$. If $P_{current}$ is found flawless, it constitutes a solution to $\Pi$ and is returned. If not, lines 9-16 organize the flaws class-wise and pass them to the $\alpha$-assigned modification generation functions, which produce plan modifications that will eliminate the flaws. Any flaw that is found unsolvable will persist and $P_{current}$ is hence discarded [17]. The plan selection strategy $f_{planSel}$ is responsible for choosing a plan from the fringe with which to continue planning.

If appropriate refinements have been found for all flaws, the modification selection function $f_{modSel}$ is called in line 17. Based on the current plan and its flaws, it selects and prioritizes those plan modifications that are to be used for generating the refinements of the current plan. The chosen modifications are applied to $P_{current}$ and the produced successor plans are inserted in the search space fringe. The algorithm is finally called recursively on an updated fringe in which the strategy function $f_{planSel}$ determines the next focal plan.

Please note that the algorithm allows for a broad variety of planning strategies [18, 19], because the planning procedure is completely independent from the flaw detection and modification generating function.

Since our approach is based on a declarative model of task abstraction, the exploitation of knowledge about hierarchical landmarks can be done transparently during the generation of the task expansion modifications: First, the respective modification generation function $f^mod$ is deployed with a reference to the landmark table of the planning problem, which has been constructed off-line in a pre-processing phase. During planning, each time an abstract task flaw indicates an abstract plan step $t$ the function $f^mod$ does not need to consider all methods provided in the domain model for the abstract task $t$. Instead, it operates on a reduced set of applicable methods according to the respective options $O(t)$ in the landmark table.

It is important to see that the overall plan generation procedure is not affected by this domain model reduction, neither in terms of functionality (flaw and modification modules do not interfere) nor in terms of search control (strategies are defined independently and completeness of search is preserved). In principle, non-declarative hierarchical planners, like the SHOP family [13] can also profit from our landmark technique. The benefit will however be reduced due to the typically extensive usage of method application conditions, which cannot be analyzed during task reachability analysis, in particular if the modeller relies on side effects of the method processing.

5 Experimental Results

In theory, it is quite intuitive that a reduced domain model leads to an improved performance of the planning system. However, in order to quantify the practical performance gained by the hierarchical landmark technique, we conducted a series of experiments in the PANDA planning environment [17]. The planning strategies we used are representatives from the rich portfolio provided by PANDA, which has been documented elsewhere [18]. We briefly review the ones on which we based our experiments.

Modification selection functions determine the shape of the fringe, because they decide about the (priority of the) newly added plan refinements. We thereby distinguish selection principles that are based
on a prioritization of certain flaw or modification classes and strategies that opportunistically choose from the presented set. The latter ones are called flexible strategies.

Representatives for inflexible strategies are the classical HTN strategy patterns that try to balance task expansion with respect to other plan refinements. The SHOP modification selection, like the system it is named after [13], prefers task expansion for the abstract plans. The evaluated scenarios are thus defined as observations which results in a domain. Please note that the resulting strategies are general domain-independent planning strategies, which are not tailored to the application of the refinement options. The fourth strategy operates on the HotSpot principle implemented on plan modifications: the Fewer Modification-based HotSpots (fmfh) function summarizes for all refinement-operators that are proposed for a plan the HotSpot values of the corresponding flaws. It then prefers those plans for which the ratio of plan modifications to accumulated HotSpot values is less. By doing so, this search schema focuses on plans that are expected to have less interfering refinement options.

Finally, since our framework's representation of the SHOP strategy solely relies on modification selection, a depth first plan selection is used for constructing a simple hierarchical ordered planner.

It is furthermore important to mention, that our strategy functions can be combined into selection cascades (denoted by the symbol +) in which succeeding components decide on those cases for which the result of the preceding ones is a tie. We have built five combinations from the components above, which can be regarded as representatives for completely different approaches to plan development. Please note that the resulting strategies are general domain-independent planning strategies, which are not tailored to the application of domain model reduction by pre-processing in any way.

We ran our experiments on two distinguished planning domains. The Satellite domain is an established benchmark in the field of non-hierarchical planning. It is inspired by the problem of managing scientific stellar observations by earth-orbiting instrument platforms. Our hybrid version regards the original primitive operators as implementations of abstract observation tasks, which results in a domain model with 3 abstract and 5 primitive tasks, related by 8 methods. The second domain is known as UM-Translog, a transportation and logistics model originally written for HTN planning systems. We adopted its type and decomposition structure to our hybrid approach which yielded a deep expansion hierarchy in 51 methods for decomposing 21 abstract tasks into 48 different primitive ones. We have chosen the above domain models because of the problem characteristics they induce: Satellite problems typically become difficult when modelling a repetition of observations, which means that a small number of methods is used multiple times in different contexts of a plan. The evaluated scenarios are thus defined as observations on one or two satellites. UM-Translog problems, on the other hand,
typically differ in terms of the decomposition structure, because specific transportation goods are treated differently, e.g., toxic liquids in trains require completely different methods than transporting regular packages in trucks. We consequently conducted our experiments on qualitatively different problems by specifying various transportation means and goods.

### Table 5: Results for the Satellite domain.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Mod. Sel.</th>
<th>Plan Sel.</th>
<th>PANDA</th>
<th>PANDA+LT</th>
<th>PANDA+L-ML</th>
</tr>
</thead>
<tbody>
<tr>
<td>1obs-1sat</td>
<td>lcf</td>
<td>lcf</td>
<td>35</td>
<td>41</td>
<td>37</td>
</tr>
<tr>
<td>1mode</td>
<td>lcf</td>
<td>lcf</td>
<td>67</td>
<td>72</td>
<td>87</td>
</tr>
<tr>
<td>2obs-1sat</td>
<td>lcf</td>
<td>lcf</td>
<td>602</td>
<td>788</td>
<td>539</td>
</tr>
<tr>
<td>1mode</td>
<td>lcf</td>
<td>lcf</td>
<td>964</td>
<td>1611</td>
<td>903</td>
</tr>
<tr>
<td>2obs-2sat</td>
<td>lcf</td>
<td>lcf</td>
<td>1408</td>
<td>1099</td>
<td>1216</td>
</tr>
<tr>
<td>1mode</td>
<td>lcf</td>
<td>lcf</td>
<td>–</td>
<td>–</td>
<td>2824</td>
</tr>
<tr>
<td>SHOP Strategy</td>
<td>–</td>
<td>–</td>
<td>257</td>
<td>257</td>
<td>264</td>
</tr>
<tr>
<td>SHOP Strategy</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>1406</td>
</tr>
</tbody>
</table>

Tables 4 and 5 show the runtime behavior of our system in terms of the size of the average search space and CPU time consumption for the problems in the UM-Translog and Satellite domains, respectively. The size of the search space is measured in the number of plans visited for obtaining the first solution. The CPU time denotes the total running time of the planning system in seconds, including the pre-processing phase. Dashes indicate that the plan generation process did not find a solution within the allowed maximum number of 5,000 plans and 9,000 seconds and has therefore been canceled. The column PANDA refers to the reference system behavior, the PANDA+LT to the version that performs a pre-processing phase.

Reviewing the overall result, it is quite obvious that the landmark pre-processing pays off in all strategy configurations and problems. It does so in terms of search space size as well as in terms of runtime. The only exceptions are two configurations in the easiest satellite problem in which the search space cannot be reduced but a neglectable overhead is introduced by pre-processing. Furthermore, the problem concerning air freight is the only one on which landmarking has a measurable negative effect (decrease of performance of 18%).

The average performance improvement over all strategies and over all problems in the UM-Translog domain is about 40% as is documented in Table 4. The biggest gain is achieved in the transportation tasks that involve special goods and transportation means, e.g., the transport of auto-mobiles, frozen goods, and mail via train saves between 53% and 71%. In general, the flexible strategies profit from the landmark technique, which gives further evidence to the previously obtained results that opportunistic planning strategies are very powerful general-purpose procedures and in addition offer potential to be improved by pre-processing methods. The SHOP-style strategy cannot take that much advantage of the reduced domain model, because it cannot adapt its focus on the reduced method alternatives.

The Satellite domain does not benefit significantly from the landmark technique due to its shallow decomposition hierarchy. We are, however, able to solve problems for which the participating strategies do not find solutions within the given resource bounds otherwise.

### 6 Conclusion

We have presented an effective landmark technique for hierarchical planning. It analyzes the planning problem by pre-processing the underlying domain and prunes those regions of the search space where a solution cannot be found. Our experiments on a number of representative hierarchical planning domains and problems give reliable evidence for the practical relevance of our approach. The performance gain went up to about 70% for problems with a deep hierarchy of tasks. Our technique is domain- and strategy-independent and can help any hierarchical planner to improve its performance.

### ACKNOWLEDGEMENTS

This work was partly supported by the Transregional Collaborative Research Centre SFB/TRR 62 funded by the German Research Foundation (DFG). We thank our colleague Pascal Bercher and the ECAI reviewers for valuable comments.

### REFERENCES