On the Extraction, Ordering, and Usage of Landmarks in Planning

Julie Porteous[1], Laura Sebastia[2], and Jörg Hoffmann[3]

[1] Department of Computer Science, The University of Durham, Durham, UK, J.M.Porteous@durham.ac.uk

[2] Dpto. Sist. Informáticos y Computación, Universidad Politécnica de Valencia, Valencia, Spain, lstarin@dsic.upv.es

[3] Institut für Informatik, Universität Freiburg, Freiburg, Germany, hoffmann@informatik.uni-freiburg.de

Abstract. Many known planning tasks have inherent constraints concerning the best order in which to achieve the goals. A number of research efforts have been made to detect such constraints and use them for guiding search, in the hope to speed up the planning process.

We go beyond the previous approaches by defining ordering constraints not only over the (top level) goals, but also over the sub-goals that will arise during planning. Landmarks are facts that must be true at some point in every valid solution plan. We show how such landmarks can be found, how their inherent ordering constraints can be approximated, and how this information can be used to decompose a given planning task into several smaller sub-tasks. Our methodology is completely domain- and planner-independent. The implementation demonstrates that the approach can yield significant performance improvements in both heuristic forward search and GRAPHPLAN-style planning.

1 Introduction

Given the inherent complexity of the general planning problem it is clearly important to develop good heuristic strategies for both managing and navigating the search space involved in solving a particular planning instance. One way in which search can be informed is by providing hints concerning the order in which planning goals should be addressed. This can make a significant difference to search efficiency by helping to focus the planner on a progressive path towards a solution. Work in this area includes that of GAM [9,10] and PRECEDE [12]. Koehler and Hoffmann [10] introduce the notion of *reasonable orders* where a pair of goals A and B can be ordered so that B is achieved before A if it isn't possible to reach a state in which A and B are both true, from a state in which just A is true, without having to temporarily destroy A. In such a situation it is reasonable to achieve B before A to avoid unnecessary effort.

The motivation of the work discussed in this paper is to extend those previous ideas on orderings by not only ordering the (top level) goals, but also the sub-goals that will arise during planning, i.e., by also taking into account what we call the *landmarks*. The key feature of a landmark is that it *must* be true on any solution path to the given planning task. Consider the *Blocksworld* task shown in Figure 1, which will be our working example throughout the paper.

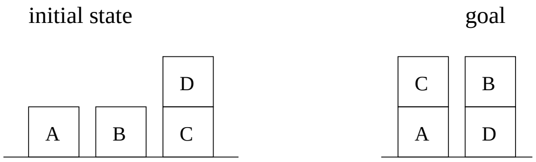

Fig. 1. Example *Blocksworld* task.

Here, *clear(C)* is a landmark because it will need to be achieved in any solution plan. Immediately stacking B on D from the initial state will achieve one of the top level goals of the task but it will result in wasted effort if *clear(C)* is not achieved first. The ordering *clear(C)* $\leq$ *on(B D)* is, however, *not* reasonable in terms of Koehler and Hoffmann's definition yet it is a sensible order to impose if we wish to reduce wasted effort during plan generation. We introduce the notion of *weakly reasonable* orderings, which captures this situation. Two landmarks L and L' are also often ordered in the sense that all valid solution plans make L true before they make L' true. We call such ordering relations *natural*. For example, *clear(C)* is naturally ordered before *holding(C)* in the above *Blocksworld* task.

We introduce techniques for extracting landmarks to a given planning task, and for approximating natural and weakly reasonable orderings between those landmarks. The resulting information can be viewed as a tree structure, which we call the *landmark generation tree*. This tree can be used to decompose the planning task into small chunks. We propose a method that does not depend on any particular planning framework. To demonstrate the usefulness of the approach, we have used the technique for control of both the forward planner FF(v1.0) [7] and the GRAPHPLAN-style planner IPP(v4.0) [11], yielding significant performance improvements in both cases.

The paper is organised as follows. Section 2 gives the basic notations. Sections 3 to 5 explain how landmarks can be extracted, ordered, and used, respectively. Empirical results are discussed in Section 6 and we conclude in Section 7.

2 Notations

We consider a propositional STRIPS [4] framework.

Definition 1. A *state* S is a finite set of logical facts. An *action* o is a triple $o = (\text{pre}(o), \text{add}(o), \text{del}(o))$ where $\text{pre}(o)$ are the preconditions, $\text{add}(o)$ is the add list, and $\text{del}(o)$ is the delete list, each being a set of facts. The result of applying a single action to a state is:

$$Result(S, \langle o \rangle) = \begin{cases} (S \cup \text{add}(o)) \setminus \text{del}(o) & \text{pre}(o) \subseteq S \\ \text{undefined} & \text{otherwise} \end{cases}$$

The result of applying a sequence of more than one action to a state is recursively defined as $Result(S, \langle o_1, \ldots, o_n \rangle) = Result(Result(S, \langle_1, \ldots, o_{n-1} \rangle), \langle o_n \rangle)$. A *planning task* $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ is a triple where $\mathcal{O}$ is the set of actions, and $\mathcal{I}$ (the initial state) and $\mathcal{G}$ (the goals) are sets of facts. A plan for a task $\mathcal{P}$ is an action sequence $P \in \mathcal{O}^*$ such that $\mathcal{G} \subseteq Result(\mathcal{I}, P)$.

3 Extracting Landmarks

In this section, we will focus on the landmarks extraction process and its properties. First of all, we define what a landmark is.

Definition 2. Given a planning task $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$. A fact L is a *landmark in* $\mathcal{P}$ iff L is true at some point in all solution plans, i.e., iff for all $P = \langle o_1, \ldots, o_n \rangle, \mathcal{G} \subseteq Result(\mathcal{I}, P) : L \in Result(\mathcal{I}, \langle o_1, \ldots, o_i \rangle)$ for some $0 \leq i \leq n$.

All initial facts are trivially landmarks (let $i = 0$ in the above definition). For the final search control, they are not considered. They *can*, however, play an important role for extracting ordering information. In the *Blocksworld* task shown in Figure 1, *clear(C)* is a landmark, but *on(A B)*, for example, is not. In general, it is PSPACE-hard to decide whether an arbitrary fact is a landmark.

Definition 3. Let LANDMARK RECOGNITION denote the following problem.
Given a planning task $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$, and a fact L. Is L a landmark in $\mathcal{P}$?

Theorem 4. *Deciding* LANDMARK RECOGNITION *is* PSPACE-*hard.*

Proof Sketch: By a reduction of (the complement of) PLANSAT, the problem of deciding whether an arbitrary STRIPS planning task is solvable [3]: add an artificial by-pass to the task, on which a new fact L must be added. ■

Due to space restrictions, we include only short proof sketches in this paper. The complete proofs can be found in a technical report [15]. The following is a simple sufficient condition for a fact being a landmark.

Proposition 5. *Given a planning task* $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$, *and a fact* L. *Define* $\mathcal{P}_L = (\mathcal{O}_L, \mathcal{I}, \mathcal{G})$ *as follows.*

$$\mathcal{O}_L := \{(pre(o), add(o), \emptyset) \mid (pre(o), add(o), del(o)) \in \mathcal{O}, L \notin add(o)\}$$

If $\mathcal{P}_L$ *is unsolvable, then* L *is a landmark in* $\mathcal{P}$.

Deciding about solvability of planning tasks with empty delete lists can be done in polynomial time by a GRAPHPLAN-style algorithm [2,8]. An idea is, consequently, to evaluate the above sufficient condition for each non-initial state fact in turn. However, this can be costly when there are many facts in a task. We use the following two-step process.

1. First, a backward chaining process extracts landmark candidates.
2. Then, evaluating Proposition 5 eliminates those candidates that are not provably landmarks.

The backward chaining process can select initial state facts, but does not necessarily select all of them. In verification, initial (and goal) facts need not be considered as they are landmarks by definition.

3.1 Extracting Landmark Candidates

Candidate landmarks are extracted using what we call the *relaxed planning graph* (RPG): relax the planning task by ignoring all delete lists, then build GRAPHPLAN's planning graph, chaining forward from the initial state of the task to a graph level where all goals are reached. Because the delete lists are empty, the graph does not contain any mutex relations [8]. Once the RPG has been built, we step backwards through it to extract what we call the *landmark-generation tree* (LGT). This is a tree (N, E) where the nodes N are candidate landmarks and an edge $(L, L') \in E$ indicates that L must be achieved as a necessary prerequisite for L'. Additionally, if several nodes $L_1, \ldots, L_k$ are ordered before the same node L', then $L_1, \ldots, L_k$ are grouped together in an AND-node in the sense that those facts must be true together at some point during the planning process. The root of the tree is the AND-node representing the top level goals.

The extraction process is straightforward. First, all top level goals are added to the LGT and are posted as goals in the first level where they were added in the RPG. Then, each goal is solved in the RPG starting from the last level. For each goal g in a level, all actions achieving g are grouped into a set and the intersection I of their preconditions is computed. For all facts p in I we: post p as a goal in the first RPG level were it is achieved; insert p as a node into the LGT; insert an edge between p and g into the LGT. When all goals in a level are achieved, we move on to the next lower level. The process stops when the first (initial) level is reached.

We also use the following technique, to obtain a larger number of candidates: when a set of actions solves a goal, we also compute the union of the preconditions that are not in the intersection. We then consider all actions achieving these facts. If the intersection of those action's preconditions is non-empty, we take the facts in that intersection as candidate landmarks as well. For example, say we are solving a *Logistics* task where there are two planes and a package to be moved from *la-post-office* to *boston-post-office*. While extracting landmarks, we will find that *at(pack1 boston-airport)* is a candidate landmark. The intersection of the actions achieving it is empty, but the union consists of *in(pack1 plane1)* and *in(pack1 plane2)*. The intersection of the preconditions of the first actions in the RPG adding these facts is *at(pack1 la-airport)*, which is a landmark that would not have been found without this more sophisticated process. More details about the process (which are not necessary for understanding the rest of this discussion) are described by Porteous and Sebastia [14].

Let us illustrate the extraction process with the *Blocksworld* example from Figure 1. The RPG corresponding to this task is shown in Figure 2. As we explained above, the extraction process starts by adding two nodes representing the goals *on(C A)* and *on(B D)* to the LGT ($N = \{$*on(C A)*,*on(B D)*$\}, E = \emptyset$). It also posts *on(C A)* as goal in level 3 and *on(B D)* in level 2. There is only one action achieving *on(C A)* in level 3: *stack C A*. So, *holding(C)* and *clear(A)* are new candidates. *holding(C)* is posted as a goal in level 2, *clear(A)* is initially true and does therefore not need to be posted as a goal. The new LGT is: $N = \{$*on(C A)*,*on(B D)*,*holding(C)*,*clear(A)*$\}, E = \{($*(holding C)*,*on(C A)*$), ($*(clear A)*,*on(C A)*$\}$. As there are no more goals in level 3, we move downwards to solve the goals in

L_0	A_1	L_1	A_2	L_2	A_3	L_3
on-table A	pick-up A	holding A	stack B A	on B A	stack C A	on C A
on-table B	pick-up B	holding B	stack B D	on B D	stack C B	on C B
on-table C	unstack D C	holding D	stack B C	on B C	stack C D	on C D
on D C		clear C	put-down B	...	...	...
clear A			...			
clear B			pick-up C	holding C		
clear D			...	...		
arm-empty						

Fig. 2. Summarised RPG for the *Blocksworld* example shown in Figure 1.

level 2. We now have two goals: *on(B D)* and *holding(C)*. In both cases, there is only one action adding each fact (*stack B D* and *pick-up C*), so their preconditions *holding(B)*, *clear(C)*, *on-table(C)*, and *arm-empty()*, as well as the respective edges, are included into the LGT. The goals at level 1 are *holding(B)* and *clear(C)*, which are added by the single actions *pick-up B* and *unstack D C*. The process ends up with the following LGT, where we leave out, for ease of reading, the initial facts and their respective edges: $N = \{$*on(C A)*, *on(B D)*, *holding(C)*, *holding(B)*, *clear(C)*, $\ldots\}$ and $E = \{$(*holding(C)*,*on(C A)*), (*holding(B)*,*on(B D)*), (*clear(C)*,*holding(C)*), $\ldots\}$. Among the parts of the LGT concerning initial facts, there is the edge (*clear(D)*,*clear(C)*) $\in E$. As we explain in Section 4, this edge plays an essential role for detecting the ordering constraint *clear(C)* $\leq$ *on(B D)* that was mentioned in the introduction. The edge is inserted as precondition of *unstack D C*, which is the first action in the RPG that adds *clear(C)*.

3.2 Verifying Landmark Candidates

Say we want to move from city A to city D on the road map shown in Figure 3, using a standard *move* operator. Landmarks extraction will come up with the following LGT: $N = \{$*at(A)*, *at(E)*, *at(D)*$\}$, $E = \{$(*at(A)*, *at(E)*), (*at(E)*, *at(D)*)$\}$—the RPG is only built until the goals are reached the first time, which happens in this example before *move C D* comes in. However, the action sequence $\langle$*move(A,B)*, *move(B,C)*, *move(C,D)*$\rangle$ achieves the goals without making *at(E)* true. Therefore, the candidate *at(E)* $\in N$ is not really a landmark.

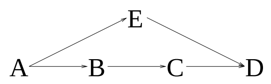

Fig. 3. An example road map.

We want to remove such candidates because they can lead to incompleteness in our search framework, which we will describe in Section 5. As was said above, we simply check Proposition 5 for each fact $L \in N$ except the initial facts and the goals, i.e., for each such L in turn we ignore the actions that add L, and check solvability of the resulting planning task when assuming that all delete lists are empty. Solvability is checked by constructing the RPG to the task. If the test fails, i.e., if the goals are reachable, then we remove L from the LGT. In the above example, *at(A)* and *at(D)* need not be verified. Ignoring all actions achieving *at(E)*, the goal is still reachable by the actions that *move* to D via B and C. So *at(E)* and its edges are removed, yielding the final LGT where $N = \{$*at(A)*, *at(D)*$\}$ and $E = \emptyset$.

4 Ordering Landmarks

In this section we define two types of ordering relations, called *natural* and *weakly reasonable* orders, and explain how they can be approximated. Firstly, consider the natural orderings. As said in the introduction, two landmarks L and L' are ordered naturally, $L \leq_n L'$, if in all solution plans L is true before L' is true. L is true before L' in a plan $\langle o_1, \ldots, o_n \rangle$ if, when i is minimal with $L \in \mathit{Result}(\mathcal{I}, \langle o_1, \ldots, o_i \rangle)$ and j is minimal with $L' \in \mathit{Result}(\mathcal{I}, \langle o_1, \ldots, o_j \rangle)$, then $i < j$. Natural orderings are characteristic of landmarks: usually, the reason why a fact is a landmark is that it is a necessary prerequisite for another landmark. For illustration consider our working example, where *clear(C)* must be true immediately before *holding(C)* in all solution plans. In general, deciding about natural orderings is PSPACE-hard.

Definition 6. Let NATURAL ORDERING denote the following problem.

Given a planning task $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$, and two atoms A and B. Is there a natural ordering between B and A, i.e., does $B \leq_n A$ hold?

Theorem 7. *Deciding* NATURAL ORDERING *is* PSPACE-*hard.*

Proof Sketch: Reduction of the complement of PLANSAT. Arrange actions for two new facts A and B such that one can either: add A, then B, then solve the original task; or add B, then A, then achieve the goal right away. ∎

The motivation for weakly reasonable orders has already been explained in the context of Figure 1. Stacking B on D from the initial state is not a good idea since $\mathit{clear}(C)$ needs to be achieved first if we are to avoid unnecessary effort. However, the ordering *clear(C)* $\leq$ *on(B D)* is *not* reasonable, in the sense of Koehler and Hoffmann's formal definition [10], since there are reachable states where B is on D and C is not clear, but C can be made clear without unstacking B. However, reaching such a state requires unstacking D from C, and (uselessly) stacking A onto C. Such states are clearly not relevant for the situation at hand. Our definition therefore weakens the reasonable orderings in the sense that only the *nearest* states are considered in which B is on D. Precisely, Koehler and Hoffmann [10] define $S_{A,\neg B}$, for two atoms A and B, as the set of reachable states where A has just been achieved, but B is still false. They order $B \leq_r A$ if all solution plans achieving B from a state in $S_{A,\neg B}$ need to destroy A. In contrast, we restrict the starting states that are considered to $S^{opt}_{A,\neg B}$, defined as those states in $S_{A,\neg B}$ that have minimal distance from the initial state. Accordingly, we define two facts B and A to have a weakly reasonable ordering constraint, $B \leq_w A$, iff

$$\forall\, s \in S^{opt}_{(A,\neg B)} : \forall\, P \in \mathcal{O}^* : \; B \in \mathit{Result}(s, P) \Rightarrow \exists\, o \in P : A \in \mathrm{del}(o)$$

Deciding about weakly reasonable orderings is PSPACE-hard.

Definition 8. Let WEAKLY REASONABLE ORDERING denote the following problem.

Given a planning task $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$, and two atoms A and B. Is there a weakly reasonable ordering between B and A, i.e., does $B \leq_w A$ hold?

Theorem 9. *Deciding* WEAKLY REASONABLE ORDERING *is* PSPACE-*hard.*

Proof Sketch: Reduction of the complement of PLANSAT. Arrange actions for two new facts A and B such that: A is never deleted, and achieved once before the original task can be started; B can be achieved only when the original goal is solved. ∎

4.1 Approximating Natural and Weakly Reasonable Orderings

As an exact decision about either of the above ordering relations is as hard as planning itself, we have used the approximation techniques described in the following. The approximation of $\leq_n$ is called $\leq_{an}$, the approximation of $\leq_w$ is called $\leq_{aw}$. The orders $\leq_{an}$ are extracted directly from the LGT. Recall that for an edge (L, L') in the LGT, we know that L and L' are landmarks and also that L is in the intersection of the preconditions of the actions achieving L' at its lowest appearance in the RPG. We therefore order a pair of landmarks L and L' $L \leq_{an} L'$, if $LGT = (N, E)$, and $(L, L') \in E$.

What about $\leq_{aw}$, the approximations to the weakly reasonable orderings? We are interested in pairs of landmarks L and L', where from all nearby states in which L' is achieved and L is not, we must delete L' in order to achieve L. Our method of approximating this looks at: pairs of landmarks within a particular AND-node of the LGT since these must be made simultaneously true in some state; landmarks that are naturally ordered with respect to one of this pair since these give an ordered sequence in which "earlier" landmarks must be achieved; and any inconsistencies[1] between these "earlier" landmarks and the other landmark at the node of interest. As the first two pieces of information are based on the RPG (from which the LGT is extracted), our approximation is biased towards those states that are close to the initial state. The situation we consider is, for a pair of landmarks in the same AND-node in the LGT, what if a landmark that is ordered before one of them is inconsistent with the other? If they are inconsistent then this means that they can't be made simultaneously true, (ie achieving one of them will result in the other being deleted). So that situation is used to form an order in one of the following two ways:

1. landmarks L and L' in the same AND-node in the LGT can be ordered $L \leq_{aw} L'$, if:

$$\exists\, x \in Landmarks : x \leq_{an} L \wedge inconsistent(x, L')$$

2. a pair of landmarks L and L' can be ordered $L \leq_{aw} L'$ if there exists some other landmark x which is: in the same AND-node in the LGT as L'; and there is an ordered sequence of $\leq_{an}$ orders that order L before x. In this situation, L and L' are ordered, if

$$\exists\, y \in Landmarks : y \leq_{an} L \wedge inconsistent(y, L')$$

In both cases the rationale is: look for an ordered sequence of landmarks required to achieve a landmark x at a node. For any landmark L in the sequence, if L is inconsistent with another landmark L' at the same AND-node as x then there is no point in achieving L' before L (effort will be wasted since it will need to be re-achieved). If L must be achieved before some other landmark y (its successor in the sequence) then the order is $y \leq_{aw} L'$.

A final way in which we derive ordering constraints is based on analysis of any $\leq_{an}$ and $\leq_{aw}$ orders already identified. A pair of landmarks L and L' is ordered $L \leq_{aw} L'$ if:

$$\exists\, x \in Landmarks : L \leq_{an} x \wedge L' \leq_{aw} x \wedge inconsistent(side\_effects(L'), L)$$

where the *side_effects* of a landmark L' are:

$$\text{side\_effects}(L') := \{\text{add}(o) \setminus \{L'\} \mid o \in \mathcal{O}, L' \in \text{add}(o)\}$$

[1] A pair of facts is inconsistent if they can't be made simultaneously true. We approximate inconsistency using the respective function provided by the TIM API [5] available from: `http://www.dur.ac.uk/computer.science/research/stanstuff/planpage.html`

The basis for this is: L must be true in the state immediately before x is achieved (given by $\leq_{an}$) and L' can be achieved before x (given by $\leq_{aw}$). But can L' and L be ordered? If the side effects of achieving L' are inconsistent with L then achieving L first would waste effort (it would have to be re-achieved for x). Hence the order $L' \leq_{aw} L$.

4.2 Extracting Natural and Weakly Reasonable Orderings

The LGT is used for extracting orders as follows: (i) identify the $\leq_{an}$ orders; (ii) identify the $\leq_{aw}$ orders; (iii) analyse those orders to identify remaining $\leq_{aw}$ orders; (iv) remove cycles in the graph of orders; (vi) finally, add all orders as edges in the LGT for later use during planning.

As an illustration of this process, consider again the example shown in figure 1. First, the set of $\leq_{an}$ orders are extracted directly from the LGT. The set contains, amongst other things: *clear D* $\leq_{an}$ *clear C*, *clear(C)* $\leq_{an}$ *holding(C)*, and *holding(C)* $\leq_{an}$ *on(C A)* (see Section 3). In the next step, the $\leq_{aw}$ orders are identified. Let us focus on how the order *clear(C)* $\leq_{aw}$ *on(B D)* (our motivating example) is found. From the $\leq_{an}$ orders we have the ordered sequence ⟨*clear(D)*, *clear(C)*, *holding(C)*, *on(C A)*⟩ and the fact that *on(C A)* is in the same node as *on(B D)*. Since *clear(D)* and *on(B D)* are inconsistent and *clear(D)* $\leq_{an}$ *clear(C)*, the order *clear(C)* $\leq_{aw}$ *on(B D)* is imposed. Note here the crucial point that we have the order *clear(D)* $\leq_{an}$ *clear(C)*. We have that order because *unstack D C* is the first action in the RPG that adds *clear(C)*. The nearest possibility, from the initial state, of clearing C is to unstack D from C. This can only be done when D is still clear. Our approximation methods recognise this, and correctly conclude that stacking B on D immediately is not a good idea.

The next stage is a check to identify and remove any cycles that appear in the graph of orderings. A cycle (or strongly connected component) such as, $L \leq_{an} L'$ and $L' \leq_{aw} L$, might arise if a landmark must be achieved more than once in a solution plan (for example, in the *Blocksworld* domain this is frequently the case for *arm_empty()*). At present, any cycles in the orders are removed since they aren't used by the search process. They are removed by firstly identifying for each cycle the set of articulation points for it (a node in a connected component is an articulation point if the component that remains, after the node and all edges incident upon it are removed, is no longer connected). The cycles are broken by iteratively removing the articulation points and all edges incident upon these points until no more strongly connected components remain. For our small example no cycles are present so the final step is to add the $\leq_{aw}$ orders to the LGT.

5 Using Landmarks

Having settled on algorithms for computing the LGT, there is still the question of how to use this information during planning. For use in forward state space planning, Porteous and Sebastia [14] have proposed a method that prunes states where some landmark has been achieved too early. If applying an action achieves a landmark L that is not a leaf of the current LGT, then do not use that action. If an action achieves a landmark L that *is* a leaf, then remove L (and all ordering relations it is part of) from the LGT. In short, do not allow achieving a landmark unless all of its predecessors have been achieved already.

Here, we explore an idea that uses the LGT to decompose a planning task into smaller sub-tasks, which can be handed over to any planning algorithm. The idea is similar to the above described method in terms of how the LGT is looked at: each sub-task results from considering the leaf nodes of the current LGT, and when a sub-task has been processed, then the LGT is updated by removing achieved leaf nodes. The main problem is that the leaf nodes of the LGT can often not be achieved as a conjunction. The main idea is to pose those leaf nodes as a *disjunctive* goal instead. See the algorithm in Figure 4.

$S := \mathcal{I},\ P := \langle\, \rangle$
remove from LGT all initial facts and their edges
repeat
  $Disj$:= leaf nodes of LGT
  call base planner with actions $\mathcal{O}$, initial state S and goal condition $\bigvee Disj$
  if base planner did not find a solution P' **then** fail **endif**
  $P := P \circ P'$, S := result of executing P' in S
  remove from LGT all $L \in Disj$ with $L \in \text{add}(o)$ for some o in P'
until LGT is empty
call base planner with actions $\mathcal{O}$, initial state S and goal $\bigwedge \mathcal{G}$
if base planner did not find a solution P' **then** fail **endif**
$P := P \circ P'$, output P

Fig. 4. Disjunctive search control algorithm for a planning task $(\mathcal{O}, \mathcal{I}, \mathcal{G})$, repeatedly calling an arbitrary planner on a small sub-task.

The depicted algorithm keeps track of the current state S, the current plan prefix P, and the current disjunctive goal $Disj$, which is always made up out of the current leaf nodes of the LGT. The initial facts are immediately removed because they are true anyway. When the LGT is empty—all landmarks have been processed—then the algorithm stops, and calls the underlying base planner from the current state with the original (top level) goals. The algorithm fails if at some point the planner did not find a solution.

Looking at Figure 4, one might wonder why the top level goals are no sooner given special consideration than when all landmarks have been processed. Remember that all top level goals are also landmarks. An idea might be to force the algorithm, once a top level goal G has been achieved, to keep G true throughout the rest of the process. We have experimented with a number of variations of this idea. The problem with this is that one or a set of already achieved original goals might be inconsistent with a leaf landmark. Forcing the achieved goals to be true together with the disjunction yields in this case an unsolvable sub-task, making the control algorithm fail. In contrast to this, we will see below that the simple control algorithm depicted above is completeness preserving under certain conditions fulfilled by many of the current benchmarks. Besides this, keeping the top level goals true did not yield better runtime or solution length behaviour in our experiments. This may be due to the fact that, unless such a goal is inconsistent with some landmark ahead, it is kept true anyway.

5.1 Theoretical Properties

The presented disjunctive search control is obviously planner-independent in the sense that it can be used within any (STRIPS) planning paradigm—a disjunctive goal can be simulated by using an artificial new fact G as the goal, and adding one action for each disjunct L, where the action's precondition is $\{L\}$ and the add list is $\{G\}$ (this was first described by Gazen and Knoblock [6]). The search control is obviously correctness preserving—eventually, the planner is run on the original goal. Likewise obviously, the method is not optimality preserving.

With respect to completeness, matters are a bit more complicated. As it turns out, the approach is completeness preserving on the large majority of the current benchmarks. The reasons for this are that there, no fatally wrong decisions can be made in solving a sub-task, that most facts which have been true once can be made true again, and that natural ordering relations are respected by any solution plan. We need two notations.

1. A *dead end* is a reachable state from which the goals can not be reached anymore [10], a task is dead-end free if there are no dead ends in the state space.
2. A fact L is *recoverable* if, when S is a reachable state with $L \in S$, and S' with $L \notin S'$ is reachable from S, then a state S'' is reachable from S' with $L \in S''$.

Many of the current benchmarks are invertible in the sense that every action o has a counterpart $\overline{o}$ that undoes o's effects [10]. Such tasks are dead-end free, and all facts in such tasks are recoverable. Completeness is preserved under the following circumstances.

Theorem 10. *Given a solvable planning task $(\mathcal{O}, \mathcal{I}, \mathcal{G})$, and an LGT (N, E) where each $L \in N$ is a landmark such that $L \notin \mathcal{I}$. If the task is dead-end free, and for $L' \in N$ it holds that either L' is recoverable, or all orders $L \leq L'$ in the tree are natural, then running any complete planner within the search control defined by Figure 4 will yield a solution.*

Proof Sketch: If search control fails, then the current state S is a dead end. If it is not, an unrecoverable landmark L' is added by the current prefix P ($L' \notin \mathcal{I}$ so it must be added at some point). L' was not a leaf node at the time it was added, so there is a landmark L with $L \leq L'$ that gets added after L' in contradiction. ■

Verifying landmarks with Proposition 5 ensures that all facts in the LGT really are landmarks; the initial facts are removed before search begins. The tasks contained in domains like *Blocksworld*, *Logistics*, *Hanoi* and many others are invertible [10]. Examples of dead-end free domains with only natural orders are *Gripper* and *Tsp*. Examples of dead-end free domains where non-natural orders apply only to recoverable facts are *Miconic*-STRIPS and *Grid*. All those domains (or rather, all tasks in those domains) fulfill the requirements for Theorem 10.

6 Results

We have implemented the extraction, ordering, and usage methods presented in the preceding sections in C, and used the resulting search control mechanism as a framework for the heuristic forward search planner FF-v1.0 [7], and the GRAPHPLAN-based planner IPP4.0 [2,11]. Our own implementation is based on FF-v1.0, so providing FF with the sub-tasks defined by the LGT, and communicating back the results, is done via function parameters. For controlling IPP, we have implemented a simple interface, where a propositional encoding of each sub-task is specified via two files in the STRIPS subset of PDDL [13,1]. We have changed the implementation of IPP4.0 to output a results file containing the spent running time, and a sequential solution plan (or a flag saying that no plan has been found). The running times given below have been measured on a Linux workstation running at 500 MHz with 128 MBytes main memory. We cut off test runs after half an hour. If no plan was found within that time, we indicate this by a dash. For IPP, we did not count the overhead for repeatedly creating and reading in the PDDL specifications of propositional sub-tasks—this interface is merely a vehicle that we used for experimental implementation. Instead, we give the running time needed by the search control plus the sum of all times needed for planning after the input files have been read. For FF, the times are simply total running times.

		FF-v1.0		FF-v1.0 + L		FF-v2.2	
domain	task	time	steps	time	steps	time	steps
Blocksworld	*bw-large-a*	0.01	12	0.17	16	0.01	14
Blocksworld	*bw-large-b*	1.12	30	0.18	24	0.01	22
Blocksworld	*bw-large-c*	-	-	0.24	38	1.02	44
Blocksworld	*bw-large-d*	7.03	56	0.31	48	0.78	54
Grid	*prob01*	0.07	14	0.26	16	0.07	14
Grid	*prob02*	0.46	39	0.44	26	0.47	39
Grid	*prob03*	3.01	58	1.30	79	2.96	58
Grid	*prob04*	2.75	49	1.30	54	2.70	49
Grid	*prob05*	28.42	149	390.01	161	29.39	145
Logistics	*prob-38-0*	38.03	223	5.93	285	39.61	223
Logistics	*prob-39-0*	101.37	244	6.22	294	98.26	239
Logistics	*prob-40-0*	69.03	245	7.49	308	31.68	251
Logistics	*prob-41-0*	129.15	255	7.73	320	29.85	248
Tyreworld	*fixit-1*	0.01	19	0.18	19	0.01	19
Tyreworld	*fixit-10*	26.87	118	3.01	136	0.71	136
Tyreworld	*fixit-20*	-	-	26.24	266	10.16	266
Tyreworld	*fixit-30*	-	-	157.74	396	46.65	396
Freecell	*prob-7-1*	11.87	56	2.05	44	4.96	48
Freecell	*prob-7-2*	4.18	50	1.99	45	4.58	52
Freecell	*prob-7-3*	2.29	43	1.88	46	4.07	42
Freecell	*prob-8-1*	19.31	63	(2.17)	-	11.32	60
Freecell	*prob-8-2*	9.89	57	2.48	49	35.52	61
Freecell	*prob-8-3*	2.64	50	2.28	51	4.16	54
Freecell	*prob-9-1*	145.60	84	3.33	72	9.55	73
Freecell	*prob-9-2*	49.17	64	3.22	60	6.77	59
Freecell	*prob-9-3*	3.29	55	2.95	54	5.53	54
Freecell	*prob-10-1*	21.89	84	(3.48)	-	61.85	87
Freecell	*prob-10-2*	15.70	70	(2.95)	-	8.45	66
Freecell	*prob-10-3*	7.68	56	3.63	61	9.32	64
Freecell	*prob-11-1*	(222.48)	-	(3.78)	-	- (160.91)	-
Freecell	*prob-11-2*	(17.76)	-	(3.42)	-	117.62 (5.62)	74
Freecell	*prob-11-3*	(35.13)	-	(4.39)	-	10.52	83

Fig. 5. Running time (in seconds) until a solution was found, and sequential solution length for FF-v1.0, FF-v1.0 with landmarks control (FF-v1.0 + L), and FF-v2.2. Times in brackets specify the running time after which a planner failed because search ended up in a dead end.

Figure 5 shows running time and solution length for FF-v1.0, FF-v1.0 controlled by our landmarks mechanism (FF-v1.0 + L), and FF-v2.2. The last system FF-v2.2 is Hoffmann and Nebel's successor system to FF-v1.0, which goes beyond the first version in terms of a number of goal ordering techniques, and a complete search mechanism that is invoked in case the planner runs into a dead end [8]. Let us consider the domains in Figure 5 from top to bottom. In the *Blocksworld* tasks taken from the BLACKBOX distribution, FF-v1.0 + L clearly outperforms the original version. The running time values are also better than those for FF-v2.2. Solution lengths show some variance, making it hard to draw conclusions. In the *Grid* examples used in the AIPS-1998 competition, running time with landmarks control is better than that of both FF versions on the first four tasks. In *prob05*, however, the controlled version takes much longer time, so it seems that the behaviour of our technique depends on the individual structure of tasks in the *Grid* domain. Solution length performance is again somewhat varied, with a tendency to be longer when using landmarks. In *Logistics*, where we look at some of the largest examples from the AIPS-2000 competition, the results are unmistakable: the control mechanism dramatically improves runtime performance, but degrades solution length performance. The increase in solution length is due to unnecessarily many airplane moves: once the packages have arrived at the nearest airports, they are transported to their destination airports one by one (we outline below an approach how this can be overcome). In the *Tyreworld*, where an increasing number of tyres need to be replaced, runtime performance of FF-v1.0 improves dramatically when using landmarks. FFv2.2, however, is still superior in terms of running time. In terms of solution lengths our method and FF-v2.2 behave equally, i.e., slightly worse than FF-v1.0.

We have obtained especially interesting results in the *Freecell* domain. Data is given for some of the larger examples used in the AIPS-2000 competition. In *Freecell*, tasks can contain dead ends. Like our landmarks control, the FF search mechanism is incomplete in the presence of such dead ends [7,8]. When FF-v1.0 or our enhanced version encounter a dead end, they simply stop without finding a plan. When FF-v2.2 encounters a dead end, it invokes a complete heuristic search engine that tries to solve the task from scratch [8]. This is why FF-v2.2 can solve *prob-11-2*. For all planners, if they encountered a dead end, then we specify in brackets the running time after which they did so. The following observations can be made: on the tasks that FF-v1.0 + L can solve, it is much faster than both uncontrolled FF versions; with landmarks, some more trials run into dead ends, but this happens very fast, so that one could invoke a complete search engine without wasting much time; finally, solution length with landmarks control is in most cases better than without.

		IPP		IPP + L	
domain	task	time	steps	time	steps
Blocksworld	*bw-large-a*	0.17	12	0.36	16
Blocksworld	*bw-large-b*	11.05	18	0.79	26
Blocksworld	*bw-large-c*	-	-	3.17	38
Blocksworld	*bw-large-d*	-	-	11.73	54
Grid	*prob01*	1.84	14	1.15	14
Grid	*prob02*	30.61	29	5.11	30
Grid	*prob03*	-	-	56.08	79
Gripper	*prob01*	0.02	11	0.20	15
Gripper	*prob03*	3.25	23	0.29	31
Gripper	*prob20*	-	-	20.85	167
Logistics	*log-a*	-	-	1.42	61
Logistics	*log-b*	-	-	0.91	45
Logistics	*log-c*	-	-	1.54	56
Logistics	*log-d*	-	-	6.80	80
Tyreworld	*fixit-1*	0.20	19	0.23	19
Tyreworld	*fixit-2*	18.55	30	0.67	32
Freecell	*prob-2-1*	8.98	9	0.48	10
Freecell	*prob-2-2*	9.73	8	0.52	10
Freecell	*prob-2-3*	8.37	9	0.53	11
Freecell	*prob-2-4*	9.17	8	0.49	10
Freecell	*prob-2-5*	8.78	9	0.53	10
Freecell	*prob-3-1*	-	-	1.15	21
Freecell	*prob-4-1*	-	-	1.92	29
Freecell	*prob-5-1*	-	-	3.01	36
Freecell	*prob-6-1*	-	-	3.76	45

Fig. 6. Running time (in seconds) until a solution was found, and sequential solution length for IPP and IPP with landmarks control (IPP + L).

Figure 6 shows the data that we have obtained by running IPP against a version controlled by our landmarks algorithm. IPP normally finds plans that are guaranteed to be optimal in terms of the number of parallel time steps. Using our landmarks control, there is no such optimality guarantee. As a measure of solution quality we show, like in the previous figure, the number of actions in the plans found. Quite obviously, our landmarks control mechanism speeds IPP up by some orders of magnitude across all listed domains. In the *Blocksworld*, solutions appear to get slightly longer. In *Grid*, solution length differs only by one more action used in *prob02*. Running IPP + L on the larger examples *prob04* and *prob05* failed due to a parse error, i.e., IPP's parsing routine failed when reading in one of the sub-tasks specified by our landmarks control algorithm. This is probably because IPP's parsing routine is not intended to read in propositional encodings of planning tasks, which are of course much larger than the uninstantiated encodings that are usually used. So this failure is due to the preliminary implementation that we used for experimentation. In *Gripper*, the control algorithm comes down to transporting the balls one by one, which is why IPP + L can solve even the largest task *prob-20* from the AIPS-1998 competition, but returns unnecessarily long plans. In the *Logistics* examples from the BLACKBOX distribution, the solutions contain—like we observed for FF in the experiments described above—unnecessarily many airplane moves; those tasks were, however, previously unsolvable for IPP. In one long testing run, IPP + L solved even the comparatively large *Logistics* task *prob-38-0* (shown above for the FF variants) within 6571 seconds, finding a plan with 251 steps. In the *Tyreworld*, there is a small increase in solution length to be observed (probably the same increase that we observed in our experiments with FF). Running *fixit-3* failed due to a parse error similar to the one described above for the larger *Grid* tasks. In *Freecell*, where we show some of the smaller tasks from the AIPS-2000 competition, the plans found by IPP + L are only slightly longer than IPP's ones for those few small tasks that IPP can solve. Running IPP + L on any task larger than *prob-6-1* produced parse errors.

In *Gripper*, and partly also in *Logistics*, the disjunctive search control from Figure 4 results in a trivialisation of the planning task, where goals are simply achieved one by one. While this speeds up the planning process, the usefulness of the found solutions is questionable. The problem is there that our approximate LGT does not capture the structure of the tasks well enough—some goals (like a ball being in room B in *Gripper*) become leaf nodes of the LGT though there are other subgoals which should be cared for first (like some other ball being picked up in *Gripper*). One way around this is trying to improve on the information that is provided by the LGT (we will say a few words on this in Section 7). Another way is to change the search strategy: instead of posing all leaf nodes to the planner as a disjunctive goal, one can pose a disjunction of *maximal consistent subsets* of those leaf nodes (consistency of a fact set here is approximated as pairwise consistency according to the TIM API). In *Gripper*, FF and IPP with landmarks control find the optimal solutions with that strategy, in *Logistics*, the solutions are similar to those found without landmarks control. This result is of course obtained at the cost of higher running times than with the fully disjunctive method. What's more, posing maximal consistent subsets as goals can lead to incompleteness when an inconsistency remains undetected.

7 Conclusion and Outlook

We have presented a way of extracting and using information on ordered landmarks in STRIPS planning. The approach is independent of the planning framework one wants to use, and maintains completeness under circumstances fulfilled by many of the current benchmarks. Our results on a range of domains show that significant, sometimes dramatic, runtime improvements can be achieved for heuristic forward search as well as GRAPHPLAN-style planners, as exemplified by the systems FF and IPP. The approach does not maintain optimality, and empirically the improvement in runtime behaviour is sometimes (like in *Logistics*) obtained at the cost of worse solution length behaviour. There are however (like in *Freecell* for FF) also cases where our technique improves solution length behaviour.

Possible future work includes the following topics: firstly, one can try to improve on the landmarks and orderings information, for example by taking into account the different "roles" that a top level goal can play (i.e. as a top level goal, or as a landmark for some other goal), or by a more informed treatment of cycles. Secondly, post-processing procedures for improving solution length in cases like *Logistics* might be useful for getting better plans after finding a first plan quickly. Finally, we want to extend our methodology so that it can handle conditional effects.

References

1. Fahiem Bacchus. *Subset of PDDL for the AIPS2000 Planning Competition*. The AIPS-00 Planning Competition Comitee, 2000.
2. Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):279–298, 1997.
3. Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994.
4. Richard E. Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
5. Maria Fox and Derek Long. The automatic inference of state invariants in tim. *Journal of Artificial Intelligence Research*, 9:367–421, 1998.
6. B. Cenk Gazen and Craig Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In Steel and Alami [16], pages 221–233.
7. Jörg Hoffmann. A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In *Proceedings of the 12th International Symposium on Methodologies for Intelligent Systems (ISMIS-00)*, pages 216–227. Springer-Verlag, October 2000.
8. Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 2001. Accepted for Publication.
9. Jana Koehler. Solving complex planning tasks through extraction of subproblems. In R. Simmons, M. Veloso, and S. Smith, editors, *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, pages 62–69. AAAI Press, Menlo Park, 1998.
10. Jana Koehler and Jörg Hoffmann. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *Journal of Artificial Intelligence Research*, 12:338–386, 2000.
11. Jana Koehler, Bernhard Nebel, Jörg Hoffmann, and Yannis Dimopoulos. Extending planning graphs to an ADL subset. In Steel and Alami [16], pages 273–285.
12. T. L. McCluskey and J. M. Porteous. Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence*, 95, 1997.
13. Drew McDermott et al. *The PDDL Planning Domain Definition Language*. The AIPS-98 Planning Competition Comitee, 1998.
14. Julie Porteous and Laura Sebastia. Extracting and ordering landmarks for planning. In *Proceedings UK Planning and Scheduling SIG Workshop*, 2000.
15. Julie Porteous, Laura Sebastia, and Jörg Hoffmann. On the extraction, ordering, and usage of landmarks in planning. Technical Report 4/01, Department of Computer Science, University of Durham, Durham, England, May 2001. Available at http://www.dur.ac.uk/ dcs0www/research/stanstuff/planpage.html
16. S. Steel and R. Alami, editors. *Recent Advances in AI Planning. 4th European Conference on Planning (ECP'97)*, volume 1348 of *Lecture Notes in Artificial Intelligence*, Toulouse, France, September 1997. Springer-Verlag.