# Pruning Methods for Optimal Delete-Free Planning

Avitan Gefen Ronen I. Brafman

Department of Computer Science

Ben-Gurion University of The Negev, Israel

June 21, 2012

## Outline

1 Motivation
2 Overview
3 RCG
4 Delete-Free Planning
5 Find Action Landmark
6 Disjoint-Path
7 Summery of Procedure
8 Empirical results

## Why pruning?

- Heuristic search: method of choice for solving satisfycing and optimal planning problems.
- Relaxation-based methods: extremely popular, effective, and influential.
- **But,** good, but imperfect heuristics estimates, such as $h+$, are insufficient for solving hard problems (e.g., *How good is almost perfect? (Helmert and Roger 2008)*.
- Additional tools needed: pruning methods, decomposition methods etc.

## Why delete-free?

We focus on delete-free problems for three reasons:

- Some naturally delete-free planning problems cannot be solved by standard planning techniques (e.g., minimal seed-set).
- New methods for delete-free planning could inspire better delete-free heuristics.
- Ideas used for delete-free planning could inspire new techniques for standard planing.

## Contribution

- New pruning methods that enhance the coverage of state-of-the-art algorithms when applied to delete-free problems.
- New ideas on landmark orderings and commitment to sub-goals
- Interesting exploitation of the problem's Causal/Relaxed And/Or Graph

## Outline

1 Motivation
2 Overview
3 RCG
4 Delete-Free Planning
5 Find Action Landmark
6 Disjoint-Path
7 Summery of Procedure
8 Empirical results

## Summery of Procedure: Preprocess

1. Find fact and action landmarks *(Keyder, Richter, and Helmert 2010).*
2. *Order fact/action landmarks (algorithm sortLndmrx).*
3. *Label disjoint actions (algorithm labelArcs).*

## Summery of Procedure: Pruning in each state

**During search, for a current state $s$:**

1. Apply any applicable action landmark immediately.
2. If none exists:
   1. Build RCG (And/Or graph) for current state $s$, $\mathscr{G}(s)$.
   2. Find closest fact landmark $l$ that was not yet achieved.
   3. Compute applicable disjunctive action landmark $X$ w.r.t. $l$
   4. Minimize $X$ to find set-inclusion minimal disjunctive action landmark.
   5. Filter the set $X$ using path-commitment information
   6. Expand $s$ using the filtered set.

## Outline

1 Motivation

2 Overview

3 RCG

4 Delete-Free Planning

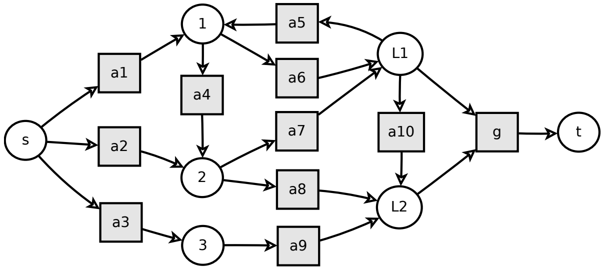
5 Find Action Landmark

6 Disjoint-Path

7 Summery of Procedure

8 Empirical results

## The Relaxed Causal And/Or Graph – *(Keyder, Richter, and Helmert 2010)*

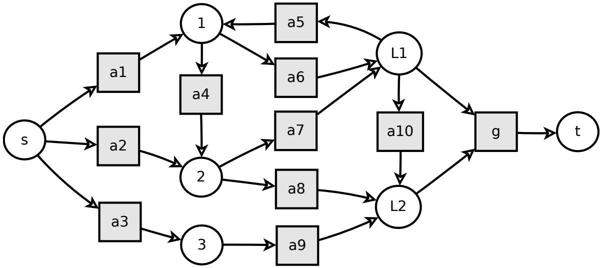


(A)

**The *Relaxed Causal Graph (RCG)*:**

- $\mathscr{G} = \langle V_I, V_{and}, V_{or}, E \rangle$.
- A rooted And/Or graph associated with a planning problem.
- Single initial node, that corresponds to the initial/current state.
- Special goal (AND) node (g) that achieves end node (t).

## The Relaxed Causal And/Or Graph

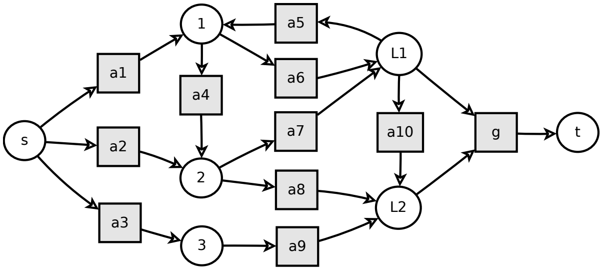


(A)

The *Relaxed Causal Graph (RCG)*:

- *AND* nodes correspond to actions (grey).
- *OR* nodes correspond to variables/fluents (white).
- An action's parents are its preconditions.
- An action's children are its effects.

## The Relaxed Causal And/Or Graph

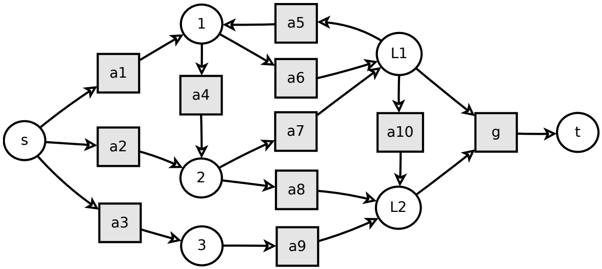


(A)

**The *Relaxed Causal Graph (RCG)*:**

- $pre(a) = \{v \in V_{or} : (v, a) \in E\}$. Contains all preconditions of $a$.
- $pre(a_9) = \{3\}$.
- $add(a) = \{v \in V_{or} : (a, v) \in E\}$. Contains all the effects of action $a$.
- $add(a_9) = \{L2\}$.

## The Relaxed Causal And/Or Graph

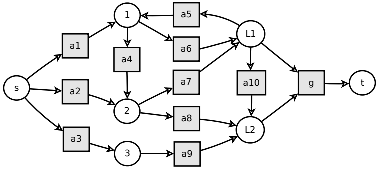


(A)

**The *Relaxed Causal Graph (RCG)*:**

- $ach(v) = \{a \in V_{and} | \; v \in add(a)\}$. Contains all actions achieving $v$.
- $ach(2) = \{a_2, a_4\}$.
- $consumers(v) = \{a \in V_{and} | \; v \in pre(a)\}$. Contains all actions using $v$.
- $consumers(2) = \{a_7, a_8\}$.

## The Relaxed Causal And/Or Graph



(A)

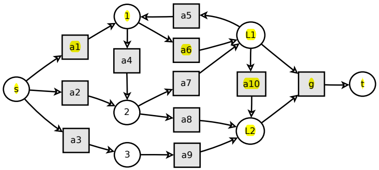
**Relaxed plan: subgraph $J = \langle V^J, E^J \rangle$ of $\mathscr{G}$ that *justifies* $V_G \subseteq V$**

- Includes $s$ and $t$
- If And node $a \in V^J$, then $pre(a) \cup add(a) \in V^J$
- If Or node $p \in V^J$, then $ach(p) \cap V^J \neq \emptyset$

**Example**

- $V^J \cap V_{or} = \{s, 1, L1, L2, t\}$.
- $V^J \cap V_{and} = \{a_1, a_6, a_{10}, g\}$.

## The Relaxed Causal And/Or Graph



(A)

**Relaxed plan: subgraph $J = \langle V^J, E^J \rangle$ of $\mathscr{G}$ that *justifies* $V_G \subseteq V$**

- Includes $s$ and $t$
- If And node $a \in V^J$, then $pre(a) \cup add(a) \in V^J$
- If Or node $p \in V^J$, then $ach(p) \cap V^J \neq \emptyset$

**Example**

- $V^J \cap V_{or} = \{s, 1, L1, L2, t\}$.
- $V^J \cap V_{and} = \{a_1, a_6, a_{10}, g\}$.

## Outline

1. Motivation
2. Overview
3. RCG
4. Delete-Free Planning
5. Find Action Landmark
6. Disjoint-Path
7. Summery of Procedure
8. Empirical results

## Properties of Delete-Free Planning Problems

**Fact Landmark:**

- A *fact landmark* for a state *s* is a fact (= variable value) that holds at some point in every legal plan from state *s* to the goal.
- A fact landmark is an *OR* node that is part of every justification sub-graph from the start node to the target node.

## Properties of Delete-Free Planning Problems

### Action Landmark

- an *action landmark* for a state $s$ is an action that must be part of any plan from $s$ to the goal.

### Disjunctive Action Landmark

- A *disjunctive action landmark* for state $s$ is a set of actions, at least one of which appears in any legal plan from $s$ to the goal.
- In the RCG, a disjunctive action landmark is a set of actions $A_{dal} = \{a \in V_{and}\}$ s.t. $\mathcal{G} = \langle s, V_{and} \setminus A_{dal}, V_{or}, E\rangle$ has no justification sub-graph that includes $t$.

## Properties of Delete-Free Planning Problems

$L_I$: the set of fact landmarks for the initial state
$L_s$: the set of fact landmarks for state $s$.

**Observations:**

- Propositions achieved remain true forever.
- Applicable actions always remain applicable since their preconditions remain true.
- The order of actions in a specific plan $(a_1, a_2, \dots)$ is not important as long as the plan is valid – $pre(a_i) \subseteq a_{i-1}(\cdots(a_1(I))\cdots)$.
- We can immediately apply any applicable action landmark.
- For any state $s$ reachable from the initial state, (1) $L_s \subseteq L_I$.
  (2) $L_I \setminus L_s$ are the landmarks achieved so far (on route to state $s$).

## Properties of Delete-Free Planning Problems

**Definition:**

- A plan $\pi$ for $G$ is **minimal** if no strict subset of $\pi$ is a plan for $G$.

We can focus on minimal plans because every optimal plan is minimal or has a minimal subplan.

## Properties of Delete-Free Planning Problems

**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 $l_1, \ldots, l_k$ of $L$ and a minimal plan $\pi$ for $\Pi$ such that $\pi = \pi_1, \ldots, \pi_k$, where $\pi_i$ is a minimal plan for $(P, A, \pi_{i-1}(\cdots(\pi_1(I))\cdots), l_i)$.

## Properties of Delete-Free Planning Problems

**Corollary 1**

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

**Practical implication:**

Instead of planning for $G$, we can plan for $l$, and maintain minimality.

## Properties of Delete-Free Planning Problems

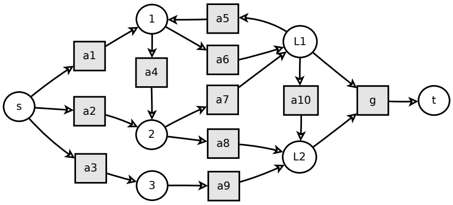
**Sort topologically nodes of the RCG**

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

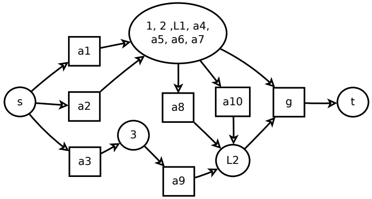
**Description:**

- We topologically order the fact landmarks of $\mathscr{G}$ using its $G_{scc}$.
- As a secondary sort we use inner sort by depth of B-Visit (*Gallo et al. 1993*) (a type of BFS for directed hypergraphs).

## Properties of Delete-Free Planning Problems



(A)



(B)

- (A) RCG of a planning problem.
- (B) $G_{scc}$ of $\mathscr{G}$: nodes are SCC's (shapes are kept as a visual aid)

## Outline

1 Motivation
2 Overview
3 RCG
4 Delete-Free Planning
5 Find Action Landmark
6 Disjoint-Path
7 Summery of Procedure
8 Empirical results

## First pruning method - Find Action Landmark

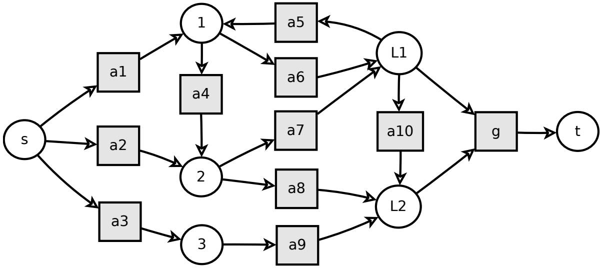
**Intuition:**

We can focus on actions within an **applicable** disjunctive landmark, pruning all other actions.

**Intuition:**

By focusing on the closest landmark $l$, rather than on $G$ or a more distant landmark, we are likely to find a smaller disjunctive action landmark, and will be able to prune more actions.

## Find Applicable Disjunctive Action Landmark



(A)

- In (A), $\{a_1, a_2\}$ is an applicable disjunctive action landmark for $L1$. Therefore, we can prune action $a_3$.

## First pruning method - Find Action Landmark

**Algorithm findDAL:**

- The input to algorithm *findDAL* is the RCG $\mathscr{G}(s)$ for our current state and a fact landmark *l*.
- We move backwards from *l*, collecting actions in a backward chaining manner.
- The set of actions which is being built, represents a sub-graph in $\mathscr{G}$ that contains all the minimal justification sub-graphs that contain/reach *l* (minimal in the sense of set inclusion).
- Therefore the applicable actions in the sub-graph compose a disjunctive action landmark (the output).
- Focusing on closest landmark vs. goal or an arbitrary landmark considerably reduces the size of the DAL in practice!

## Find Applicable Disjunctive Action Landmark

**Find disjunctive action landmark**

```
1: findDAL($\mathscr{G}(s), l$)  {$\mathscr{G}(s)$ 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 $a \in S$
```

## Find Applicable Disjunctive Action Landmark

### Set-inclusion minimal disjunctive action landmark

If the applicable action landmark $X$ returned from algorithm *findDAL* is not set-inclusion minimal, we can try to minimize $X$ by iterating and trying to remove actions from $X$.

### Obtaining a set-inclusion minimal landmark:

1. Iterate over the set $X \subseteq A_{app}$ returned from Algorithm *findDAL*.
2. For each $a \in X$ we ask whether $X \setminus \{a\}$ is still a disjunctive action landmark:
    1. We remove $X \setminus \{a\}$ from the graph
    2. If $I$ or $t$ are unreachable, $X \setminus \{a\}$ **is a disjunctive action landmark** and we can replace $X$ with $X \setminus \{a\}$ .

## Outline

1 Motivation

2 Overview

3 RCG

4 Delete-Free Planning

5 Find Action Landmark

6 Disjoint-Path

7 Summery of Procedure

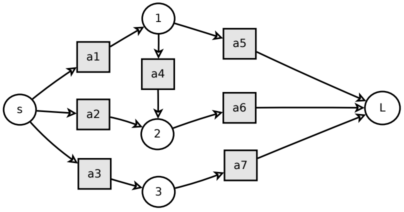
8 Empirical results

## Disjoint-Path Commitment For Fact Landmark

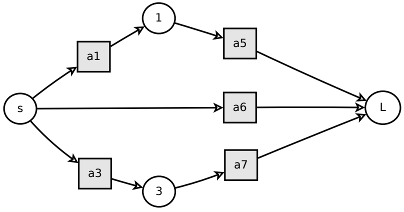
During A* search, if there are only two disjoint plans that reach some landmark *l*, each optimal plan will contain actions only from one of these plans. However:

- During search (especially in delete-free planning), after executing some actions from the first plan, we also have the opportunity of branching on actions from the second plan.
- However, there's no need to consider both branches each time. We can safely focus on two distinct branches that correspond to the two plans without interleaving actions from both.

## Disjoint-Path Commitment For Fact Landmark



(A)

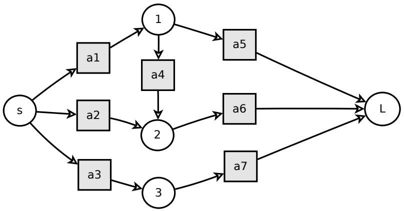


(B)

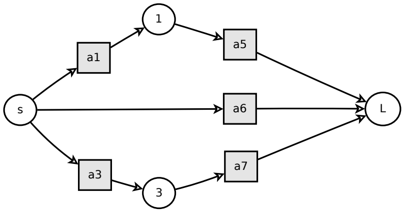
**RCG:**

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

## Disjoint-Path Commitment For Fact Landmark



(A)



(B)

**RCG:**

- Notice that in (A), there is no minimal plan containing both $a_2, a_3$ or both $a_2, a_1$.
- Therefore, after executing action $a_2$, we can in theory prune $a_1, a_3$.

## Disjoint-Path Commitment For Fact Landmark

**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 $l$.

## Disjoint-Path Commitment For Fact Landmark

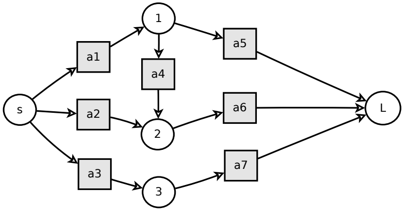
**Problem**

The number of minimal plans can be exponential.

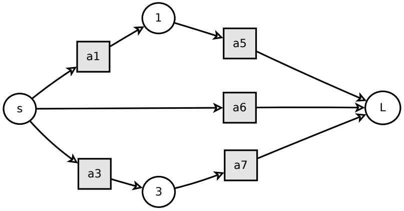
**Solution**

Approximation – using actions achieving a fact landmark as “markers”.

## Disjoint-Path Commitment For Fact Landmark



(A)



(B)

### Algorithms *labelArcs and filterAppOps*

- We propagate labels of actions achieving a fact landmark backwards.
- $\mu'(a_1) = \{a_5, a_6\}$, $\mu'(a_2) = \{a_6\}$, $\mu'(a_3) = \{a_7\}$.
- If $\mu'(a_{last}) \cap \mu'(a) = \emptyset$ we can prune $a$.
- In (B), $a_{last} = a_2$ so we can prune $a_3$.
- Notice that, $\mu'(a_2) \cap \mu'(a_1) \neq \emptyset$ and therefore we cannot prune $a_1$.

## Disjoint-Path Commitment For Fact Landmark

**Arc labeling for disjoint-path-sets**

```
1: labelArcs($\mathscr{G}$)  {$\mathscr{G}$ is a RCG}
2: $S = \emptyset$
3: $G_{scc} \leftarrow$ build from $\mathscr{G}$
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
```

## Disjoint-Path Commitment For Fact Landmark

**Filter focused applicable ops for disjoint-path**

```
1: filterAppOps($\mathscr{G}, l, a_{last}$)  {$\mathscr{G}$ 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($\mathscr{G}, l$)  {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
```

## Outline

1 Motivation
2 Overview
3 RCG
4 Delete-Free Planning
5 Find Action Landmark
6 Disjoint-Path
7 Summery of Procedure
8 Empirical results

## Summery of Procedure

**In the preprocessing stage we do the following:**

1. Find fact and action landmarks *(Keyder, Richter, and Helmert 2010).*
2. *Order fact/action landmarks (algorithm sortLndmrx).*
3. *Label disjoint actions (algorithm labelArcs).*

## Summery of Procedure

**During search, for a current state $s$:**

1. Apply any applicable action landmark immediately.
2. If none exists:
   1. Build RCG (And/Or graph) for current state $s$, $\mathscr{G}(s)$.
   2. Find closest fact landmark $l$ that was not yet achieved.
   3. Compute applicable disjunctive action landmark $X$ w.r.t. $l$
   4. Minimize $X$ to find set-inclusion minimal disjunctive action landmark.
   5. Filter the set $X$ using path-commitment information
   6. Expand $s$ using the filtered set.

## Outline

1. Motivation
2. Overview
3. RCG
4. Delete-Free Planning
5. Find Action Landmark
6. Disjoint-Path
7. Summery of Procedure
8. Empirical results

## Empirical Results – Time Limit 5 Minutes

| 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 |

- **Pruning + blind** is better than **No pruning + M&S** in almost every domain (except freecell).
- **Pruning + M&S** is better than both (except freecell).
- **Pruning + LM-cut** is better than **No pruning + LM-cut** in almost every domain (except freecell, equal in gripper).
- **Pruning + LM-cut** perform better than all in coverage and speed in almost every domain (except freecell, equal in gripper).

## Empirical Results – Time Limit 30 Minutes

Scores based on (Roger and Helmert 2010). **Higher values are better**. **100 is highest**.

| Domain | No pruning + LM-cut | | | Pruning + LM-cut | | |
|---|---|---|---|---|---|---|
| | # solved | exp avg | time avg | # solved | exp avg | time avg |
| 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 |

- Pruning reduced the number of expansions, in some domains significantly compared to LM-Cut alone, like rovers and depot.
- In rovers and depot the search-time score were also significantly better.
- Not in all domains where the expansion-score is noticeably better, does the search-time score behaves the same.
- 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.

## Future work

There are various potential ways our pruning procedure could be improved:

1. Minimization step (computing minimal disjunctive action landmark) can be expensive.
   - Improve, or learn to recognize domains in which it is not worthwhile.
2. Consider analyzing the topology of the RCG at each state (during search) instead of doing so only in the initial state.
   - Might improve landmark ordering and/or disjoint-path labels
3. Extend these ideas general planning.
   - Some ideas (finding good disjunctive action landmark) can probably be used in regular planning (e.g., extending the SAC algorithms).
   - The idea of disjoint path may be generalizable to regular planning.

## Thank You

- Thank You!