Cost-optimal Planning with Landmarks

E. Karpas C. Domshlak

Faculty of Industrial Engineering and Management
Technion

16.7.2009 - IJCAI

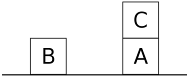
Outline

Landmarks

- A landmark is a fact that must be true at some point in every valid plan (Hoffmann, Porteous and Sebastia 2004)
- Landmarks can be (partially) ordered, according to the order they must be achieved
- *Some* landmarks and ordering can be discovered automatically

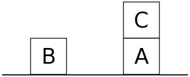
Example Planning Problem - Blocks

- The BLOCKSWORLD domain deals with arranging blocks in a specific way using a crane

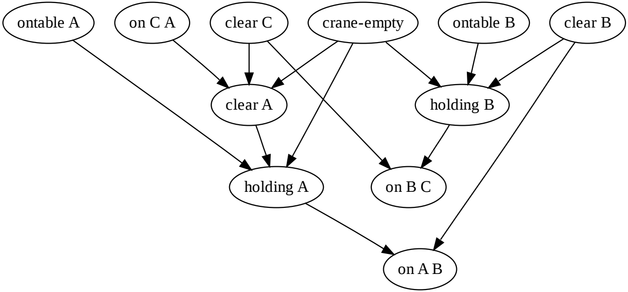


- Variables: *crane-empty*, *holding(X)*, *clear(X)*, *ontable(X)*, *on(X,Y)*
- Operators:
  - *pickup(X)*
    - Pre: *ontable(X)*, *clear(X)*, *crane-empty*
    - Add: *holding(X)*
    - Del: *ontable(X)*, *clear(X)*, *crane-empty*
  - *putdown(X)*...
  - *stack(X,Y)*...
  - *unstack(X,Y)*...

Landmarks Example

- Consider the following blocks problem ("The Sussman Anomaly")
- Initial State 

- Goal: $on(A, B)$, $on(B, C)$



Using Landmarks - Heuristic

- The number of landmarks that still need to be achieved can be used as an (inadmissible) heuristic function (Richter et. al. 2008)
- This is the heuristic used by *LAMA* - a state of the art satisficing planner, and winner of the IPC-2008 sequential satisficing track
- Note that this a path-dependent heuristic

LAMA

- The landmarks that still need to be achieved after reaching state s via path π are

$$L(s, \pi) =$$

- L is the set of all (discovered) landmarks
- $\text{Accepted}(s, \pi) \subset L$ is the set of *accepted* landmarks
- $\text{ReqAgain}(s, \pi) \subseteq \text{Accepted}(s, \pi)$ is the set of *required again* landmarks - landmarks that must be achieved again (computed from landmark orderings)

LAMA

- The landmarks that still need to be achieved after reaching state s via path π are

$$L(s,\pi) = L$$

- L is the set of all (discovered) landmarks
- $\text{Accepted}(s,\pi) \subset L$ is the set of *accepted* landmarks
- $\text{ReqAgain}(s,\pi) \subseteq \text{Accepted}(s,\pi)$ is the set of *required again* landmarks - landmarks that must be achieved again (computed from landmark orderings)

LAMA

- The landmarks that still need to be achieved after reaching state s via path π are

$$L(s,\pi) = L \setminus \text{Accepted}(s,\pi)$$

- L is the set of all (discovered) landmarks
- $\text{Accepted}(s,\pi) \subset L$ is the set of *accepted* landmarks
- $\text{ReqAgain}(s,\pi) \subseteq \text{Accepted}(s,\pi)$ is the set of *required again* landmarks - landmarks that must be achieved again (computed from landmark orderings)

LAMA

- The landmarks that still need to be achieved after reaching state s via path π are

$$L(s,\pi) = L \setminus (\text{Accepted}(s,\pi) \setminus \text{ReqAgain}(s,\pi))$$

- L is the set of all (discovered) landmarks
- $\text{Accepted}(s,\pi) \subset L$ is the set of *accepted* landmarks
- $\text{ReqAgain}(s,\pi) \subseteq \text{Accepted}(s,\pi)$ is the set of *required again* landmarks - landmarks that must be achieved again (computed from landmark orderings)

Problem - Inadmissibility

- LAMA is inadmissible because a single action can achieve multiple landmarks
- Example: *stack(A,B)* achieves both *crane-empty* and *on(A,B)*.

Outline

Admissibility - how?

- We are interested in admissible heuristics, in order to perform cost-optimal planning
- Problem: the number of landmarks is inadmissibile
- Solution: assign a cost to each landmark, and sum over the costs of landmarks

Conditions for Admissible Cost Sharing

- Each action shares its cost between all the landmarks it achieves
- Each landmark is assigned the cheapest cost any action assigned it

Proposition 1

Given costs that obey these constraints, the sum of costs of landmarks that still need to be achieved is an admissible heuristic:

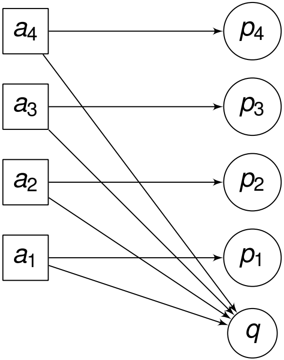
$$h_L(s,\pi) := cost(L(s,\pi)) = \sum_{\phi \in L(s,\pi)} cost(\phi)$$

Cost Partitioning - how?

- How can we assign costs to landmarks?
- Easy answer - uniform cost sharing - each action shares its cost equally between the landmarks it achieves

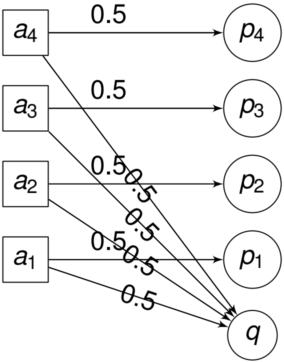
- Advantage: easy and fast to compute
- Disadvantage: can be much worse than the optimal cost partitioning

Example



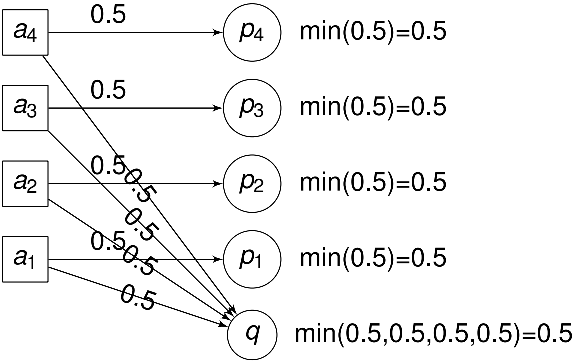
Example

Uniform cost sharing



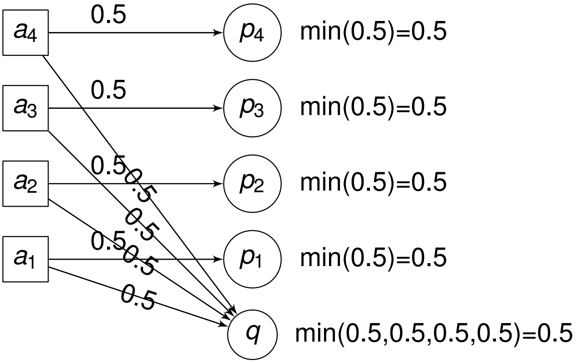
Example

Uniform cost sharing



Example

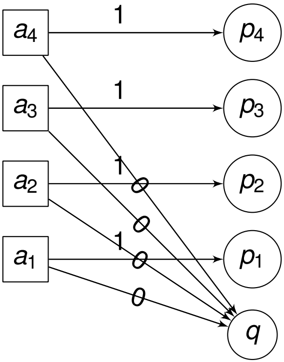
Uniform cost sharing $\qquad h_L = 2.5$



Example

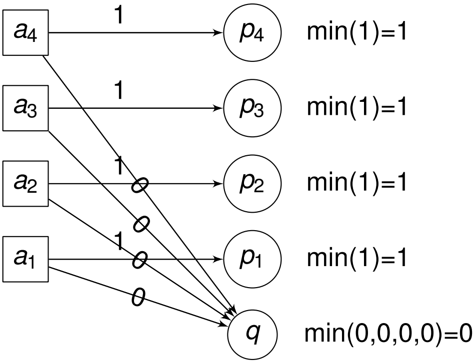
Optimal cost sharing

$h_L = 2.5$



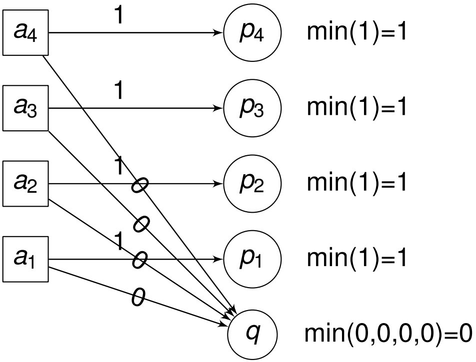
Example

Optimal cost sharing $h_L = 2.5$



Example

Optimal cost sharing $h_L = 4$ $h_L = 2.5$



Optimal Cost Sharing

- The good news: the optimal cost partitioning can be computed in poly-time
  - The constraints for admissibility are linear, and can be used in a Linear Programming (LP) problem
  - The objective is to maximize the sum of landmark costs
- The bad news: poly-time can still take a long time

How can we do better?

- So far:
  - Uniform cost sharing is easy to compute, but suboptimal
  - Optimal cost sharing takes a long time to compute
- Q: How can we get better heuristic estimates that don't take a long time to compute?
- A: Exploit additional information (action landmarks)
- An action landmark is an action that must be taken in every valid plan (Zhu and Givan 2004, Vidal and Geffner 2006)

Outline

1. Background - Inadmissible Landmarks Based Heuristics
2. Admissible Landmarks Based Heuristics
   - Admissibility
   - Cost Sharing Schemes
3. Search Algorithm (hint: not A^*)
4. Experimental Evaluation
5. Summary

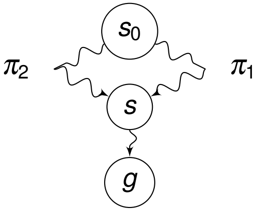
A^* with Path Dependent Heuristics

- Regular A^* evaluates a state only the first time it is reached (via path π_1)
- When the same state is reached again (via a different path π_2), it is not evaluated again

A^* with Path-Dependent Heuristic

1. Evaluate a state every time it is reached
2. Use the maximum estimate

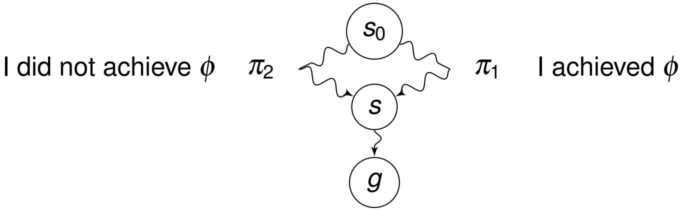
A^* with Landmarks Heuristics



- Suppose state s was reached by paths π_1, π_2
- Suppose π_1 achieved landmark ϕ and π_2 did not
- Then ϕ needs to be achieved after state s

We call this multi-path-dependence

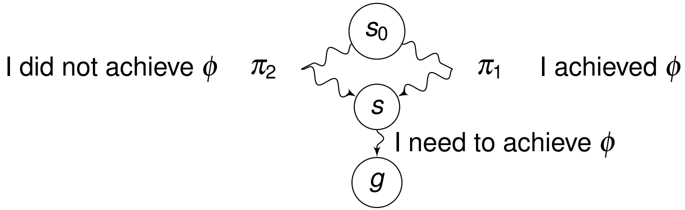
A^* with Landmarks Heuristics



- Suppose state s was reached by paths π_1, π_2
- Suppose π_1 achieved landmark ϕ and π_2 did not
- Then ϕ needs to be achieved after state s

We call this multi-path-dependence

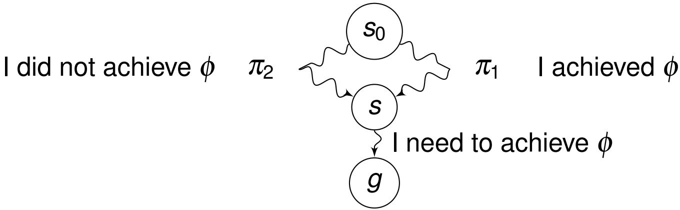
A^* with Landmarks Heuristics



- Suppose state s was reached by paths π_1, π_2
- Suppose π_1 achieved landmark ϕ and π_2 did not
- Then ϕ needs to be achieved after state s

We call this multi-path-dependence

A^* with Landmarks Heuristics



- Suppose state s was reached by paths π_1, π_2
- Suppose π_1 achieved landmark ϕ and π_2 did not
- Then ϕ needs to be achieved after state s

We call this multi-path-dependence

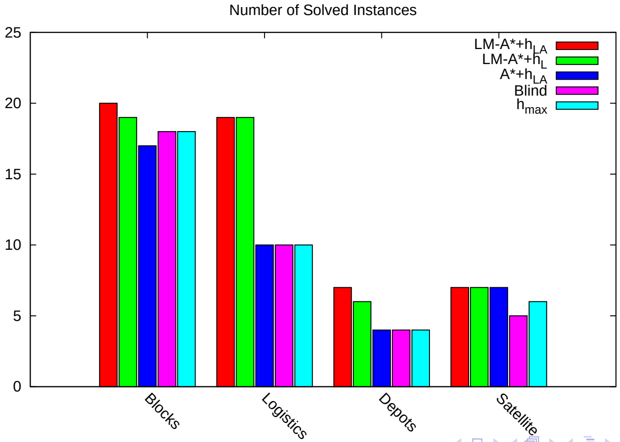
Fusing Data from Multiple Paths

- Suppose we have a set of paths from s_0 to a state s
  - The accepted landmarks are those that were accepted by all paths
- We compute h_L using the accepted landmarks from all the paths
- We call the resulting search algoritm *LM-A**

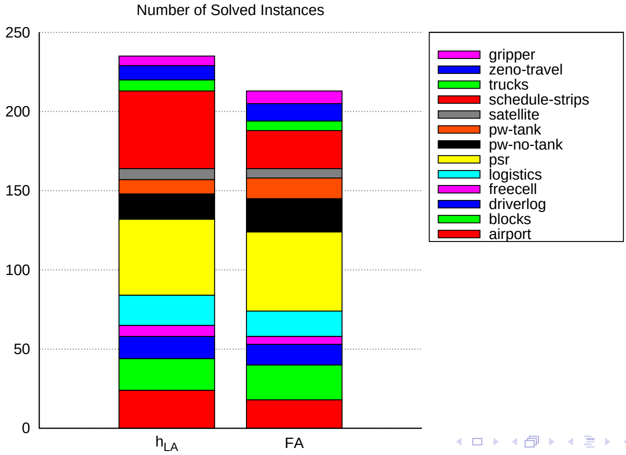
Outline

1. Background - Inadmissible Landmarks Based Heuristics
2. Admissible Landmarks Based Heuristics
   - Admissibility
   - Cost Sharing Schemes
3. Search Algorithm (hint: not A^*)
4. Experimental Evaluation
5. Summary

Evaluation - Baseline



Evaluation - State-of-the-art



Outline

Summary

- We introduced an admissible landmarks-based heuristic
- We introduced the *LM-*A^* search algorithm that uses multi-path-dependent heuristics more effectively than A^*
- *LM-*A^* with h_{LA} favorably compete with state-of-the-art admissible heuristics

Thank You

- Thank You