This version is available at https://strathprints.strath.ac.uk/1874/
Strathprints is designed to allow users to access the research output of the University of Strathclyde. Unless otherwise explicitly stated on the manuscript, Copyright © and Moral Rights for the papers on this site are retained by the individual authors and/or other copyright owners. Please check the manuscript for details of any other licences that may have been applied. You may not engage in further distribution of the material for any profitmaking activities or any commercial gain. You may freely distribute both the url (https://strathprints.strath.ac.uk/) and the content of this paper for research or private study, educational, or not-for-profit purposes without prior permission or charge.
Any correspondence concerning this service should be sent to the Strathprints administrator: strathprints@strath.ac.uk
On the Extraction, Ordering, and Usage of Landmarks in Planning

Julie Porteous\textsuperscript{1}, Laura Sebastia\textsuperscript{2}, and Jörg Hoffmann\textsuperscript{3}

\textsuperscript{1} Department of Computer Science, The University of Durham, Durham, UK, J.M.Porteous@durham.ac.uk
\textsuperscript{2} Dpto. Sist. Informáticos y Computación, Universidad Politécnica de Valencia, Valencia, Spain, lstarin@dsic.upv.es
\textsuperscript{3} Institut für Informatik, Universität Freiburg, Freiburg, Germany, hoffmann@informatik.uni-freiburg.de

\textbf{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 Blockworld task shown in Figure 1, which will be our working example throughout the paper.

![Diagram of initial state and goal](image)

**Fig. 1.** Example Blockworld 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 $clear(C) \leq on(BD)$ 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 Blockworld 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:

\[
\text{Result}(S, \langle o \rangle) = \left\{
\begin{array}{ll}
(S \cup \text{add}(o)) \setminus \text{del}(o) & \text{pre}(o) \subseteq S \\
\text{undefined} & \text{otherwise}
\end{array}
\right.
\]

The result of applying a sequence of more than one action to a state is recursively defined as \( \text{Result}(S, \langle o_1, \ldots, o_n \rangle) = \text{Result}(\text{Result}(S, \langle o_1, \ldots, o_{n-1} \rangle), \langle o_n \rangle) \). 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 \( \mathcal{G} \subseteq \text{Result}(\mathcal{I}, P) \).

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 \( P = \langle o_1, \ldots, o_n \rangle, \mathcal{G} \subseteq \text{Result}(\mathcal{I}, P) : L \in \text{Result}(\mathcal{I}, \langle o_1, \ldots, o_i \rangle) \) 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, \( \text{clear}(C) \) is a landmark, but \( \text{on}(A, B) \), 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.

\[
\mathcal{O}_L := \{ (\text{pre}(o), \text{add}(o), \emptyset) \mid (\text{pre}(o), \text{add}(o), \text{del}(o)) \in \mathcal{O}, L \not\subseteq \text{add}(o) \}
\]

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
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), ...} and E = {(holding(C), on(C A)), (holding(B), on(B D)), (clear(C), holding(C)), ...}. Among the parts of the LGT concerning initial facts, there is the edge (clear(D), clear(C)) ∈ E. As we explain in Section 4, this edge plays an essential role for detecting the ordering constraint clear(C) ≤ 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(B)}, {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 ⟨move(A, B), move(B, C), move(C, D)⟩ achieves the goals without making at(E) true. Therefore, the candidate at(E) ∈ N is not really a landmark.

![Fig. 3. An example road map.](image)

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 ∈ 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 = ∅.

### 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 $L \in \text{Result}(I, \langle o_1, \ldots, o_i \rangle)$ and $j$ is minimal with $L' \in \text{Result}(I, \langle o_1, \ldots, o_j \rangle)$, 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 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 $B \preceq_r A$ 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 $S^\text{opt}_{A, \neg B}$, 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

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

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 latest appearance in the RPG. We therefore order a pair of landmarks $L$ and $L'$ 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:

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

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

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

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:

$$\exists x \in \text{Landmarks} : L \leq_{an} x \land L' \leq_{aw} x \land \text{inconsistent}(\text{side effects}(L'), L)$$

where the \textit{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; (v) 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 \( \langle \text{clear}(D), \text{clear}(C), \text{holding}(C), \text{on}(C \ A) \rangle \) and the fact that \( \text{on}(C \ A) \) is in the same node as \( \text{on}(B \ D) \). 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 Sebasta [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.

\[ S := I, \quad P := \langle \rangle \]

repeat
  \[ \text{Disj} := \text{leaf nodes of LGT} \]
  call base planner with actions \( \mathcal{O} \), initial state \( S \) and goal condition \( \sqrt{\text{Disj}} \)
  if base planner did not find a solution \( P' \) then fail endif
  \[ P := P \circ P', \quad S := \text{result of executing } P' \text{ in } S \]
  remove from LGT all \( L \in \text{Disj} \) 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 \( \land G \)
if base planner did not find a solution \( P' \) then fail endif
\[ P := P \circ P', \quad \text{output } P \]

Fig. 4. Disjunctive search control algorithm for a planning task \((\mathcal{O}, I, G)\), 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 \( \text{Disj} \), 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 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 (STIPS) 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 \((O,I,G)\), and an LGT \((N,E)\) where each \( L \in N \) is a landmark such that \( L \notin I \). 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 \( P \) (\( L' \notin I \) 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.
<table>
<thead>
<tr>
<th>domain</th>
<th>task</th>
<th>PP-v1.0 time steps</th>
<th>PP-v1.0 + L time steps</th>
<th>PP-v2.2 time steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blockworld</td>
<td>lw-large-a</td>
<td>0.01</td>
<td>0.17</td>
<td>0.01</td>
</tr>
<tr>
<td>Blockworld</td>
<td>lw-large-b</td>
<td>1.12</td>
<td>0.18</td>
<td>0.01</td>
</tr>
<tr>
<td>Blockworld</td>
<td>lw-large-c</td>
<td>-</td>
<td>0.24</td>
<td>1.02</td>
</tr>
<tr>
<td>Blockworld</td>
<td>lw-large-d</td>
<td>7.03</td>
<td>0.31</td>
<td>0.78</td>
</tr>
<tr>
<td>Grid</td>
<td>prob01</td>
<td>0.01</td>
<td>0.12</td>
<td>0.01</td>
</tr>
<tr>
<td>Grid</td>
<td>prob02</td>
<td>0.46</td>
<td>0.14</td>
<td>0.43</td>
</tr>
<tr>
<td>Grid</td>
<td>prob03</td>
<td>3.01</td>
<td>1.30</td>
<td>2.96</td>
</tr>
<tr>
<td>Grid</td>
<td>prob04</td>
<td>2.75</td>
<td>1.30</td>
<td>2.70</td>
</tr>
<tr>
<td>Grid</td>
<td>prob05</td>
<td>28.42</td>
<td>9.01</td>
<td>28.39</td>
</tr>
<tr>
<td>Logistics</td>
<td>prob-b-32</td>
<td>38.03</td>
<td>5.93</td>
<td>39.61</td>
</tr>
<tr>
<td>Logistics</td>
<td>prob-b-33</td>
<td>101.37</td>
<td>6.22</td>
<td>98.26</td>
</tr>
<tr>
<td>Logistics</td>
<td>prob-b-60</td>
<td>65.03</td>
<td>7.49</td>
<td>61.98</td>
</tr>
<tr>
<td>Logistics</td>
<td>prob-b-140</td>
<td>129.15</td>
<td>7.73</td>
<td>129.85</td>
</tr>
<tr>
<td>Tyreworld</td>
<td>fast-1</td>
<td>0.01</td>
<td>0.18</td>
<td>0.01</td>
</tr>
<tr>
<td>Tyreworld</td>
<td>fast-10</td>
<td>26.37</td>
<td>3.01</td>
<td>0.71</td>
</tr>
<tr>
<td>Tyreworld</td>
<td>fast-20</td>
<td>-</td>
<td>26.24</td>
<td>10.18</td>
</tr>
<tr>
<td>Tyreworld</td>
<td>fast-30</td>
<td>-</td>
<td>39.74</td>
<td>40.06</td>
</tr>
<tr>
<td>Treebell</td>
<td>prob-b-1</td>
<td>11.87</td>
<td>2.06</td>
<td>4.99</td>
</tr>
<tr>
<td>Treebell</td>
<td>prob-b-2</td>
<td>4.18</td>
<td>1.89</td>
<td>4.58</td>
</tr>
<tr>
<td>Treebell</td>
<td>prob-b-3</td>
<td>2.29</td>
<td>1.88</td>
<td>4.07</td>
</tr>
<tr>
<td>Treebell</td>
<td>prob-b-4</td>
<td>19.34</td>
<td>2.17</td>
<td>11.32</td>
</tr>
<tr>
<td>Treebell</td>
<td>prob-b-5</td>
<td>9.48</td>
<td>2.18</td>
<td>5.52</td>
</tr>
<tr>
<td>Treebell</td>
<td>prob-b-6</td>
<td>2.64</td>
<td>2.28</td>
<td>4.16</td>
</tr>
<tr>
<td>Treebell</td>
<td>prob-b-7</td>
<td>145.60</td>
<td>3.33</td>
<td>9.58</td>
</tr>
<tr>
<td>Treebell</td>
<td>prob-b-8</td>
<td>49.17</td>
<td>3.22</td>
<td>6.77</td>
</tr>
<tr>
<td>Treebell</td>
<td>prob-b-9</td>
<td>3.29</td>
<td>2.96</td>
<td>5.63</td>
</tr>
<tr>
<td>Treebell</td>
<td>prob-b-10</td>
<td>21.89</td>
<td>3.48</td>
<td>61.86</td>
</tr>
<tr>
<td>Treebell</td>
<td>prob-b-12</td>
<td>15.70</td>
<td>2.95</td>
<td>8.45</td>
</tr>
<tr>
<td>Treebell</td>
<td>prob-b-13</td>
<td>7.68</td>
<td>3.63</td>
<td>9.32</td>
</tr>
<tr>
<td>Treebell</td>
<td>prob-b-14</td>
<td>222.48</td>
<td>3.78</td>
<td>160.91</td>
</tr>
<tr>
<td>Treebell</td>
<td>prob-b-15</td>
<td>117.68</td>
<td>3.47</td>
<td>117.92</td>
</tr>
<tr>
<td>Treebell</td>
<td>prob-b-16</td>
<td>35.13</td>
<td>4.39</td>
<td>10.52</td>
</tr>
</tbody>
</table>

Fig. 5. Running time (in seconds) until a solution was found, and sequential solution length for PP-v1.0, PP-v1.0 with landmarks control (PP-v1.0 + L), and PP-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 PP-v1.0, PP-v1.0 controlled by our landmarks mechanism (PP-v1.0 + L), and PP-v2.2. The last system PP-v2.2 is Hoffmann and Nebel's successor system to PP-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 Blockworld tasks taken from the BLACKBOX distribution, PP-v1.0 + L clearly outperforms the original version. The running time values are also better than those for PP-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 PP 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 PP-v1.0 improves dramatically when using landmarks. PP-
v2.2, 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.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Task</th>
<th>IPP</th>
<th>IPP + L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freecell</td>
<td>prob-21</td>
<td>8.98 9 0.48 10</td>
<td></td>
</tr>
<tr>
<td>Freecell</td>
<td>prob-22</td>
<td>9.73 8 0.52 10</td>
<td></td>
</tr>
<tr>
<td>Freecell</td>
<td>prob-23</td>
<td>8.37 9 0.53 11</td>
<td></td>
</tr>
<tr>
<td>Freecell</td>
<td>prob-24</td>
<td>9.17 8 0.49 10</td>
<td></td>
</tr>
<tr>
<td>Freecell</td>
<td>prob-25</td>
<td>8.78 9 0.53 10</td>
<td></td>
</tr>
<tr>
<td>Freecell</td>
<td>prob-26</td>
<td>- 1.15 21</td>
<td></td>
</tr>
<tr>
<td>Freecell</td>
<td>prob-27</td>
<td>- 1.92 20</td>
<td></td>
</tr>
<tr>
<td>Freecell</td>
<td>prob-28</td>
<td>- 3.01 28</td>
<td></td>
</tr>
<tr>
<td>Freecell</td>
<td>prob-29</td>
<td>- 3.76 45</td>
<td></td>
</tr>
</tbody>
</table>

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 *fixit3* 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 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