Automated Action Planning

Implicit Planning Task Structure: Landmark Heuristics

Carmel Domshlak

Finite Domain Representation (FDR) Language

Definition (FDR planning tasks)

An FDR planning task is a tuple $\Pi = \langle V, A, I, G \rangle$

- V is a finite set of state variables with finite domains $dom(v_i)$
- initial state I is a complete assignment to V
- goal G is a partial assignment to V
- A is a finite set of actions a specified via pre(a) and eff(a), both being partial assignments to V

In cost-sensitive planning, each action a is also associated with a cost $C(a)$

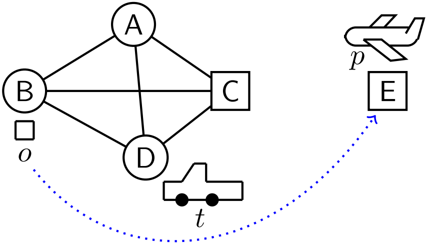
Landmarks

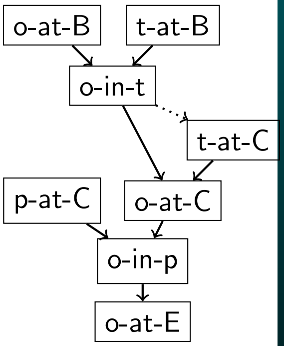
- A landmark is a formula that must be true at some point in every plan
- Landmarks can be (partially) ordered according to the order in which they must be achieved
- Some landmarks and orderings can be discovered automatically
- Most current approaches consider only landmarks that are facts or disjunctions of facts
  (Some recent work on conjunctive landmarks)

Action Landmarks

- An action landmark is an action which occurs in every valid plan
- Landmarks may imply actions landmarks (e.g., sole achievers)
- Action landmarks imply landmarks (e.g., preconditions and effects)
- Some action landmarks can be discovered automatically

Example Planning Problem - Logistics





Partial landmarks graph

Sound Landmark Orderings

Sound landmark orderings are guaranteed to hold - they do not prune the solution space

- *Natural* ordering $A \rightarrow B$, iff A true some time before B
- *Necessary* ordering $A \rightarrow_n B$, iff A always true one step before B becomes true
- *Greedy-necessary* ordering $A \rightarrow_{gn} B$, iff A true one step before B becomes true for the first time

Note that $A \rightarrow_n B \implies A \rightarrow_{gn} B \implies A \rightarrow B$

Landmark Complexity

- Everything is PSPACE-complete

- Deciding if a given fact is a landmark is PSPACE-complete
- Proof Sketch: it's the same as deciding if the problem without operators that achieve this fact is unsolvable

- Deciding if there is a natural / necessary / greedy-necessary between two landmarks is PSPACE-complete

Landmark Discovery in Theory

Theory

- A is a landmark $\Longleftrightarrow$ Π'_A is unsolvable
  where Π'_A is Π without the operators that achieve A

- The delete relaxation of Π'_A is unsolvable $\Longrightarrow$ Π'_A is unsolvable (delete-relaxation landmarks)
- An abstraction of Π'_A is unsolvable $\Longrightarrow$ Π'_A is unsolvable (abstraction landmarks)

Landmark Discovery in Theory

Theory

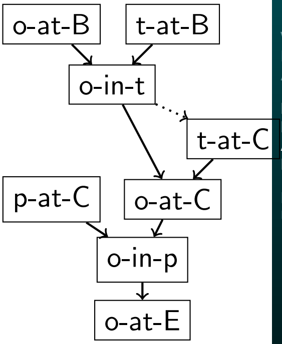
- A is a landmark $\Longleftrightarrow$ Π'_A is unsolvable
  where Π'_A is Π without the operators that achieve A

- The delete relaxation of Π'_A is unsolvable $\Longrightarrow$ Π'_A is unsolvable (delete-relaxation landmarks)
- An abstraction of Π'_A is unsolvable $\Longrightarrow$ Π'_A is unsolvable (abstraction landmarks)

Landmark Discovery I
Delete Relaxation Landmarks

Find landmarks and orderings by backchaining

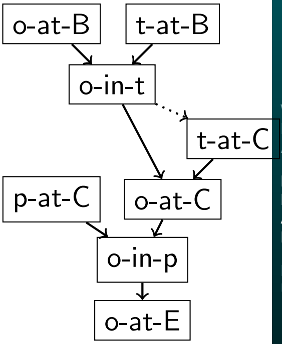
- Every goal is a landmark
- If B is landmark and all actions that achieve B share A as precondition, then
  - A is a landmark
  - $A \rightarrow_n B$



Useful restriction: consider only the case where B is achieved for the first time $\rightsquigarrow$ find more landmarks (and $A \rightarrow_{\mathrm{gn}} B$)

PSPACE-complete to find first achievers
$\rightsquigarrow$ over-approximation by building relaxed planning graph for Π'_B

- This graph contains no actions that add B
- Any action applicable in this graph can possibly be executed before B first becomes true $\rightsquigarrow$ possible first achievers



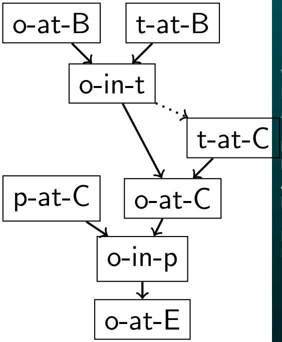
Additionally, if C not in the graph and C later proven to be a landmark, introduce $B \rightarrow C$

Landmark Discovery I

Delete Relaxation Landmarks

Disjunctive landmarks also possible, e.g., (o-in-p_1 $\vee$ o-in-p_2):

- If B is landmark and all actions that (first) achieve B have A or C as precondition, then $A \vee C$ is a landmark
- Generalises to any number of disjuncts
- Large number of possible disjunctive landmarks $\rightsquigarrow$ must be restricted



Domain Transition Graphs (DTGs)

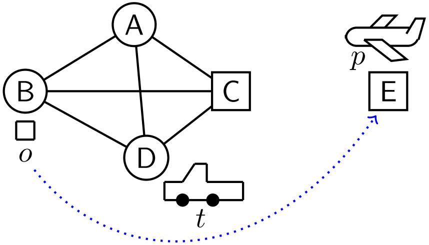
Find landmarks through DTGs (Richter et al. 2008)

The domain transition graph of $v \in V$ (DTG$_v$) represents how the value of v can change.

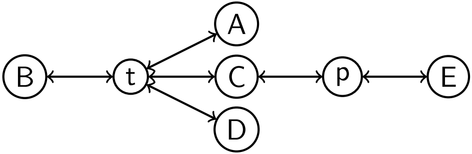
Given: an FDR task $\langle V, A, s_0, G \rangle$
DTG$_v$ is a directed graph with nodes $\mathcal{D}_v$ that has arc $\langle d, d' \rangle$ iff

- $d \neq d'$, and
- $\exists$ action with $v \mapsto d'$ as effect, and either
  - $v \mapsto d$ as precondition, or
  - no precondition on v

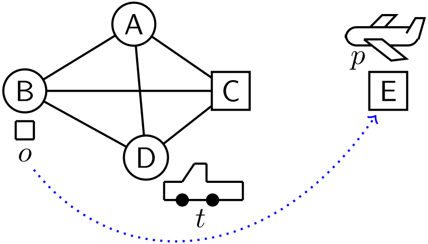
DTG Example



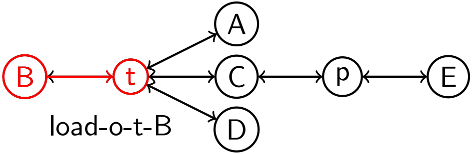
DTG$_{v_o}$:



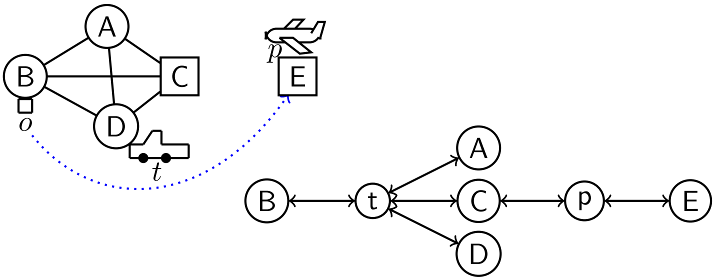
DTG Example



DTG_{v_o}:



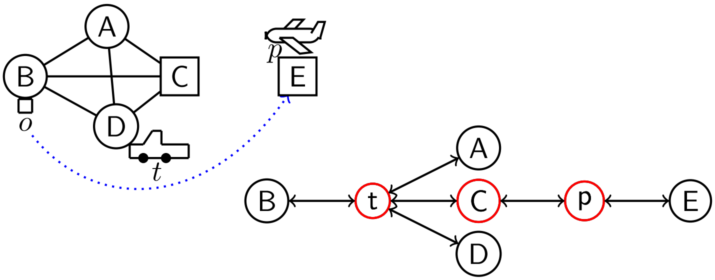
Landmark Discovery II
Abstraction Landmarks



- Find landmarks through DTGs: if
  - $s_0(v) = d_0$,
  - $v \mapsto d$ landmark, and
  - every path from d_0 to d passes through d',

then $v \mapsto d'$ landmark, and $\quad (v \mapsto d') \rightarrow (v \mapsto d)$

Landmark Discovery II
Abstraction Landmarks



- Find landmarks through DTGs: if
  - $s_0(v) = d_0$,
  - $v \mapsto d$ landmark, and
  - every path from d_0 to d passes through d',

then $v \mapsto d'$ landmark, and $\quad (v \mapsto d') \rightarrow (v \mapsto d)$

Using Landmarks

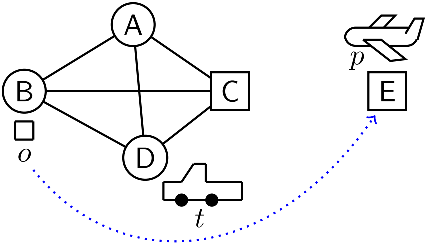
- Some landmarks and orderings can be discovered efficiently
- So what can we do once we have these landmarks?

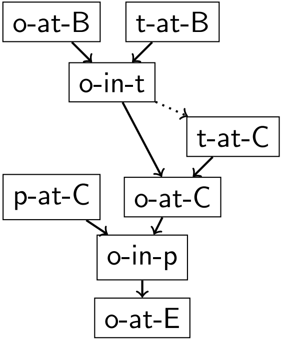
- We assume that landmarks and orderings are discovered in a pre-processing phase, and the same landmark graph is used throughout the planning phase

Using Landmarks as Subgoals

- Landmarks can be used as subgoals for a base planner
- The first layer of landmarks that have not yet been achieved is passed as a disjunctive goal to a base planner

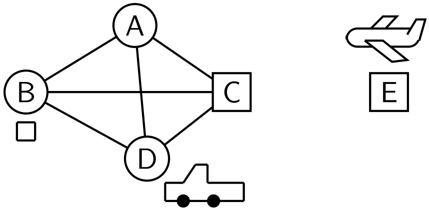
Using Landmarks as Subgoals - Logistics Example

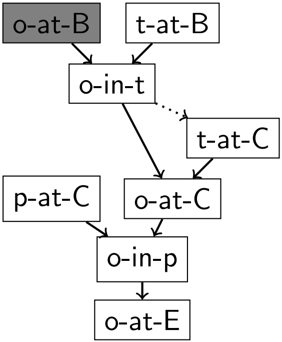




- Partial plan:
- Goal:

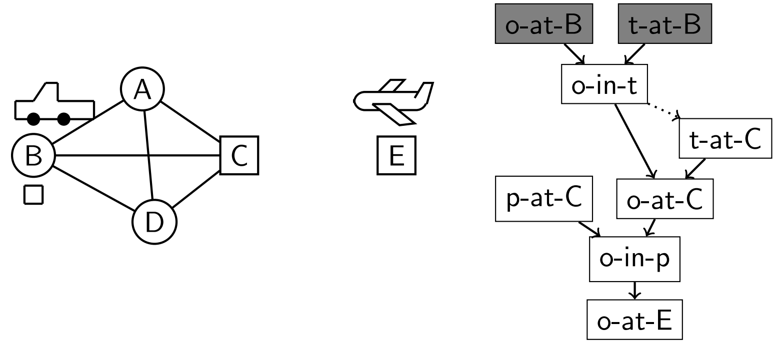
Using Landmarks as Subgoals - Logistics Example





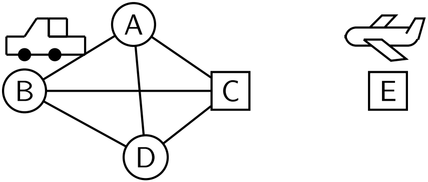
- Partial plan: $\emptyset$
- Goal: t-at-B $\vee$ p-at-C

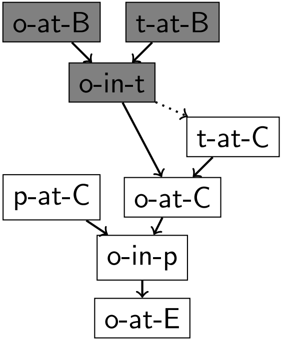
Using Landmarks as Subgoals - Logistics Example



- Partial plan: Drive-t-B
- Goal: o-in-t $\vee$ p-at-C

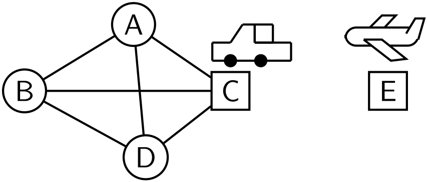
Using Landmarks as Subgoals - Logistics Example

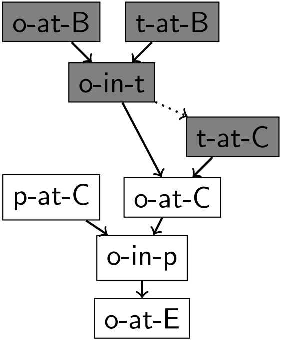




- Partial plan: Drive-t-B, Load-o-B
- Goal: t-at-C $\lor$ p-at-C

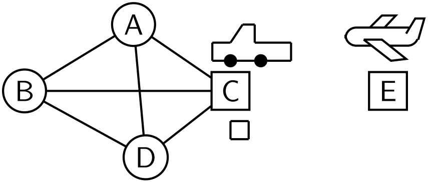
Using Landmarks as Subgoals - Logistics Example

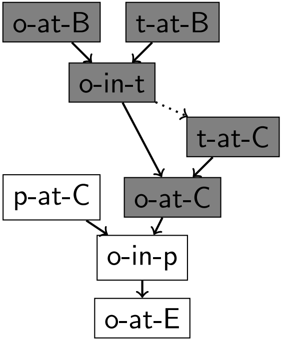




- Partial plan: Drive-t-B, Load-o-B, Drive-t-C
- Goal: o-at-C $\vee$ p-at-C

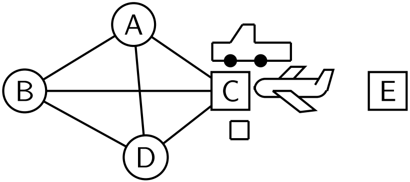
Using Landmarks as Subgoals - Logistics Example

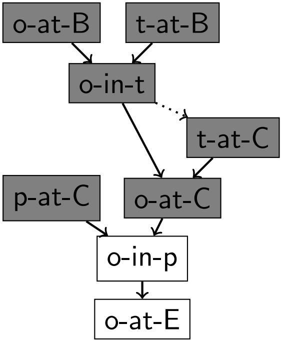




- Partial plan: Drive-t-B, Load-o-B, Drive-t-C, Unload-o-C
- Goal: p-at-C

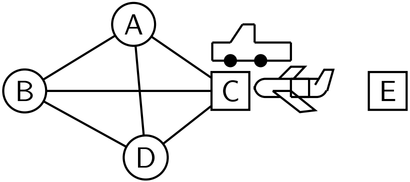
Using Landmarks as Subgoals - Logistics Example

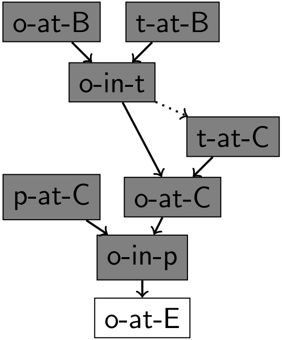




- Partial plan: Drive-t-B, Load-o-B, Drive-t-C, Unload-o-C, Fly-p-C
- Goal: o-in-p

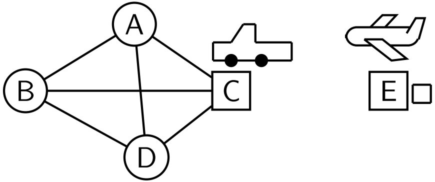
Using Landmarks as Subgoals - Logistics Example

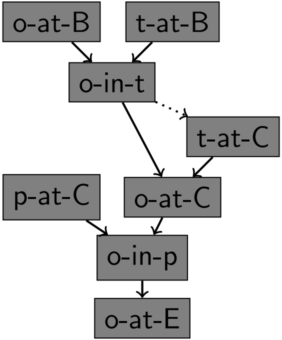




- Partial plan: Drive-t-B, Load-o-B, Drive-t-C, Unload-o-C, Fly-p-C, Load-o-p
- Goal: o-at-E

Using Landmarks as Subgoals - Logistics Example



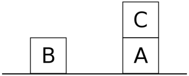


- Partial plan: Drive-t-B, Load-o-B, Drive-t-C, Unload-o-C, Fly-p-C, Load-o-p, Fly-p-E, Unload-o-E
- Goal: $\emptyset$

Using Landmarks as Subgoals

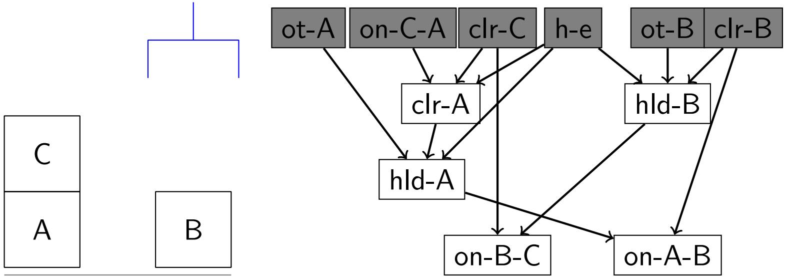
- That was a good example
- Now let's see a bad one

Using Landmarks as Subgoals - Sussman Example

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

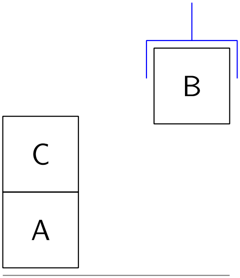
- Goal: *on-A-B*, *on-B-C*

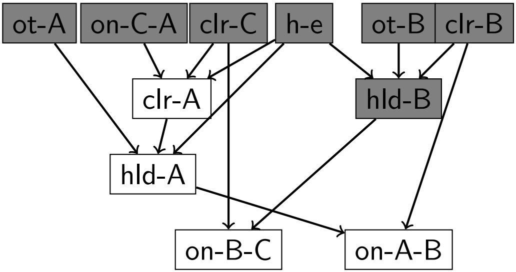
Using Landmarks as Subgoals - Sussman Example



- Partial plan: $\emptyset$
- Goal: clear-A $\vee$ holding-B

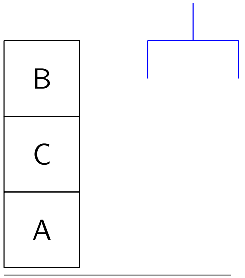
Using Landmarks as Subgoals - Sussman Example

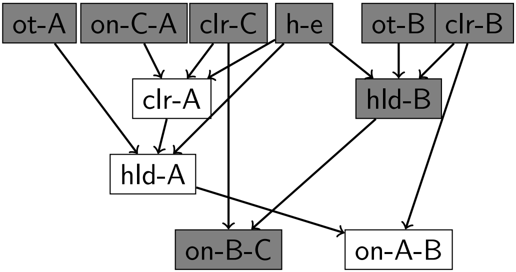




- Partial plan: Pickup-B
- Goal: clear-A $\vee$ on-B-C

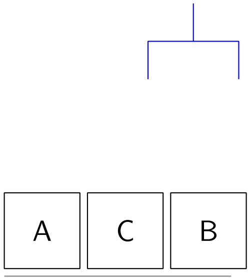
Using Landmarks as Subgoals - Sussman Example

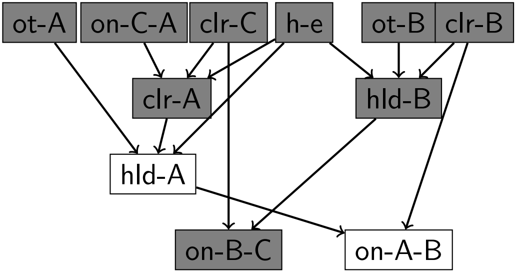




- Partial plan: Pickup-B, Stack-B-C
- Goal: clear-A

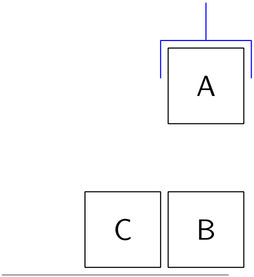
Using Landmarks as Subgoals - Sussman Example

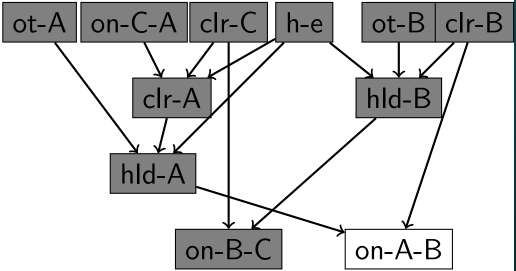




- Partial plan: Pickup-B, Stack-B-C, Unstack-B-C, Putdown-B, Unstack-C-A, Putdown-C
- Goal: holding-A

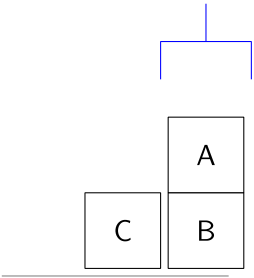
Using Landmarks as Subgoals - Sussman Example

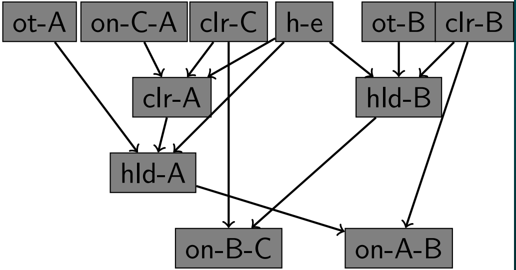




- Partial plan: Pickup-B, Stack-B-C, Unstack-B-C, Putdown-B, Unstack-C-A, Putdown-C, Pickup-A
- Goal: on-A-B

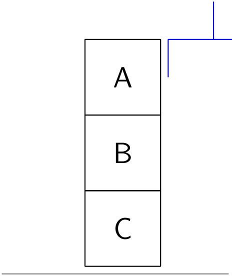
Using Landmarks as Subgoals - Sussman Example

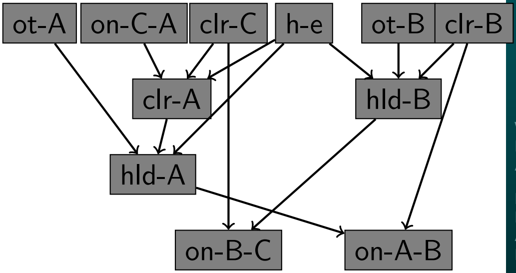




- Partial plan: Pickup-B, Stack-B-C, Unstack-B-C, Putdown-B, Unstack-C-A, Putdown-C, Pickup-A, Stack-A-B
- Goal: Still need to achieve on-B-C

Using Landmarks as Subgoals - Sussman Example





- Partial plan: Pickup-B, Stack-B-C, Unstack-B-C, Putdown-B, Unstack-C-A, Putdown-C, Pickup-A, Stack-A-B, Unstack-A-B, Putdown-A, Pickup-B, Stack-B-C, Pickup-A, Stack-A-B
- Goal: $\emptyset$

Using Landmarks as Subgoals - Pros and Cons

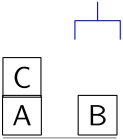
- Pros:
  - Planning is very fast - the base planner needs to plan to a lesser depth
- Cons:
  - Can lead to much longer plans
  - Not complete in the presence of dead-ends

Using Landmarks for Heuristic Estimates

- The number of landmarks that still need to be achieved is a heuristic estimate
- Used by LAMA (Richter, Helmert and Westphal 2008), winner of the IPC-2008 and IPC-2011 sequential satisficing track!

Path-dependent Heuristics

- Suppose we are in state s. Did we achieve landmark A yet?
- Example: did we achieve holding(B)?



- There is no way to tell just by looking at s
- Achieved landmarks are a function of path, not state
- The number of landmarks that still need to be achieved is a path-dependent heuristic

The Landmark Heuristic

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

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

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

The Landmark Heuristic

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

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

- 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

The Landmark Heuristic

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

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

- 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

The Landmark Heuristic

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

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

- 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

Accepted Landmarks

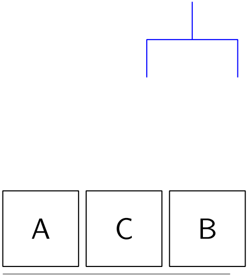
- In LAMA, a landmark A is first accepted by path π in state s if
  - all predecessors of A in the landmark graph have been accepted, and
  - A becomes true in s
- Once a landmark has been accepted, it remains accepted

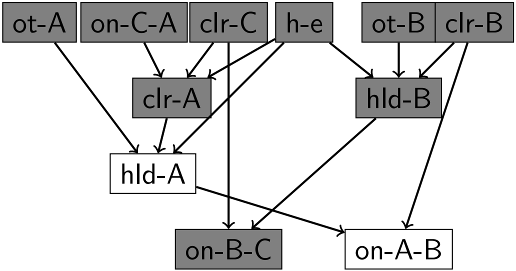
Required Again Landmarks

- A landmark A is required again by path π in state s if:
  - **false-goal** A is false in s and is a goal, or
  - **open-prerequisite** A is false in s and is a greedy-necessary predecessor of some landmark that is not accepted

Accepted and Required Again Landmarks - Example

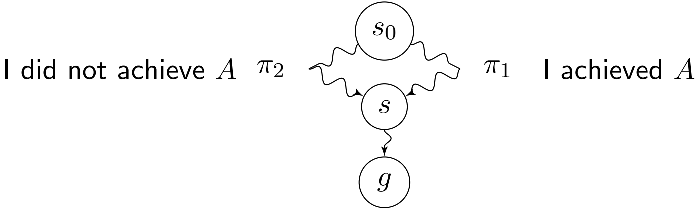
- In the Sussman anomaly, after performing: Pickup-B, Stack-B-C, Unstack-B-C, Putdown-B, Unstack-C-A, Putdown-C





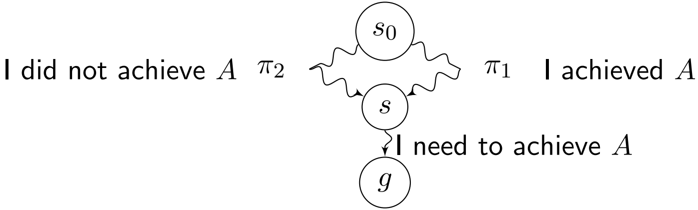
- on-B-C is a *false-goal*, and so it is required again

Multi-path Dependence



- Suppose state s was reached by paths π_1, π_2
- Suppose π_1 achieved landmark A and π_2 did not
- Conclusion:

Multi-path Dependence



- Suppose state s was reached by paths π_1, π_2
- Suppose π_1 achieved landmark A and π_2 did not
- Conclusion: A needs to be achieved after state s
- Proof: A 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 $\mathcal{P}$ is a set of paths from s_0 to a state s. Define

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

  where

  - $\text{Accepted}(s, \mathcal{P}) = \bigcap_{\pi \in \mathcal{P}} \text{Accepted}(s, \pi)$
  - $\text{ReqAgain}(s, \mathcal{P}) \subseteq \text{Accepted}(s, \mathcal{P})$ is specified as before by s and the various rules

- $L(s, \mathcal{P})$ is the set of landmarks that we know still needs to be achieved after reaching state s via the paths in $\mathcal{P}$

Admissible Heuristic Estimates

- LAMA's heuristic: the number of landmarks that still need to be achieved (Richter, Helmert and Westphal 2008)
- LAMA's heuristic is inadmissible - a single action can achieve multiple landmarks
  - Example: *hand-empty* and *on-A-B* can both be achieved by *stack-A-B*
- Admissible heuristic: assign a cost to each landmark, sum over the costs of landmarks (Karpas and Domshlak, 2009)

Conditions for Admissibility

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

$$\forall a \in A : \sum_{A \in L(a|s,\mathcal{P})} cost(a, A) \leq C(a)$$

$cost(a, A)$: cost "assigned" by action a to A
$L(a|s, \mathcal{P})$: the set of landmarks achieved by a

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

$$\forall A \in L(s, \mathcal{P}) : \ cost(A) \leq \min_{a \ \in \ \text{ach}(A|s,\mathcal{P})} cost(a, A)$$

$cost(A)$: cost assigned to landmark A
$\text{ach}(A|s, \mathcal{P})$: the set of actions that can achieve A

Conditions for Admissibility

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

$$\forall a \in A: \sum_{A \in L(a|s,\mathcal{P})} cost(a, A) \leq C(a)$$

$cost(a, A)$: cost "assigned" by action a to A
$L(a|s, \mathcal{P})$: the set of landmarks achieved by a

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

$$\forall A \in L(s, \mathcal{P}): \; cost(A) \leq \min_{a \in \mathsf{ach}(A|s,\mathcal{P})} cost(a, A)$$

$cost(A)$: cost assigned to landmark A
$\mathsf{ach}(A|s, \mathcal{P})$: the set of actions that can achieve A

Admissible Cost Sharing

- Idea: the cost of a set of landmarks is no greater than the cost of any single action that achieves them
- Given that, the sum of costs of landmarks that still need to be achieved is an admissible heuristic, h_L

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

- Proof: Homework $\ddot{\smile}$

Cost Partitioning - how?

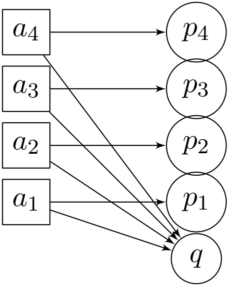
- How can we find such a partitioning?
- Easy answer - uniform cost sharing - each action shares its cost equally between the landmarks it achieves

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

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

Uniform Cost Sharing

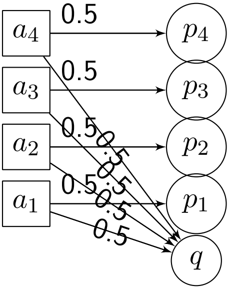
- Advantage: Easy and fast to compute
- Disadvantage: Can be much worse than the optimal cost partitioning



Uniform Cost Sharing

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

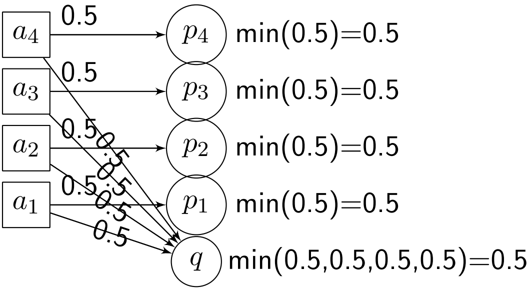
Uniform cost sharing



Uniform Cost Sharing

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

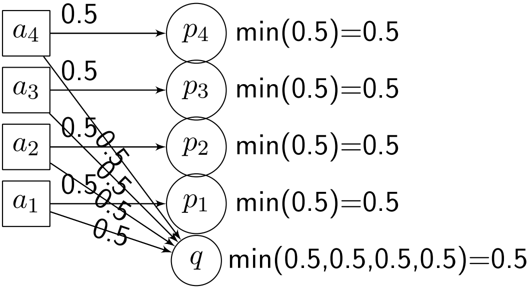
Uniform cost sharing



Uniform Cost Sharing

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

Uniform cost sharing $h_L = 2.5$

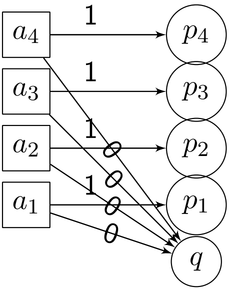


Uniform Cost Sharing

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

Optimal cost sharing

uniform $h_L = 2.5$

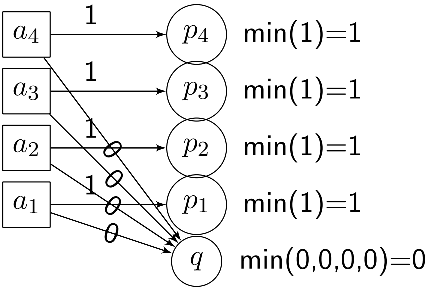


Uniform Cost Sharing

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

Optimal cost sharing

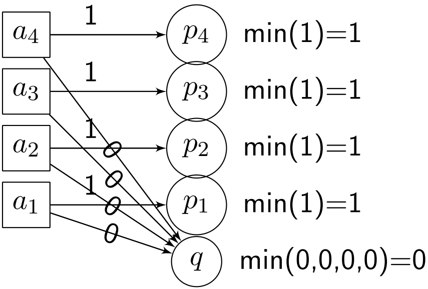
uniform $h_L = 2.5$



Uniform Cost Sharing

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

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



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 Program (LP)
  - Objective: 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

- Sounds familiar?
  1. Indeed, but can be just a coincidence.

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 Program (LP)
  - Objective: 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
- Sounds familiar?
  1. Indeed, but can be just a coincidence.

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 Program (LP)
  - Objective: 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
- Sounds familiar?
  1. Indeed, but can be just a coincidence.
  2. Not a coincidence: special case of action-cost partitioning for abstractions.

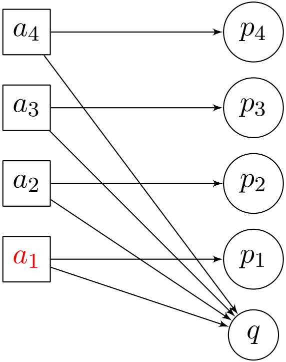
How can we do better?

- 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:

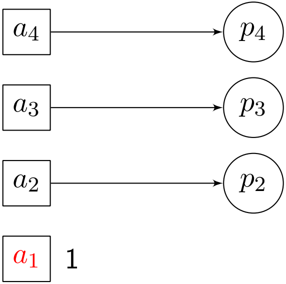
How can we do better?

- 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

Using Action Landmarks - by Example



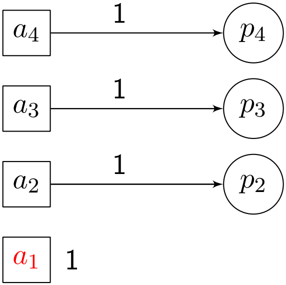
Using Action Landmarks - by Example



Using Action Landmarks - by Example

Uniform Cost Sharing

$h_{LA} = 4$



min(1)=1

min(1)=1

min(1)=1

Summary

- Landmarks provide a way to utilize the implicit structure of a planning problem
- They can (and have been) used successfully for both satisficing and optimal planning
- The envelope of heuristic functions has been pushed!