Exploiting Landmarks for Hybrid Planning [1]

Mohamed Elkawkagy and **Pascal Bercher**
and **Bernd Schattenberg** and **Susanne Biundo** [2]

Abstract. Very recently, the well-known concept of landmarks has been adapted from the classical planning setting to hierarchical planning. It was shown how a pre-processing step that extracts local landmarks from a planning domain and problem description can be used in order to prune the search space that is to be explored before the actual search is performed. This pruning technique eliminates all branches of the task decomposition tree, for which can be proven that they will never lead to a solution. In this paper, we investigate this technique in more detail and extend it by introducing search strategies which use these local landmarks in order to guide the planning process more effectively towards a solution. Our empirical evaluation shows that the pre-processing step dramatically improves performance because dead ends can be detected much earlier than without pruning and that our search strategies using the local landmarks outperform many other possible search strategies.

1 Introduction

In recent years, the exploitation of knowledge gained by pre-processing a planning domain and/or problem description has proven to be an effective means to reduce planning effort. Various pre-processing procedures, like effect relaxation [3, 13], abstractions [11], and landmarks [20] have been proposed for classical planning, where they serve to compute strong search heuristics. However, pre-processing techniques can also be used to perform some pruning of the search space before the actual search is performed. Very recently, different techniques have been introduced which restrict the domain and problem description of an Hierarchical Task Network (HTN) problem to a smaller subset, since some parts of the domain description might be irrelevant for the given problem at hand [5, 10]. In this paper, we investigate our previously introduced landmark technique [5] in more detail, which uses local landmarks to prune the search space that is to be explored before the actual search is performed. We further investigate, how search strategies can take advantage of these extracted local landmarks.

While the use of landmark tasks is a novelty in hierarchical planning, it is 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 by Porteous et al. [20] and further developed by Hoffmann et al. [14] and Zhu and Givan [27], 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 [25] and extended the landmark concept to so-called disjunctive landmarks [9, 19]. 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 landmarks resulted in the notion of so-called action landmarks [16, 26]. An action is an action landmark if it occurs in every solution plan. Most of the recent landmark approaches use landmark information to compute heuristic functions for a forward searching planner [16, 21] and investigate their relations to critical-path-, relaxation-, and abstraction-heuristics [12]. In summary, it turned out that the use of landmark information can significantly improve the performance of classical state-based planners.

In hierarchical planning, *landmarks* are mandatory abstract or primitive tasks, i.e. tasks that have to be performed by any solution plan. *Local landmarks* are abstract or primitive tasks that are mandatory, given their parent task is mandatory (where a parent task is the abstract task that introduced the local landmark by decomposition). That is, a local landmark is also a landmark if its parent is one, too. For an initial task network that states a current planning problem, a pre-processing procedure computes the corresponding local 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 local landmarks. Using the precondition and effects of primitive tasks, one can perform a relaxed reachability test [8]. A failure indicates that the method which introduced the primitive task is no longer eligible. If the tested primitive task was a local landmark, we can even get further: its parent abstract task can never be decomposed into a solution because one of its local landmarks cannot be achieved. Hence, this abstract (parent) task can also be pruned without the need of inspecting the primitive tasks in the other methods for this abstract task. Being able to prune useless regions of the search space this way, a hierarchical planner performs significantly better than it does without exploiting the local landmark information.

Before introducing the local landmark extraction procedure for hierarchical planning in Section 3, we will briefly review HTN planning in general and our underlying framework and planning procedure in particular (Section 2). Afterwards, Section 4 shows how the information about local landmarks can be used during planning. It presents experimental results from

---

[1] Revised 4.12.2010.

[2] Institute of Artificial Intelligence, Ulm University, D-89069 Ulm, Germany, email: *forename.surname*@uni-ulm.de

a set of benchmark problems of the *UM-Translog* [1] and *Satellite* domains, which give evidence for the considerable performance increase gained by pre-processing the planning problem to prune unnecessary parts and by the use of the novel search strategies using the local landmarks. The paper ends with possible extensions to our approach (Section 5) and with some concluding remarks in Section 6.

2 Formal Framework

Hierarchical Task Network (HTN) planning is based on the concepts of tasks and methods [6]. 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. A planning problem consists of finding a decomposition of the initial task network, using the tasks and methods provided by the domain model. Thus, the planning problem is solved by incrementally decomposing the abstract tasks in the initial task network until it contains only primitive tasks and is consistent w.r.t. 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.

Our approach [2] relies on a *hybrid* planning framework [7, 15], which combines HTN planning 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 can 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 (i.e., a method's task network) 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 [4, 7].

In our framework, a task network or partial plan $P = \langle S, \prec, V, C\rangle$ consists of a set of plan steps S, i.e., (partially) instantiated task schemata, augmented with a unique label to differentiate between multiple occurrences of the same task. We denote by $\mathit{Tasks}(P)$ the set of (partially) instantiated task schemata in the plan steps S of P, i.e., S without labels. It also contains a set of ordering constraints $\prec$ that impose a partial order on the plan steps, a set of variable constraints V, and a set C of causal links. Variable constraints are (in-) equations between variables or between variables and constants. A causal link $s_i \rightarrow_\varphi s_j$ indicates that the precondition φ of plan step s_j is an effect of plan step s_i and is *supported* this way. A domain model $D = \langle T, M\rangle$ includes a set of task schemata and a set of decomposition methods. A task schema $t(\overline{\tau}) = \langle \mathrm{prec}(t(\overline{\tau})), \mathrm{add}(t(\overline{\tau})), \mathrm{del}(t(\overline{\tau}))\rangle$ specifies the preconditions as well as the positive and negative effects of a task. Preconditions and effects are sets of literals and $\overline{\tau} = \tau_1 \ldots \tau_n$ are the task parameters. In the hybrid setting, both primitive and abstract tasks show preconditions and effects. This enables the use of POCL planning operations even on abstract levels. However, in this paper we restrict our language to pure HTN; preconditions and effects are thus omitted for abstract tasks. 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 other representatives of HTN-style planning. A planning problem $\Pi = \langle D, s_{init}, P_{init}\rangle$ includes a domain model D, an initial state s_{init}, and P_{init}, which represents an initial partial plan. Please note, that in our *hybrid* planning framework, one can also specify a goal state. However, since we restrict ourselves in this paper to pure HTN planning, the goal state is omitted.

Based on these strictly declarative specifications of planning domains and problems, hybrid and HTN planning is performed by refining an initial partial plan P_{init} of Π stepwise until a partial plan $P = \langle S, \prec, V, C\rangle$ is obtained that satisfies the following solution criteria:

1. P is a refinement of P_{init}, i.e., it is a successor of the initial plan in the induced search space (cf. Definition 1),
2. each precondition of a plan step in P is supported by a causal link in C,
3. the ordering and variable constraints are consistent, i.e., the ordering does not induce cycles on the plan steps and the (in-) equations of variable constraints are free of contradiction,
4. none of the causal links in C is threatened, i.e., for each causal link $s_i \rightarrow_\varphi s_j$ 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, and
5. all plan steps in S are primitive tasks.

Please note that we encode the initial state description via the effects of an artificial primitive task, as it is usually done in POCL planning. In doing so, the second criterion guarantees that the solution is executable in the initial state.

Before we present our planning algorithm in more detail, we define the search space induced by the HTN planning problem Π. 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 ordering and variable constraints. We call such a refinement step a *plan modification*.

Definition 1 (Induced Search Space)**.** *Let $\mathcal{P}_\Pi = \langle \mathcal{V}, \mathcal{E}\rangle$ be the directed acyclic graph which represents the (possibly infinite) search space induced by a planning problem $\Pi = \langle D, s_{init}, P_{init}\rangle$. Then, the set of vertexes $\mathcal{V}$ is the set of plans in the induced search space and the set of edges $\mathcal{E}$ corresponds to the set of plan used modifications. By abuse of notation, we write $P \in \mathcal{P}_\Pi$ to state $P \in \mathcal{V}$. The root of $\mathcal{P}_\Pi$ is the initial plan of Π; thus, $P_{init} \in \mathcal{P}_\Pi$. The direct successors of a plan $P \in \mathcal{P}_\Pi$ are all Plans P', such that P' resulted from P by applying a plan modification m to P. Then, $m \in \mathcal{E}$.*

Now, we present our planning algorithm (Algorithm 1) which takes the initial plan of the planning problem Π as an input and refines it stepwise until a solution is found. Our algorithm performs an informed search, guided by so-called search strategies, in the search space induced by the HTN planning problem Π (cf. Definition 1).

The fringe of the algorithm is a plan sequence $\langle P_1 \ldots P_n\rangle$ ordered by the used search strategy. It contains all non-visited

Algorithm 1: Planning Algorithm

Input : The sequence $\texttt{Fringe} = \langle P_{init} \rangle$.
Output: A solution or $\texttt{fail}$.

1 **while** $\mathit{Fringe} = \langle P_1 \dots P_n \rangle \neq \varepsilon$ **do**
2 $\quad F \leftarrow f^{\text{FlawDet}}(P_1)$
3 $\quad$ **if** $F = \emptyset$ **then return** P_1
4 $\quad \langle m_1 \dots m_{n'} \rangle \leftarrow f^{\text{ModOrd}}(\bigcup_{\mathtt{f} \in F} f^{\text{ModGen}}(\mathtt{f}))$
5 $\quad \mathtt{succ} \leftarrow \langle \mathtt{app}(m_1, P_1) \dots \mathtt{app}(m_{n'}, P_1) \rangle$
6 $\quad \mathtt{Fringe} \leftarrow f^{\text{PlanOrd}}(\mathtt{succ} \circ \langle P_2 \dots P_n \rangle)$
7 **return** fail

plans that are direct successors of visited non-solution plans. According to the used search strategy, a plan P_i leads more quickly to a solution than plans P_j for $j > i$. The current plan under consideration is always the first plan of the fringe. The planning algorithm loops as long as no solution is found and there are still plans to refine (line 1). Hence, in line 2, the flaw detection function f^{FlawDet} calculates all flaws of the current plan. A flaw is a plan component that violates a solution criterion. For instance, in the HTN planning setting, (the occurrence of) an abstract task is a flaw. If no flaws can be found, the plan is a solution and is returned (line 3). In line 4, all plan modifications are calculated by the modification generating function f^{ModGen}, which address all found flaws. Afterwards, the modification ordering function f^{ModOrd} orders all these modifications according to a given strategy. Finally, this fringe is updated in two steps: First, the plans resulting from applying the modifications are calculated (line 5) to be inserted at the head of the fringe in line 6. Afterwards, the plan ordering function f^{PlanOrd} takes the updated fringe and orders it according to its strategy. This step can also be used in order to discard some plans (i.e., to delete some plans permanently from the fringe). This is useful for plans which contain unresolvable flaws like an inconsistent ordering of tasks. If the fringe becomes empty, no solution exists and $\texttt{fail}$ gets returned.

In contrast to other systems, which *implicitly* define their search strategy by their search procedure, our approach – implemented in the planning environment PANDA [22] (**P**lanning and **A**cting in a **N**etwork **D**ecomposition **A**rchitecture) – *explicitly* defines the search strategy: It is the result of the combination of the used modification and plan ordering functions. Let us take a look at a simple example strategy for clarification: To perform a depth first strategy, the plan ordering strategy has to be the identity (i.e., $f^{\text{PlanOrd}}(\overline{P}) = \overline{P}$ for any plan sequence $\overline{P}$), whereas the modification ordering strategy f^{ModOrd} can be arbitrary (but decides, which branches to visit first). Thus, the plan ordering strategy is used to prioritize the plans; several strategies can be concatenated for tie-braking. The plan ordering strategy uses also its input sequence for tie braking: If two plans are still invariant after application of the plan ordering function, the order given in the input is used.

Many different plan ordering strategies[3] have been described and evaluated in our previous work [22, 23, 24]. In this work, we will only sketch those used in the experiments (cf. Subsection 4.1).

[3] In previous work, we called the plan and modification ordering functions plan and modification *selection* functions, respectively.

3 Local Landmark Extraction

For a given hierarchical planning problem $\Pi = \langle D, s_{init}, P_{init} \rangle$, *global landmarks*[4] are the tasks that occur in every sequence of decompositions leading from the initial task network P_{init} to a solution plan. However, we do not calculate (global) landmarks, but – what we call – *local* landmarks. Local landmarks are landmarks with respect to a given abstract task. We will define them more formally in this section. The local landmark extraction is performed using a so-called *task decomposition tree* (TDT) of Π. Figure 1 depicts such a tree schematically. The TDT of Π 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 somewhere in the TDT. Each level of a TDT consists of two parts: a task and a method level. Method nodes are AND nodes, because their children are the tasks that occur in the partial plan of the respective method, all of which have to be performed in order to apply the corresponding method. Task nodes, on the other side, are OR nodes, because their children are the methods that can be used to decompose the respective task. To avoid loops, each abstract task is decomposed only once in the TDT; hence, all but one identical and fully grounded abstract tasks become leaf nodes in the TDT. Other leaf nodes are the primitive tasks. A TDT is built by forward chaining from the (grounded) abstract tasks in the initial task network until all nodes of the fringe are leaf nodes. The root node on level 0 is an artificial method node that represents the initial partial plan P_{init}.

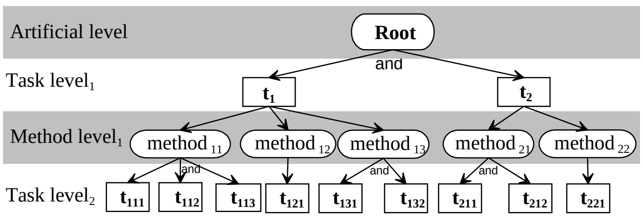

Figure 1: A schematic task decomposition tree.

Before we can formally define landmarks and local landmarks, we first need to define plan and solution sequences, respectively.

Definition 2 (Plan and Solution Sequences)**.** *For a planning problem Π and a given plan $P \in \mathcal{P}_\Pi$, let $\mathcal{S}eq_\Pi(P)$ be the set of all plan sequences in $\mathcal{P}_\Pi$ rooted in that plan, i.e., $\mathcal{S}eq_\Pi(P) = \{\langle P_1 \dots P_n \rangle | P = P_1, P_1 \in \mathcal{P}_\Pi$ and P_{i+1} is a direct successor of $P_i \in \mathcal{P}_\Pi$ for all $1 \leq i < n\}$.*

The set of all solution sequences rooted in P is then $\mathcal{S}ol\mathcal{S}eq_\Pi(P) = \{\langle P_1 \dots P_n \rangle \in \mathcal{S}eq_\Pi(P) | P_n$ solution of $\Pi, n \geq 1\} \subseteq \mathcal{S}eq_\Pi(P)$.

Definition 3 (Landmark)**.** *A landmark is a grounded (i.e., fully instantiated) task that occurs in every sequence of decompositions leading from the initial task network to a solution.*

[4] In the following, we will only call them *landmarks*.

That is, the task t is called a landmark of $\Pi = \langle D, s_{init}, P_{init}\rangle$, if for every sequence $\langle P_1 \dots P_n\rangle \in \mathcal{S}ol\mathcal{S}eq_\Pi(P_{init})$ there is an $1 \le i \le n$, such that $t \in Tasks(P_i)$.

Whereas a landmark has to occur in every decomposition sequence of a solution (which is rooted in the initial plan), a *local* landmark only has to occur in each solution sequence rooted in a plan containing a specific task t.

Definition 4 (Local Landmark of an Abstract Task)**.** *For a given grounded abstract task t, let $\mathcal{P}_\Pi(t)$ be the set of all plans in $\mathcal{P}_\Pi$ containing t, i.e., $\mathcal{P}_\Pi(t) = \{P \in \mathcal{P}_\Pi | t \in Tasks(P)\}$.*

We call the grounded task t' a local landmark of t, if for all $P \in \mathcal{P}_\Pi(t)$ holds, that for all sequences $\langle P_1 \dots P_n\rangle \in \mathcal{S}ol\mathcal{S}eq_\Pi(P)$ there is a P_i with $i > 1$ such that $t' \in Tasks(P_i)$.

We use the next definition to calculate all tasks that occur in all available methods for the same abstract task.

Definition 5 (Common Task Set Operator $\widehat{\cap}$)**.** *Let t be an abstract task in the TDT and $m_i = \langle t, \langle S_i, \prec_i, V_i, C_i\rangle\rangle$ and $m_j = \langle t, \langle S_j, \prec_j, V_j, C_j\rangle\rangle$ two of its methods in the TDT. That is, both t and its methods are fully grounded. Then, the Common Task Set Operator $\widehat{\cap}$ of m_i and m_j is defined as*

$$m_i \,\widehat{\cap}\, m_j = Tasks(S_i) \cap Tasks(S_j)$$

Using this definition, we can calculate the mandatory tasks of an abstract task t, $M(t)$, by intersecting all available methods. Obviously, the tasks contained in $M(t)$ are local landmarks of t, because these tasks are contained in all solution sequences that are rooted in a plan containing t. It is also notable, that all tasks in $M(t)$ are local landmarks of t if t is not contained in any solution sequence[5] (i.e., if for all $\langle P_1 \dots P_n\rangle \in \mathcal{S}ol\mathcal{S}eq_\Pi(P_{init})$ holds, that there is no $P_i, 1 \le i \le n$ such that $t \in Tasks(P_j)$).

However, not all local landmarks of an abstract task can be detected that way because not all local landmarks have to be in such an intersection.

We would also like to emphasize, that local landmarks are in general no landmarks. This is obvious, because one can calculate the local landmarks of an abstract task which is not contained in all valid decompositions (or even in *any* valid decomposition) of the initial plan.

Based on the definition of the common task set operator, we will now define the remaining task set operator which calculates the set of tasks in which two (grounded) methods differ.

Definition 6 (Remaining Task Set Operator $\widehat{\cup}$)**.** *Let t be an abstract task in the TDT and $m_i = \langle t, \langle S_i, \prec_i, V_i, C_i\rangle\rangle$ and $m_j = \langle t, \langle S_j, \prec_j, V_j, C_j\rangle\rangle$ two of its methods in the TDT. Then, the Remaining Task Set Operator $\widehat{\cup}$ of m_i and m_j is defined as*

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

Analogously to the mandatory tasks $M(t)$ of an abstract task t, we can define its optional tasks $O(t)$, by applying the remaining task set operator to all methods of t in the TDT. $M(t)$ and $O(t)$ can be regarded as a partition of the methods of t in the TDT, i.e., it holds:

$$\{Tasks(P) \mid \textit{there is a method } m = \langle t, P\rangle \textit{ in the TDT}\} = \{M(t) \cup o \mid o \in O(t), \text{if } O(t) \neq \emptyset \text{ or } o = \emptyset, \text{ else}\}.$$

Table 1: A schematic landmark table, showing in each line an ground instance of an abstract task, its mandatory abstract tasks and its optional tasks.

abstr. Tasks	Mandatory	Optional
t_1	$M(t_1)$	$O(t_1)$
t_2	$M(t_2)$	$O(t_2)$
$\vdots$	$\vdots$	$\vdots$

The landmark extraction algorithm (Algorithm 2) calculates for each abstract task occurring in the TDT these two sets and stores it into a so-called landmark table. Table 1 shows such a landmark table schematically. The algorithm takes a TDT[6], which is computed before the algorithm is called, as input and returns a landmark table after its termination.

Intuitively, the algorithm simply tests all primitive tasks for relaxed reachability, starting with the initial plan (the root of the TDT) and proceeding level by level of the TDT. If a task can be proven unreachable, the method introducing this task is pruned from the TDT and all its sub-nodes (and so forth). After all infeasible methods of an abstract task t have been pruned from the TDT, this task, its mandatory tasks, and its optional tasks are stored into the landmark table.

Now we take a look how this is achieved by our algorithm: First, the landmark table and a set for backward propagation get initialized (line 1). Afterwards, each abstract task, which is not yet stored into the landmark table is considered level by level of the TDT (line 2 to 4). For the current abstract task at hand, line 6 to 8 calculate the mandatory and the optional tasks in the yet unpruned TDT according to Definition 5 and 6. After the tasks introduced by decomposition of t have been partitioned into $M(t)$ and $O(t)$, these sets are analyzed for infeasibility. This test is performed by a relaxed reachability analysis. First, we study the primitive tasks of $M(t)$ (line 9). If such a task can be proven to be infeasible, *all methods* of t become obsolete and can hence be pruned from the TDT[7] (line 10 and 12). After this test, each optional task set is tested for reachability. If an infeasible task can be found, only this specific method gets pruned from the TDT (line 13 to 17). If something was pruned, the loop (line 5 to 18) enters another cycle, because the set $M(t)$ might have grown. If no more pruning is possible, the mandatory and optional task sets for t are stored into the landmark table in line 19. When storing an entry in line 21, it is checked whether the stored abstract task is feasible or not (an abstract task is infeasible if it does not have any methods left, i.e., if $M(t)$ and $O(t)$ are empty). If some abstract task could actually be proven infeasible, it is stored for backward propagation, because again all methods containing this abstract task can be pruned from the TDT and from the landmark table. Finally, if all abstract tasks are checked, the backward propagation procedure is called with the current landmark table and TDT in line 22.

[5] In fact, *all* grounded tasks t' are local landmarks of t if t is not contained in any solution sequence.

[6] We use the indefinite article, because only the task decomposition *graph* is unique, whereas the resulting task decomposition *tree* depends on the chosen order in which tasks get decomposed.

[7] In the presented algorithm, the optional task sets would still be tested, which is obviously not necessary. However, for the sake of readability, we did not handle this case in the algorithm.

Procedure propagate takes as input the already filled landmark table, the possibly pruned TDT and a set `infeasible` of abstract tasks which have been proved infeasible due to no remaining methods in the TDT. It works tail-recursively and returns the final landmark table as soon as no propagation is possible (line 1). To this end, it first takes and removes some arbitrary task t' from the set `infeasible`. Because this abstract task was proven infeasible, its landmark table entry can be removed (line 3); also, all methods containing this task can be pruned from the TDT (line 6). To calculate the methods that can possibly be pruned, all parent tasks of t' are identified (line 4). Then, for all these parents (line 5), the respective methods are removed in line 6. Because methods were removed, the mandatory and the optional task sets could have changed again. Hence, they are recalculated in line 7 to 9. Next, the the old landmark table entry of the current parent t is removed and replaced by the new one (line 10). In line 12, it is tested again, whether the new landmark table entry corresponds to an infeasible abstract task. If so, it is put into the set `infeasible` for later processing. The procedure is then called with the modified parameters in line 13.

Without a formal proof, we want to mention that Algorithm 2 (i.e., the initial landmark table calculation as well as the backward propagation) always terminates. For the first part of the algorithm, this is easy to see because both loop conditions (line 2 and 3) cannot be modified within the loops. For the second part, i.e., the propagate procedure, we have to show that the set `infeasible` becomes empty eventually. This is the case because each task gets inserted at most once and will be removed at some point.

After the algorithm terminated, the TDT does not have to be considered anymore. All necessary information is encoded in the landmark table.

As we have already pointed out, we only calculate *local* landmarks. That is, given a landmark table entry $(t,M(t),O(t))$, $M(t)$ contains some of the local landmarks of t, which, in general, don't have to be actual landmarks because t was not proven to be a landmark. However, all local landmarks of the abstract tasks in the task level 1 of the TDT are also actual landmarks, because all tasks in the task level 1 are those contained in the initial plan and hence landmarks. Thus, if we restrict our local landmark extraction procedure to calculate only the local landmarks of tasks which are (local) landmarks by themselves, all tasks in the $M(t)$ sets in the local landmark table returned are actual landmarks, too. These landmarks are, however, of limited use because every decomposition contains them anyway. Thus, a "guiding" towards these landmarks as done in classical planning does not bring any benefit.

Algorithm 2: Local landmark Extraction Algorithm

Input : A task decomposition tree TDT.
Output: The filled landmark table LT.

1 LT $\leftarrow \emptyset$, `infeasible` $\leftarrow \emptyset$
2 **for** $i \leftarrow 1$ **to** *TDT.maxDepth()* **do**
3   **foreach** *abstract task* t *in level* i *of TDT* **do**
4   **if** *LT contains an entry for* t **then continue**
5   **repeat**
6   Let M be the methods of t in the TDT.
7       $M(t) \leftarrow \widehat{\bigcap}_{m \in M} m$
8       $O(t) \leftarrow \widehat{\bigcup}_{m \in M} m$
9   **foreach** *primitive task* $t' \in M(t)$ **do**
10   **if** t' *can be proven infeasible* **then**
11   remove all $m \in M$ from the TDT, including all sub-nodes.
12   **break**
13   **foreach** *optional task set* $o \in O(t)$ **do**
14   **foreach** *primitive task* $t' \in o$ **do**
15   **if** t' *can be proven infeasible* **then**
16   remove the method $m = \langle t, P \rangle$, with $\mathit{Tasks}(P) = M(t) \cup o$ from the TDT, including all sub-nodes.
17   **continue**
18   **until** *no method was removed from TDT*
19   LT $\leftarrow$ LT $\cup \{(t, M(t), O(t))\}$
20   **if** $M(t) = O(t) = \emptyset$ **then**
21   `infeasible` $\leftarrow$ `infeasible` $\cup \{t\}$
22 **return** ***propagate****(LT, TDT, infeasible)*

Example

In order to illustrate our landmark extraction technique, let us consider a simple example from the UM-Translog domain [1]. 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. Figure 2 shows a part of the task decomposition tree of this example.

The local landmark extraction algorithm detects that the first level in the TDT contains only one abstract task t = *transport*(P_1,L_1,L_3) and that there is only one method, *Pi_ca_de*, that can decompose the task into a partial plan, which contains the subtasks *pickup*(P_1), *carry*(P_1, L_1, L_3), and *deliver*(P_1). Hence, $M(t)$ becomes $\{$*pickup*(P_1),*carry*(P_1, L_1, L_3),*deliver*$(P_1)\}$ and $O(t) = \emptyset$. The current abstract task and the sets $M(t)$ and $O(t)$ are entered as the first row of the landmark table as shown in Table 2.

The landmark extraction algorithm then takes the (unchanged) TDT to investigate the next tree level. The abstract tasks to be inspected on this level are *pickup*(P_1) *and carry*(P_1, L_1, L_3). The primitive task *deliver*(P_1) is tested and considered executable. Suppose, the task t = *pickup*(P_1) is chosen first in line 3 of Algorithm 2. As shown in Figure 2, the TDT accounts for three methods to decompose this task: *Pickup_hazardous*, *Pickup_normal*, and *Pickup_valuable*. Therefore, we get $M(t) = \{$*collect_fees*$(P_1)\}$, and $O(t) = \{\{$*have_permit*$(P_1)\}, \emptyset, \{$*collect_insurance*$(P_1)\}\}$. At this point, the relaxed reachability analysis is performed. First, *collect_fees*(P_1) is being tested, because it is contained in the intersection $M(t)$. Suppose, this task can not be proven to be infeasible. Then, each primitive task in each set $r \in O(t)$ has to be checked. Assume the primitive task *have_permit*(P_1) is feasible, whereas *collect_insurance*(P_1) is not. The method *Pickup_valuable* is therefore deleted from the TDT. After an additional iteration in which $M(t)$ and $O(t)$ get recalculated, the current abstract task t = *pickup*(P_1), the set $M(t)$, and the modified set $O(t)$ are added to the landmark table as depicted in the second line of Table 2.

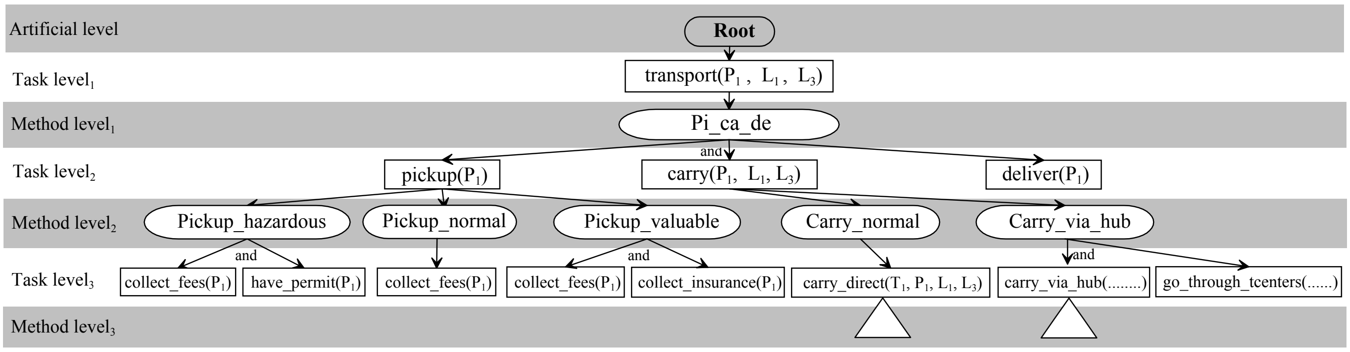

Figure 2: Part of the TDT for the transportation task

Table 2: Example landmark table containing the first three entries for the transportation task illustrated in Figure 2. The sets in the right most column are indexed by the method's name that contains its tasks.

abstr. Task	Mandatory	Optional
transport(P_1, L_1, L_3)	{pickup(P_1),carry(P_1, L_1, L_3),deliver(P_1)}	$\emptyset$
pickup(P_1)	{collect_fees(P_1)}	$\{\{\text{have\_permit}(P_1)\}_{\text{Pickup\_hazardous}}, \emptyset_{\text{Pickup\_normal}}\}$
carry(P_1, L_1, L_3)	{carry_direct(T_1, P_1, L_1, L_3)}	$\emptyset$

Procedure propagate(LT,TDT,`infeasible`)

Input : A landmark table LT, a task decomposition tree TDT, possibly pruned, and a set of abstract tasks `infeasible`, which have been proved infeasible.

Output: the updated landmark table LT, in which methods are pruned that contain infeasible abstract tasks.

1 **if** *infeasible* $= \emptyset$ **then return** *LT*
2 `infeasible` $\leftarrow$ `infeasible` $\setminus \{t'\}$, where $t' \in$ `infeasible`.
3 LT $\leftarrow$ LT $\setminus\{(t', M(t'), O(t')) \in \text{LT}\}$
4 `parents` $\leftarrow \{t | (t, M(t), O(t)) \in LT, t' \in M(t) \cup \bigcup_{o \in O(t)} o\}$
5 **foreach** $t \in$ *parents* **do**
6   Remove all methods from the TDT, that contain t' in its plan, i.e., all $m = \langle t, P \rangle$ with $t' \in Tasks(P)$.
7   Let M be the methods of t in the TDT.
8     $M(t) \leftarrow \widehat{\bigcap}_{m \in M} m$
9     $O(t) \leftarrow \widehat{\bigcup}_{m \in M} m$
10   LT $\leftarrow$ (LT$\setminus\{(t, M'(t), O'(t)) \in \text{LT}\}) \cup \{(t, M(t), O(t))\}$
11   **if** $M(t) = O(t) = \emptyset$ **then**
12   `infeasible` $\leftarrow$ `infeasible` $\cup \{t\}$
13 **return** *propagate(LT,TDT,infeasible)*

In the second iteration (line 3) the abstract task t = *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 $M(t) = \emptyset$ and $O(t) = \{\{$*carry_direct*$(T_1, P_1, L_1, L_3)\}$, $\{$*carry_via_hub*$(\dots)$, *go_through_tcenters*$(\dots)\}\}$. Suppose the primitive task *go_through_tcenters*$(\dots)$ is infeasible. The sub tree with root *carry_via_hub*$(\dots)$ has then to be removed from the TDT. Because the TDT was changed, the iteration (line 5 to 18) enters another cycle. Because there is now only one method left, $M(t)$ now contains all tasks of this remaining method. Hence, the current abstract task t = *carry*(P_1, L_1, L_3) together with the modified $M(t)$ and $O(t)$ are added to the landmark table as depicted in the last line of Table 2.

4 Landmark Exploitation

The information about landmarks can be exploited in two ways: The first is to deduce heuristic guidance from the knowledge about which tasks have to be decomposed on refinement paths that lead towards a solution. But before we investigate into this matter, we will present a second way of landmark exploitation, namely the reduction of domain models or, more precisely, the transformation of a universal domain model into one that includes problem-specific pruning information.

4.1 Domain Model Reduction

During the construction of the landmark table, the feasibility check and the consecutive propagation of its result into the abstract task level lead to a pruning of the task decomposition tree. The result of this analysis implies that if a method is removed from the TDT during the operation of our landmark extraction algorithm, it can be safely ignored as a refinement option during plan generation.

We consequently supply our refinement generating module with the landmark table for the current planning problem and verify for every incoming abstract task flaw, which of the methods specified in the domain model are reasonably applicable. However, the landmark table is built from grounded tasks, while the plan generation procedure operates on lifted instances for which the final grounding is yet to be computed. We therefore calculate all groundings of the abstract task at hand that are consistent with the current variable constraints and match these grounded tasks t with the entries in the landmark table. The union of the (lifted) method schemata that constitute the (grounded) instances in the optional task sets

$O(t)$ is the set of method schemata that we consider for application to the currently flawed abstract task. Obviously, the earlier a task is addressed in the planning process, the less variable constraints are typically introduced in the partial plan, the more task groundings are implied by the lifted instance, and consequently the less probable is one of its methods pruned by this technique.

In order to quantify the effect of this landmark exploitation technique, we have performed several benchmark tests on the UM-Translog and the Satellite domain with various different search strategies. Table 3 shows the domain model sizes of the UM-Translog domain after our pruning process. For the Satellite domain, our pruning technique did not change the domain model. According to this table, the pruning technique achieves a reduction of the number of abstract task instances that ranges between 33% and 43%, while the reduction of the number of inapplicable methods per instance varied between 27% and 41%.

Table 3: This table shows the remaining sizes of the domain model after our reduction for typical problems from the UM-Translog domain. On all problems that are grouped together the same reduction was achieved.

Problem	**abstr. Tasks** (of 21)	**Methods** (of 51)
Regular Truck Problems Hopper Truck, Auto Truck, Regular Truck (a)[8] Regular Truck (b) Regular Truck (c) Regular Truck (d)	12 (57%)	30 (59%)
Various Truck Type Problems Flatbed Truck, Armored R-Truck	12 (57%)	32 (63%)
Traincar Problems Auto Traincar (a), Auto Traincar (b), Mail Traincar, Refrigerated Regular Traincar	14 (67%)	32 (63%)
Airplane Problem Airplane	14 (67%)	37 (73%)

In theory, it is quite intuitive that a reduced domain model leads to an improved performance of the planning system. It is however hard to predict the actual effect the pruning information on the grounded instance level has on the lifted computations, in particular taking into account that the landmark table typically contains a number of "distracting" local landmarks that are located on non-solution paths. In order to quantify the practical performance gained by the hierarchical landmark technique, we therefore conducted a series of experiments in the PANDA planning environment [22]. The planning strategies we used are representatives from the rich portfolio provided by PANDA, which has been discussed in previous work [22, 23, 24]. We briefly review the ones on which we based our experiments.

As was already mentioned in Section 2, the search strategy is encoded by the combination of the modification and plan ordering functions. We distinguish ordering principles that are based on a prioritization of certain flaw or modification classes and strategies that opportunistically choose from the presented set. We call the latter ones *flexible strategies*.

Representatives for *inflexible* strategies are the classical HTN strategy patterns that try to balance task expansion with respect to other plan refinements. A classical HTN strategy is the preference of expansion the way it has been realized in the UMCP system [6]: Plans are developed into completely primitive task networks in which causal interactions are dealt with afterwards. This technique presumably benefits most from a reduced method set. The *SHOP strategy*, like the system it is named after [18], prefers task expansion for the abstract tasks in the order in which they are to be executed. The expand-then-make-sound *(ems)* modification ordering strategy [17] alternates task expansion modifications with other classes, resulting in a "level-wise" concretion of all plan steps.

As for the *flexible* modification orderings, we included the well-established Least Committing First *(lcf)* paradigm, a generalization of POCL strategies that prioritizes those modifications higher that address flaws for which the smallest number of alternative solutions has been proposed. From previous work on planning strategy development we deployed two HotSpot-based strategies: HotSpots denote those components in a plan that are referred to by multiple flaws, thereby quantifying to which extent solving one deficiency may interfere with the solution options for coupled components. The Direct Uniform HotSpot *(du)* strategy consequently avoids those modifications which address flaws that refer to HotSpot plan components. While the *du* heuristic takes all flaws uniformly into account when calculating their interference potential, the Direct Adaptive HotSpot *(da)* strategy puts problem-specific weights on the binary combinations of flaw types that occur in the plan. The strategy adapts to a repeated occurrence of type combinations by increasing their weights: If abstract task flaws happen to coincide with causal threats, their combined occurrence becomes more important for this plan generation episode. As a generalization of singular HotSpots to commonly affected areas of plan components, the HotZone *(hz)* modification ordering takes into account connections between HotSpots and tries to give modifications that deal with these clusters a low priority.

Plan ordering functions control the traversal through the refinement space that is provided by the modification ordering functions. The strategies in our experimental evaluation were based on the following five components: The least commitment principle on the plan ordering level is represented in two different ways, namely the Fewer Modifications First *(fmf)* strategy, which prefers plans for which a smaller number of refinement options has been announced, and the Less Constrained Plan *(lcp)* strategy, which is based on the ratio of plan steps to the number of variable constraints and causal links in the plan.

The HotSpot concept can be lifted on the plan ordering level: The Fewer HotZone *(fhz)* strategy prefers plans with fewer HotZone clusters. The rationale for this search principle is to focus on plans in which the deficiencies are more closely related and that are hence candidates for an early decision concerning the compatibility of the refinement options. The fourth strategy operates on the HotSpot principle

[8] Different sub-versions of a problem differ in the number of parcels to transport and the number and kind of involved locations.

implemented on plan modifications: the Fewer Modification-based HotSpots *(fmh)* 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 ordering, a depth first plan selection is used for constructing a simple hierarchical ordered planner (that is, the plan ordering function is the identity function).

It is furthermore important to mention that our strategy functions can be combined into selection cascades in which succeeding components decide on those cases for which the result of the preceding ones is a tie: With $s_1 \circ s_2$ we denote, that the strategy s_1 is applied first and afterwards strategy s_2 for tie-braking.

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 means that they 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 encoding of this domain 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 representation 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 modeling 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 4 shows 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. 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 exception to this is the UM-Translog problem concerning air freight, on which using the pruned domain model has a measurable negative effect (decrease of performance of 18%). In two configurations in the easiest Satellite problem the search space cannot be reduced but a negligible overhead is introduced by pre-processing.

The average performance improvement over all strategies and over all problems in the UM-Translog domain is about 40% as is documented in Table 4a. 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 does not adapt its focus on the reduced method alternatives. It continues to address the abstract tasks in the order of their intended execution, regardless of the opportunities that the changes in the method structure may offer. We believe, however, that there may be other possibilities for a SHOP strategy to take into account the reduced domain models.

The Satellite domain does not benefit significantly from the landmark technique due to its shallow decomposition hierarchy (cf. Table 4b). We are, however, able to solve problems for which the participating strategies do not find solutions within the given resource bounds otherwise.

4.2 Landmark-Aware Strategies

Our landmark-aware strategies are based on the idea that the refinement options, which are basically stored in the optional task set column of the landmark table, estimate an upper bound for the number of expansion refinements that an abstract task requires before a solution is found. In the previous example (see Table 2), the implementation options for the abstract task *pickup* can be completely explored via the *Pickup_hazardous* and *Pickup_normal* methods. This heuristic is only a rough estimation for the "expansion effort" because the table may contain tasks that turn out to be un-achievable and it does not take into account the refinement effort it takes to make an implementation operationable on the primitive level. For our first strategies, we assume that all methods deviate more or less to the same amount in terms of both factors. We will see that this simplification already yields a heuristic with good performance.

For our landmark strategies, we first need to define the *landmark cardinality* of a set o of tasks to be the number of its (abstract) tasks, for which there is a landmark table entry. That is, let $|o|_{LT} = |\{t \in o \mid (t, M(t), O(t)) \in LT\}|$.

Our first modification ordering function *lm* is defined as follows:

Definition 7 (Landmark-Aware Ordering lm)**.** *Let f_i and f_j be two abstract task flaws in a plan P and let t_i and t_j be ground instances of the abstract tasks that are compatible with the (in-) equations in the variable constraints of P and that are referenced by f_i and f_j, respectively. Furthermore, let the landmark table contain corresponding entries $(t_i, M(t_i), O(t_i))$ and $(t_j, M(t_j), O(t_j))$.*

Table 4: This table shows the impact of the introduced pruning technique. The column *pruned* refers to the reduced domain models, whereas *unpruned* refers to the original ones (cf. Table 3). The tests were run with the planning environment PANDA [22] on a machine with a 3 GHz CPU and 256 MB Heap memory for the Java VM. *Space* refers to the number of created plans and *Time* refers to the used time in seconds including pre-processing. Values are the averages of three runs. Dashes indicate that no solution was found within a limitation of 5,000 created nodes and a time limit of 150 minutes.

(a) Results for the UM-Translog domain.

Problem	**Modification ordering function** f^{ModOrd}	**Plan ordering function** f^{PlanOrd}	**unpruned** Space	Time	**pruned** Space	Time
Hopper Truck	lcf ∘ hz	fmh ∘ fmf	72	147	41	95
	lcf ∘ ems	fmh ∘ fmf	101	211	72	174
	lcf ∘ du	fhz ∘ fmf	75	155	46	99
	hz ∘ lcf	fhz ∘ lcp ∘ fmf	71	143	54	115
	SHOP Strategy		160	323	89	212
	UMCP Strategy		96	187	58	122
Auto Truck	lcf ∘ hz	fmh ∘ fmf	119	301	85	236
	lcf ∘ ems	fmh ∘ fmf	191	443	114	298
	lcf ∘ du	fhz ∘ fmf	129	314	92	251
	hz ∘ lcf	fhz ∘ lcp ∘ fmf	183	469	157	413
	SHOP Strategy		226	558	164	433
	UMCP Strategy		216	535	156	474
Regular Truck (a)	lcf ∘ hz	fmh ∘ fmf	149	377	73	203
	lcf ∘ ems	fmh ∘ fmf	234	613	105	206
	lcf ∘ du	fhz ∘ fmf	241	483	131	370
	hz ∘ lcf	fhz ∘ lcp ∘ fmf	190	458	115	307
	SHOP Strategy		163	479	146	406
	UMCP Strategy		216	550	177	506
Regular Truck (b)	lcf ∘ hz	fmh ∘ fmf	70	142	42	98
	lcf ∘ ems	fmh ∘ fmf	106	216	81	182
	lcf ∘ du	fhz ∘ fmf	83	160	46	105
	hz ∘ lcf	fhz ∘ lcp ∘ fmf	75	152	54	122
	SHOP Strategy		146	283	106	241
	UMCP Strategy		146	278	55	113
Regular Truck (c)	lcf ∘ hz	fmh ∘ fmf	72	149	41	92
	lcf ∘ ems	fmh ∘ fmf	109	225	78	179
	hz ∘ lcf	fhz ∘ lcp ∘ fmf	74	153	54	120
	lcf ∘ du	fhz ∘ fmf	84	173	46	104
	SHOP Strategy		409	911	80	177
	UMCP Strategy		110	215	57	127
Regular Truck (d)	lcf ∘ hz	fmh ∘ fmf	–	–	275	1237
	lcf ∘ ems	fmh ∘ fmf	–	–	293	1144
	lcf ∘ du	fhz ∘ fmf	753	2755	295	1262
	hz ∘ lcf	fhz ∘ lcp ∘ fmf	–	–	787	3544
	SHOP Strategy		–	–	926	4005
	UMCP Strategy		1396	4893	308	1263
Flatebed Truck	lcf ∘ hz	fmh ∘ fmf	81	182	58	40
	lcf ∘ ems	fmh ∘ fmf	120	269	90	216
	lcf ∘ du	fhz ∘ fmf	96	216	54	129
	hz ∘ lcf	fhz ∘ lcp ∘ fmf	130	299	69	162
	SHOP Strategy		243	595	98	257
	UMCP Strategy		164	344	63	149
Armored R-Truck	lcf ∘ hz	fmh ∘ fmf	73	176	52	138
	lcf ∘ ems	fmh ∘ fmf	147	326	142	317
	lcf ∘ du	fhz ∘ fmf	84	203	56	135
	hz ∘ lcf	fhz ∘ lcp ∘ fmf	72	174	57	150
	SHOP Strategy		144	369	100	253
	UMCP Strategy		134	330	75	172

Table 4: (continued)

(a) Results for the UM-Translog domain (continued).

Problem	**Modification ordering function** f^{ModOrd}	**Plan ordering function** f^{PlanOrd}	**unpruned** Space	**unpruned** Time	**pruned** Space	**pruned** Time
Auto Traincar (a)	lcf ∘ hz	fmh ∘ fmf	454	1547	218	742
	lcf ∘ ems	fmh ∘ fmf	945	3057	243	728
	lcf ∘ du	fhz ∘ fmf	–	–	–	–
	hz ∘ lcf	fhz ∘ lcp ∘ fmf	–	–	–	–
	SHOP Strategy		–	–	–	–
	UMCP Strategy		282	1300	220	739
Auto Traincar (b)	lcf ∘ hz	fmh ∘ fmf	342	1137	144	421
	lcf ∘ ems	fmh ∘ fmf	460	1425	177	477
	lcf ∘ du	fhz ∘ fmf	365	1044	107	328
	hz ∘ lcf	fhz ∘ lcp ∘ fmf	357	958	278	770
	SHOP Strategy		541	1282	247	963
	UMCP Strategy		413	1168	161	546
Mail Traincar	lcf ∘ hz	fmh ∘ fmf	380	1241	89	221
	lcf ∘ ems	fmh ∘ fmf	590	1805	138	313
	lcf ∘ du	fhz ∘ fmf	559	1450	64	160
	hz ∘ lcf	fhz ∘ lcp ∘ fmf	93	213	70	171
	SHOP Strategy		832	1911	121	274
	UMCP Strategy		397	994	92	229
Refrigerated Regular Traincar	lcf ∘ hz	fmh ∘ fmf	384	1240	89	215
	lcf ∘ ems	fmh ∘ fmf	634	1861	138	315
	lcf ∘ du	fhz ∘ fmf	446	1074	64	159
	hz ∘ lcf	fhz ∘ lcp ∘ fmf	92	198	70	172
	SHOP Strategy		777	1735	173	353
	UMCP Strategy		400	952	90	244
Airplane	lcf ∘ hz	fmh ∘ fmf	164	507	141	435
	lcf ∘ ems	fmh ∘ fmf	142	413	167	471
	lcf ∘ du	fhz ∘ fmf	257	749	200	621
	hz ∘ lcf	fhz ∘ lcp ∘ fmf	280	777	240	700
	SHOP Strategy		335	821	150	450
	UMCP Strategy		91	253	70	215

(b) Results for the Satellite domain.

Problem	**Modification ordering function** f^{ModOrd}	**Plan ordering function** f^{PlanOrd}	**unpruned** Space	**unpruned** Time	**pruned** Space	**pruned** Time
1obs-1sat-1mode	lcf ∘ hz	fmh ∘ fmf	38	41	37	42
	lcf ∘ ems	fmh ∘ fmf	46	51	46	53
	lcf ∘ du	fhz ∘ fmf	67	72	67	72
	hz ∘ lcf	fhz ∘ lcp ∘ fmf	58	62	53	60
	SHOP Strategy		61	67	57	61
	UMCP Strategy		83	91	83	91
2obs-1sat-1mode	lcf ∘ hz	fmh ∘ fmf	602	788	539	708
	lcf ∘ ems	fmh ∘ fmf	964	1631	903	1428
	lcf ∘ du	fhz ∘ fmf	1135	1319	901	1030
	hz ∘ lcf	fhz ∘ lcp ∘ fmf	1468	1699	1216	1474
	SHOP Strategy		251	270	237	264
	UMCP Strategy		1132	1336	883	1035
2obs-2sat-1mode	lcf ∘ hz	fmh ∘ fmf	–	–	–	–
	lcf ∘ ems	fmh ∘ fmf	–	–	–	–
	lcf ∘ du	fhz ∘ fmf	–	–	2821	3353
	hz ∘ lcf	fhz ∘ lcp ∘ fmf	–	–	–	–
	SHOP Strategy		–	–	1406	1780
	UMCP Strategy		448	510	278	1097

The modification ordering function lm then orders a plan modification m_i before m_j if and only if m_i addresses f_i, m_j addresses f_j, and $\sum_{o \in O(t_i)} |o|_{LT} < \sum_{o \in O(t_j)} |o|_{LT}$ holds.

This strategy implements a rationale that is similar to the least commitment principle of the *lcf*, because it favors those plan refinements that impose less successor plans, that means, is reduces the effective branching factor of the search space. We note, that the proper choice of the grounded task instances t_i and t_j in the above definition is crucial for the actual performance, because the plan modifications typically operate on the lifted abstract tasks and method definitions. For our first experiments, we implemented a random choice on the compatible grounded landmark table entries, future work will however focus on a better informed candidate selection.

While the above heuristic focuses on the very next level of refinement, the following definition also takes into account estimates for subsequent refinement levels thus minimizing the number of refinement choices until no more decompositions are necessary.

Definition 8 (Indirect Landmark-Aware Ordering lm^*)**.** *Let f_i and f_j be two abstract task flaws in a plan P and let t_i and t_j be ground instances of the abstract tasks that are compatible with the (in-) equations in the variable constraints of P and that are referenced by f_i and f_j, respectively.*

Furthermore, let $O^*(t)$ be the transitive closure of a recursive traversal of the landmark table that begins in t. More formally: $O^*(t) = \{o | o \in O(t)$ for $(t, M(t), O(t)) \in \mathrm{LT}$ or $o \in O(t')$ for $(t', M(t'), O(t')) \in \mathrm{LT}, t' \in o'$, and $o' \in O^*(t)\}$.

The modification ordering function lm^* then orders a plan modification m_i before m_j if and only if m_i addresses f_i, m_j addresses f_j, and $\sum_{o \in O^*(t_i)} |o|_{LT} < \sum_{o \in O^*(t_j)} |o|_{LT}$ holds.

We would like to point out that O^* is always finite due to the finiteness of the landmark table, even for cyclic method definitions.

The results of our first experimental evaluation of these landmark-aware strategies is given in Table 5: lm and lm^* do outperform the other strategies on practically all problems in the UM-Translog domain (cf. Table 5a) in terms of both size of the explored search space and computational time. We believe that it is because of the relatively unreliable random choice of grounded candidates for the lifted task instances that the supposedly better informed lm^* does not consistently perform better than lm. We will address this crucial issue in future work by focusing the computational methods for lifting the landmark information: We will investigate into calculating the average optional task set sizes of the compatible ground task instances, use the minimal set sizes for consistently underestimating the effort (analog to admissible heuristics), and the like. In the Satellite domain, our landmark-aware strategies show only average performance, as there is hardly any landmark information available due to the shallow decomposition hierarchy of this domain.

5 Outlook

We have empirically shown that pruning the planning problem can significantly reduce the explored search space. This pruning relies on a relaxed reachability analysis of fully grounded primitive tasks. So far, a very basic reachability test has been used, which only tests for unsatisfied rigid predicates[9]. In future work, we will use a more elaborated technique as, for instance, explained by Fox and Long [8].

Our empirical evaluation has also shown the success of the two introduced search strategies which use the calculated local landmarks in order to guide the search process. As has been shown earlier [22, 23, 24], many different search strategies can be developed. Future work will introduce additional landmark strategies and discuss the results in more detail.

The techniques discussed in this paper directly apply to hierarchical planning. However, there are various extensions possible that apply to *hybrid* planning. One of the main differences between those two approaches is that in hybrid planning, not only primitive tasks show preconditions and effects, but also abstract tasks. This allows to test even the abstract tasks for reachability. Another difference is that hybrid planning problems also specify a goal state that has to be accomplished. Using this goal state, one can use techniques from classical planning in order to generate classical (action) landmarks which can then be used in the hybrid setting.

6 Conclusion

We have presented an effective landmark technique for hierarchical planning. It analyzes the planning problem by preprocessing 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 best performance gain could be achieved 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. We have also introduced two search strategies which use the local landmarks in order to guide the search process more efficiently towards a solution. In our empirical evaluation, both strategies outperform the other strategies we chose for comparison in most cases.

ACKNOWLEDGEMENTS

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

[1] Scott Andrews, Brian Kettler, Kutluhan Erol, and James A. Hendler, ‘UM Translog: A planning domain for the development and benchmarking of planning systems’, Technical Report CS-TR-3487, University of Maryland, (1995).

[2] Susanne Biundo and Bernd Schattenberg, ‘From abstract crisis to concrete relief (a preliminary report on combining state abstraction and HTN planning)’, in *Proc. of the 6th European Conference on Planning (ECP 2001)*, pp. 157–168. Springer Verlag, (2001).

[9] Rigid predicates are predicates that are interpreted state-independently and hence their truth value cannot be changed by actions. Their usage is therefore restricted to preconditions and the initial state specification.

Table 5: This table shows the impact of the used modification ordering functions on the planning process when using the reduced domain models. In all experiments, except in the SHOP and UMCP case, the plan ordering function fmf was used. In the case of SHOP and UMCP, the plan and modification ordering functions were used that simulate SHOP's and UMCP's search process, respectively. The tests were run with the planning environment PANDA [22] on a machine with a 3 GHz CPU and 256 MB Heap memory for the Java VM. *Space* refers to the number of created plans and *Time* refers to the used time in seconds including pre-processing. Values are the averages of three runs. Dashes indicate that no solution was found within a limitation of 5,000 created nodes and a time limit of 150 minutes. The best result for a given problem is emphasized bold, the second best bold and italic.

(a) Results for the UM-Translog domain

Mod. ordering function f^{ModOrd}	Hopper Truck		Auto Truck		Regular Truck (a)		Regular Truck (b)	
	Space	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**
lcf	55	118	155	470	162	463	78	173
hz	55	121	197	527	191	473	***55***	117
lm	***52***	***111***	**133**	**329**	**145**	**374**	62	135
lm*	**51**	**109**	***135***	***462***	154	430	**52**	**112**
ems	147	295	405	976	211	507	127	262
da	144	352	644	2077	239	562	114	257
du	101	224	459	1304	1508	4097	160	460
SHOP	89	212	164	433	***146***	***406***	106	241
UMCP	58	122	156	474	177	506	***55***	***113***

Mod. ordering function f^{ModOrd}	Regular Truck (c)		Regular Truck (d)		Flatbed Truck		Armored R-Truck	
	Space	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**
lcf	127	222	327	1278	***62***	179	86	198
hz	***55***	137	–	–	159	399	122	355
lm	**53**	**122**	***291***	***1172***	63	155	***71***	***177***
lm*	65	142	**266**	**1162**	**61**	**144**	**61**	**155**
ems	114	235	–	–	1571	3797	113	269
da	148	352	723	2560	99	237	120	359
du	117	258	–	–	1047	2601	75	201
SHOP	83	190	926	4005	98	257	95	227
UMCP	57	***127***	308	1263	63	***149***	75	172

Mod. ordering function f^{ModOrd}	Auto Traincar (a)		Auto Traincar (b)		Mail Traincar	
	Space	**Time**	**Space**	**Time**	**Space**	**Time**
lcf	–	–	227	926	79	209
hz	–	–	701	1616	81	224
lm	**158**	**596**	183	608	**75**	**184**
lm*	304	1473	**158**	**543**	***78***	***205***
ems	–	–	2558	6447	879	1806
da	–	–	184	705	641	2031
du	–	–	1390	4018	424	1090
SHOP	–	–	247	963	121	274
UMCP	***220***	***739***	***161***	***546***	92	229

Mod. ordering function f^{ModOrd}	Refrigerated Regular Traincar		Airplane	
	Space	**Time**	**Space**	**Time**
lcf	90	225	247	798
hz	***76***	***196***	345	1323
lm	**72**	**180**	***142***	***441***
lm*	89	212	189	676
ems	500	1048	784	2517
da	588	1958	172	620
du	307	775	643	2134
SHOP	173	353	150	450
UMCP	90	244	**70**	**215**

Table 5: (continued)

(b) Results for the Satellite domain.

Mod. ordering function f^{ModOrd}	1obs-1sat-1mode **Space**	**Time**	2obs-1sat-1mode **Space**	**Time**	2obs-2sat-1mode **Space**	**Time**
lcf	86	93	1120	1338	***407***	**701**
hz	***61***	**60**	1281	4764	1094	1338
lm	73	80	***560***	***652***	693	***785***
lm*	78	85	847	969	739	813
ems	65	64	1586	2608	1219	1579
da	**56**	**60**	782	1131	2186	6841
du	100	107	–	–	–	–
SHOP	62	66	**138**	**155**	1406	1780
UMCP	83	91	883	1035	**278**	1097

[3] Blai Bonet and Héctor Geffner, 'Planning as heuristic search', *Artificial Intelligence*, **129**, 5–33, (2001).

[4] Luis Castillo, Juan Fdez-Olivares, and Antonio González, 'On the adequacy of hierarchical planning characteristics for real-world problem solving', in *Proc. of the 6th European Conference on Planning (ECP 2001)*, (2001).

[5] Mohamed Elkawkagy, Bernd Schattenberg, and Susanne Biundo, 'Landmarks in hierarchical planning', in *Proc. of the 20th European Conference on Artificial Intelligence (ECAI 2010)*, (2010).

[6] Kutluhan Erol, James Hendler, and Dana S. Nau, 'UMCP: A sound and complete procedure for hierarchical task-network planning', in *Proc. of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS 1994)*, pp. 249–254, (1994).

[7] Tara A. Estlin, Steve A. Chien, and Xuemei Wang, 'An argument for a hybrid HTN/operator-based approach to planning', in *Proc. of the 4th European Conference on Planning: Recent Advances in AI Planning*, pp. 182–194, (1997).

[8] Maria Fox and Derek Long, 'The automatic inference of state invariants in TIM', *Journal of Artificial Intelligence Research (JAIR)*, **9**, 367–421, (1998).

[9] Peter Gregory, Stephen Cresswell, Derek Long, and Julie Porteous, 'On the extraction of disjunctive landmarks from planning problems via symmetry reduction', in *Proc. of the 4th International Workshop on Symmetry and Constraint Satisfaction Problems*, eds., W. Harvey and Z. Kiziltan, pp. 34–41, (2004).

[10] Ronny Hartanto and Joachim Hertzberg, 'On the benefit of fusing DL-reasoning with HTN-planning', in *Advances in Artificial Intelligence, Proc. of the 32nd German Conference on Artificial Intelligence (KI 2009)*, eds., Bärbel Mertsching, Marcus Hund, and Zaheer Aziz, pp. 41–48, (2009).

[11] Patrik Haslum, Blai Bonet, and Héctor Geffner, 'New admissible heuristics for domain-independent planning', in *Proc. of the Twentieth National Conference on Artificial Intelligence*, volume 3, pp. 1163–1168, (2005).

[12] Malte Helmert and Carmen Domshlak, 'Landmarks, critical paths and abstractions: What's the difference anyway?', in *Proc. of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pp. 162–169, (2009).

[13] Jörg Hoffmann and Berhard Nebel, 'The FF planning system: Fast plan generation through heuristic search', *Journal of Artificial Intelligence Research*, **14**, 253–302, (2001).

[14] Jörg Hoffmann, Julie Porteous, and Laura Sebastia, 'Ordered landmarks in planning', *Journal of Artificial Intelligence Research*, **22**, 215–278, (2004).

[15] Subbarao Kambhampati, Amol Mali, and Biplav Srivastava, 'Hybrid planning for partially hierarchical domains', in *Proc. of the 15th National Conference on Artificial Intelligence*, pp. 882–888. American Association for Artificial Intelligence (AAAI Press), (1998).

[16] Erez Karpas and Carmel Domshlak, 'Cost-optimal planning with landmarks', in *Proc. of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pp. 1728–1733, (2009).

[17] Thomas L. McCluskey, 'Object transition sequences: A new form of abstraction for HTN planners', in *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI 00)*, eds., Steve Chien, Subbarao Kambhampati, and Craig Knoblock, pp. 216–225, Breckenridge, CO, USA, (2000). AAAI Press, Menlo Park, CA, USA.

[18] Dana S. Nau, Yue Cao, Ammon Lotem, and Héctor Muñoz-Avila, 'SHOP: Simple hierarchical ordered planner', in *Proc. of the 16th International Joint Conference on Artificial Intelligence (IJCAI 1999)*, pp. 968–975, (1999).

[19] Julie Porteous and Stephen Cresswell, 'Extending landmarks analysis to reason about resources and repetition', in *In Proc. of the 21st Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG 2002)*, pp. 45–54, (2002).

[20] Julie Porteous, Laura Sebastia, and Jörg Hoffmann, 'On the extraction, ordering, and usage of landmarks in planning', in *Proc. of the 6th European Conference on Planning (ECP 2001)*, eds., A. Cesta and D. Borrajo, pp. 37–48, (2001).

[21] Silvia Richter, Malte Helmert, and Matthias Westphal, 'Landmarks revisited', in *Proc. of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*. AAAI Press, (2008).

[22] Bernd Schattenberg, *Hybrid Planning & Scheduling*, Ph.D. dissertation, Ulm University, Germany, 2009.

[23] Bernd Schattenberg, Julien Bidot, and Susanne Biundo, 'On the construction and evaluation of flexible plan-refinement strategies', in *Advances in Artificial Intelligence, Proc. of the 30th German Conference on Artificial Intelligence (KI 2007)*, eds., Joachim Hertzberg, Michael Beetz, and Roman Englert, volume 4667 of *Lecture Notes in Artificial Intelligence*, pp. 367–381. Springer, (2007).

[24] Bernd Schattenberg, Andreas Weigl, and Susanne Biundo, 'Hybrid planning using flexible strategies', in *Advances in Artificial Intelligence, Proc. of the 28th German Conference on Artificial Intelligence (KI 2005)*, pp. 249–263. Springer-Verlag Berlin Heidelberg, (2005).

[25] Laura Sebastia, Eva Onaindia, and Eliseo Marzal, 'Decomposition of planning problems', *AI Communications*, **19**(1), 49–81, (2006).

[26] Vincent Vidal and Héctor Geffner, 'Branching and pruning: An optimal temporal POCL planner based on constraint programming', *Artificial Intelligence*, **170**, 298–335, (2006).

[27] Lin Zhu and Robert Givan, 'Landmark extraction via planning graph propagation', in *Proc. of the ICAPS 2003 Doctoral Consortium*, pp. 156–160, (2003).