Cost-optimal Planning with Landmarks

E. Karpas C. Domshlak

Faculty of Industrial Engineering and Management
Technion

24.3.2009 - Information Systems Seminar

Outline

Planning

- The object of (classical) planning is to find a sequence of actions that leads from an initial state to some goal
- Usually divided into:
  - Satisficing planning
  - Optimal planning

Planning as Search

- Planning can be viewed as a search problem
- $\Pi = \langle S, A, tr, s_0, S_g \rangle$ where
  - S is the set of states
  - A is the set of actions
  - $tr : S \times A \longrightarrow S$ is the transition function
  - s_0 is the initial state
  - S_g is a set of goal states

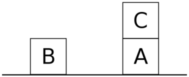
- Satisficing planning: find a sequence of actions $\langle a_0 \ldots a_n \rangle$ s.t. $tr(a_n, \ldots tr(tr(s_0, a_0), a_1) \ldots) \in S_g$
- Optimal planning: find one of the shortest such sequences
  - A^* with an admissible heuristic

Planning in STRIPS

- STRIPS is a language for describing planning problems compactly
- A STRIPS task is a 5-tuple $\Pi = \langle V, A, \mathscr{C}, s_0, G \rangle$
  - $V = \{v_1, \ldots, v_n\}$ is a set of binary *state variables*
  - s_0 is an *initial state*
  - $G \subseteq V$ is the goal
  - A is a finite set of *actions*
    - Each action a is a triple $\langle \mathrm{pre}, \mathrm{add}, \mathrm{del} \rangle$ of sets of variables
    - Each action has a non-negative cost $\mathscr{C}(a)$
- A complete assignment to V is called a *state*
- A *fact* is an assignment to a single variable (i.e. $v_i = T/F$)

Planning Example - 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)*...

Outline

Landmarks

- A landmark is a fact that must be true at some point in every valid plan (Hoffmann, Porteous and Sebastia 2004)
- Example: if *on(A,B)* is a goal, then *clear(B)* must be true sometime before that in every plan
- *Some* landmarks can be discovered automatically (Hoffmann, Porteous and Sebastia 2004, Richter, Helmert and Westphal 2008)

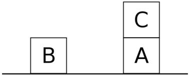
Landmark Orderings

- There is a (partial) order between landmarks
- $\phi < \psi$ means that landmark ϕ must be achieved before landmark ψ
- Orderings can also be discovered automatically

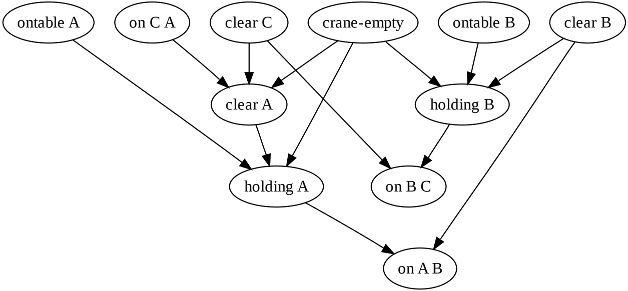
Landmark Discovery

- Backchaining - if all actions that achieve landmark ϕ have a common precondition ψ, then ψ is also a landmark, and is ordered greedy-necessarily before ϕ
- Goal facts are (trivially) landmarks
- Landmark discovery is done by backchaining from goal facts
- Small disjunctive landmarks are also allowed
- More ways to discover (more) landmarks also exist

Landmarks Example

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

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



Using Landmarks - Subgoals

- Landmarks were originally used as *subgoals* in search
- Advantages: considerable speedup in search
- Disadvantage:longer plans, incompleteness (sometimes)

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

- Suppose we are in state s. Did we achieve landmark ϕ yet?
- There is no way to tell. Achieved landmarks are a function of path, not state
- Solution: make the heuristic path-dependent, instead of state-dependent

Using Landmarks - Heuristic

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

- Suppose we are in state s. Did we achieve landmark ϕ yet?
- There is no way to tell. Achieved landmarks are a function of path, not state
- Solution: make the heuristic path-dependent, instead of state-dependent

Using Landmarks - Heuristic

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

- Suppose we are in state s. Did we achieve landmark ϕ yet?
- There is no way to tell. Achieved landmarks are a function of path, not state
- Solution: make the heuristic path-dependent, instead of state-dependent

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

- 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
- $\text{ReqAgain}(s,\pi)$ is 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
- $\text{ReqAgain}(s,\pi)$ is computed from landmark orderings

Outline

Admissibility - how?

- We are interested in admissible heuristics, in order to perform cost-optimal planning
- LAMA is inadmissible because a single action can achieve multiple landmarks
- Example: *crane-empty* and *on(A,B)* can both be achieved by *stack(A,B)*
- Solution: assign a cost to each landmark, and sum over the costs of landmarks

Conditions for Admissibile Cost Sharing

- Each action shares its cost between all the landmarks it achieves

$$\forall a \in A: \sum_{\phi \in L(a|s,\pi)} cost(a,\phi) \leq \mathscr{C}(a)$$

$cost(a,\phi)$ is the cost "assigned" by action a to ϕ
$L(a|s,\pi)$ is the set of landmarks achieved by a

- Each landmark is assigned the cheapest cost any action assigned it

$$\forall \phi \in L(s,\pi): \; cost(\phi) \leq \min_{a \in \mathrm{ach}(\phi|s,\pi)} cost(a,\phi)$$

$cost(\phi)$ is the cost assigned to landmark ϕ
$\mathrm{ach}(\phi|s,\pi) \subseteq A$ is the set of actions that can achieve ϕ

Conditions for Admissibile Cost Sharing

- Each action shares its cost between all the landmarks it achieves

$$\forall a \in A : \sum_{\phi \in L(a|s,\pi)} cost(a,\phi) \leq \mathscr{C}(a)$$

  $cost(a,\phi)$ is the cost "assigned" by action a to ϕ
  $L(a|s,\pi)$ is the set of landmarks achieved by a

- Each landmark is assigned the cheapest cost any action assigned it

$$\forall \phi \in L(s,\pi) : \; cost(\phi) \leq \min_{a \in \mathrm{ach}(\phi|s,\pi)} cost(a,\phi)$$

  $cost(\phi)$ is the cost assigned to landmark ϕ
  $\mathrm{ach}(\phi|s,\pi) \subseteq A$ is the set of actions that can achieve ϕ

Admissibile Cost Sharing

- The idea is that the cost of a set of landmarks is no greater than the cost of any single action that achieves them
- Given costs that obey these constraints, the sum of costs of landmarks that still need to be achieved is an admissible heuristic, which we call h_L

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

- Proof: left up to the reader $\ddot\smile$

Outline

Cost Partitioning - how?

- How can we find such a partitioning?
- Easy answer - uniform cost sharing - each action shares its cost equally between the landmarks it achieves

$$cost(a,\phi) = \frac{\mathscr{C}(a)}{|L(a|s,\pi)|}$$

$$cost(\phi) = \min_{a \in ach(\phi|s,\pi)} cost(a,\phi)$$

Uniform Cost Sharing

- Advantage: Easy and fast to compute
- Disadvantage: can be much worse than the optimal cost partitioning
- Example:
  - The known landmarks are $\{p_1, \ldots, p_k, q\}$
  - The possible actions are a_i $(1 \le i \le k)$ with $\mathrm{eff}(a_i) = \{p_i, q\}$
  - With uniform cost sharing:
    - $cost(a_i, p_i) = cost(a_i, q) = 0.5$
    - $cost(p_i) = cost(q) = 0.5$
    - $h_L(s, \pi) = \frac{k+1}{2}$
  - The optimal cost partitioning is:
    - $cost(a_i, p_i) = 1$, $cost(a_i, q) = 0$
    - $cost(p_i) = 1$, $cost(q) = 0$
    - $h_L(s, \pi) = k$

Uniform Cost Sharing

- Advantage: Easy and fast to compute
- Disadvantage: can be much worse than the optimal cost partitioning
- Example:
  - The known landmarks are $\{p_1, \ldots, p_k, q\}$
  - The possible actions are a_i $(1 \leq i \leq k)$ with $\text{eff}(a_i) = \{p_i, q\}$
  - With uniform cost sharing:
    - $cost(a_i, p_i) = cost(a_i, q) = 0.5$
    - $cost(p_i) = cost(q) = 0.5$
    - $h_L(s, \pi) = \frac{k+1}{2}$
  - The optimal cost partitioning is:
    - $cost(a_i, p_i) = 1$, $cost(a_i, q) = 0$
    - $cost(p_i) = 1$, $cost(q) = 0$
    - $h_L(s, \pi) = k$

Uniform Cost Sharing

- Advantage: Easy and fast to compute
- Disadvantage: can be much worse than the optimal cost partitioning
- Example:
  - The known landmarks are $\{p_1, \ldots, p_k, q\}$
  - The possible actions are a_i $(1 \leq i \leq k)$ with $\text{eff}(a_i) = \{p_i, q\}$
  - With uniform cost sharing:
    - $cost(a_i, p_i) = cost(a_i, q) = 0.5$
    - $cost(p_i) = cost(q) = 0.5$
    - $h_L(s, \pi) = \frac{k+1}{2}$
  - The optimal cost partitioning is:
    - $cost(a_i, p_i) = 1$, $cost(a_i, q) = 0$
    - $cost(p_i) = 1$, $cost(q) = 0$
    - $h_L(s, \pi) = k$

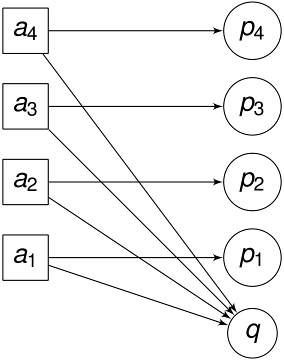
Uniform Cost Sharing

- Advantage: Easy and fast to compute
- Disadvantage: can be much worse than the optimal cost partitioning
- Example:
  - The known landmarks are $\{p_1, \ldots, p_k, q\}$
  - The possible actions are a_i $(1 \leq i \leq k)$ with $\mathrm{eff}(a_i) = \{p_i, q\}$
  - With uniform cost sharing:
    - $cost(a_i, p_i) = cost(a_i, q) = 0.5$
    - $cost(p_i) = cost(q) = 0.5$
    - $h_L(s, \pi) = \frac{k+1}{2}$
  - The optimal cost partitioning is:
    - $cost(a_i, p_i) = 1$, $cost(a_i, q) = 0$
    - $cost(p_i) = 1$, $cost(q) = 0$
    - $h_L(s, \pi) = k$

Uniform Cost Sharing

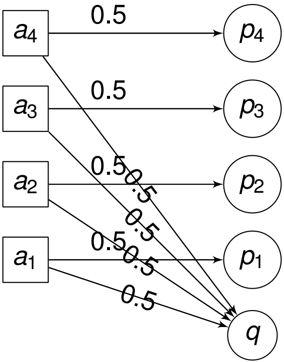
- Advantage: Easy and fast to compute
- Disadvantage: can be much worse than the optimal cost partitioning
- Example:
  - The known landmarks are $\{p_1, \ldots, p_k, q\}$
  - The possible actions are a_i $(1 \leq i \leq k)$ with $\text{eff}(a_i) = \{p_i, q\}$
  - With uniform cost sharing:
    - $cost(a_i, p_i) = cost(a_i, q) = 0.5$
    - $cost(p_i) = cost(q) = 0.5$
    - $h_L(s, \pi) = \frac{k+1}{2}$
  - The optimal cost partitioning is:
    - $cost(a_i, p_i) = 1$, $cost(a_i, q) = 0$
    - $cost(p_i) = 1$, $cost(q) = 0$
    - $h_L(s, \pi) = k$

Example Illustrated



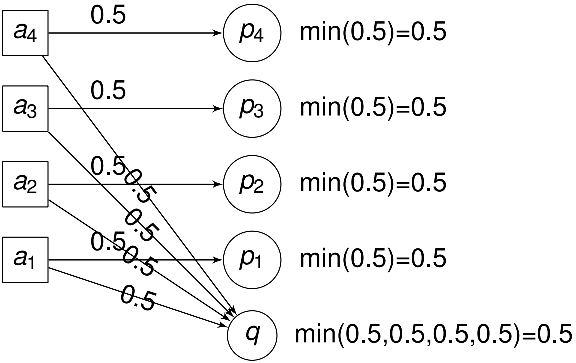
Example Illustrated

Uniform cost sharing



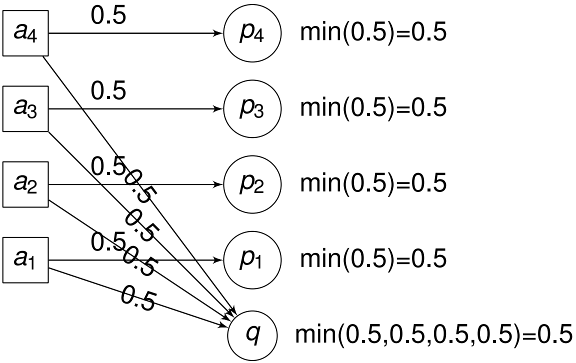
Example Illustrated

Uniform cost sharing



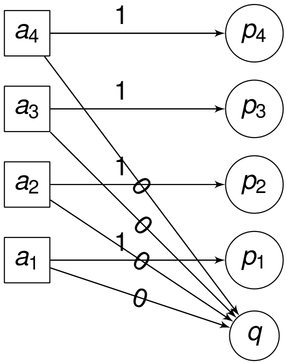
Example Illustrated

Uniform cost sharing $h_L = 2.5$



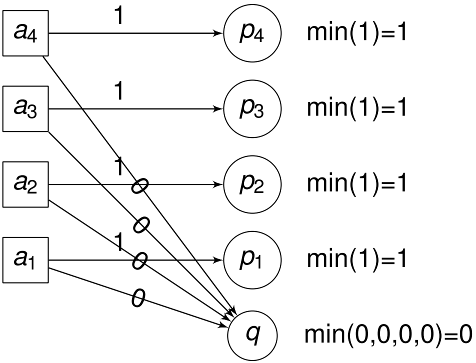
Example Illustrated

Optimal cost sharing



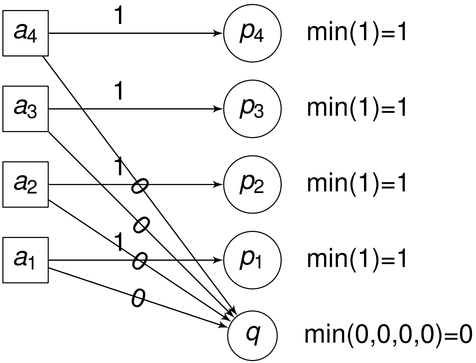
Example Illustrated

Optimal cost sharing



Example Illustrated

Optimal cost sharing $\quad h_L = 4$

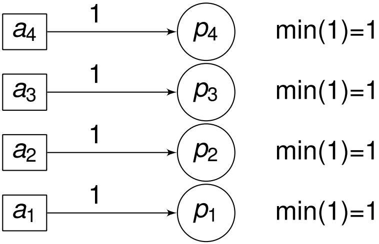


Optimal Cost Sharing

- The good news: the optimal cost partitioning is poly-time to compute
  - 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 solution to the LP gives us the optimal cost partitioning
- The bad news: poly-time can still take a long time

Optimal Cost Sharing - Properties

- Monotonicity along the inclusion relation of the landmark sets
  - If L and L' are sets of landmarks, and $L \subseteq L'$, the optimal cost sharing ensures $cost(L') \geq cost(L)$
  - This does not hold for uniform cost sharing
  - Consider the previous example - landmarks $\{p_1, \ldots, p_k, q\}$, $\text{eff}(a_i) = \{p_i, q\}$
  - If we exclude landmark q, uniform cost sharing assigns $cost(p_i) = cost(a_i, p_i) = 1$, so $h_L = k$



Outline

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

Introducing - Action Landmarks

- An action landmark is an action that must be taken in every valid plan (Zhu and Givan 2004, Vidal and Geffner 2006)
- Example: if *on(A,B)* is a goal, then *stack(A,B)* must occur in every plan
- Checking if action *a* is an action landmark can be done efficiently

Using Action Landmarks

- Discover (some) action landmarks during preprocessing
- Keep track of which action landmarks were *not* used along the current path π - call this set $U(s, \pi)$
- Every action in $U(s, \pi)$ *must* occur after s, so we should account for their cost in our heuristic estimate
- Action landmark covering
  - Sum up the costs of actions in $U(s, \pi)$
  - Remove from $L(s, \pi)$ the landmarks that are achieved by actions in $U(s, \pi)$ - call this $L_U(s, \pi)$
  - Perform action cost sharing over $L_U(s, \pi)$

Introducing - h_{LA}

- The resulting heuristic is

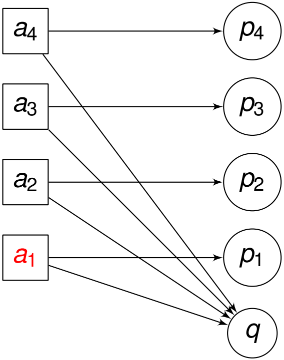
$$h_{LA}(s,\pi) := cost(L_U(s,\pi)) + \sum_{a \in U(s,\pi)} \mathscr{C}(a)$$

- h_{LA} is admissible, and dominates h_L
  - $\mathscr{C}(a)$ dominates the sum of costs of landmarks achieved by a (because of our contraints)
  - Removing these landmarks and adding $\mathscr{C}(a)$ never lowers the heuristic estimate

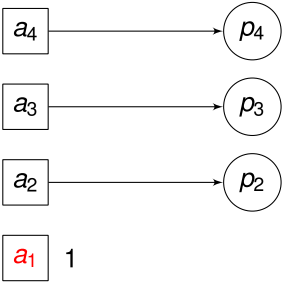
Side Benefits of h_{LA}

- Side Benefit: action landmark covering with uniform cost sharing
  - Consider the same example as before - landmarks $\{p_1, \ldots, p_k, q\}$, $\mathrm{eff}(a_i) = \{p_i, q\}$
  - Recall that with uniform cost sharing $h_L = \frac{k+1}{2}$
  - Suppose one of these actions (say a_1) is discovered to be an action landmark
  - p_1 and q are removed from our landmarks for cost sharing
  - $p_2, \ldots, p_k$ are assigned a cost of 1, and a_1 adds 1
  - With uniform cost sharing $h_{LA} = k$

Side Benefits of h_{LA} - Illustrated

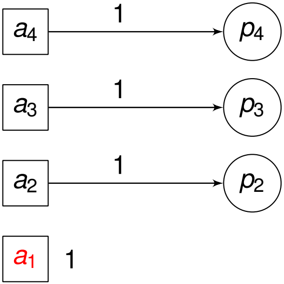


Action landmark covering



Side Benefits of h_{LA} - Illustrated

Uniform Cost Sharing $\qquad h_{LA} = 4$



min(1)=1

min(1)=1

min(1)=1

Outline

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 - fine for state-dependent heuristics
- Using A^* with an admissible path dependent heuristic still guarantees that an optimal solution is found

- Q: Can we do better with a path-dependent heuristic?
- A: Yes. Evaluate states every time they are reached, and take the maximum
- This works for *any* path-dependent heuristic

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 - fine for state-dependent heuristics
- Using A^* with an admissible path dependent heuristic still guarantees that an optimal solution is found

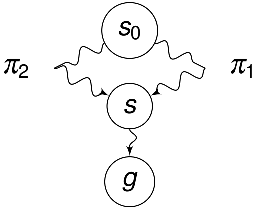
- Q: Can we do better with a path-dependent heuristic?
- A: Yes. Evaluate states every time they are reached, and take the maximum
- This works for *any* path-dependent heuristic

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 - fine for state-dependent heuristics
- Using A^* with an admissible path dependent heuristic still guarantees that an optimal solution is found

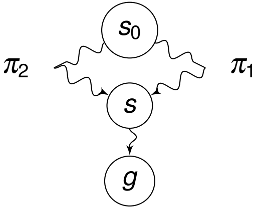
- Q: Can we do better with a path-dependent heuristic?
- A: Yes. Evaluate states every time they are reached, and take the maximum
- This works for *any* path-dependent heuristic

- Q: Can we do even better with landmarks?



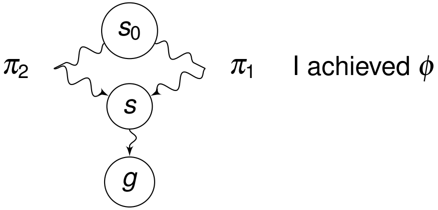
- Suppose state s was reached by paths π_1, π_2
- Suppose π_1 achieved landmark ϕ and π_2 did not
- Then ϕ needs to be achieved after state s
- Proof: ϕ is a landmark, therefore it needs to be true in all valid plans, including valid plans that start with π_2

- Q: Can we do even better with landmarks?



- 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
- Proof: ϕ is a landmark, therefore it needs to be true in all valid plans, including valid plans that start with π_2

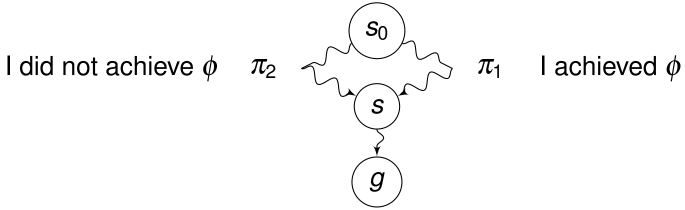
- Q: Can we do even better with landmarks?



- 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
- Proof: ϕ is a landmark, therefore it needs to be true in all valid plans, including valid plans that start with π_2

A^* with Landmarks Heuristics

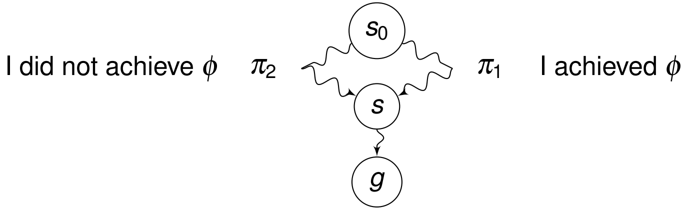
- Q: Can we do even better with landmarks?



- 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
- Proof: ϕ is a landmark, therefore it needs to be true in all valid plans, including valid plans that start with π_2

A^* with Landmarks Heuristics

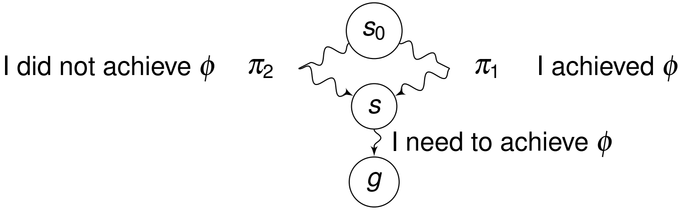
- Q: Can we do even better with landmarks?



- 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
- Proof: ϕ is a landmark, therefore it needs to be true in all valid plans, including valid plans that start with π_2

A^* with Landmarks Heuristics

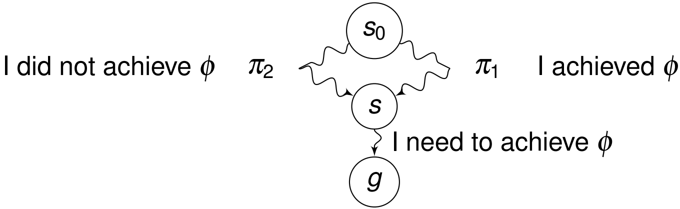
- Q: Can we do even better with landmarks?



- 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
- Proof: ϕ is a landmark, therefore it needs to be true in all valid plans, including valid plans that start with π_2

A^* with Landmarks Heuristics

- Q: Can we do even better with landmarks?



- 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
- Proof: ϕ is a landmark, therefore it needs to be true in all valid plans, including valid plans that start with π_2

Fusing Data from Multiple Paths

- Suppose $\mathscr{P}$ is a set of paths from s_0 to a state s. Define

$$U(s,\mathscr{P}) = \bigcup_{\pi \in \mathscr{P}} U(s,\pi)$$

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

- Where:
  - $\text{Accepted}(s,\mathscr{P}) = \bigcap_{\pi \in \mathscr{P}} \text{Accepted}(s,\pi)$
  - $\text{ReqAgain}(s,\mathscr{P}) \subseteq \text{Accepted}(s,\mathscr{P})$ is specified as before by s and the greedy-necessary orderings over L
- The multi-path-dependent version of our heuristics are

$$h_L(s,\mathscr{P}) = cost(L(s,\mathscr{P}))$$

$$h_{LA}(s,\mathscr{P}) = cost(L_U(s,\mathscr{P})) + \sum_{a \in U(s,\mathscr{P})} \mathscr{C}(a)$$

Multi-path-dependent Heuristics

- We call the resulting search algoritm *LM-A**
- In general, any heuristic that can use information from multiple paths can be used in such a way
- However, storing all the paths that reach all states is not feasible
- *LM-A** exploits the monotonicity of set union and intersection to store the information from all paths compactly

- Note: one of the reasons that almost-perfect heuristics are not good in many planning domains is transpositions (Helmert and Röger 2008). This is our little payback

Outline

Evaluation

- We want to evaluate how our heuristics fare in the "real" world of IPC benchmarks

- We evaluate *LM-A** and A^* with h_L and h_{LA}
- We evaluate h_{LA} with blind search and a baseline admissible heuristic h_{max}
- We evaluate h_{LA} with the state-of-the-art *FA* heuristic
- We evaluate how h_{LA} and *FA* scale as problem size increases

- *FA* is an abstraction based heuristic (Helmert, Haslum and Hoffmann 2007), which is one of the best-performing forward-search planners

*LM-A** vs. *A**

domain	N^+/N^1	*solved*			*nodes*			*time*		
		1	2	3	1	2	3	1	2	3
Blocks	17/20	17	19	**20**	2772	3779	2354	0.74	1.1	0.98
Logistics	10/19	10	**19**	**19**	101274	347	347	11	0.04	0.2
Depots	4/7	4	6	**7**	425340	64159	28243	101	47	20
Satellite	7/7	7	7	7	27925	27917	27917	27	44	46
TOTAL	38/53	38	51	**53**	480649	13678	9259	19	14	11

- 1 - $A^* + h_{LA}$
- 2 - *LM-A** $+ h_L$
- 3 - *LM-A** $+ h_{LA}$

Baseline Comparison

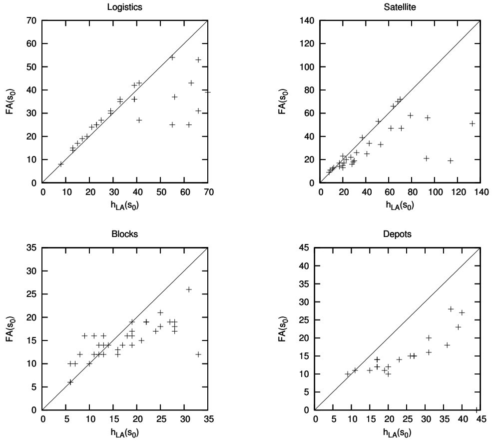
domain	N^+/N^1	*solved*			*nodes*			*time*		
		1	2	3	1	2	3	1	2	3
Blocks	18/20	18	18	**20**	1202791	370074	16715	5.73	8.41	8.31
Logistics	10/19	10	10	**19**	157261	63081	347	1	0.6	0.2
Depots	4/7	4	4	**7**	2361807	635063	28243	24	65	20
Satellite	5/7	5	6	**7**	5157775	2187930	5142	129	74	5
TOTAL	37/53	37	38	**53**	1937465	654160	16497	23	21	7

- 1 - Blind Search
- 2 - $A^* + h_{max}$
- 3 - $LM\text{-}A^* + h_{LA}$

State-of-the-art Comparison

domain	N^+/N^1	*solved*		*nodes*		*time*	
		h_{LA}	*FA*	h_{LA}	*FA*	h_{LA}	*FA*
airport	18/24	**24**	18	1395	528152	8	123
blocks	19/23	20	**22**	89179	1319533	56	13
driverlog	13/14	**14**	13	109611	765930	53	14
freecell	5/7	**7**	5	8487	1406793	10	232
logistics	16/19	**19**	16	14265	44111	20	0
psr	48/50	48	**50**	14541	3267	3	0
pw-no-tank	16/21	16	**21**	122455	295857	48	20
pw-tank	9/13	9	**13**	127383	85165	211	142
satellite	6/7	**7**	6	6287	437238	5	13
schedule-strips	23/50	**49**	24	3932	152	15	713
trucks	6/7	**7**	6	249518	4586761	198	80
zeno-travel	9/11	9	**11**	6658	36030	9	1
total	226/284	**267**	243	99838	458689	43	103

Scaling - Initial State Estimate



Outline

Summary

- We introduced the h_L and h_{LA} heuristics
- 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