Pruning Methods for Optimal Delete-Free Planning

Avitan Gefen and **Ronen I. Brafman**
Department of Computer Science
Ben-Gurion University of The Negev, Israel

Abstract

Delete-free planning underlies many popular relaxation ($h+$) based heuristics used in state-of-the-art planners; it provides a simpler setting for exploring new pruning methods and other ideas; and a number of interesting recent planning domains are naturally delete-free. In this paper we explore new pruning methods for planning in delete-free planning domains. First, we observe that optimal delete-free plans can be composed from contiguous sub-plans that focus on one fact landmark at a time. Thus, instead of attempting to achieve the goal, the planner can focus on more easily achievable landmarks at each stage. Then, we suggest a number of complementary pruning techniques that are made more powerful with this observation. To carry out these pruning techniques efficiently, we make heavy use of an And/Or graph depicting the planning problem. We empirically evaluate these ideas using the FD framework, and show that they lead to clear improvements.

Introduction

Heuristic search is currently the preferred method for solving satisfycing and optimal planning problems. Among the different methods for generating heuristic functions, relaxation-based methods are extremely popular, effective, and influential (Hoffmann and Nebel 2001; Helmert and Domshlak 2009). These methods attempt to estimate the true minimal distance-to-goal of a state in the delete-free problem generated from a given instance by removing all delete-effects. Computing this value, known as $h+$, is NP-hard (Bylander 1994), but various approximations of it, starting with the highly influential FF heuristic (Hoffmann and Nebel 2001), and more recently LM-Cut (Helmert and Domshlak 2009) have been shown to be very effective in practice. Nevertheless, good, but imperfect heuristic estimates, such as $h+$, are not sufficient alone for solving hard problems, as demonstrated in (Helmert and Röger 2008). Thus, it is important to introduce to our arsenal of tools additional methods that help in other ways, such as pruning methods (Chen, Xu, and Yao 2009; Chen and Yao 2009; Coles and Coles 2010) methods for problem decomposition (Brafman and Domshlak 2006), and others.

In this paper we investigate new pruning techniques that exploit landmarks within delete-free planning problems. We focus on delete-free problems for four reasons. First, delete-free problems offer special structure and flexibility that we can exploit for this purpose. This structure also allows us to model delete-free problems using And/Or graphs (or directed hypergraphs), whose structure we exploit in this paper. Moreover, this structure leads to the existence of multiple equivalent permutations of each plan, the pruning of which is important if we wish to prevent redundant work. Second, we hope our techniques will be applicable, eventually, to standard planning problems, although this requires non-trivial extensions. Third, solving (optimally, or not) delete-free problems, i.e., computing $h+$, is one of the most effective contemporary techniques for generating heuristic functions. Thus, any method that effectively solves delete-free planning problems has immediate use in standard planning. Finally, a number of interesting, naturally delete-free planning problems (Gefen and Brafman 2011; Gallo et al. 1993) cannot be solved by standard planning techniques.

In the next section we present an overview of our pruning techniques. They are based on two principles: 1. There exists a minimal plan to the goal whose prefix is a minimal plan to some landmark, l. 2. Given a disjunctive action landmark consisting of actions applicable at s, we can ignore all other actions at s. To operationalize these ideas, we provide techniques for identifying useful landmark orderings, small disjunctive actions landmarks, and some properties of minimal plans. These techniques make heavy use of an And/Or graph depicting the (delete-free) planning problem. Due to space limitations, we omit some of the proofs, which can be found in (Gefen and Brafman 2012).

We evaluate the new pruning methods by measuring their impact on the performance of A^* search with the admissible LM-Cut (Helmert and Domshlak 2009) and Merge& Shrink heuristics (Helmert, Haslum, and Hoffmann 2007) on delete-free problems obtained from standard benchmark problems. Our experiments show that pruning noticeably improved the performance of state-of-the-art heuristic search for many domains, and that pruning with blind search did better on most domains than Merge& Shrink.

Background and Overview

A *STRIPS planning task*, or *planning task* for short, is a 4-tuple (P, A, I, G). P is a finite set of *propositions*. A state s is represented by the set of propositions that are *true* in it. $I \subseteq P$ is the *initial state*. $G \subseteq P$ is the set of propositions that must be true at any *goal state*. A is the set of *actions*. Each action $a \in A$ has the form: $a = \langle pre(a), add(a), del(a) \rangle$ denoting its preconditions, add effects, and delete effects. An action a is applicable in a state $s \subseteq P$ iff $pre(a) \subseteq s$. Applying a in s transforms the system to the state $(s \setminus del(a)) \cup add(a)$. We use $a(s)$ to denote the resulting state. When a is not applicable in s then $a(s)$ is undefined.

A *solution* to a planning task is a plan $\rho = (a_1, \dots, a_k)$ such that $G \subseteq a_k(\cdots(a_1(I))\cdots)$. That is, it is a sequence of actions that transforms the initial state into a state satisfying the goal conditions. *From here on we will be discussing only delete-free problems, i.e., problems in which all actions have an empty delete list.*

Landmarks play an important role in solving planning tasks. In this paper, they play an important role in decomposing the planning problem. A *fact landmark* for a state s is a proposition that holds at some point in every legal plan from state s to the goal (Hoffmann, Porteous, and Sebastia 2004). An *action landmark* for a state s is an action that must be part of any plan from s to the goal. A *disjunctive action landmark* for a state s is a set of actions, at least one of which must be part of every legitimate plan from state s to the goal (Helmert and Domshlak 2009).

Let L_I denote the set of fact landmarks for the initial state, and let L_s denote the set of fact landmarks for state s. The following observations are useful for understanding the properties of delete-free planning problems. Except for 6, all are immediate.

1. Achieved facts are never deleted, so once achieved they remain true.
2. Applicable actions always remain applicable since their preconditions are never deleted.
3. Each action should appear only once in a plan.[1]
4. The order of actions in a specific plan $(a_1, a_2, \dots)$ is not important as long as the plan is valid. That is, as long as $pre(a_i) \subseteq a_{i-1}(\cdots(a_1(I))\cdots)$.
5. Thus, if an action landmark is applicable in state s, we can immediately apply it.
6. For any state s reachable from the initial state, $L_s \subseteq L_I$, and $L_I \setminus L_s$ are the landmarks achieved so far (on route to state s). Proof in (Gefen and Brafman 2012).
7. Consequently, fact landmarks need only be computed once in the initial state. This is not generally true when delete lists are not empty.

In this paper, we focus on finding *optimal* delete-free plans. The notion of *minimal* plans plays a central role in this. A plan π for G is *minimal* if no strict subset of π is a plan for G. Clearly, any optimal plan must be minimal, unless zero-cost actions exist, in which case it must have a sub-plan which is minimal. Therefore, when seeking an optimal plan for G, *we can prune any plan that is not minimal without sacrificing optimality.*

[1]We do not consider actions with conditional effects.

We now present a number of theoretical results and observations which will provide the underlying intuitions and theoretical justifications for the pruning algorithms we present in later sections.

Lemma 1. *Let L be a set of fact landmarks for a delete-free planning problem $\Pi = (P, A, I, G)$, such that $G \subseteq L$. Then, there exists an ordering $l_1, \dots, l_k$ of L and a minimal plan π for Π such that $\pi = \pi_1, \dots, \pi_k$, where π_i is a minimal plan for $(P, A, \pi_{i-1}(\cdots(\pi_1(I))\cdots), l_i)$.*

Lemma 1 justifies a divide and conquer approach to optimal delete-free planning, but is not constructive. A similar idea was used in the context of general planning as a heuristic method (Hoffmann, Porteous, and Sebastia 2004). However, in delete-free planning, it can be used as a completeness and minimality preserving approach to pruning. The following corollary makes it applicable.

Corollary 1. *Let l be an arbitrary fact landmark for the delete-free planning problem $\Pi = (P, A, I, G)$. Let π_l be a minimal plan for (P, A, I, l). Let π' be a minimal plan for $(P, A, \pi_l(I), G)$. Then, π_l, π' is a minimal plan for Π.*

Corollary 1 has an important practical implication: *instead of planning for G, we can plan for l, and maintain minimality.* Later on, we will show how to find such an l that is easy to achieve. In fact, we will derive a whole sequence of landmarks, as Lemma 1 suggests.

When seeking a minimal plan for l, we will employ two pruning techniques. In theory, these pruning methods can be used directly to plan for G, rather than l. But, they are much more effective when applied to a closer ”goal” – thus the practical importance of choosing an ”easily achieved” landmark.

The first method allows us to focus on actions within an applicable disjunctive landmark, pruning all other actions. It is justified by the following theorem:

Theorem 1. *Consider the search tree T for a delete-free planning problem and some state s in it. Let X be a disjunctive action landmark for s, all of whose actions are applicable in s. Let T' be the result of pruning all subtrees rooted at s that commence with an action outside X. T' contains a minimal solution iff T contains a solution.*

Proof. Consider an optimal plan π from s to the goal. It must contain an action $a \in X$ and this action is applicable at s. As observed above, we can reorder the actions in π, provided we make sure that each action remains applicable in the new plan. Since a is applicable in s, there is a plan that is a permutation of π in which a is the first action after s. This plan is optimal as well, and it is not pruned by the above procedure. □

Our second method focuses on recognizing non-minimal plans. During the search we say that we are *pursuing* l from state s_p if starting in state s_p, the landmark we are currently trying to achieve is l. The proof of the following Theorem is immediate.

Theorem 2. *Let a_{last} be the last action taken to reach the current state s while pursuing landmark l from state s_p. We can safely prune actions that are not included in any minimal plan that contains a_{last} and leads from state s_p to a fact landmark l, provided a_{last} did not achieve any fact landmark (with respect to s_p) including l.*

To operationalize this idea, we will later describe techniques that allow us to recognize sets of actions that ”go together” in minimal plans.

The Relaxed Causal And/Or graph

We can fully capture the structure of a delete-free problem using graphical models such as And/Or graphs or directed hypergraphs. As the former are more familiar to AI and planning researchers, we will focus on them, relying heavily on the framework introduced in (Keyder, Richter, and Helmert 2010) using And/Or graphs. A *rooted And/Or graph* is a $\mathcal{G} = \langle V_I, V_{and}, V_{or}, E\rangle$ with vertices $V := V_I \cup V_{and} \cup V_{or}$ and edges E. V_I, V_{and} and V_{or} are disjoint sets called the *Initial* nodes, *AND* nodes, and *OR* nodes, respectively. That is, it is a directed graph with two types of vertices: the *AND* nodes and the *OR* nodes, as well as set (usually of size one) of initial nodes, which are *OR* nodes. The *Relaxed Causal Graph (RCG)* is a rooted And/Or graph associated with a planning problem. It has a single initial node, that intuitively corresponds to the initial state. The *AND* nodes correspond to actions and the *OR* nodes correspond to propositions. The preconditions of an action are connected to the action, and the action is connected to its effects.

Formally, it is a directed graph $\mathcal{G} = \langle s, V_{and}, V_{or}, E\rangle$ with vertices $V := \{s\} \cup V_{and} \cup V_{or}$ and edges E. $\{s\}, V_{and}$ and V_{or} are disjoint sets called the *start* node, *AND* nodes, and *OR* nodes, respectively. The *OR* nodes represent propositions and *AND* nodes represent actions. The start node s is a special *OR* node that represents the current (or initial) state. There is a special *OR* node t attached to the goal via a special AND node g or (a_{end}). See Figure 1 for an example.

More specifically – assuming we are in the initial state I and all propositions are false (I is empty):

- There are two special *OR* nodes s, t for *start* and *end* nodes, respectively.
- There is an *OR* node for every proposition $p \in P$.
- There is an *AND* node for every action $a \in A$.
- There is a special *AND* node g or (a_{end}) to connect the goal propositions to t.
- There is a directed edge $(v, a) \in E$ connecting every proposition v with every action a such that $v \in pre(a)$.
- There is a directed edge $(a, v) \in E$ connecting every action a with every proposition v such that $v \in add(a)$.
- There is a directed edge (s, a) if a is applicable in the current (initial) state. Notice that in this case (s, a) will be the only edge entering a.
- There is a directed edge $(v_g, a_{end}) \in E$ where v_g is a goal proposition.
- There is a directed edge $(a_{end}, t) \in E$.

We will use the following notations:

1. $pre(a) = \{v \in V_{or} : (v, a) \in E\}$. Contains all preconditions of a.
2. $add(a) = \{v \in V_{or} : (a, v) \in E\}$. Contains all the effects of action a.
3. $ach(v) = \{a \in V_{and} | \ v \in add(a)\}$. It contains all actions that can achieve v.
4. $consumers(v) = \{a \in V_{and} | \ v \in pre(a)\}$. It contains all actions that need v.

Now, we can extend the definition above to an arbitrary, non-empty initial state, as follows, starting from the graph generated for the empty initial state.

1. Remove all true (achieved) propositions (in the current state) from the graph.
2. Remove any action $a \in V_{and}$ for which $add(a) = \emptyset$.
3. If after step two an action $a \in V_{and}$ has $pre(a) = \emptyset$, connect it to the start node s by an edge (s, a).

A (delete-free) plan is a subgraph $J = \langle V^J, E^J\rangle$ of $\mathcal{G}$ which is said to *justify* $V_G \subseteq V$ if and only if the following are true for J:

1. $s \subseteq V^J$
2. $V_G \subseteq V^J$
3. $\forall a \in V^J \cap V_{and} : \forall \langle v, a\rangle \in E : v \in V^J \wedge \langle v, a\rangle \in E^J$
4. $\forall v \in V^J \cap V_{or} : \exists \langle a, v\rangle \in E : a \in V^J \wedge \langle a, v\rangle \in E^J$
5. J is acyclic.

As the name suggests, J is a justification for V_G if J contains a proof that all nodes in V_G are true (V_G are reachable via a hyperpath in J), under the assumption that s is true. The set V^J represents the nodes that are proven to be true by J, and the edges E^J represent the arguments for why they are true. The five conditions then state that: (1) the start node is true (2) all nodes in V_G must be proven true (3) *AND* nodes are proven true if all their predecessors are true (4) *OR* nodes are proven true if they have some true predecessor, and (5) the proof must be well-founded.

Notice that each J can be represented as $J = (a_1, a_2, \ldots, a_{i-1}, a_i, \ldots, a_k)$, where every $a_i \in V_{and}$ and $pre(a_i) \subseteq \{s\} \cup add(a_1) \cup \cdots \cup add(a_{i-1})$. An optimal relaxed plan is a lowest-cost justification (sub-graph) J that starts at s and contains (reaches) t and therefore the goal set. For a node $v \in V$ we say that v is *unreachable* in $\mathcal{G}$ if there is no justification sub-graph J that proves it (contain v).

Finally, we note that unlike the *relaxed planning graph* (RPG) (Hoffmann and Nebel 2001) the RCG contains a single copy of each proposition and action. In this respect it is closely related to the bi-level planning graph (Long and Fox 1999).

Pruning with the RCG

We now provide methods for carrying out the pruning ideas discussed earlier. Our first step will be to recognize the next fact landmark to pursue. Then, we explain how to recognize a small applicable disjunctive action landmark. Finally, we will explain how we can recognize that certain actions cannot co-exist in the same minimal plan, and how we can use it to provide additional pruning. We note that any (non-disjunctive) action landmark discovered can be applied as soon as possible.

Discovering and Ordering Fact Landmarks

We use the graph of strongly connected components (SCC) of the RCG to identify landmark orderings and, specifically, the next landmark to pursue. The SCCs of a graph $\mathcal{G}$ form a directed acyclic graph (DAG), denoted G_{scc}. A node in the G_{scc} is a SCC of $\mathcal{G}$. There is an edge from node n to n' if there is an edge from some node in the SCC represented by n to some node in the SCC represented by n'. An example appears in Figure 1.B. Since the G_{scc} is a DAG, we can sort its nodes in a topological order. As a secondary sort we use the depth of nodes established by the B-Visit procedure described in (Gallo et al. 1993) (a type of BFS for directed hypergraphs).

Algorithm 1 Sort topologically nodes of the RCG

```
1: sortLndmrx($\mathcal{G}$)  {$\mathcal{G}$ is a RCG}
2: Find fact landmarks using algorithm of (Keyder, Richter, and
   Helmert 2010)
3: $G_{scc} \leftarrow$ build from $\mathcal{G}$
4: $T \leftarrow$ Topological sort of $G_{scc}$
5: $T \leftarrow$ replace each SCC in $T$ with its related nodes from $\mathcal{G}$
   {(inner sort by depth of B-Visit)}
6: $T \leftarrow$ remove from $T$ nodes which are not landmarks
7: return $T$
```

The result of Algorithm sortLndmrx is an ordered list of landmarks, T. The first landmark l in T that has not been achieved in s is our candidate for "nearest" landmark to s. We note that fact and (non-disjunctive) action landmarks are found using algorithm of (Keyder, Richter, and Helmert 2010).

Finding Applicable Disjunctive Action Landmark

Theorem 1 can be used to reduce the branching factor for forward-search planners on delete-free problems substantially. To be effective, we need a method for finding small, applicable disjunctive action landmarks. Let us assume we have built the RCG of some delete-free problem starting with state s. Let $l \in L_s$ be the fact landmark we are pursuing (the first in the list above) for state s. Algorithm findDAL finds a disjunctive action landmark consisting of applicable actions for state s with respect to l.

The input to algorithm findDAL is the RCG $\mathcal{G}(s)$ for our current state and a fact landmark l. The set S which is being built, represents a sub-graph in $\mathcal{G}$ that contains all the minimal justification sub-graphs that contain/reach l (minimal in the sense of set inclusion). For this reason all the applicable

Algorithm 2 Find disjunctive action landmark

```
1: findDAL($\mathcal{G}(s), l$)  {$\mathcal{G}(s)$ is a current state RCG and $l$ is a fact
   landmark}
2: $Q = \emptyset,\ S = \emptyset$
3: for all $a \in ach(l)$ do
4:     $Q \leftarrow Q \cup \{a\},\ S \leftarrow S \cup \{a\}$
5: end for
6: while $Q \neq \emptyset$ do
7:     Select and remove $a \in Q$
8:     for all $v \in pre(a)$ do
9:         for all $e \in ach(v)$ do
10:            if $e \notin S$ then
11:                $Q \leftarrow Q \cup \{e\},\ S \leftarrow S \cup \{e\}$
12:            end if
13:        end for
14:    end for
15: end while
16: return all applicable $a \in S$
```

actions in S compose a disjunctive action landmark. S is being built by moving backwards from l on $\mathcal{G}$ and collecting all the actions that "touch" l. Then for each action a that was removed from Q, we examine all actions that can achieve a precondition of a. If the action was encountered for the first time, we add it to Q and S and continue to our next iteration. Therefore we remove each action only once and the complexity of this algorithm is at most $O(C_1 C_2 V_{and})$. Where C_1 is the largest $pre(a), a \in V_{and}(\mathcal{G})$ and C_2 is the largest $ach(v), v \in pre(a)$ where $a \in V_{and}(\mathcal{G})$. It is important to notice that the target/end node t is also a fact landmark and if we will use it in the algorithm we will get all the current applicable actions.

Sometimes, this method yields an applicable action landmark X that is not set-inclusion minimal. To further minimize X, we proceed as follows: (i) We iterate over the set $X \subseteq A_{app}$ returned from Algorithm findDAL. (ii) For each $a \in X$ we ask whether $X \setminus \{a\}$ is still a disjunctive action landmark, by performing the B-Visit procedure, mentioned before (a kind of BFS for directed hypergraphs), and checking if l or t are unreachable when the actions $X \setminus \{a\}$ are removed temporarily from $\mathcal{G}(s)$. Notice, this means the action a is in the RCG. (iii) If l or t are unreachable, the set $X \setminus \{a\}$ **is a disjunctive action landmark** , since $\mathcal{G} = \langle s, \{V_{and} \setminus X\} \cup a, V_{or}, E\rangle$ has no justification subgraph that includes t or l. Therefore, we can remove a from X and continue to the next iteration. We note that, the minimization process can theoretically find a disjunctive action landmark that reaches l but not t, because, if l is unreachable then t must be unreachable, since, l is a landmark. But, t can be unreachable while l remains reachable – this distinction will be important in next sections.

The result of this procedure will be a set $X' \subseteq X$ which is still a disjunctive action landmark and set-inclusion minimal.

Disjoint-Path Commitment

The next method complements the method of applicable disjunctive action landmarks, and follows the idea in Theo-

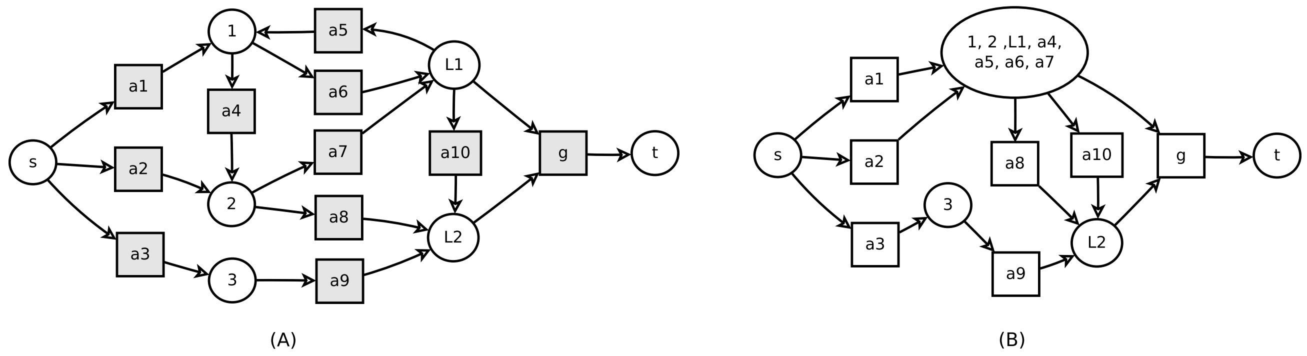

Figure 1: (A) RCG as an And/Or graph $\mathcal{G}$ of a planning problem: facts in white, actions in grey, L1 and L2 are the goal and hence fact landmarks, g is a special goal action. (B) G_{scc} of $\mathcal{G}$: nodes are SCC's (shapes are kept just as a visual aid)

rem 2. A naive way of using Theorem 2 would be to find all minimal plans for achieving landmark l from a parent state s_p. Then, partition these plans into sets of plans that are (action-wise) disjoint. In that case, if we just applied an action a_{last}, we will consider only plans from the partition containing a_{last}. As the number of minimal plans could be exponential, we require a more clever approximation technique. We will use the actions achieving a fact landmark as "markers" for a subset of all minimal plans.

Let the current state be s reached by a_{last} from state s_p. Let M be all the minimal plans from state s_p to a fact landmark l. Let $M(a)$ be all minimal plans that use action a. Let A_{app} be all applicable actions that can get a new proposition in state s. Let $A_l = \{a_1, \cdots, a_k\}$ be all the actions that achieve l, then:

- $M(a_i) \cap M(a_j) = \emptyset$, where $a_i, a_j \in A_l, a_i \neq a_j$.

  Notice that even if a_i achieves a precondition for a_j, a minimal plan to l that include a_i would not include a_j. For example, in Figure 2.A: $M(a_5) = \{(a_1, a_5)\}$, $M(a_6) = \{(a_1, a_4, a_6), (a_2, a_6)\}$, $M(a_7) = \{(a_3, a_7)\}$.
- If there exist an action $a_L \in A_l$ and $a \in A_{app}$, where $M(a_{last}) \cap M(a_L) \neq \emptyset \wedge M(a) \cap M(a_L) \neq \emptyset$ then it is possible that $M(a_{last}) \cap M(a) \neq \emptyset$, but if there is no such a_L then it is obvious that $M(a_{last}) \cap M(a) = \emptyset$. For example, in Figure 2.B, $M(a_{last}) = \{(a_2, a_6)\}$ and there is only one $a_L = a_6$ and one applicable action a_6 where $M(a_2) \cap M(a_6) \neq \emptyset \wedge M(a_6) \cap M(a_6) \neq \emptyset$.
- Let $\mu(a) = \{a_L | a_L \in A_l, M(a) \cap M(a_L) \neq \emptyset\}$, i.e. the set of all actions that can achieve l and reside in a minimal plan with action a. Let $a \in A_{app}$, if $\mu(a_{last}) \cap \mu(a) = \emptyset$, then $M(a_{last}) \cap M(a) = \emptyset$.

The set $\mu(a)$ is approximated by a super set $\mu'(a) \supseteq \mu(a)$ in algorithm labelArcs using a back propagation procedure that will propagate backwards $a_L \in A_l$ as labels. So, instead of looking at state s (reached by a_{last} from s_p) and verifying whether for some applicable action $a \in A_{app}$, $M(a_{last}) \cap M(a) \neq \emptyset$, we ask if $\mu'(a_{last}) \cap \mu'(a) \neq \emptyset$.

We call the division of M according to the actions achieving the fact landmark l a *disjoint-path-sets* where the disjointness refers to the actions achieving a fact landmark. To use the idea of disjoint-path-sets in practice, we use a two step procedure: (i) Algorithm labelArcs is executed before the search begins. It propagates labels of arcs achieving fact landmarks backwards on the RCG $\mathcal{G}(I)$ and computes for each action $v_a \in V_{and}$ a set $\mu'(v_a)$ where $\mu(v_a) \subseteq \mu'(v_a)$. When it terminates we can easily check for each v_a which are the actions a_L that achieve some fact landmark l and $\mu'(v_a) \cap \mu'(a_L) \neq \emptyset$. (ii) Algorithm filterAppOps is executed during search (if a_{last} did not achieve a new fact landmark) and compares an applicable disjunctive action landmark (with respect to l) with the last action taken a_{last} to see which actions can be safely pruned, i.e. they do not share a label of a landmark achieving action with a_{last}.

Algorithm 3 Arc labeling for disjoint-path-sets

```
1: labelArcs($\mathcal{G}$)  {$\mathcal{G}$ is a RCG}
2: $S = \emptyset$
3: $G_{scc} \leftarrow$ build from $\mathcal{G}$
4: for all $a \in A$ do
5:     if there is a fact landmark $l \in add(a)$ then
6:         $n \leftarrow scc(a)$  {$scc(a)$ denotes the SCC node containing $a$ in $G_{scc}$}
7:         $dp(n) \leftarrow \{a\}$  {$dp(n)$ set of arc labels of $n$}
8:         $S \leftarrow \{n\}$
9:     end if
10: end for
11: while $S \neq \emptyset$ do
12:     Select and remove $n \in S$
13:     for all $n' \in predeccessors(n)$ do
14:         if $dp(n) \not\subseteq dp(n')$ then
15:             $dp(n') \leftarrow dp(n') \cup dp(n)$
16:             $S \leftarrow S \cup \{n'\}$
17:         end if
18:     end for
19: end while
```

Algorithm labelArcs starts by constructing the G_{scc} of $\mathcal{G}$.[2] Next, we mark each SCC node $n \in G_{scc}$ by labels which are the arcs that reside in the SCC n – but only if these arcs are fact landmark achieving actions. More specifically, if an action $a \in V_{and}$ has a fact landmark $l \in add(a)$, we add the

[2]It is possible, in theory, to devise an algorithm that will get the same output going over $\mathcal{G}$, but when we have many cycles in $\mathcal{G}$, going over G_{scc} will be quicker.

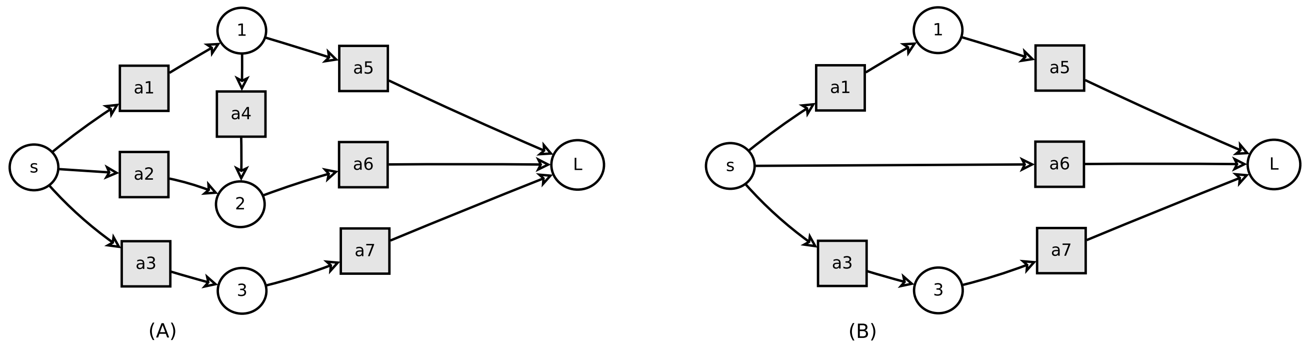

Figure 2: (A) A sub-graph of a RCG of a planning problem: propositions in white, actions in grey, L is fact landmarks that needs to be achieved. (B) The same sub-graph after executing action a_2.

label a to the set $dp(n)$, the set of labels for SCC node n in G_{scc} The SCC nodes that were updated will also be inserted to the set S, which will serve for the back propagation procedure that takes place over the G_{scc}. Now, we start the back propagation procedure, taking out SCC nodes from S, and propagating information to their predecessors in the G_{scc}. An update will take place if $dp(n) \not\subseteq dp(n')$ where n' is a predecessor of n in G_{scc}. If there was an update we need to propagate it further so we add n' to S. At the end of the run of algorithm labelArcs, for each action $a \in A$ we can check all the actions in $dp(scc(a))$ that "enter" a fact landmark and are reachable from a.

For example, the graph in Figure 2.A will have the following labels at the end of the run of Algorithm labelArcs (notice that the G_{scc} of this graph will look the same): $dp(scc(a_1)) = \{a_5, a_6\}$, $dp(scc(a_2)) = \{a_6\}$, $dp(scc(a_3)) = \{a_7\}$, $dp(scc(a_4)) = \{a_6\}$, $dp(scc(a_5)) = \{a_5\}$, $dp(scc(a_6)) = \{a_6\}$, $dp(scc(a_7)) = \{a_7\}$.

Algorithm 4 Filter focused applicable ops for disjoint-path

```
1: filterAppOps($\mathcal{G}, l, a_{last}$)  {$\mathcal{G}$ is a RCG and $l$ is a fact landmark, $a_{last}$ is the last action taken to reach the current state}
2: $F \leftarrow$ A($\mathcal{G}, l$)  {return $X \subseteq A_{app}(s)$}
3: if $a_{last}$ didn't achieve any fact landmark then
4:     $S = \emptyset$  {new set of actions for expansion}
5:     $dpl \leftarrow dp(a_{last}) \cap ach(l)$  {$dpl$ = committed labels}
6:     for all $a \in F$ do
7:         if $dp(a) \cap dpl \neq \emptyset$ then
8:             $S \leftarrow S \cup \{a\}$
9:         end if
10:    end for
11:    return S
12: end if
13: return F
```

To use the labels gathered in Algorithm labelArcs during search, we use Algorithm filterAppOps. Algorithm filterAppOps filters out applicable actions (returned from algorithm findDAL) which we can ignore because of our commitment to some set of paths. This filtration is with respect to the last action a_{last} taken to reach from s_p to the current state s. We filter out all actions that will never appear with a_{last} in a minimal plan that starts at s_p and achieve l. This is the reason why filtration can occur only when a_{last} did not achieve a fact landmark. If it did, we need first to consider all sets of paths before we commit. Notice, that for this procedure to be sound the set $X \subseteq A_{app}$ that we get from algorithm findDAL, must be a disjunctive action landmark with respect to the fact landmark l and not only t (the end node).

Summary/Putting It Altogether

We now summarize and demonstrate the entire procedure using the example $\mathcal{G}$ in Figure 1.

Preprocessing:

1. Find fact and action landmarks (Keyder, Richter, and Helmert 2010): we find $\{L1, L2, t\}$ to be fact landmarks of the problem. The only action landmark is g (a_{end}).
2. Run Algorithm sortLndmrx to order fact/action landmarks: The order of fact landmarks is: $(L1, L2, t)$. Notice that the SCC of $L1$ must appear before the SCC of $L2$ in a topological sort of the G_{scc} of $\mathcal{G}$. Both appear before t.
3. Run Algorithm labelArcs to label disjoint actions: Before propagation, the SCC of $L1$ is labeled with $\{a_6, a_7\}$. a_8, a_9, a_{10}, which enter $L2$, are in independent SCCs, so each is labeled by itself. g (a_{end}) is also labeled by itself. Next, we add to S all SCC's that have an action that achieves some fact landmark: $S = \{scc(a_6), scc(a_8), scc(a_9), scc(a_{10}), scc(g)\}$. As an example of an update during propagation, assume the $scc(a_6)$ is taken out of S. Thus, we need to check for both $scc(a_1), scc(a_2)$ if their dp set already contains the labels in $dp(scc(a_6))$. If it does not, as in the first time, we propagate the labels and add the updated dp sets to S.

   After backward propagation we have: $dp(scc(a_1)) = \{a_6, a_7, a_8, a_{10}, g\}$, $dp(scc(a_2)) = \{a_6, a_7, a_8, a_{10}, g\}$, $dp(scc(a_3)) = \{a_9, g\}$.

During search – Constructing filtered action state for state s:

1. Apply any applicable action landmark immediately.
2. If none exists, find closest fact landmark l that was not yet achieved. Specifically, if our current state is the initial state, we choose $L1$.

3. Build RCG for current state s $\mathcal{G}(s)$.
4. Run Algorithm findDAL to get an applicable disjunctive action landmark X with respect to l: We get back $X = \{a_1, a_2\}$.
5. Try to minimize X by iterating all actions $a \in X$ (minimize with respect to l): Nothing to remove from $X = \{a_1, a_2\}$.
6. Run Algorithm filterAppOps to filter the set X: In the initial state we don't have a last action, so let us consider two scenarios: (i) we performed a_2 in s, the returned set $X = \{a_1, a_7\}$, both a_1 and a_7 have the same labels with a_2, so there is nothing to filter out. (ii) we performed (a_1, a_6, a_2) and achieved $L1$ using a_6. Thus, we are now pursuing L_2 and the return set $X = \{a_3, a_8, a_{10}\}$. a_8 and a_{10} both have a common label with a_2. However, a_3 does not have a common label with them, so we can filter it out.
7. Expand s using the filtered set.

To get possibly even more pruning we can alter step 5, to minimize the set X once with respect to t – call it X_1, and once with respect to l – call it X_2. The set X_2 can then continue to stage 6, and in stage 7 we will take the minimal set for expansion.

Duplicates in A*: Our pruning technique is agnostic to the search algorithm used, but obtaining an optimal plan requires an appropriate search algorithm, with A* being the natural choice. Here, some caution is required. One of our pruning technique (disjoint-path commitment) uses the last action executed to prune the actions considered in the current state. Planning algorithms using A* maintain this information, so no change in the implementation of A* is required. However, a delicate question arises in the case of graph search. What happens if we reach the same state s with two path ending with different actions: first a and later a'? Clearly, if the path to s ending in a' has better g value, we need to re-expand s pruning based on a'. If both paths have the same g value, it may seen that we need to reconsider actions that were pruned by a but are not pruned by a'. Fortunately, we can show that no re-expansion is needed in that case.

Theorem 3. *When using Theorem 2 for pruning in A* search, there is no need to reopen an already visited state, unless we reached it with a lower f cost.*

Empirical Results

We implemented the ideas described in the previous sections on the FD framework (Helmert 2006). We evaluated their performance both in the context of blind search and heuristic search (using A*) using two of best existing admissible heuristics: LM-Cut and Merge& Shrink (Helmert, Haslum, and Hoffmann 2007), comparing them to the performance of these optimal planning algorithms without our pruning method.

As delete-free benchmark domains we used the set of problems available in the FD distribution to which a new delete-removal procedure can be applied.[3] These domains include domains that are known to be solvable, like blocks, and some that are considered very hard to solve even in their delete-free form, like freecell.

Table 1: search time limit used is 5 minutes per problem. As expected, the results show that LM-Cut is superior to Merge& Shrink in delete-free problems. Freecell is the only domain in which pruning+X had lower coverage than X, specifically, pruning reduces the coverage of LM-Cut. In all other domains, pruning improves coverage and reduces the number of expansion. It is interesting to see that, except for the freecell domain, pruning+blind behaves better than Merge& Shrink. In fact, the only reason why pruning+Merge& Shrink does not solve all of the blocks problems is because Merge& Shrink did not finish its pre-process step. It is therefore not a surprise that pruning+Merge& Shrink behaves only slightly better than pruning+blind.

Table 2: search time limit used is 30 minutes per problem. We continued to explore the performances of LM-Cut alone versus pruning+LM-Cut. As expected, we can see that in all domains pruning reduced the number of expansions, in some domains significantly compared to LM-Cut alone, like rovers and depot. In these domains the search-time score was also significantly better. But not in all domains where the expansion-score is noticeably better, does the search-time score behaves the same. In general, the computation overhead depends on the domain's (or specific instance) structure. It seems that the most time intensive overhead occurs where the disjunctive action landmark is very large and can not be minimized. This makes the minimization step (minimal set-inclusion) both time consuming and futile.

We also tried our pruning procedure on the seed-set problems with zero cost actions. Zero cost actions are challenging to existing solvers, and as reported by Gefen and Brafman (2011), they cannot solve any instance of this problem. Pruning, in this case, is not sufficient to overcome this method. However, when we augment pruning with a simple procedure that applies all applicable zero-cost action, then pruning+blind-search+zero-cost-procedure, solves all the seed-set problems.

Summary and Related Work

Building on a number of observations with respect to delete-free planning, we extended the use of the RCG (Keyder, Richter, and Helmert 2010) to provide more information in preprocessing time and during planning. Specifically, we showed how to generate an ordered list of fact landmarks, how to generate small disjunctive action landmarks from them, and how to update this information during search. By focusing on the actions in these landmarks, we obtained some initial pruning. Then, we described how information can be propagated in the RCG in order to identify disjoint paths and we utilized this information to provide additional pruning which is based on the idea of focusing on one path

[3]This procedure, not implemented by us, has difficulty with some formats, and so this is only a subset of available domains.

Domain (# of problems)	Pruning + blind			No pruning + M&S			Pruning + M&S			No pruning + LM-cut			Pruning + LM-cut		
	a	b	c	a	b	c	a	b	c	a	b	c	a	b	c
blocks(35)	35	643	0.96957	24	15945	31.31	24	339	0.99	35	18052	1	35	643	0.96
freecell(80)	0	–	–	2	256081	2.98	0	–	–	4	54710	1	1	3379	0.98
gripper(20)	6	974166	5564.16	3	242095	4493.78	7	387312	1669.07	20	960	1	20	960	1
logistics00(28)	22	1133	1.44	14	1385231	2974.63	22	1039	1.33	23	741	1	28	1075	1
logistics98(35)	5	4310	13.41	2	4431	128.50	6	14486	89.18	9	833	1	11	2614	0.80
rovers(40)	12	108096	12.60	7	1935626	710.13	14	204795	0.93	12	836893	1	19	24501	0.36

Table 1: Search time limit per problem: 5 minutes. Column a: # of solved problems. Column b: total number of expansions for all solved instances. Column c: Avg improvement w.r.t. baseline of No pruning + Lm-cut, computed as $(\sum_i X_i/Y_i)/Z$ where: i – problem solved by both planners; Z – # of problems solved by both planners; X_i – # of expansion executed by current planner in problem i; Y_i – # of expansion for No pruning + Lm-cut planner. Lower values are better.

Domain	No pruning + LM-cut			Pruning + LM-cut		
	a	b	c	a	b	c
blocks(35)	35	98.57	98.25	35	100	100
freecell (80)	6	54.20	63.15	2	56.00	26.23
Gripper (20)	20	100	99.62	20	100	100
Logistics00 (28)	23	100	100	28	100	100
Logistics98 (35)	10	94.43	90.46	16	100	95.09
rovers (40)	13	59.62	71.87	23	100	100
depot (22)	7	59.1	62.88	12	97.18	96.38
driverlog (20)	14	88.31	92.60	15	94.32	92.04
Miconic (150)	150	99.99	94.30	150	99.99	84.41
Mystery (30)	26	91.98	85.04	26	96.57	79.11
pipesworld-notankage (50)	17	86.59	93.01	9	86.66	83.48
pipesworld-tankage (50)	10	90.6	90.06	9	92.79	79.12

Table 2: Search-time limit per problem: 30 minutes. Column a: # of solved problems. Column b: expansion score avg. Column c: search-time score avg. b, c values are calculated for instances solved by both planners. Scores based on (Röger and Helmert 2010). Higher values are better. 100 is highest.

at a time. Our procedure show variable computational overhead, working well on many domains (extremely well on some) and less on others. For some domains the overhead is too large and the procedure performs worse than heuristic search alone.

To our knowledge no other work in the literature focuses on pruning in delete-free problems. However, there has been quite a few recent papers dealing with action pruning: Stratified Planning (SP) (Chen, Xu, and Yao 2009), Expansion core (EC) (Chen and Yao 2009), Bounded intention Planning (BIP) (Wolfe and Russell 2011), Symmetries (Coles and Coles 2010; Fox and Long 1999; Pochter, Zohar, and Rosenschein 2011). Of these, the only work we are aware of that is landmark based is the SAC algorithm (Xu et al. 2011). This algorithm is related to the notion of *stubborn sets* (explained in the same paper). The SAC algorithm uses a disjunctive action landmark which is then extended (to a stubborn set) to prevent conflicts between actions in the extended set to actions outside the extended set. This extended set can then be used as the expansion set of some state s, which means they can prune all other applicable action. This pruning keeps an optimal solution in the search tree because of the properties of stubborn sets. The relation of stubborn sets to delete-free planning is unclear. Nevertheless, since this method is complete in general planning, our Algorithm findDAL can be viewed as a simplification of SAC to the delete-free space. That being said, our algorithms for finding small disjunctive action landmarks that utilize a topological ordering of landmarks, finds all applicable actions related to a fact landmark that is close to search state s, and applies the the minimization step, could be used by the SAC algorithm to find smaller expansion sets. However, the extension step may potentially insert back some of the actions.

There are various potential ways our pruning procedure could be improved. First, improving the minimization step (minimal set-inclusion) would have immediate impact on currently weak domains. Another option would be to recognize such domains and avoiding the mentioned step. Other option is to analyze the topology of the RCG at each state (during search) instead of doing so only in the initial state. Another direction is extending these algorithms to general planning: some of the algorithms developed could immediately be used in regular planning (e.g., extending the SAC algorithms), the idea of disjoint path, may be generalizable to regular planning.

Acknowledgements: The authors were partly supported by ISF Grant 1101/07, the Paul Ivanier Center for Robotics Research and Production Management, and the Lynn and William Frankel Center for Computer Science.

References

Brafman, R. I., and Domshlak, C. 2006. Factored planning: How, when, and when not. In *AAAI*. AAAI Press.

Bylander, T. 1994. The computational complexity of propositional strips planning. *Artif. Intell.* 69(1-2):165–204.

Chen, Y., and Yao, G. 2009. Completeness and optimality preserving reduction for planning. In *IJCAI*, 1659–1664.

Chen, Y.; Xu, Y.; and Yao, G. 2009. Stratified planning. In *IJCAI*, 1665–1670.

Coelho, H.; Studer, R.; and Wooldridge, M., eds. 2010. *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Coles, A. J., and Coles, A. 2010. Completeness-preserving pruning for optimal planning. In Coelho et al. (2010), 965–966.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In Dean, T., ed., *IJCAI*, 956–961. Morgan Kaufmann.

Gallo, G.; Longo, G.; Pallottino, S.; and Nguyen, S. 1993. Directed hypergraphs and applications. *Discrete Applied Mathematics* 42(2-3):177–201.

Gefen, A., and Brafman, R. I. 2011. The minimal seed set problem. In Bacchus, F.; Domshlak, C.; Edelkamp, S.; and Helmert, M., eds., *ICAPS*. AAAI.

Gefen, A., and Brafman, R. I. 2012. Pruning methods for optimal delete-free planning. Technical report, Department of Computer Science Ben-Gurion University.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A. E.; Cesta, A.; and Refanidis, I., eds., *ICAPS*. AAAI.

Helmert, M., and Röger, G. 2008. How good is almost perfect? In Fox, D., and Gomes, C. P., eds., *AAAI*, 944–949. AAAI Press.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In Boddy, M. S.; Fox, M.; and Thiébaux, S., eds., *ICAPS*, 176–183. AAAI.

Helmert, M. 2006. The fast downward planning system. *J. Artif. Intell. Res. (JAIR)* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: fast plan generation through heuristic search. *J. Artif. Int. Res.* 14(1):253–302.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *J. Artif. Intell. Res. (JAIR)* 22:215–278.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for and/or graphs. In Coelho et al. (2010), 335–340.

Long, D., and Fox, M. 1999. Efficient implementation of the plan graph in stan. *J. Artif. Intell. Res. (JAIR)* 10:87–115.

Pochter, N.; Zohar, A.; and Rosenschein, J. S. 2011. Exploiting problem symmetries in state-based planners. In Burgard, W., and Roth, D., eds., *AAAI*. AAAI Press.

Röger, G., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In *ICAPS*, 246–249.

Wolfe, J., and Russell, S. J. 2011. Bounded intention planning. In Walsh, T., ed., *IJCAI*, 2039–2045. IJCAI/AAAI.

Xu, Y.; Chen, Y.; Lu, Q.; and Huang, R. 2011. Theory and algorithms for partial order based reduction in planning. *CoRR* abs/1106.5427.