Assignments

Problems Sets:

- Problem Set 2 (Scheduling) due
- Problem Set 3 (PDDL Modeling) out soon

Readings:

- Hoffman, Porteous, Sebastia, "Ordered Landmarks in Planning," Journal of Artificial Intelligence Research, 22, pp. 215-278, 2004. (Voted most influential paper during ICAPS 2013).
- Karpas, *et al.*, "Temporal Landmarks: What Must Happen, and When," 25th International Conference on Automated Planning and Scheduling, 138-146, 2015.
- Fox and Long, "PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains", Journal of Artificial Intelligence Research, 20, 61-124, 2003.

Planning with Temporal Landmarks

Tiago Vaquero and Christian Muise

Drawn from material by:
Karpas' lecture (16.412-2015)
Karpas and Richter (ICAPS Tutorial 2010)

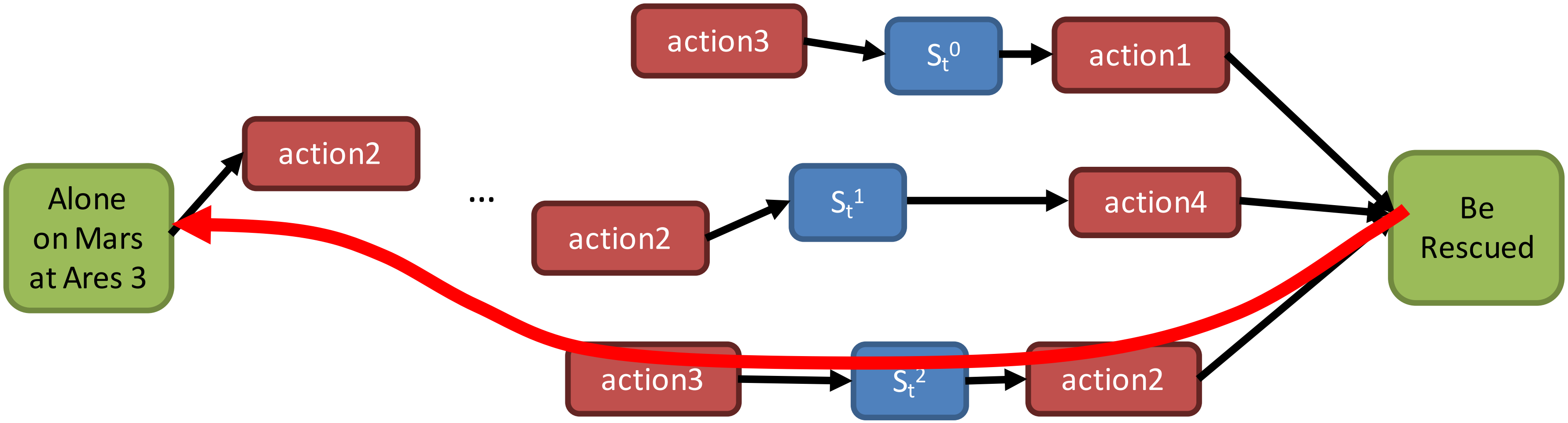
February 24, 2016

Motivation

Alone on Mars at Ares 3

Be Rescued

Motivation

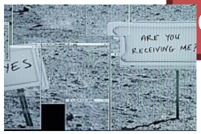


action3
S_t^0
action1
action2
...
Alone on Mars at Ares 3
action2
S_t^1
action4
Be Rescued
action3
S_t^2
action2

Motivation

Alone on Mars at Ares 3

Have enough food and water



Re-Establish communication

Get the Rover ready for long trips

Drive to Ares 4

Be at Ares 4

Have MAV ready

Be Rescued

[0, 4 years]

Outline

- **What Landmarks Are**
- How Landmarks Are Discovered
- Using Landmarks
  - Subgoals
  - Heuristic Estimates
  - Admissible Heuristic Estimates
  - Enriching the Problem
  - Beyond Classical Planning
- Summary

What Landmarks Are

- A landmark is a *logical formula* that *must* be true at some point in every plan

Types of Landmarks

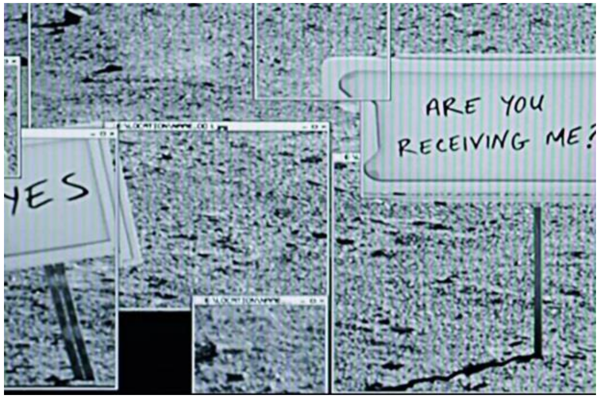
- Fact Landmarks
- Action Landmark
- Temporal Landmark

Fact Landmarks

- A fact landmark is a fact that *must* be true at some point in every plan (Hoffmann, Porteous & Sebastia 2004)
  - *To get to Ares 4, I need to have the rover ready for the trip*

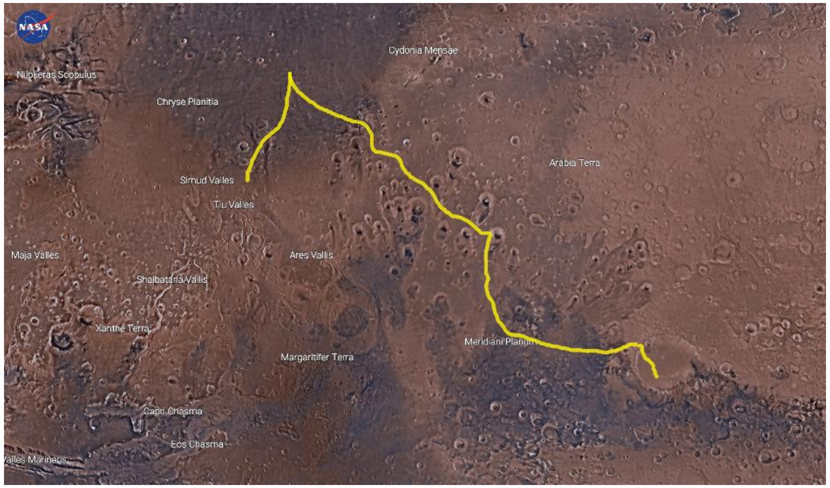
Action Landmarks

- An action landmark is an action which occurs in every valid plan
  - *To tell I am alive, I need to re-establish communication*
  - *To get to the rover, I need to exit this the Hab*



Fact and Action Landmarks

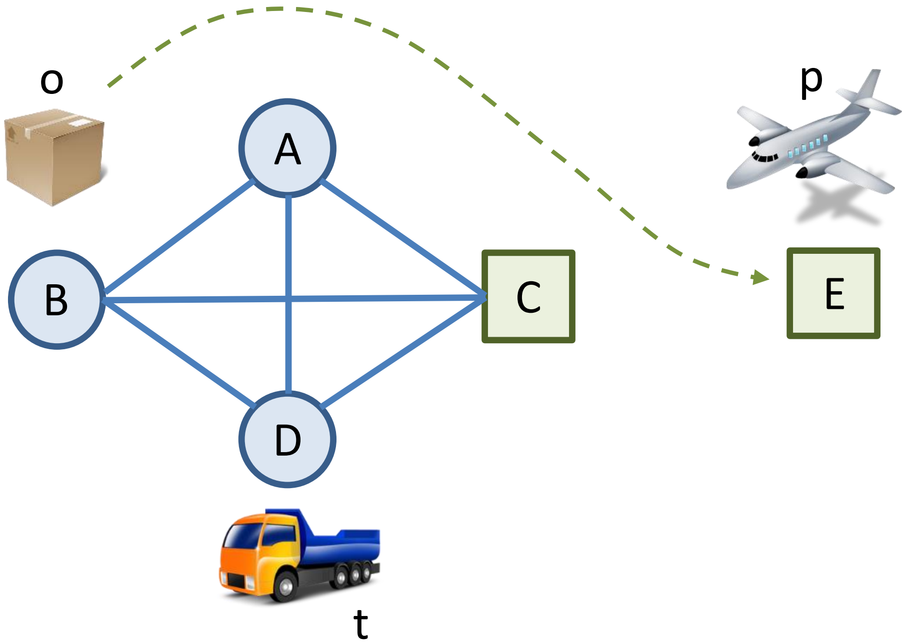
- We can also consider disjunctions over facts and/or actions
  - *To get back to Ares 4, I need to take route A or route B*

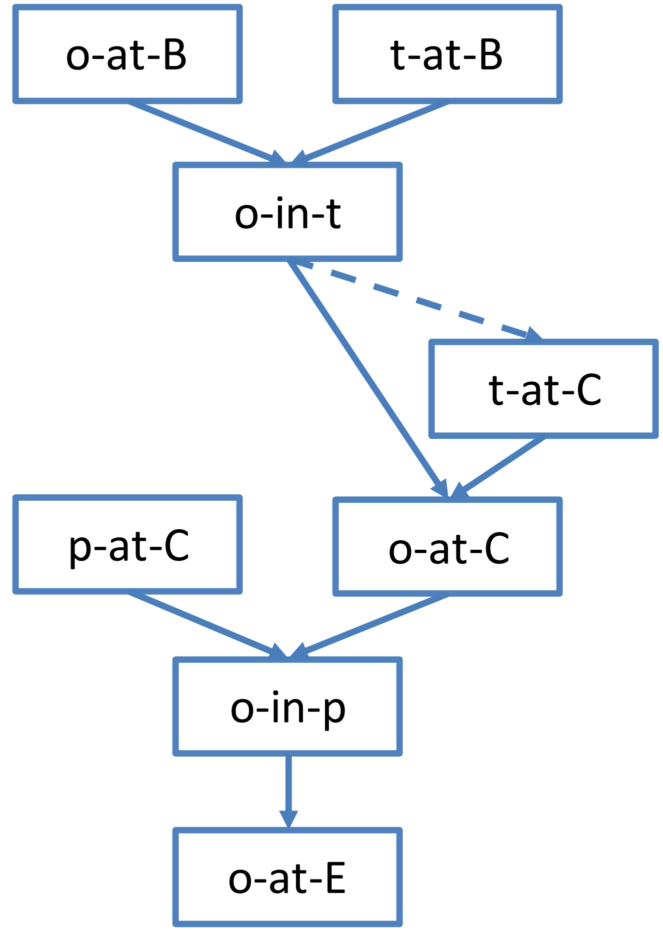


Temporal Landmarks

- A temporal fact landmark is a formula over facts that becomes true from time point t_s to t_e in every valid plan.
  - I need to be at Ares 4 within 4 years
- A temporal action landmark is an action which occurs at time point t in every valid plan
  - I have to launch the MAV from Ares 4 at 549 sols

Example

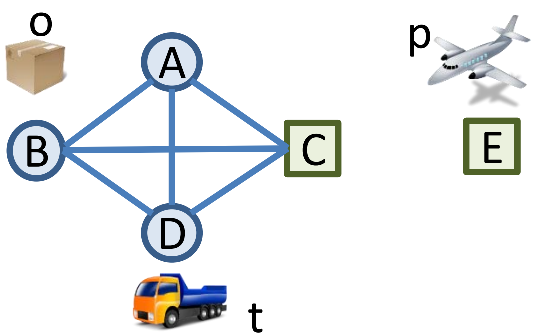




Partial Landmarks Graph

Landmark Ordering

- Landmarks can be (partially) ordered according to the order in which they must be achieved
  - *Truck needs to load package before driving to the airport*
  - *Airplane must be at the airport before loading the package*



- Some landmarks and orderings can be discovered *automatically*

Landmark Ordering

- ***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
- Other ordering types exist, which we do not discuss

- Note that:

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

Outline

- What Landmarks Are
- **How Landmarks Are Discovered**
- Using Landmarks
  - Subgoals
  - Heuristic Estimates
  - Admissible Heuristic Estimates
  - Enriching the Problem
  - Beyond Classical Planning
- Summary

Landmarks Complexity

Remember: Planning is PSPACE-complete

Landmarks:

- Everything is PSPACE-complete
- Deciding if a given fact is a landmark is PSPACE-complete
- Deciding if there is a natural / necessary / greedy-necessary / reasonable ordering between two landmarks is PSPACE-complete

Landmark Discovery

- Theory

$$A \text{ is a } \textbf{landmark} \iff \pi'_A \text{ is unsolvable}$$

where π'_A *is* π *without the operators that achieve A*

  – Delete relaxation of is π'_A unsolvable $\Longrightarrow$ π'_A unsolvable delete relaxation landmarks – but better methods exists

Landmark Discovery

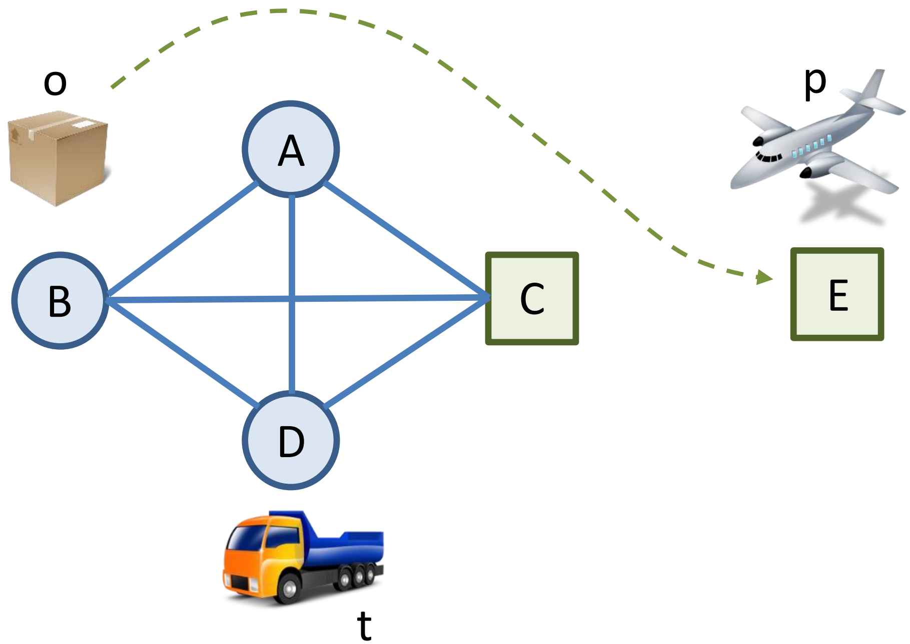
- Methods that are used in practice:
  1. Backchaining
  2. Domain Transition Graphs
  3. Forward Propagation

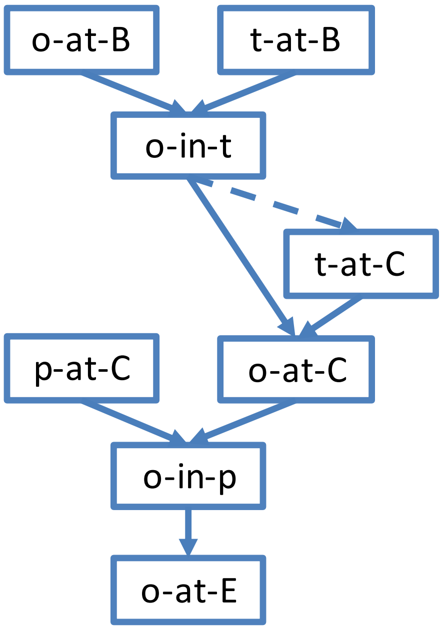
Landmark Discovery (1)

Find landmarks and orderings by backchaining (Hoffmann et al. 2004)

Step 1: Find Landmark Candidates and Orderings

Step 2: Verify Landmark Candidates



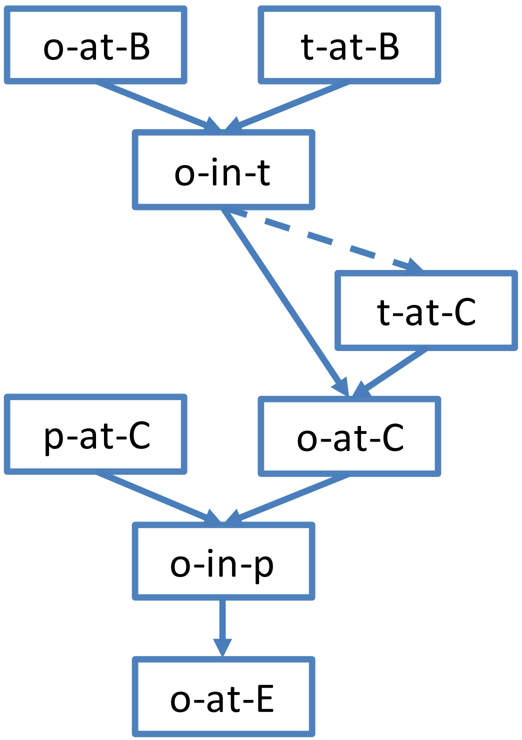


Landmark Discovery (1)

Step 1: Find landmarks candidates and orderings

- Start with the goals: every goal is a landmark
- If *B* is landmark and all actions that achieve *B* share *A* as precondition, then
  - *A* is a landmark
  - $A \rightarrow_n B$

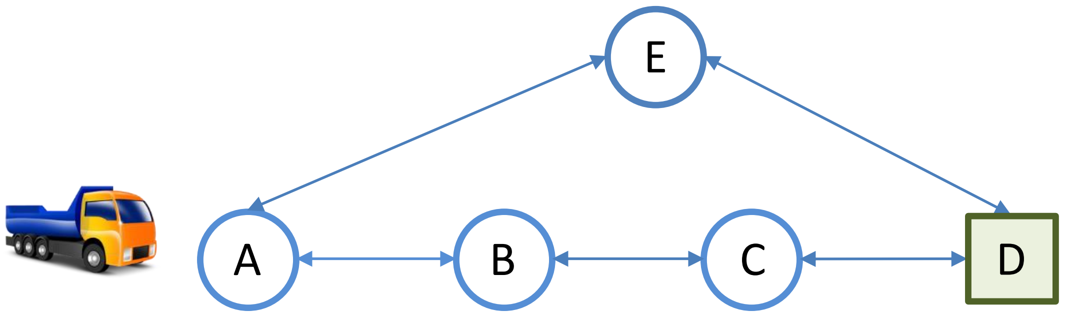
- *Useful restriction:* consider only the case where *B* is achieved for the first time
  - Relaxed Planning Graph to find first achievers
  - find more landmarks (and $A \rightarrow_{gn} B$)



Landmark Discovery (1)

Step 1: Find landmarks candidates and orderings

Example

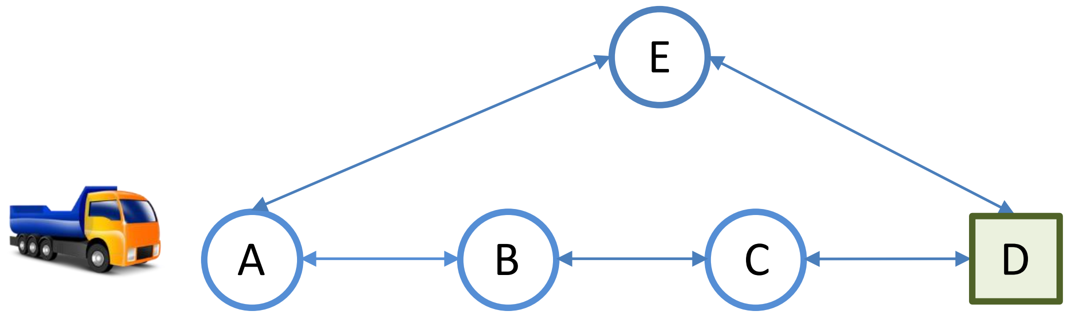


t-at-A

Landmark Discovery (1)

Step 1: Find landmarks candidates and orderings

Example



Landmark: t-at-D

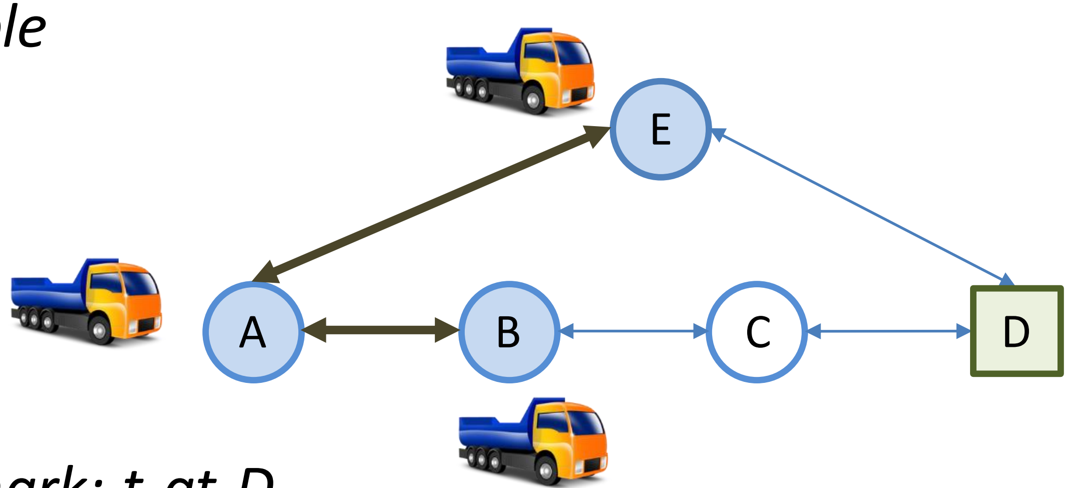
t-at-A

t-at-D

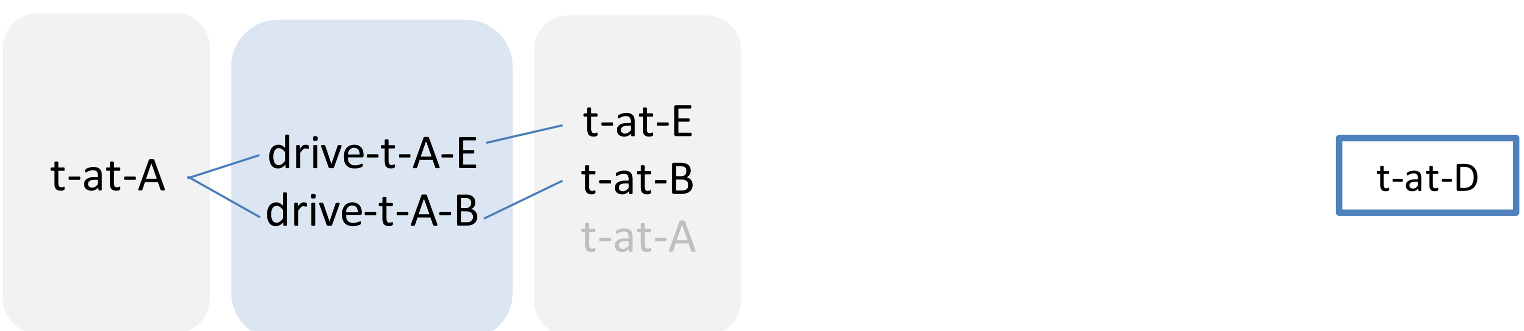
Landmark Discovery (1)

Step 1: Find landmarks candidates and orderings

Example



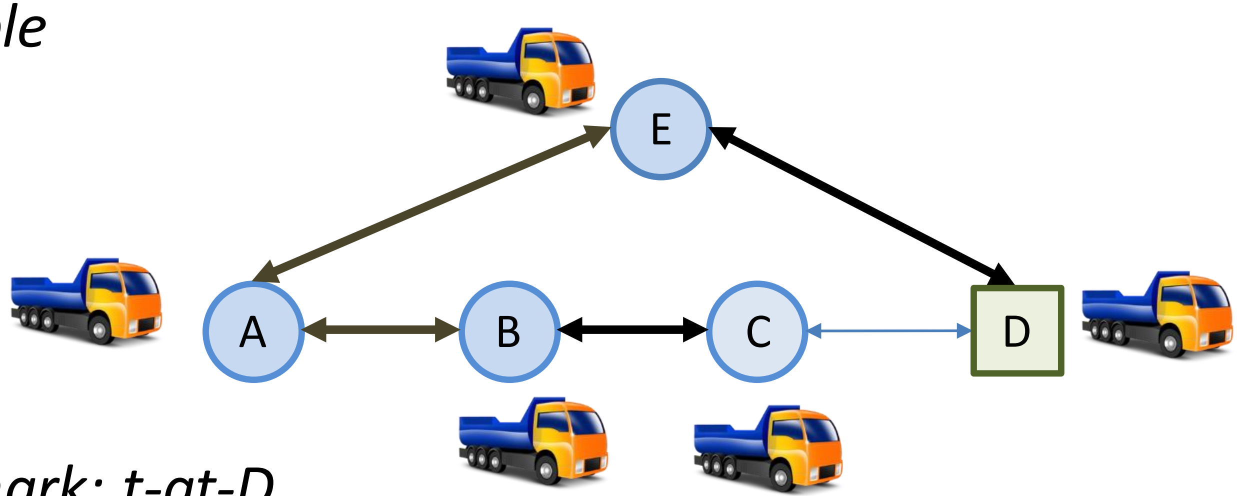
Landmark: t-at-D



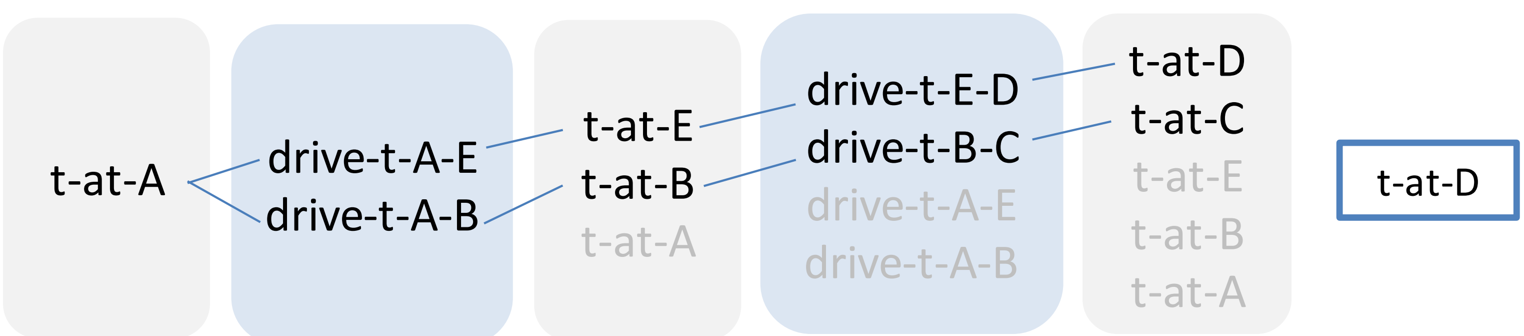
Landmark Discovery (1)

Step 1: Find landmarks candidates and orderings

Example



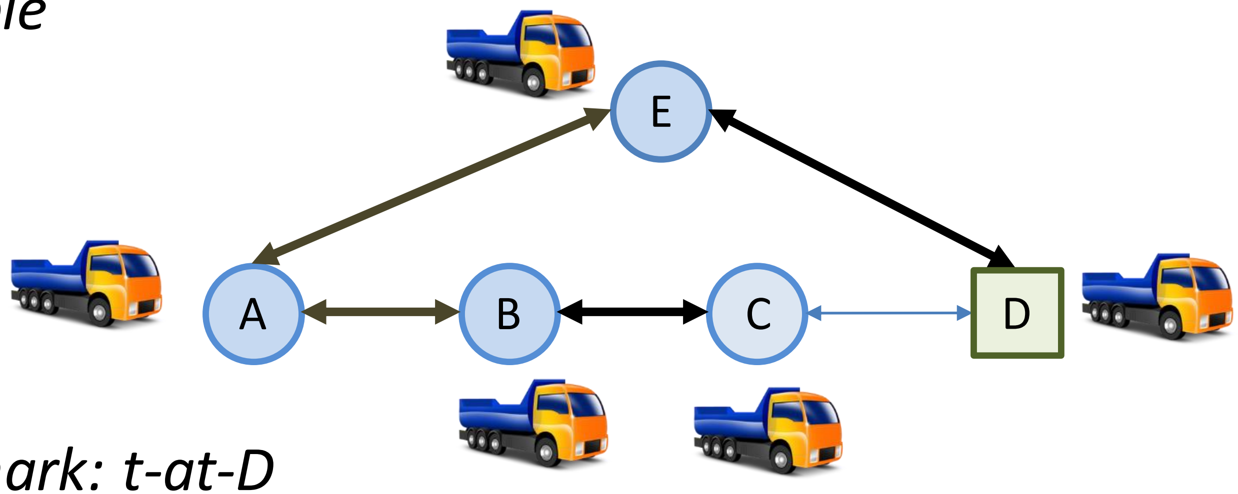
Landmark: t-at-D



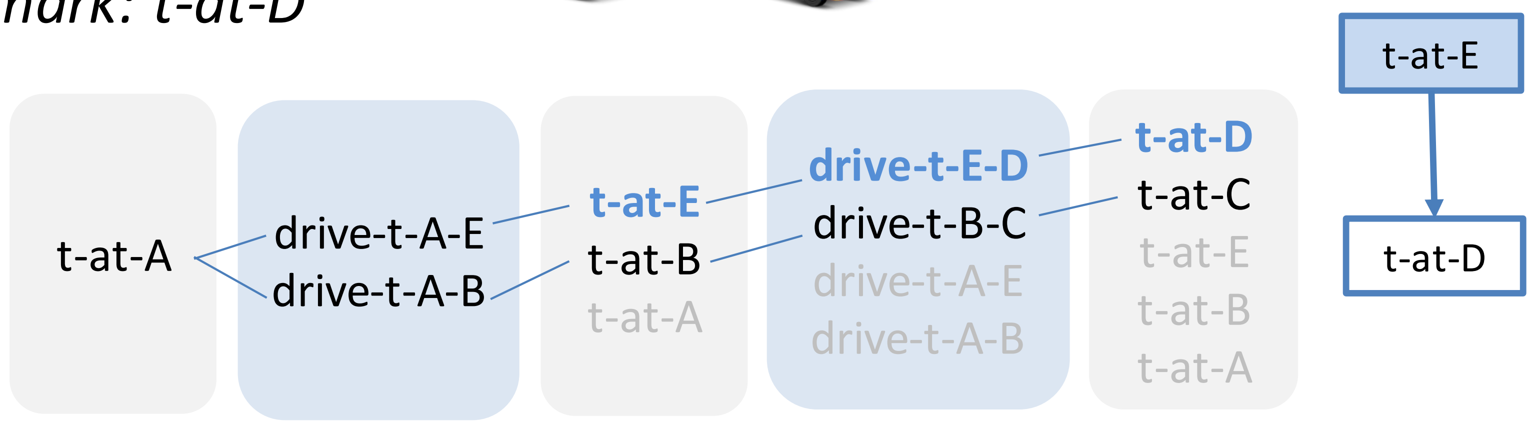
Landmark Discovery (1)

Step 1: Find landmarks candidates and orderings

Example



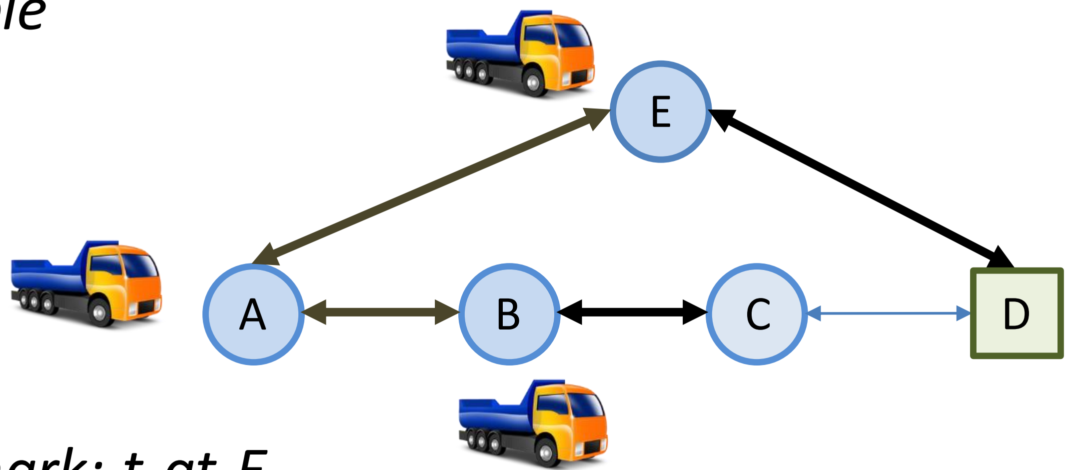
Landmark: t-at-D



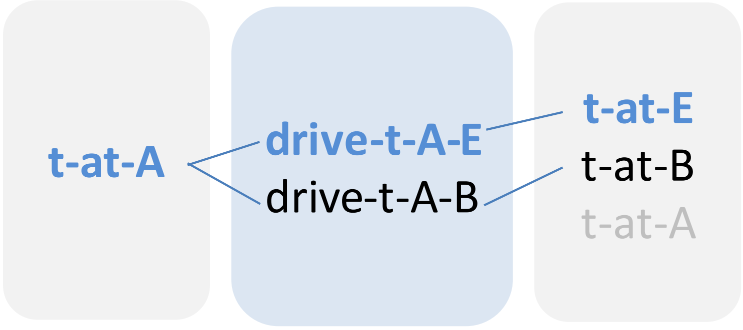
Landmark Discovery (1)

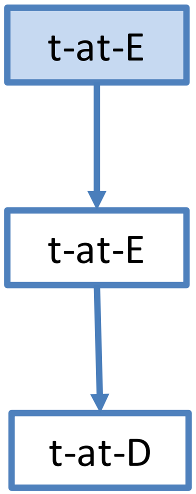
Step 1: Find landmarks candidates and orderings

Example



Landmark: t-at-E





{A, E, D}

Landmark Discovery (1)

Step 1: Find landmarks candidates and orderings

initialize the LGG to $(G, \emptyset)$, and set $C := G$

while $C = \emptyset$ **do**

  set $C' := \emptyset$

  for all $L' \in C$, $level(L') \neq 0$ **do**

    let A be the set of all actions a such that $L' \in add(a)$, and $level(a) = level(L') - 1$

    for all facts L such that $\forall a \in A : L \in pre(a)$ **do**

      if L is not yet a node in the LGG, set $C' := C' \cup \{L\}$

      if L is not yet a node in the LGG, then insert that node

      if $L \rightarrow_{gn} L'$ is not yet an edge in the LGG, then insert that edge

    endfor

  endfor

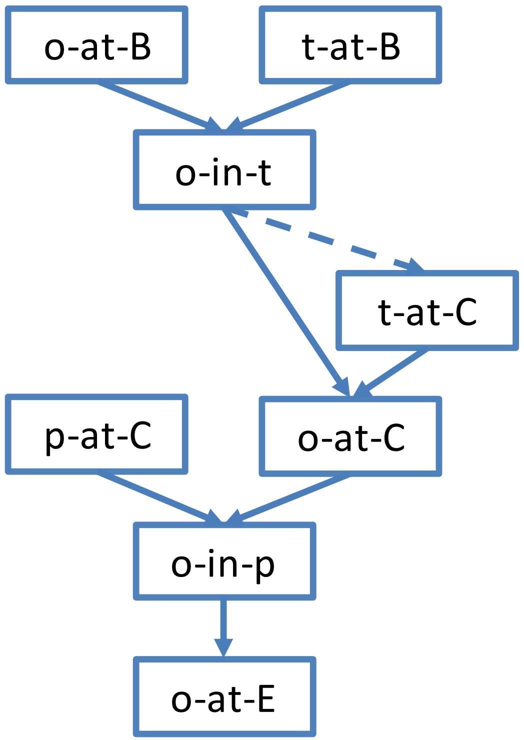
  set $C := C'$

endwhile

Landmark Discovery (1)

Step 2: Verify Landmark Candidates

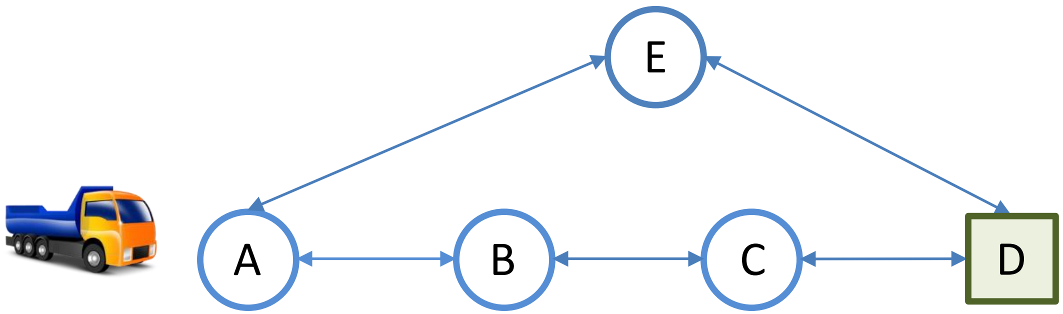
- For each landmark discovered:
  - Remove all the action that can achieve it
  - Build relaxed planning graph for π'_A and check if we can find the goals
  - If so, remove landmark and ordering from the landmark graph

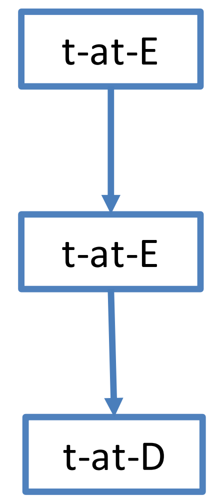


Landmark Discovery (1)

Step 2: Verify Landmark Candidates

Example



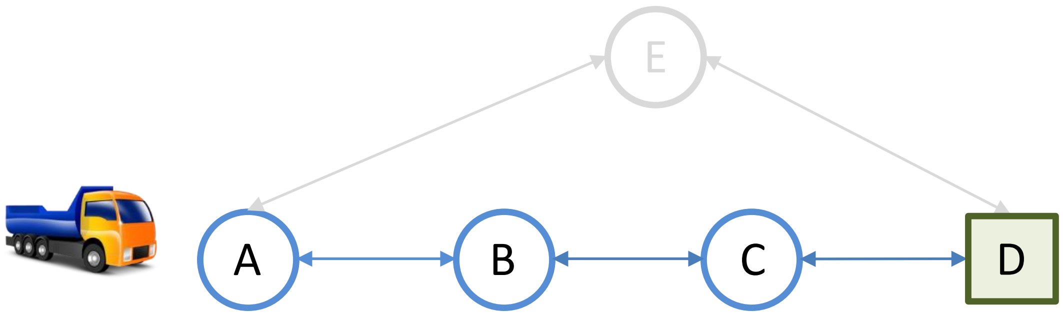


{A, E, D}

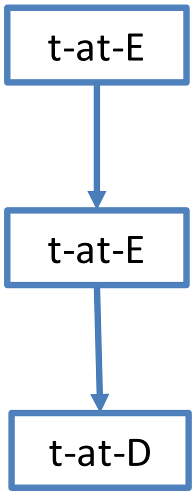
Landmark Discovery (1)

Step 2: Verify Landmark Candidates

Example



Check Landmark: t-at-E

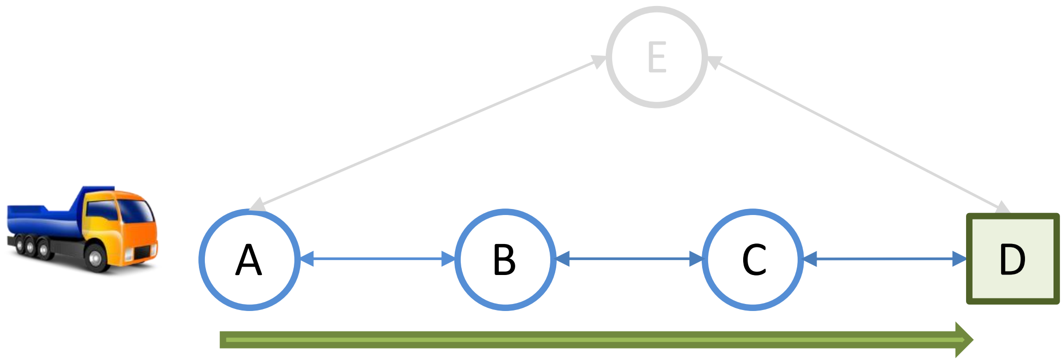


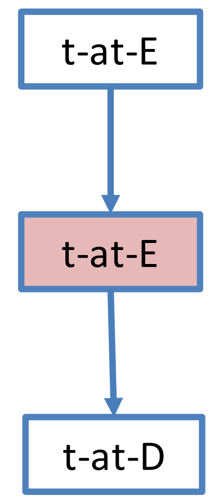
{A, E, D}

Landmark Discovery (1)

Step 2: Verify Landmark Candidates

Example





Check Landmark: t-at-E

Remove t-at-E and its orderings

Landmarks: {A, D}

{A, ~~E,~~ D}

Landmark Discovery (1)

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

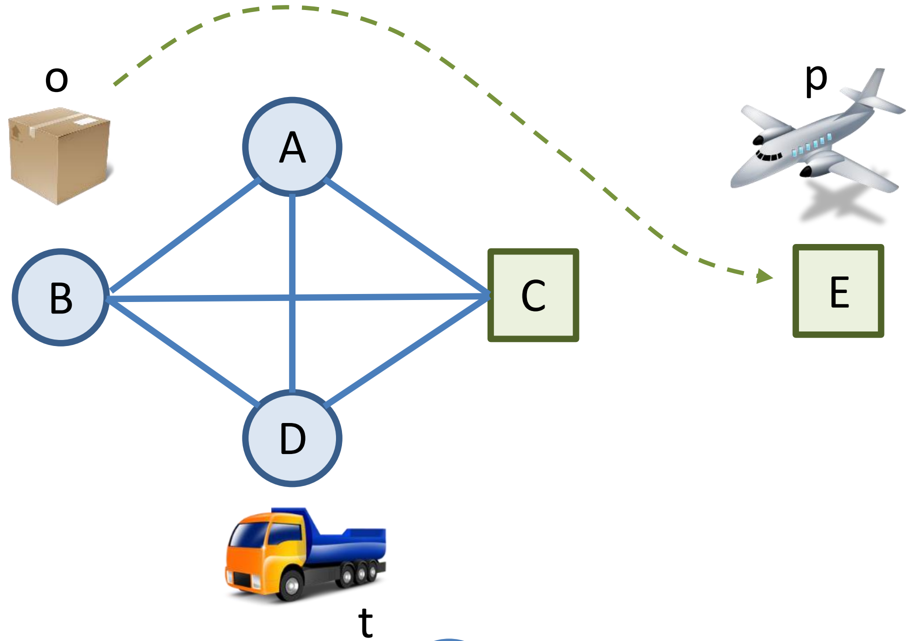
- If *B* is landmark and all actions that (first) achieve *B* have *A* or *C* as precondition, then $A \vee C$ is a landmark.

Landmark Discovery (2)

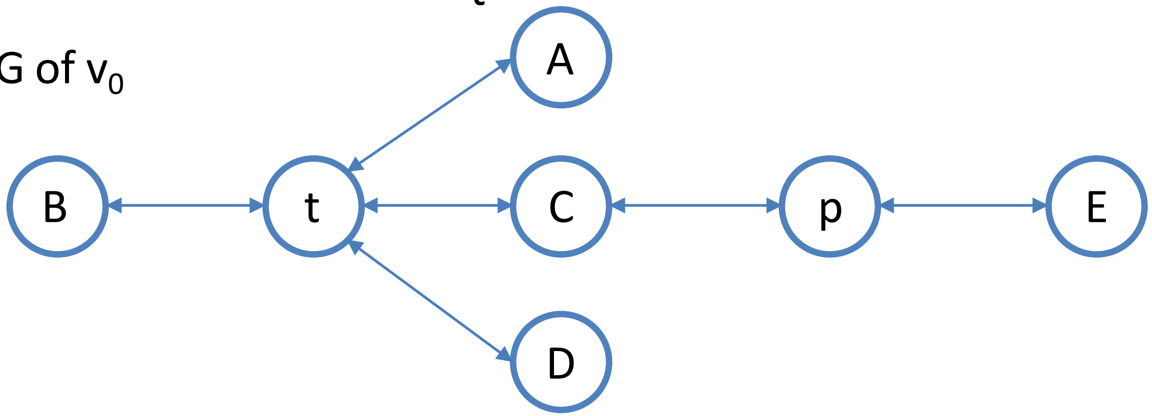
Find landmarks through Domain Transition Graphs (DTGs) (Richter et al. 2008)

- Given: a SAS+ task $< V, A, s_0, G >$
- The DTG of variable $v \in V$ (DTG_v) represents how the value of v can change.
- DTG_v is a directed graph with *nodes* D_v that has *arc* $<d, d'>$ 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

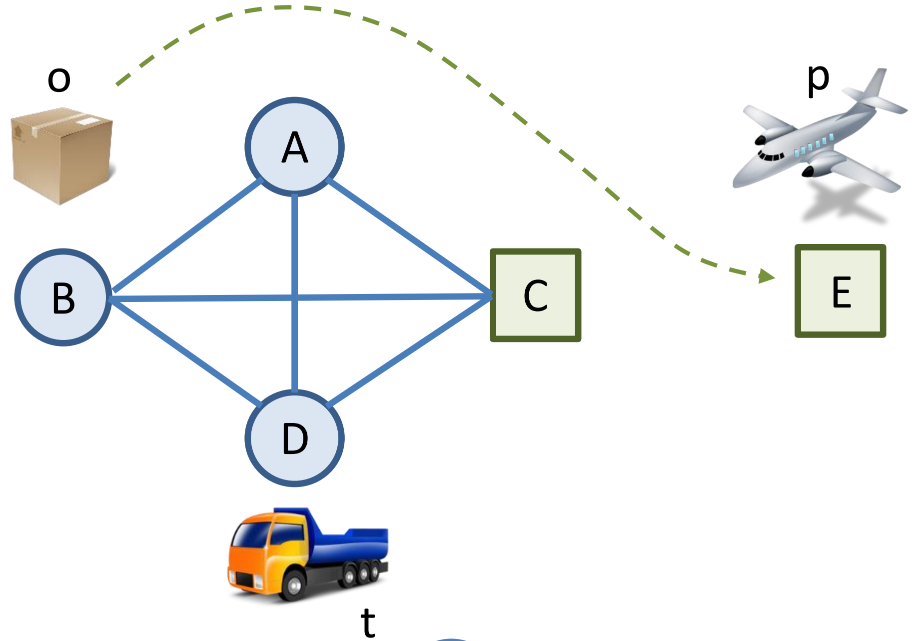
Landmark Discovery (2)



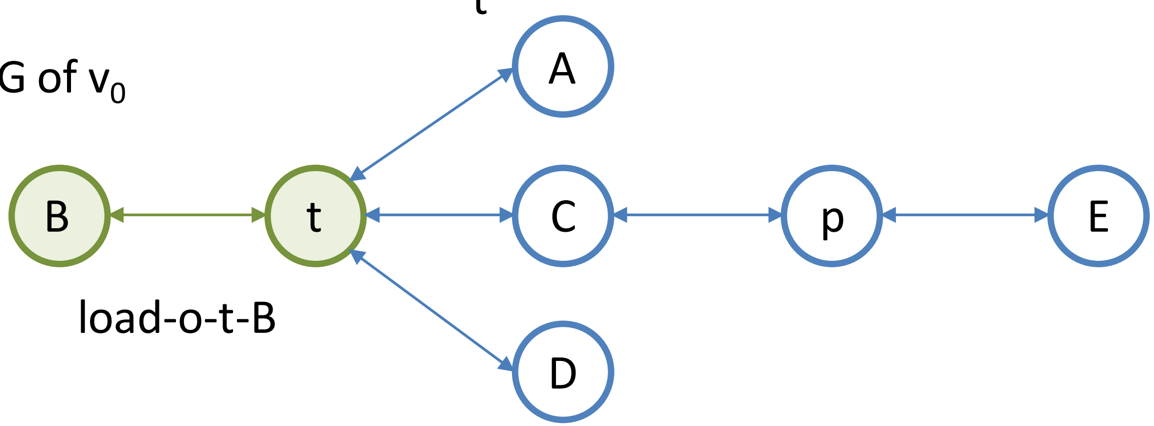
Package: DTG of v_0



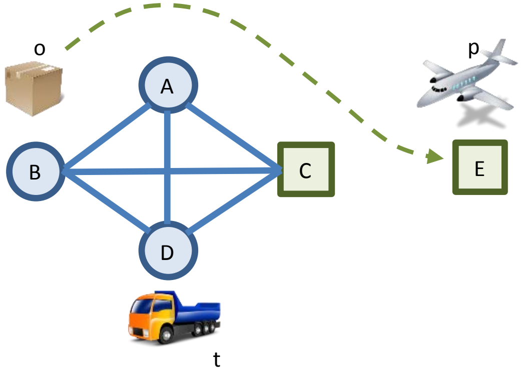
Landmark Discovery (2)

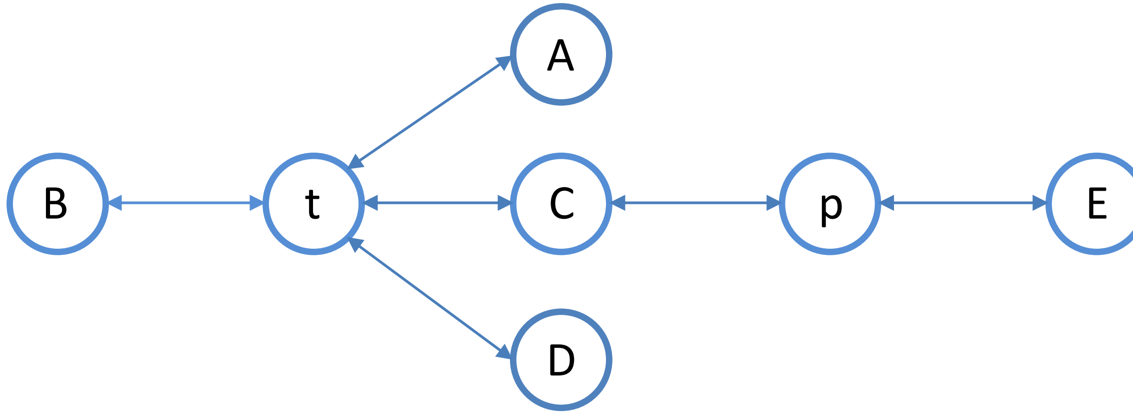


Package: DTG of v_0



Landmark Discovery (2)



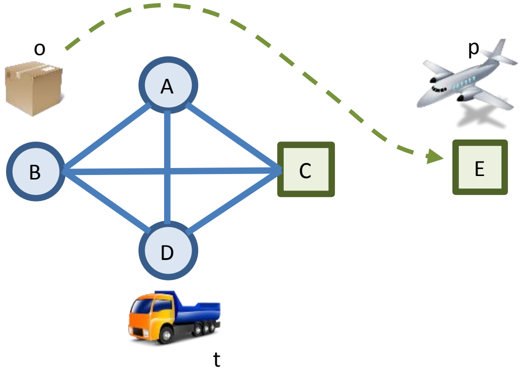


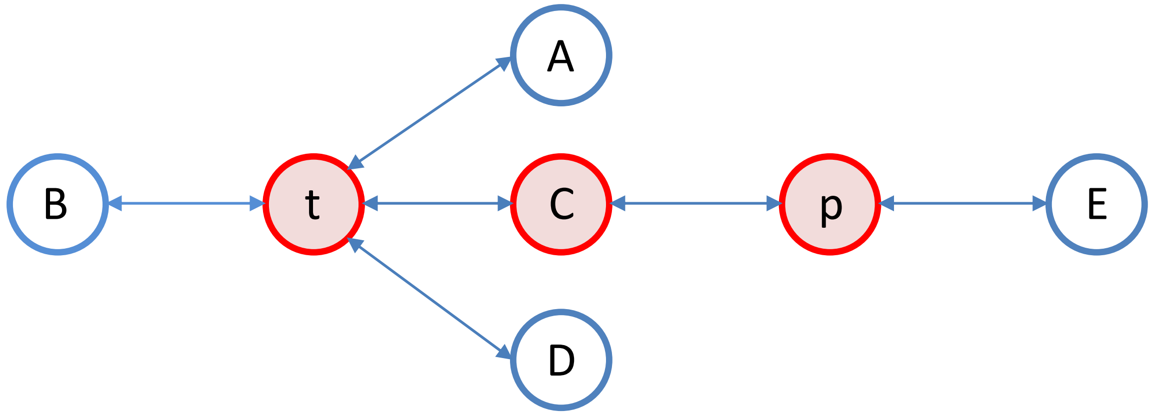
Find landmarks through DTGs, if

- $s_0(v) = d_0$,
- $v \mapsto d$ landmark (goal), and
- *every path* from d_0 to d passes through d',

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

Landmark Discovery (2)





Find landmarks through DTGs, if

- $s_0(v) = d_0$,
- $v \mapsto d$ landmark (goal), and
- *every path* from d_0 to d passes through d',

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

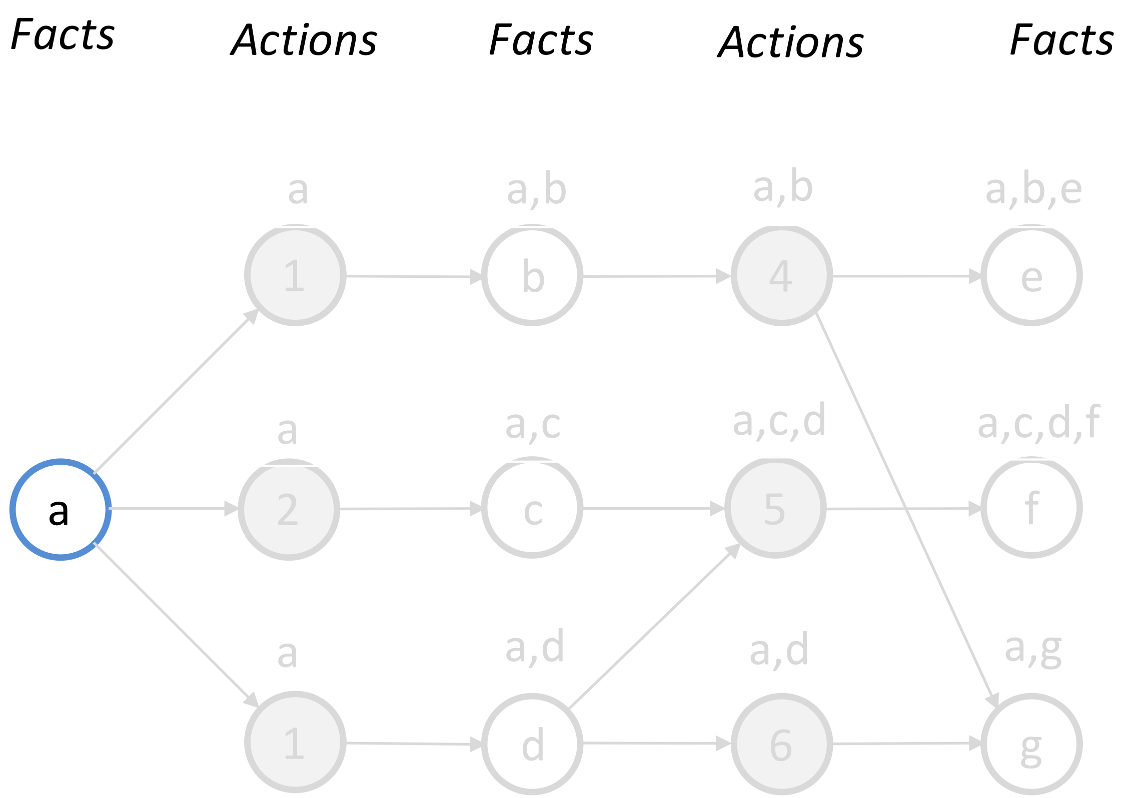
Landmark Discovery (3)

Find landmarks through forward propagation in relaxed planning graph

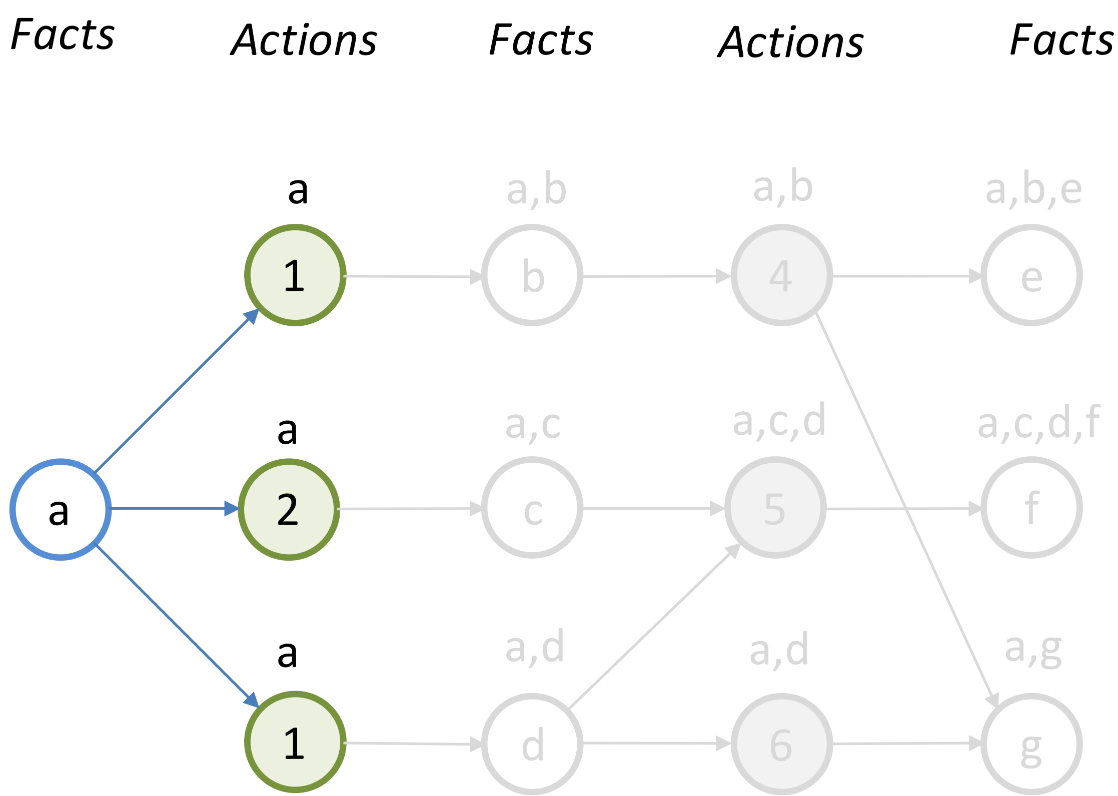
- Propagate information on necessary predecessors
  - Label each fact node with itself
  - Propagate labels along arcs

- Finds causal landmarks only (preconditions for actions)
- Finds all causal delete-relaxation landmarks in polynomial time

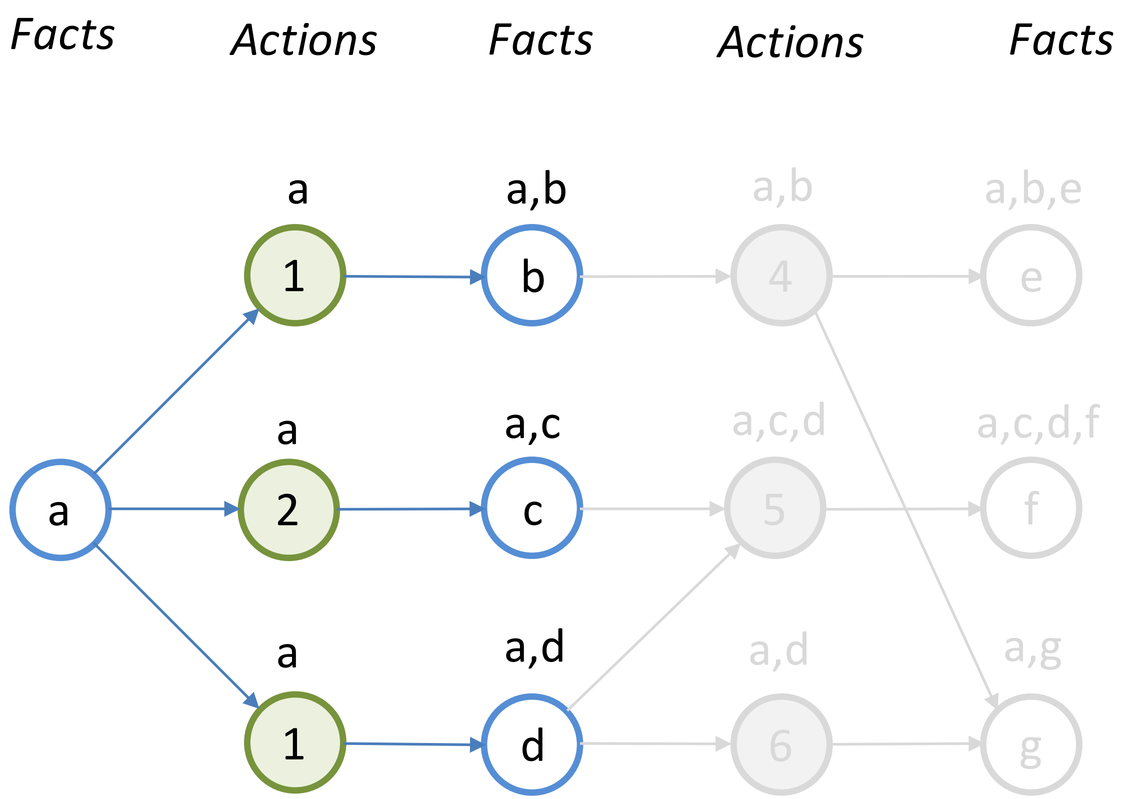
Landmark Discovery (3)



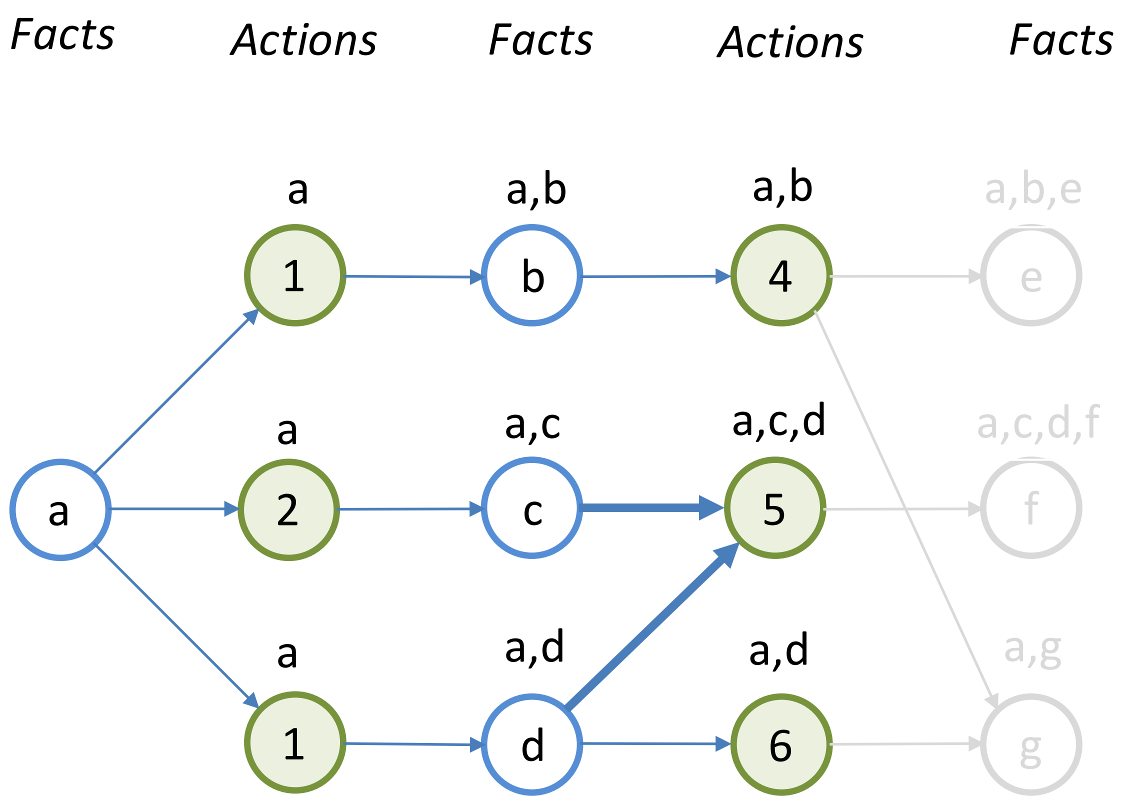
Landmark Discovery (3)



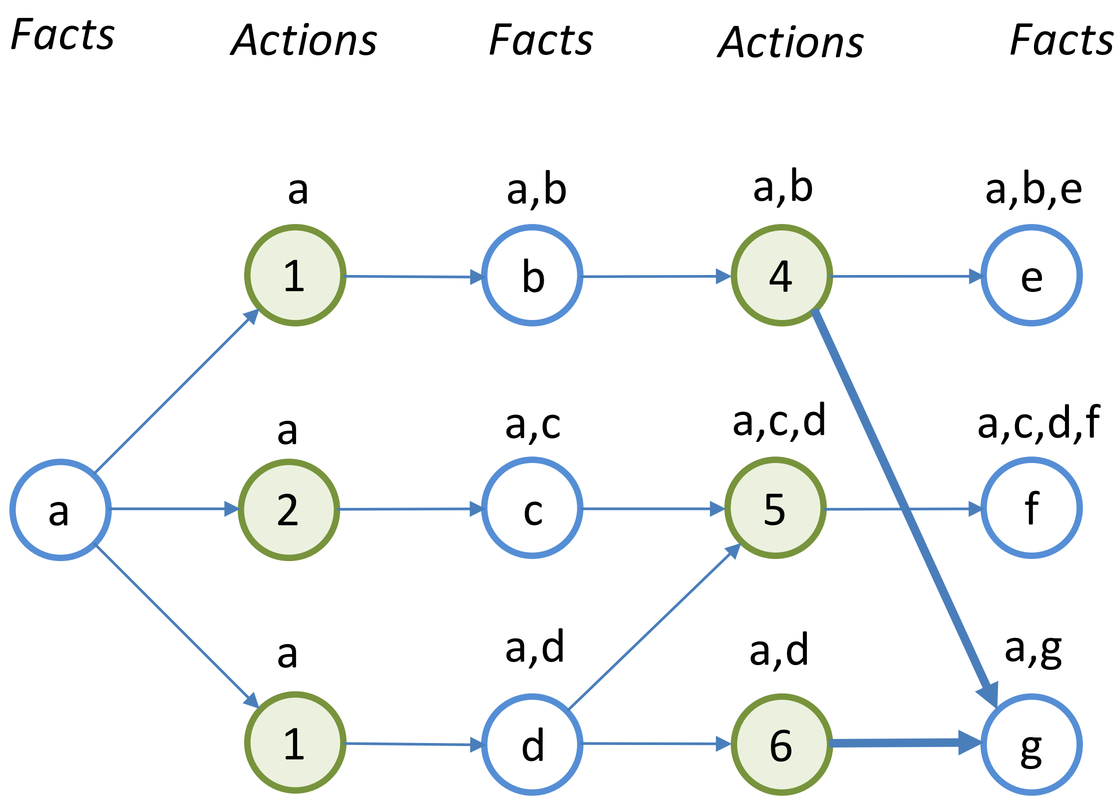
Landmark Discovery (3)



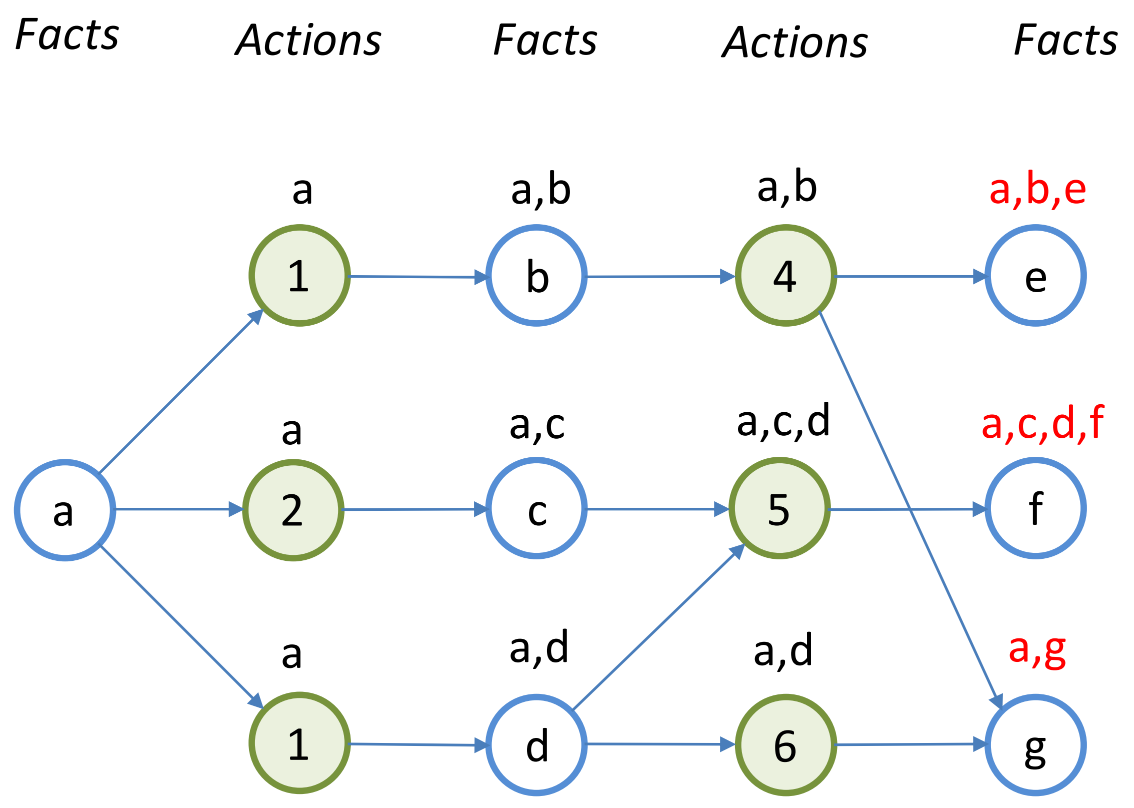
Landmark Discovery (3)



Landmark Discovery (3)

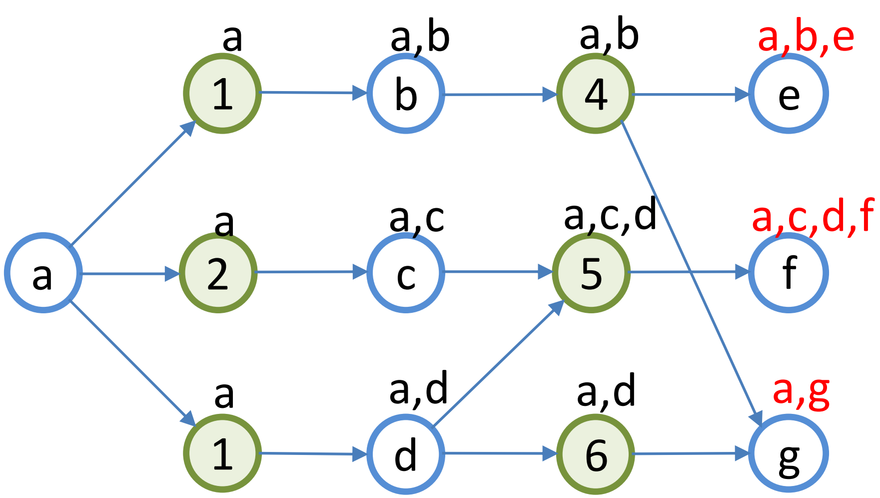


Landmark Discovery (3)



Landmark Discovery (3)

- Goal nodes in final layer: labels are landmarks
- $A \rightarrow B$ if A forms part of the label for B in the final layer
- $A \rightarrow_{gn} B$ if A is precondition for all possible first achievers of B
- Possible first achievers of B are achievers that do not have B in their label (Keyder, Richter & Helmert 2010)



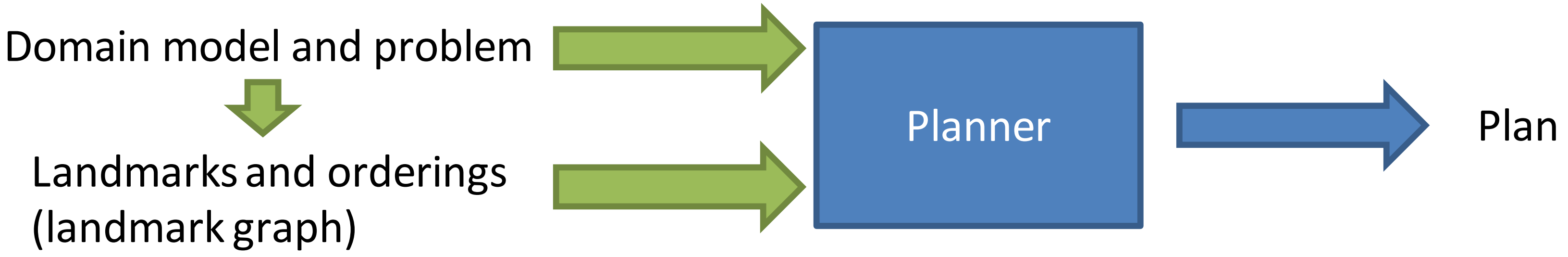
Outline

- What Landmarks Are
- How Landmarks Are Discovered
- **Using Landmarks**
  - Subgoals
  - Heuristic Estimates
  - Admissible Heuristic Estimates
  - Enriching the Problem
  - Beyond Classical Planning
- Summary

Using Landmarks

- 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



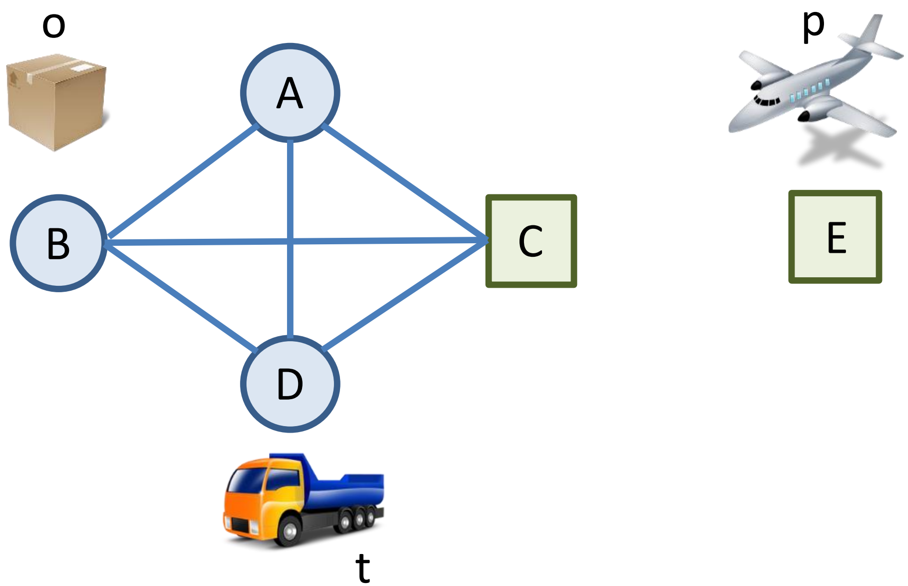
Outline

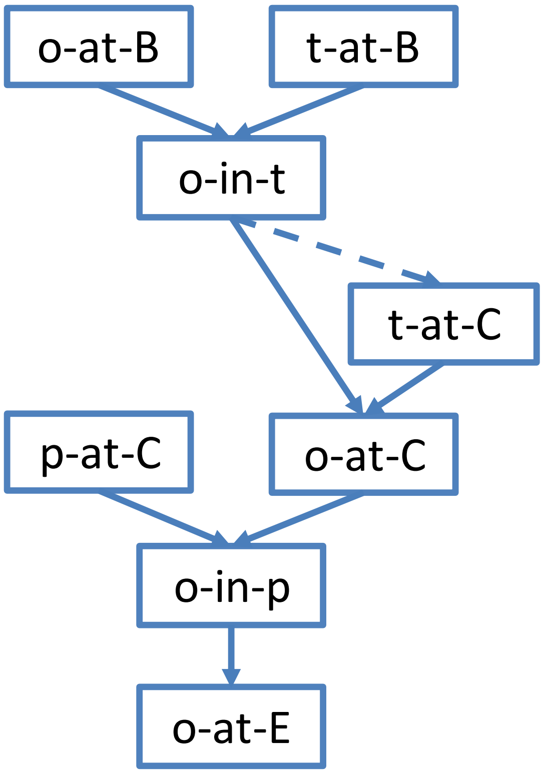
- What Landmarks Are
- How Landmarks Are Discovered
- Using Landmarks
  - **Subgoals**
  - Heuristic Estimates
  - Admissible Heuristic Estimates
  - Enriching the Problem
  - Beyond Classical Planning
- Summary

Using Landmarks: 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: Subgoals

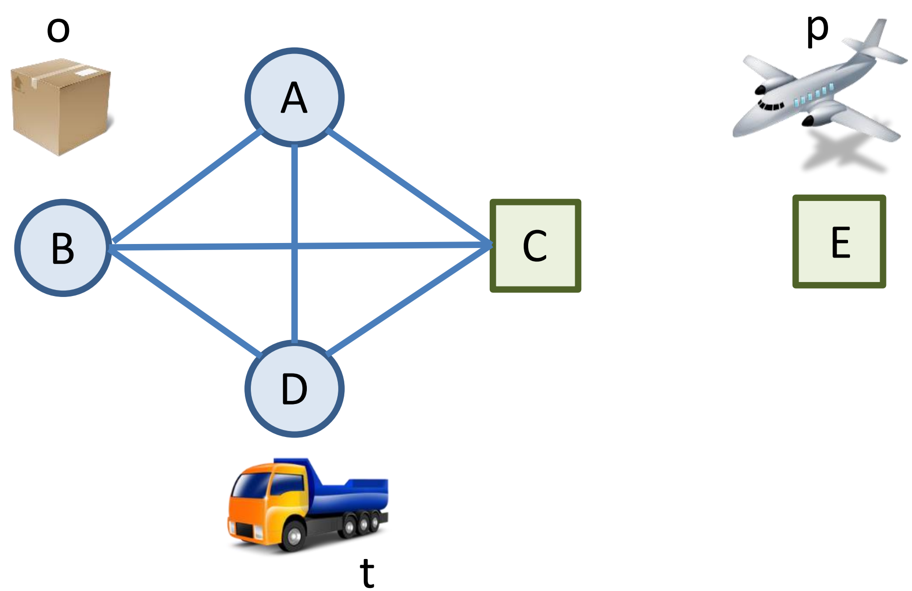


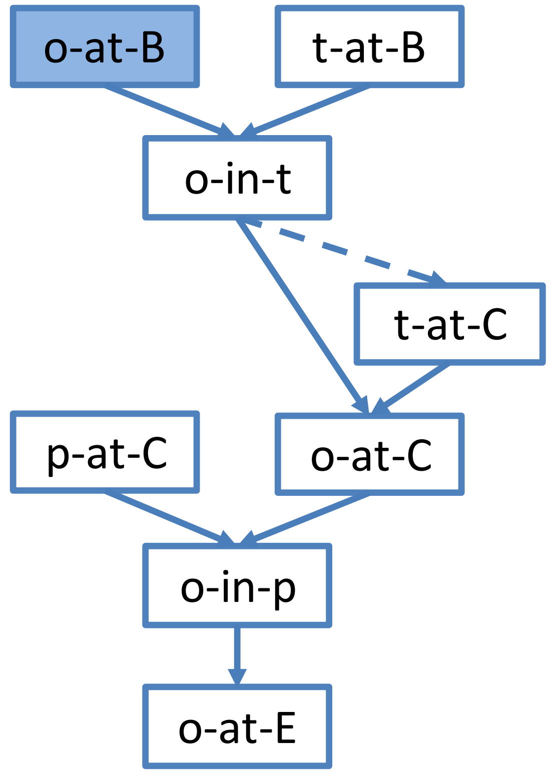


Partial Plan:

Goal:

Using Landmarks: Subgoals

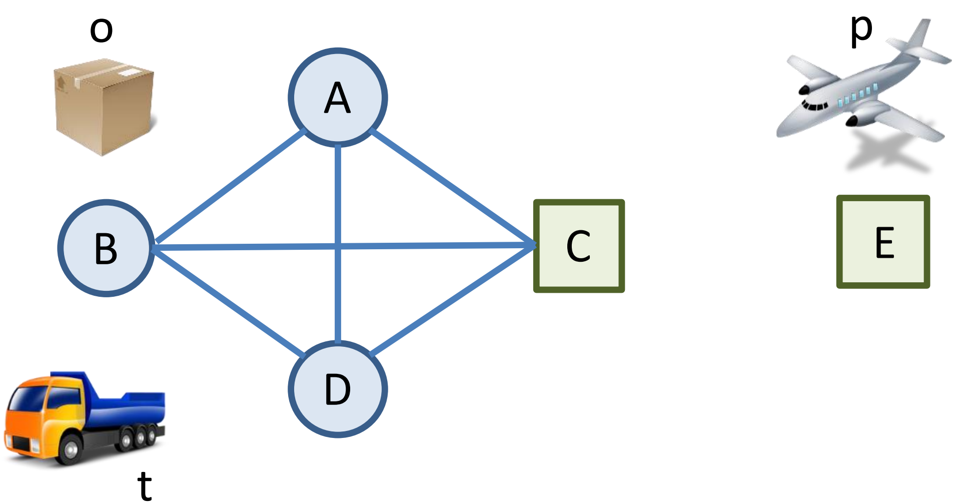


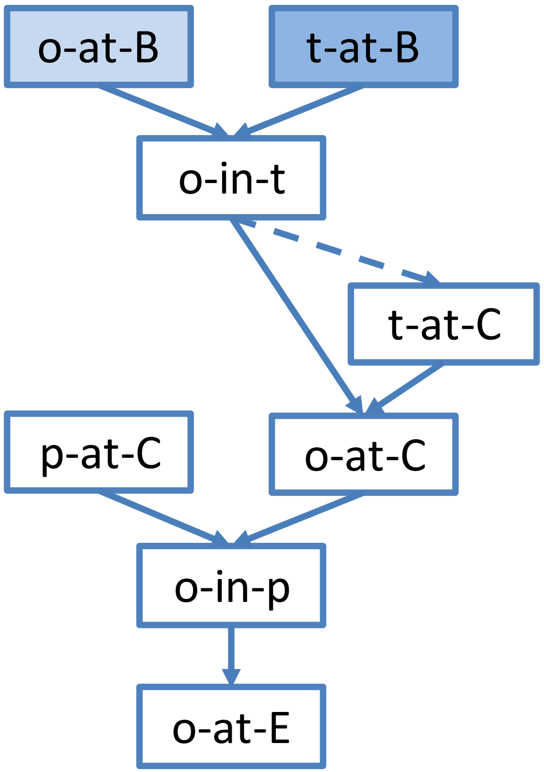


Partial Plan: Ø

Goal: p-at-C ∨ t-at-B

Using Landmarks: Subgoals

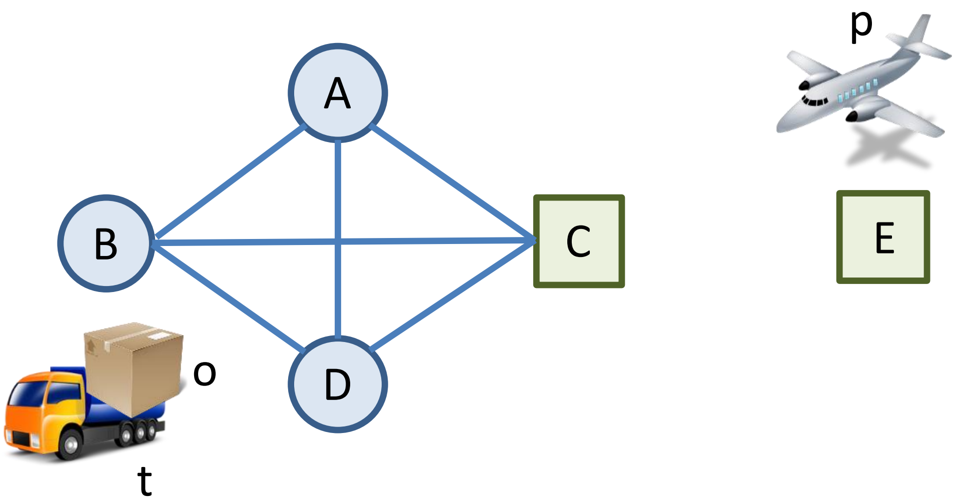


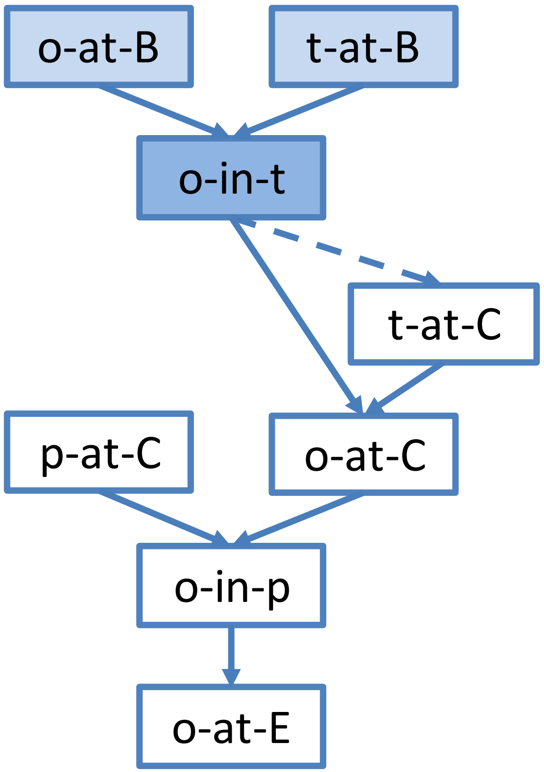


Partial Plan: drive-t-B

Goal: o-in-t V p-at-C

Using Landmarks: Subgoals

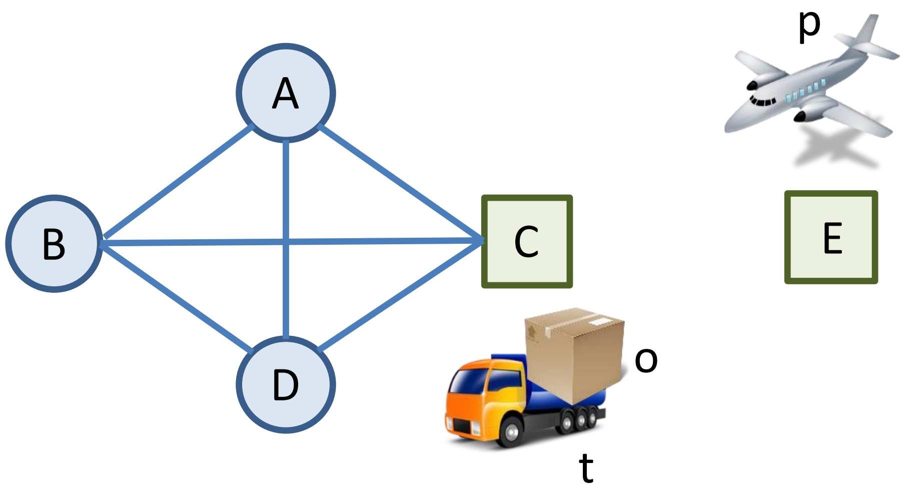


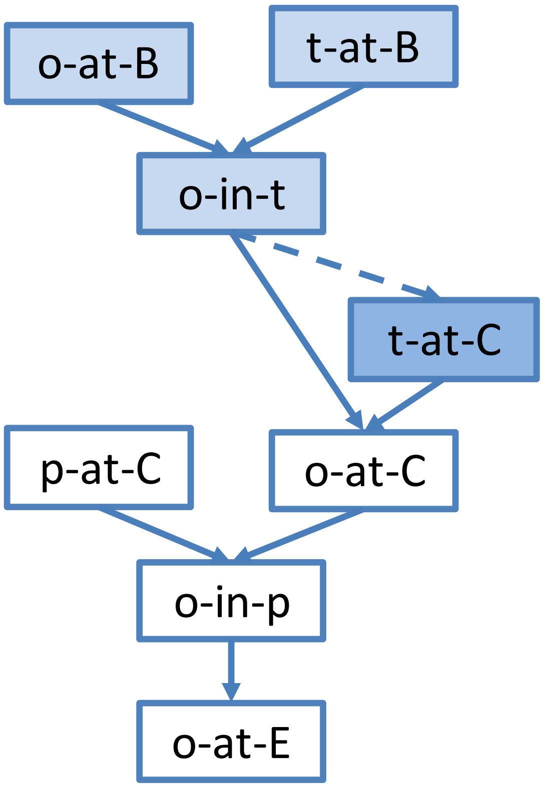


Partial Plan: drive-t-B, load-o-t

Goal: t-at-C ∨ p-at-C

Using Landmarks: Subgoals

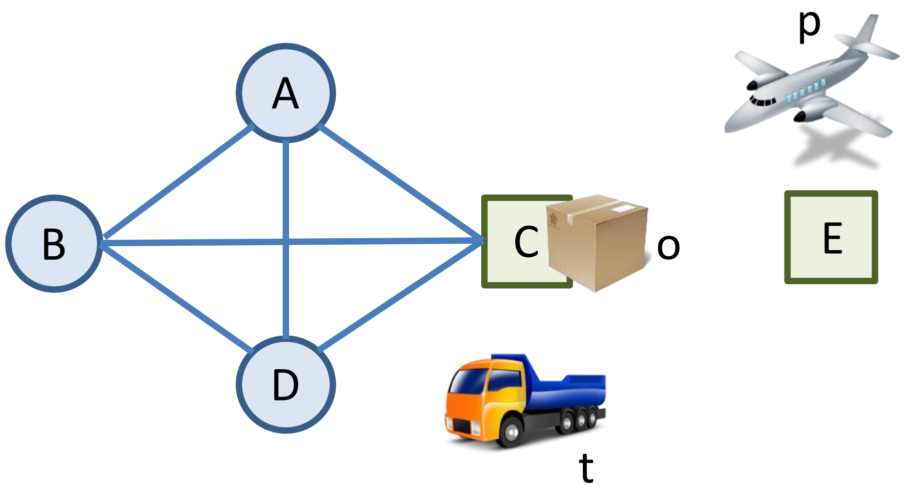


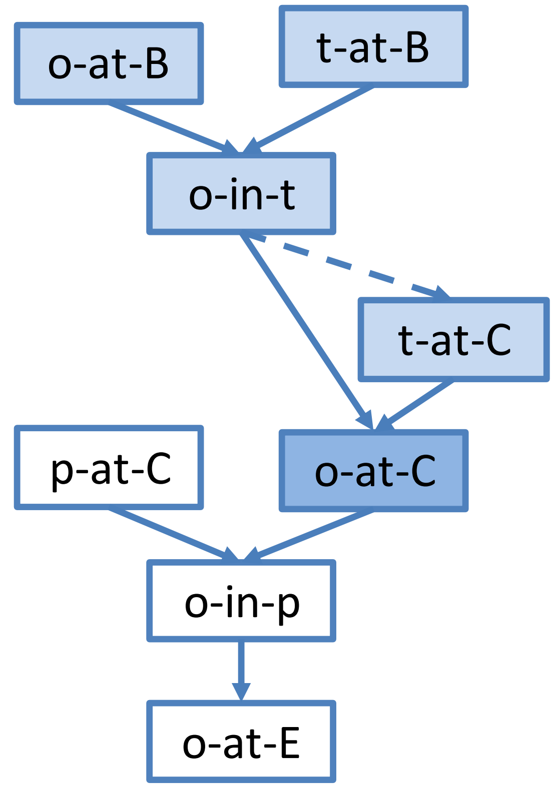


Partial Plan: drive-t-B, load-o-t, drive-t-C

Goal: o-at-C V p-at-C

Using Landmarks: Subgoals

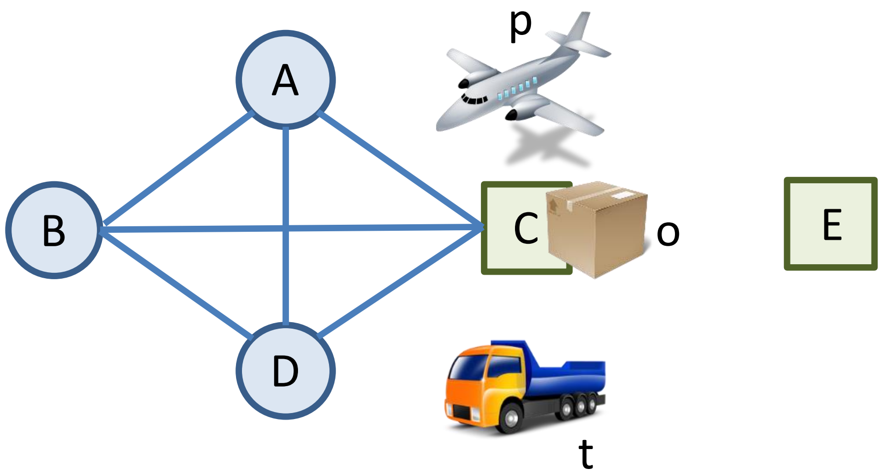


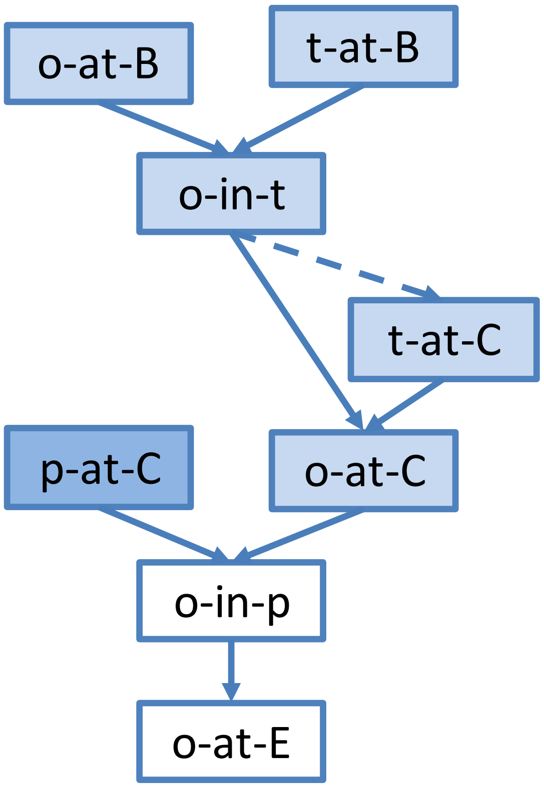


Partial Plan: drive-t-B, load-o-t, drive-t-C, unload-o-C

Goal: p-at-C

Using Landmarks: Subgoals

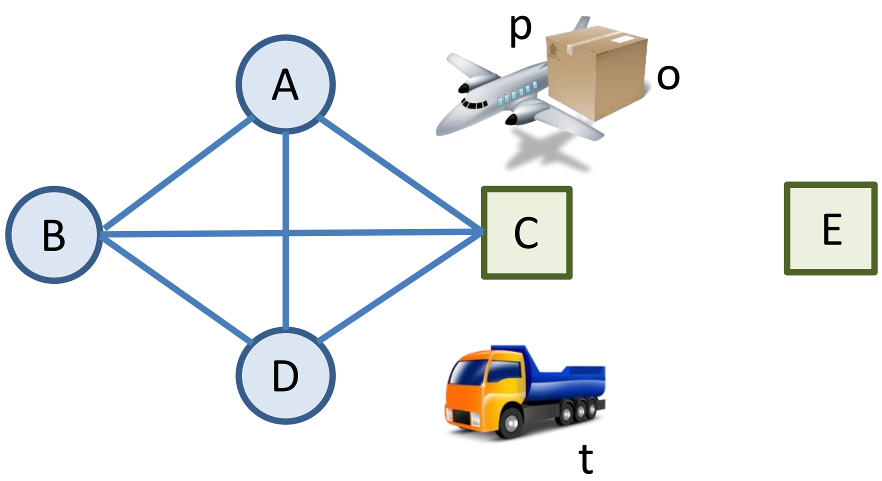


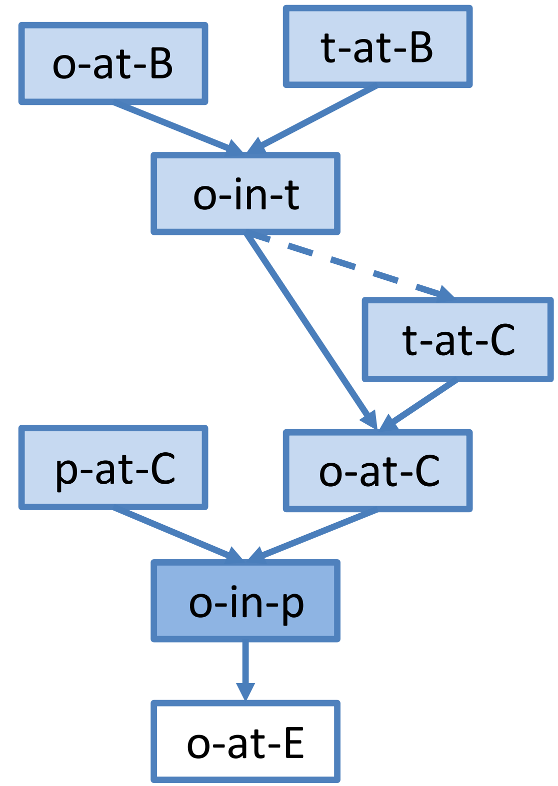


Partial Plan: drive-t-B, load-o-t, drive-t-C, unload-o-C, fly-p-C

Goal: o-in-p

Using Landmarks: Subgoals

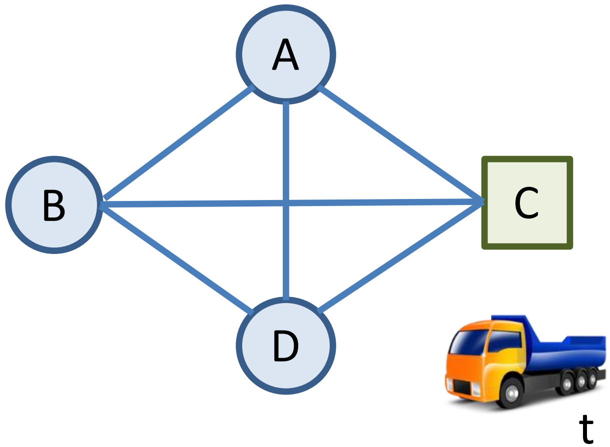


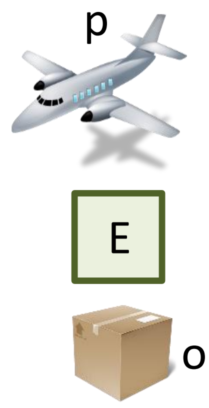


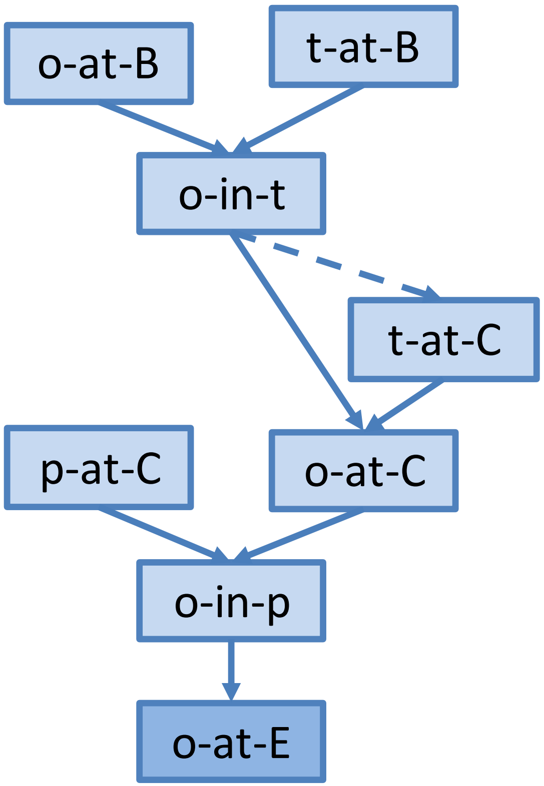
Partial Plan: drive-t-B, load-o-t, drive-t-C, unload-o-C, fly-p-C, load-o-p

Goal: o-at-E

Using Landmarks: Subgoals







Partial Plan: drive-t-B, load-o-t, drive-t-C, unload-o-C, fly-p-C, load-o-p, fly-p-E, unload-o-E

Goal: Ø

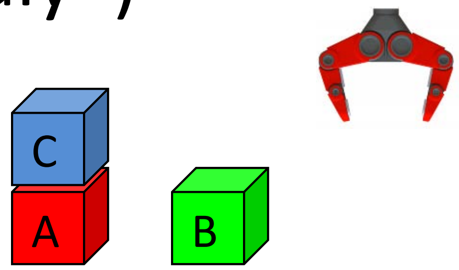
Using Landmarks: Subgoals

- That was a good example, but
- Let’s see an bad example

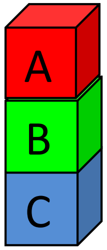
Using Landmarks: Subgoals

Consider the following Bocks World problem ("The Sussman Anomaly")

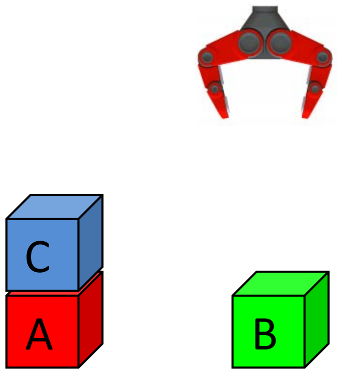
- Initial State:

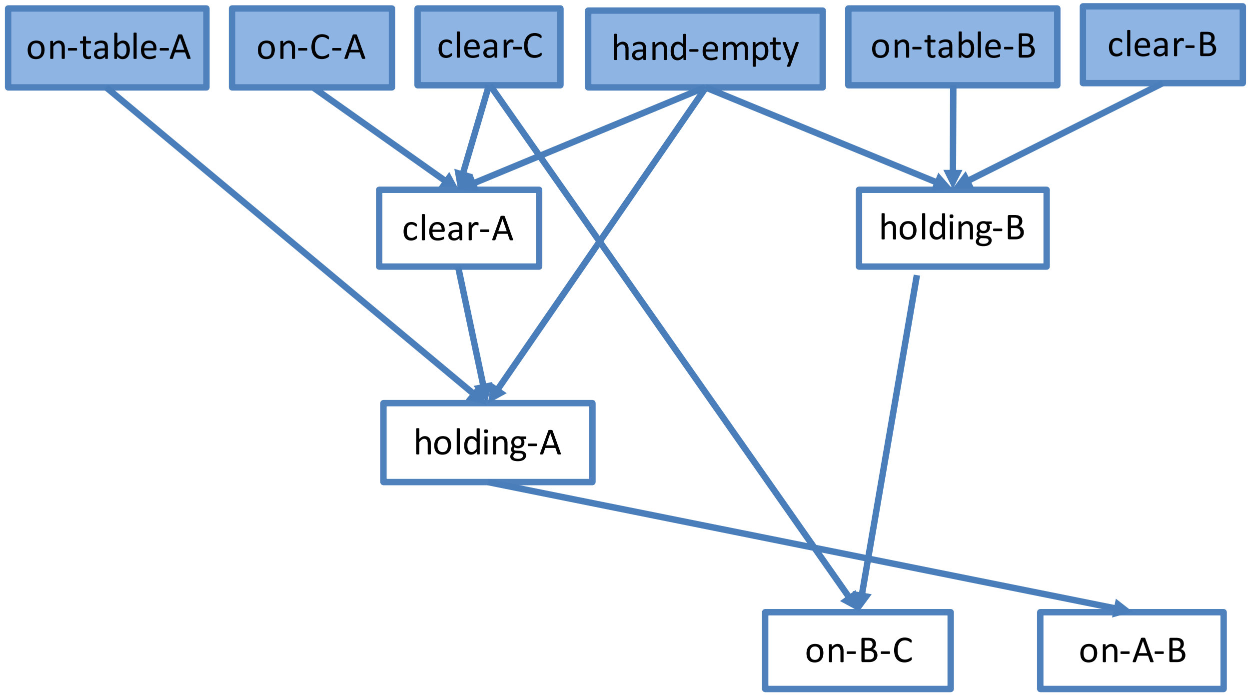


- Goal: on-A-B, on-B-C



Using Landmarks: Subgoals

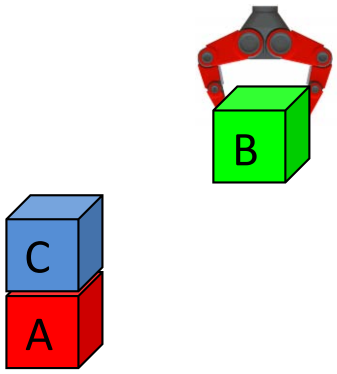


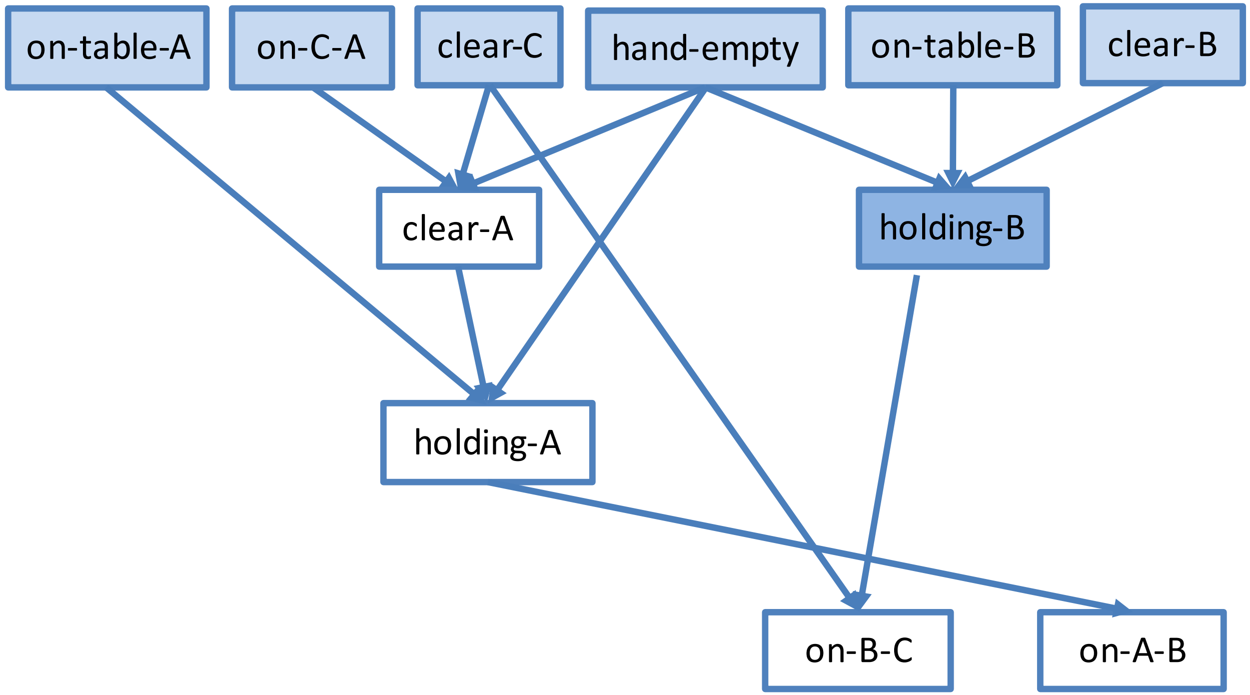


Partial Plan: Ø

Goal: clear-A V holding-B

Using Landmarks: Subgoals

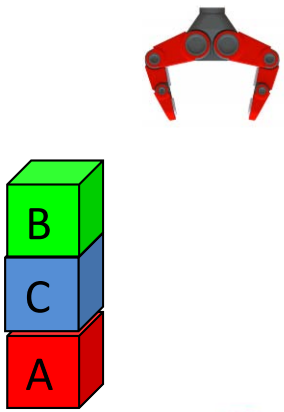


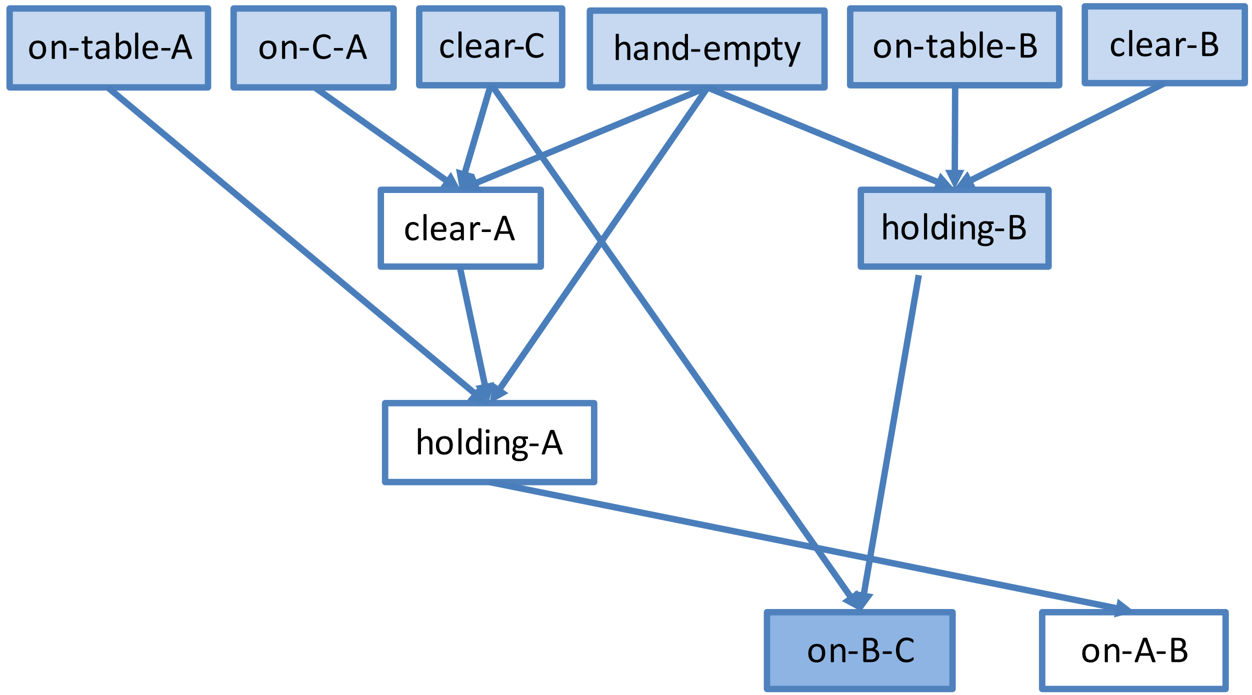


Partial Plan: pickup-B

Goal: clear-A V on-B-C

Using Landmarks: Subgoals

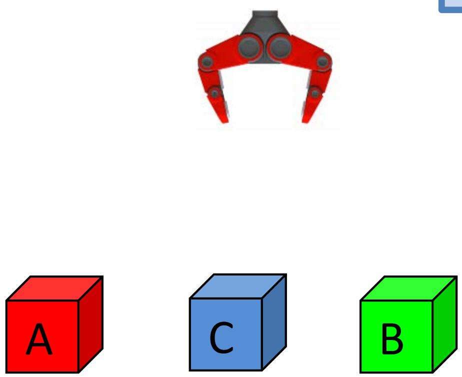


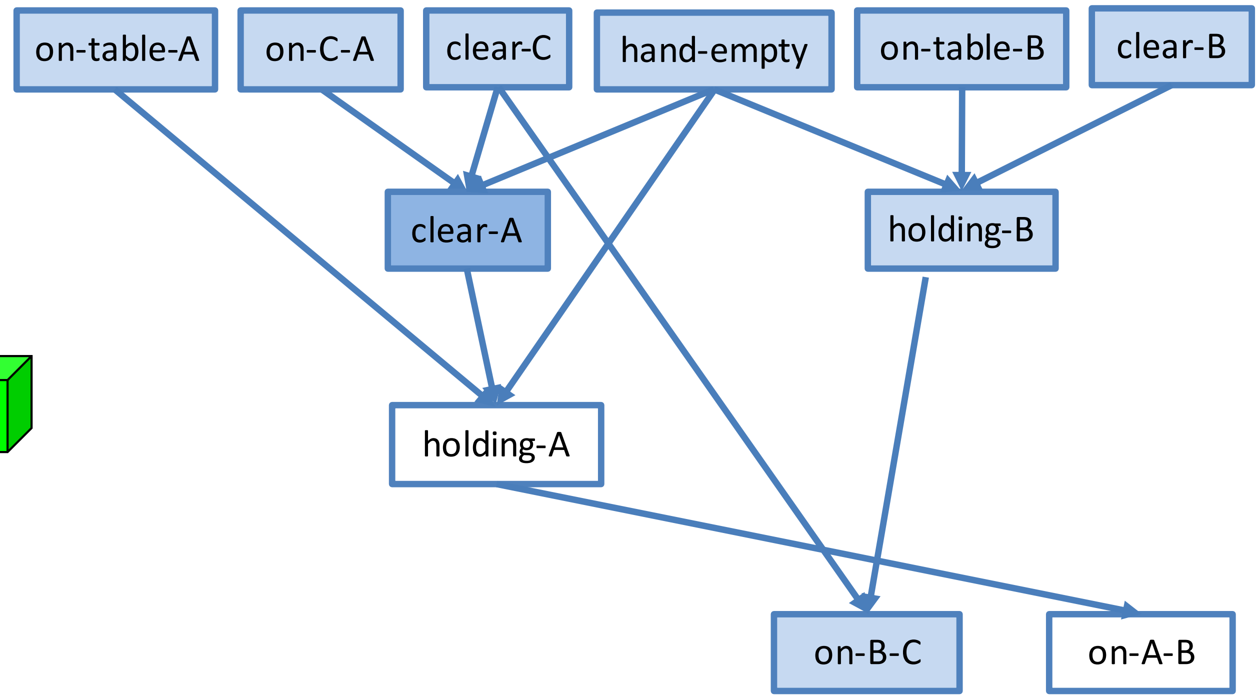


Partial Plan: pickup-B, stack-B-C

Goal: clear-A

Using Landmarks: Subgoals

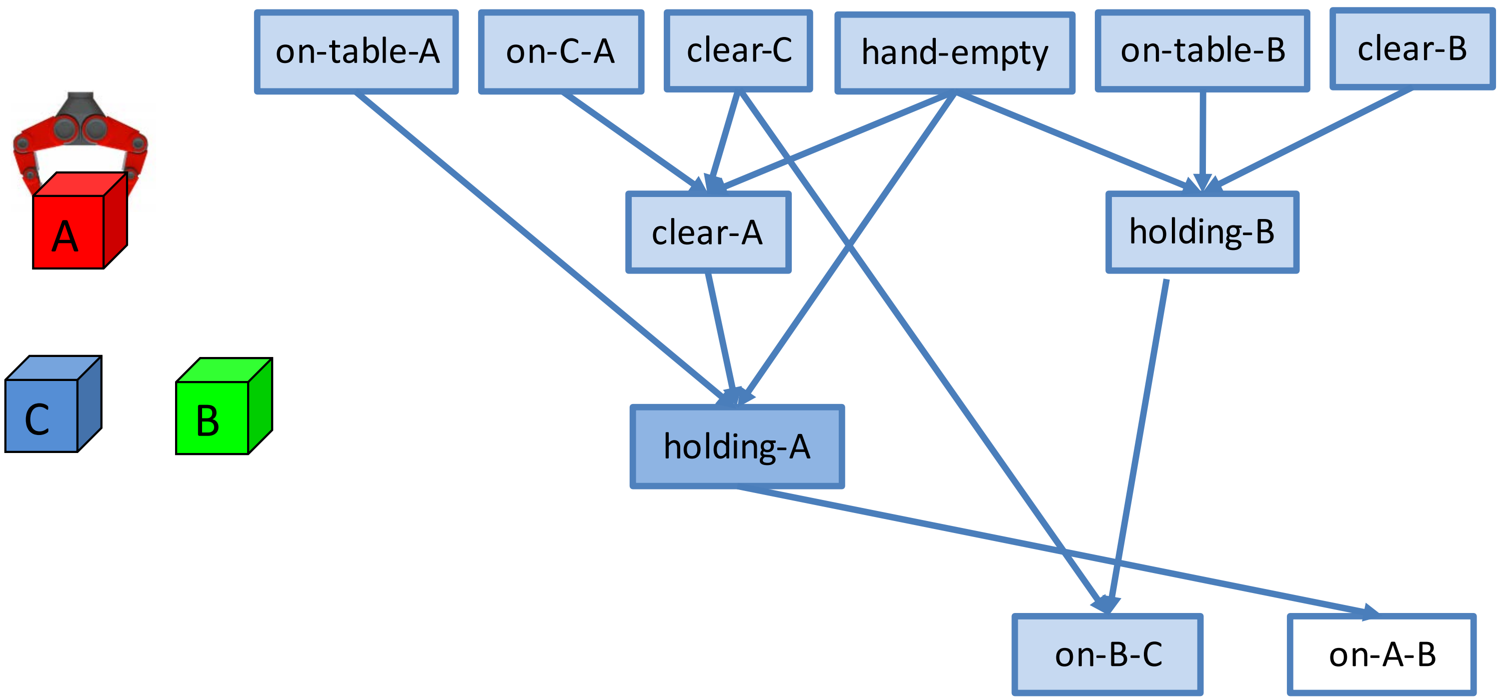




Partial Plan: pickup-B, stack-B-C, unstack-B-C, putdown-B, unstack-C-A, putdown-C

Goal: holding-A

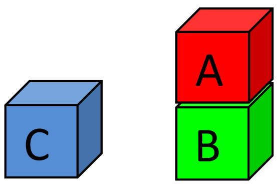
Using Landmarks: Subgoals

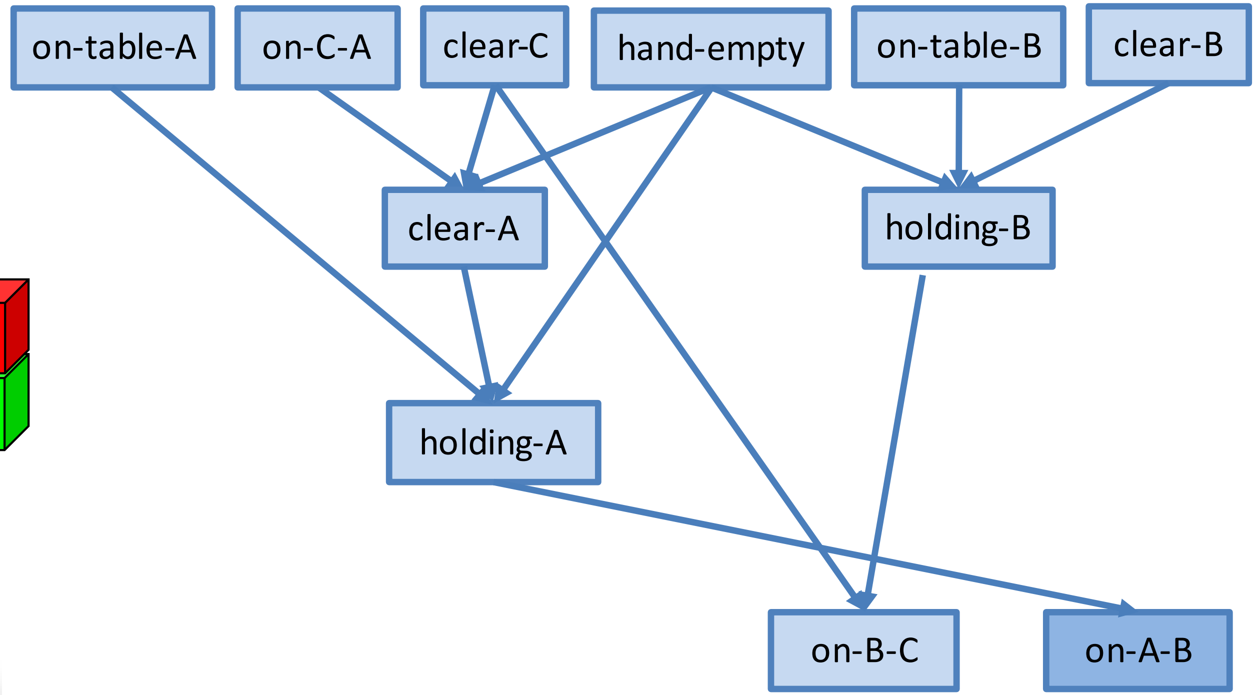


Partial Plan: pickup-B, stack-B-C, unstack-B-C, putdown-B, unstack-C-A, putdown-C, pickup-A

Goal: on-A-B

Using Landmarks: Subgoals

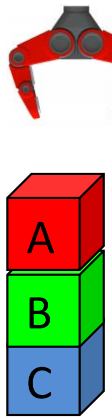


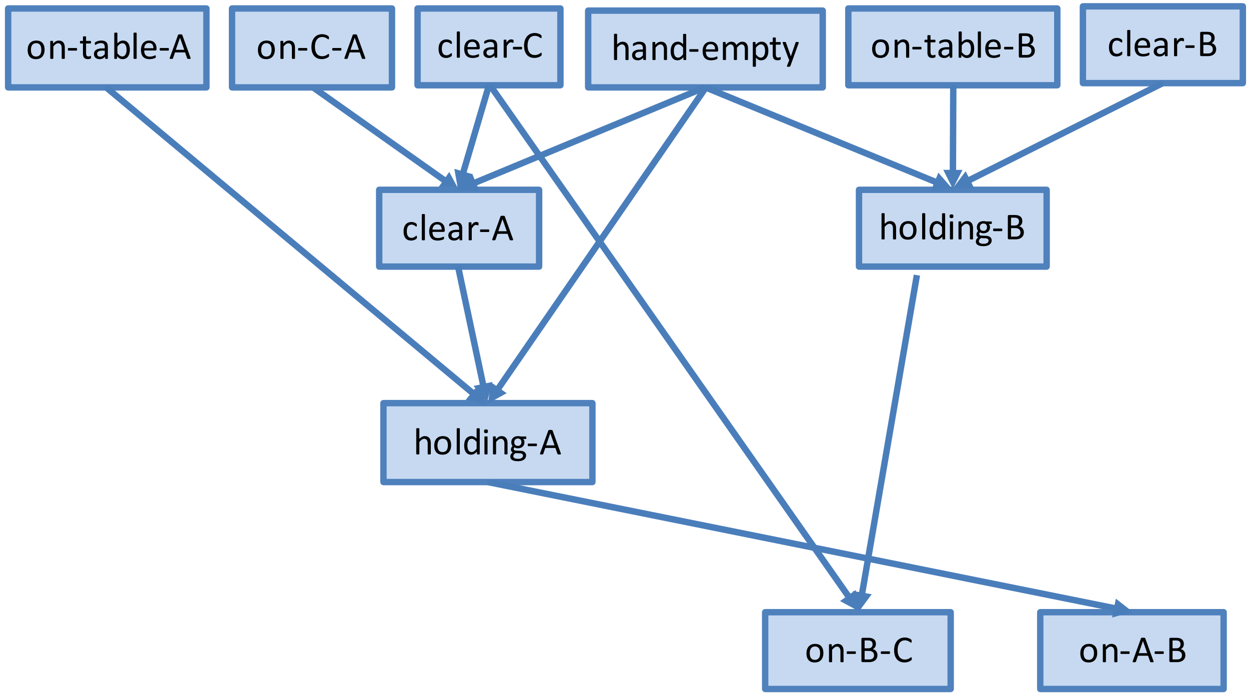


Partial Plan: pickup-B, stack-B-C, unstack-B-C, putdown-B, unstack-C-A, putdown-C, pickup-A, stack-A-B

Goal: Still need to achieve on-B-C

Using Landmarks: Subgoals





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: Ø

Using Landmarks: Subgoals

- Pro:
  - 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

Outline

- What Landmarks Are
- How Landmarks Are Discovered
- Using Landmarks
  - Subgoals
  - **Heuristic Estimates**
  - Admissible Heuristic Estimates
  - Enriching the Problem
  - Beyond Classical Planning
- Summary

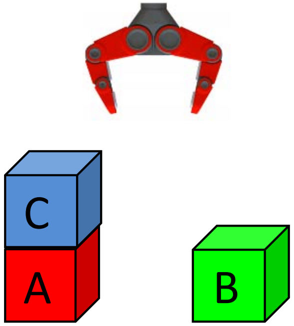
Using Landmarks: Heuristic Estimates

- The number of landmarks that still need to be achieved is a heuristic estimate (Richter, Helmert and Westphal 2008)

- Used by LAMA - winner of the IPC-2008 sequential satisficing track
  - Forward Search
  - heuristic derived from landmarks

Using Landmarks: Heuristic Estimates

- 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 state *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

LAMA Approach

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

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

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

LAMA: Accepted Landmarks

- 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*

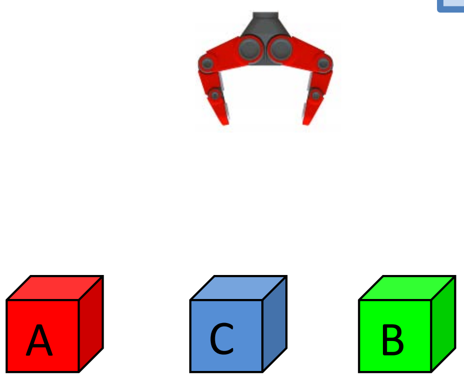
- Once a landmark has been accepted, it remains accepted

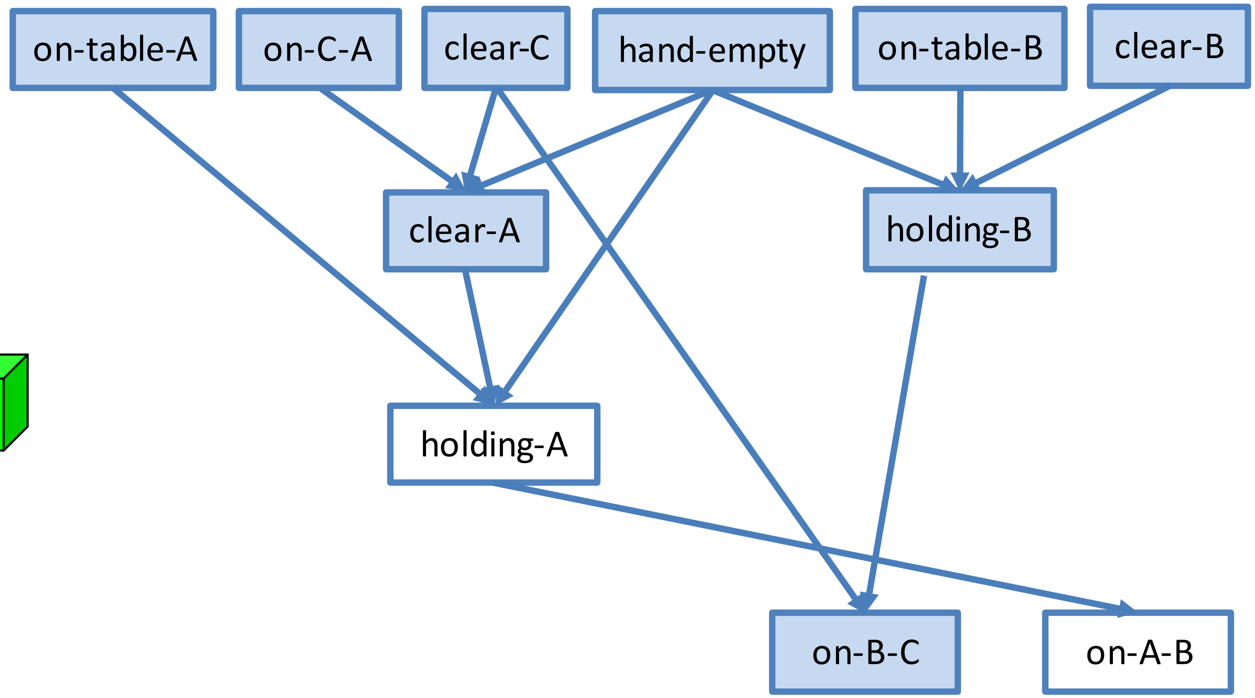
LAMA: Required Again Landmarks

- 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
- It's also possible to use (Buffet and Hoffmann, 2010):
  - **doomed-goal:** *A* is true in *s* and is a goal, but one of its greedy-necessary successors was not accepted, and is inconsistent with *A*
- Unsound rule:
  - **required-ancestor** is the transitive closure *of open-prerequisite*

LAMA: Accepted and Required Again Landmarks - Example

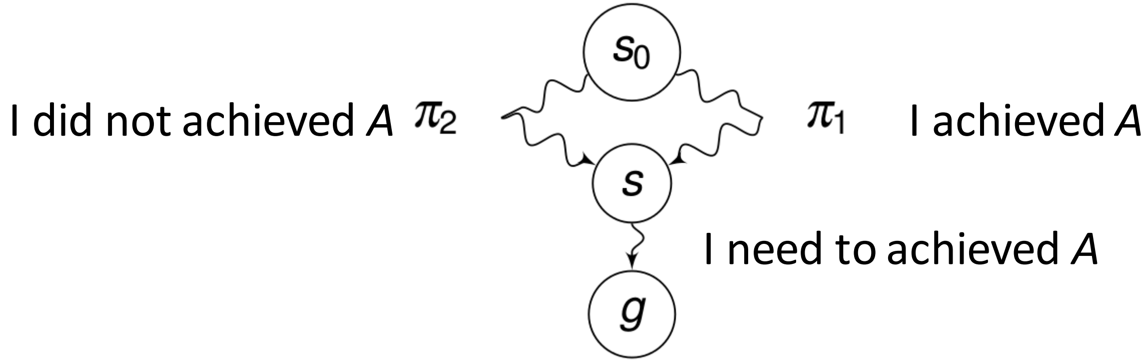
- In the Sussman anomaly, after performing: pickup-B, stack-B-C, unstack-B-C, putdown-B, unstack-C-A, putdown-C





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

Multi-path Dependency



- Suppose state s was reached by paths π_1, π_2
- Suppose π_1 achieved landmark *A* and π_2 did not
- Then *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 Different Paths

- Suppose P is a set of paths from s_0 to a state s. Define:

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

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

- $L(s, P)$ is the set of landmarks that we know still needs to be achieved after reaching state s via the paths in P
(Karpas and Domshlak, 2009)

Outline

- What Landmarks Are
- How Landmarks Are Discovered
- Using Landmarks
  - Subgoals
  - Heuristic Estimates
  - **Admissible Heuristic Estimates**
  - Enriching the Problem
  - Beyond Classical Planning
- Summary

Using Landmarks: 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 the *right* cost to each landmark, sum over the costs of landmarks (Karpas and Domshlak, 2009)

Admissible Heuristic Estimates: Conditions for Admissibility

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

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

cost(a, A): cost "assigned" by action a to A

$L(a \mid s, P)$: the set of landmarks achieved by a

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

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

cost(A) : cost assigned to landmark A

ach($A \mid s, P$) : the set of actions that can achieve A

Admissible Heuristic Estimates: 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

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

Admissible Heuristic Estimates: Admissible Cost Sharing

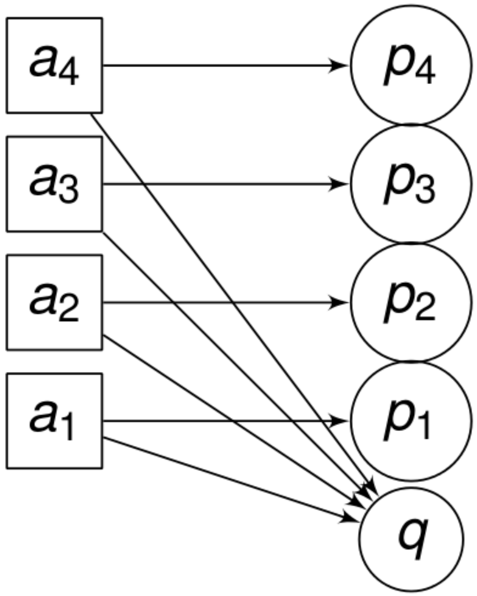
- How can we find such a partitioning?
- Easy answer - uniform cost sharing - each action shares its cost equally between the landmarks it achieves

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

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

Admissible Heuristic Estimates: Uniform Cost Sharing

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

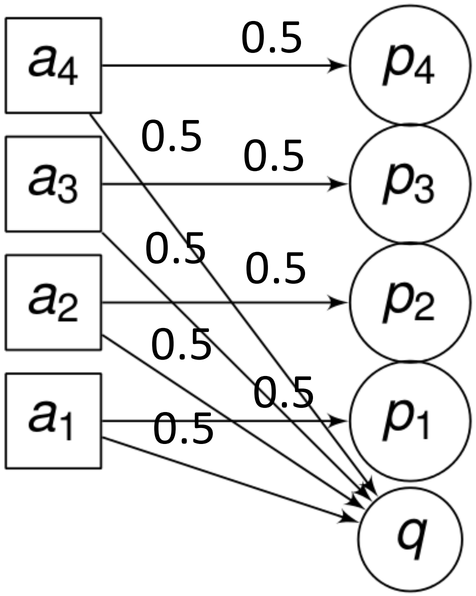


Admissible Heuristic Estimates: Uniform Cost Sharing

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

Uniform cost sharing

h_L **= 2.5**



min(0.5)=0.5

min(0.5)=0.5

min(0.5)=0.5

min(0.5)=0.5

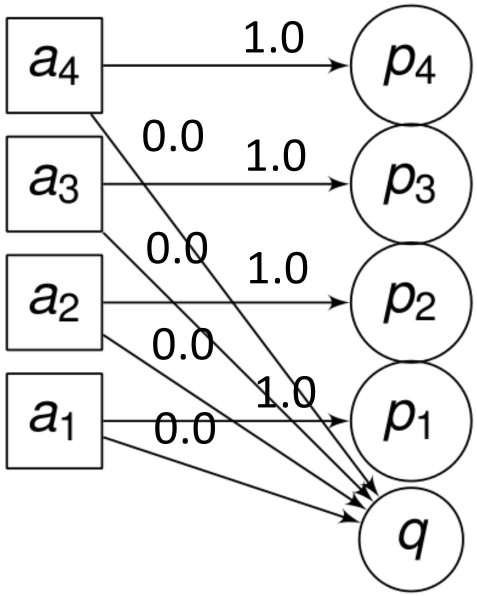
min(0.5,0.5,0.5,0.5)=0.5

Admissible Heuristic Estimates: Uniform Cost Sharing

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

Uniform cost sharing

$h_L = 4.0$ $h_L = 2.5$



min(1.0)=1.0

min(1.0)=1.0

min(1.0)=1.0

min(1.0)=1.0

min(0,0,0,0)=0

Admissible Heuristic Estimates: 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

Admissible Heuristic Estimates: How we can get better?

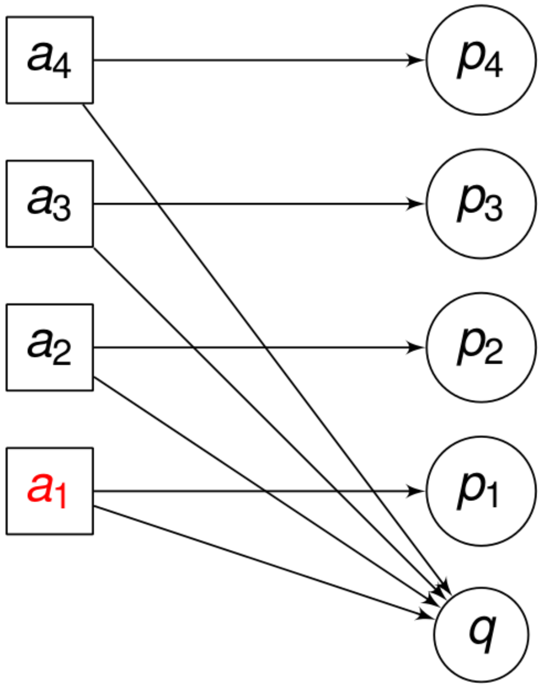
- So far:
  - Uniform cost sharing is easy to compute, but suboptimal
  - Optimal cost sharing takes a long time to compute

- Q: How can we get better heuristic estimates that don't take a long time to compute?

  A: Exploit additional information - action landmarks

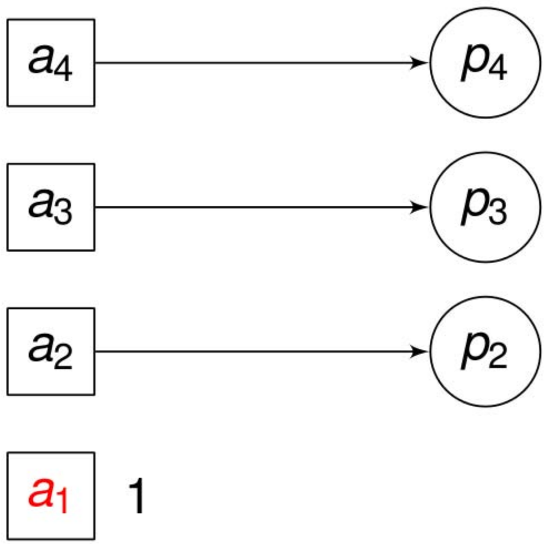
Admissible Heuristic Estimates: How we can get better?

Using action landmarks … for example:



Admissible Heuristic Estimates: How we can get better?

Using action landmarks … for example:

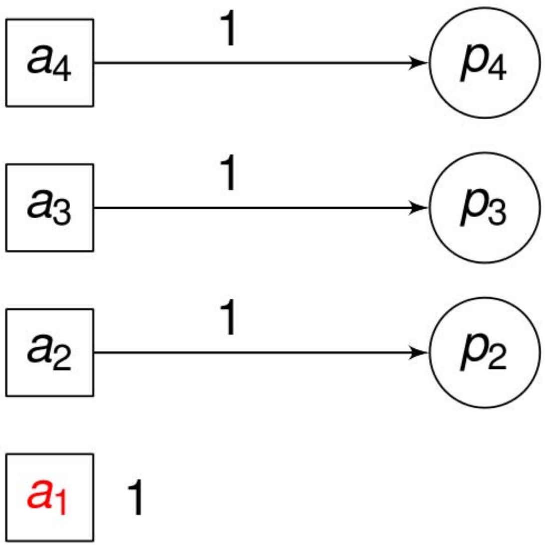


Admissible Heuristic Estimates: How we can get better?

Using action landmarks … for example:

- Uniform Cost Sharing

h_{LA} **= 4.0**



min(1.0)=1.0

min(1.0)=1.0

min(1.0)=1.0

Outline

- What Landmarks Are
- How Landmarks Are Discovered
- Using Landmarks
  - Subgoals
  - Heuristic Estimates
  - Admissible Heuristic Estimates
  - **Enriching the Problem**
  - Beyond Classical Planning
- Summary

Using Landmarks: Enriching the Problem

- Landmarks are, in essence, implicit goals
- We can make these explicit by reformulating the planning problem

- Two different methods for doing this have been proposed (Wang, Baier and McIlraith, 2009 and Domshlak, Katz and Lefler, 2010)

Using Landmarks: Enriching the Problem

Viewing Landmarks as Temporally Extended Goals:

- Landmarks and their orderings can be viewed as temporally extended goals (Wang, Baier and McIlraith, 2009)
- These temporally extended goals can be expressed in Linear Temporal Logic (LTL)
- Each LTL formula can be compiled into a finite-state automaton (FSA)
- Each FSA can be encoded as a single variable in an enriched planning problem

Using Landmarks: Enriching the Problem

A Simpler Approach:

A simpler approach of encoding landmarks into a planning problem is to encode the landmarks directly (Domshlak, Katz and Lefler, 2010)

- Each landmark is represented by a single binary state variable
- The two values represent landmark accepted / not accepted
- Each operator that achieves the landmark has an additional effect added to it, changing the landmark variable value to accepted
- The accepting value of each landmark variable is added to the goal state

Using Landmarks: Enriching the Problem

Why Enrich Problems?

- Landmarks and orderings are implicit, encoding them into the problem makes them explicit
- Allows other heuristics to use landmark information
- Example: structural pattern heuristic on the enriched problem accounts not only for explicit goals (Domshlak, Katz and Lefler, 2010)
- In fact, the landmark count heuristic can be seen as the goal count heuristic on the landmark enriched problem
- Caveat - since current landmark discovery procedures are based on delete-relaxation, this adds no information to delete-relaxation based heuristics

Outline

- What Landmarks Are
- How Landmarks Are Discovered
- Using Landmarks
  - Subgoals
  - Heuristic Estimates
  - Admissible Heuristic Estimates
  - Enriching the Problem
  - **Beyond Classical Planning**
- Summary

Using Landmarks: Beyond Classical Planning

- Probabilistic landmarks
  - a landmark is a fact which must be true in every successful trajectory (possible execution)
- Temporal Landmarks

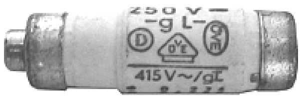
Temporal Landmarks

Landmarks for Temporal Planning

- We can treat a durative action as two "snap" actions: the *start* and the *end (Fox & Long, 2003)*
- This way, we can create a classical planning problem which is a **relaxation** of the temporal planning problem (Haslum 2009)
- The landmarks of this relaxation are called **causal landmarks**

Example Temporal Planning Problem: Flashlight Match Cellar

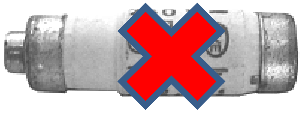
- Goal: to fix the fuse in the cellar



- Possible actions:
  - **Fix-fuse** – takes 10 seconds, requires light throughout
  - **Light-match** – requires a match, provides light for 15 seconds, consumes the match

  There is a flashlight in the dark cellar:

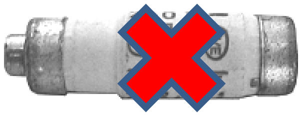
  - **find-flashlight** – needs light throughout, takes 2 seconds
  - **turn-on-flashlight** – takes 1 second (after it's found), and produces light



- Initial state: have a match

Durative Action: Fix fuse

Duration: 10 seconds



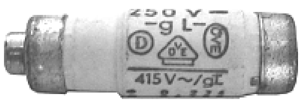
<u>Start</u>:

    Condition:

    Effect:

<u>Invariant condition</u>: *light*

<u>End</u>:



    Condition:

    Effect: *fuse-fixed*

Durative Action: Light match

Duration: 15 seconds

Start:

Condition: *have-match*

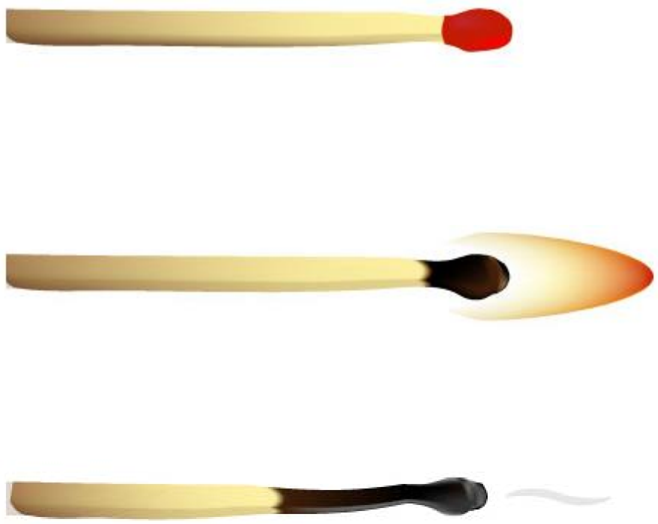
Effect: not(*have-match*), *light*

Invariant condition:

End:

Condition:

Effect: not(*light*)



Durative Action: Find Flashlight

Duration: 2 seconds

Start:

    Condition:

    Effect:

Invariant condition: *light*

End:

    Condition:

    Effect: *have-flashlight*

Durative Action: Turn On Flashlight

Duration: 1 seconds

<u>**Start**</u>:

Condition: have-flashlight

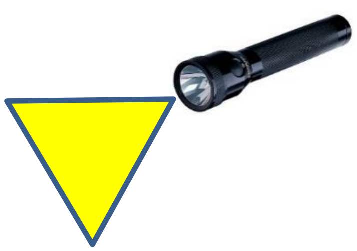
Effect:

<u>**Invariant condition**</u>:

<u>**End**</u>:

Condition:

Effect: *light*



Causal Landmarks for Flashlight Match Cellar

If we run a casual landmark discovery, we would get:

- fuse-fixed
- light
- have-match

Possible Solution: Flashlight Match Cellar

What if we change the duration of light-match to 5?

The causal landmarks do not change

Temporal Landmarks

- We define **temporal landmarks** which incorporate statements about *what* must happen with temporal constraints about *when*

- Temporal landmarks allow us to discover more about the task than causal landmarks

Temporal Landmark Definitions

- Two types of temporal landmarks:
  - Fact landmark $holds_{s:e}(F)$
    - fact F must hold from ***exactly*** time point s until ***at least*** time point e
  - Action landmark $\text{occurs}_o(e)$
    - event (start/end of action) e must occur at time point o
- Maintain a set of simple temporal constraints between time points

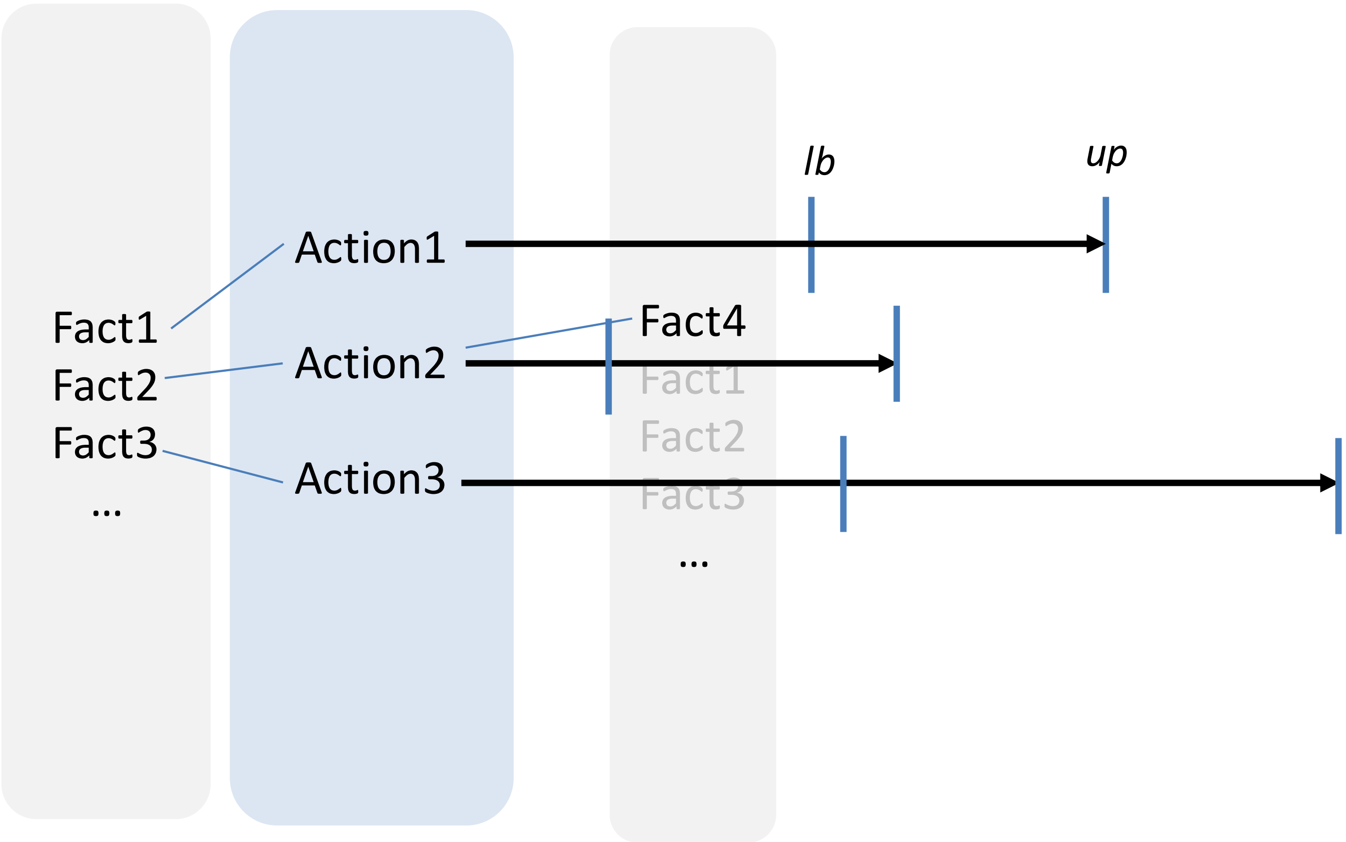
  $\text{lb} \leq t_2 - t_1 \leq \text{ub}$

Discovering Temporal Landmarks

- Similar to backchaining for classical landmarks
- Start with what must happen
  - The goal must be achieved
- Draw the logical conclusions from what we know must happen
  - We use a set of derivation rules for this

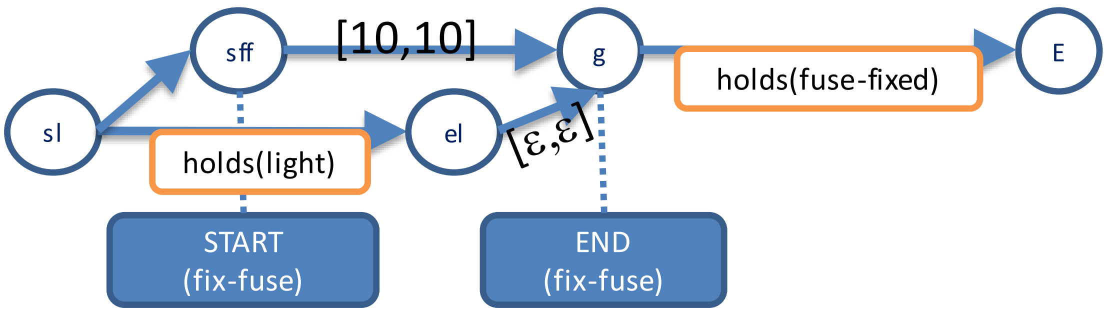
Temporal Relaxed Planning Graph

.



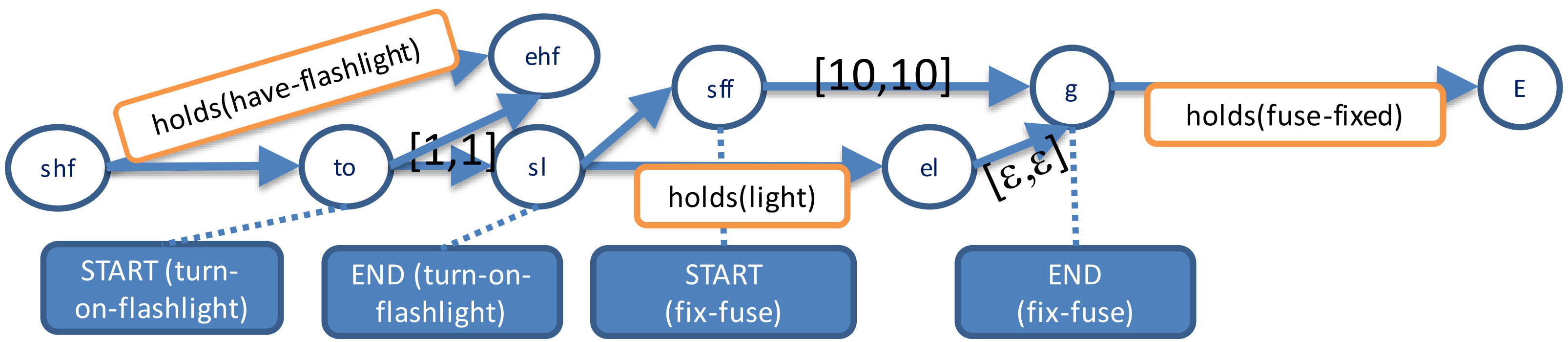
Temporal Landmarks for Flashlight Match Cellar

- The goal must hold from some time point *g* until the end *E*
  - $holds_{g:E}$(fuse-fixed)
- The only event which can achieve fuse-fixed is END(fix-fuse), which must occur exactly at *g*
  - $occurs_g$(END(fix-fuse))
- Every action that ends must start
  - $occurs_{sff}$(START(fix-fuse)), with $g - sff = 10$
- The invariant condition of fix-fuse must hold from *sl* to *el*
  - $holds_{sl:g}(light)$, with $sl \le sff$ and $el = g - \varepsilon$



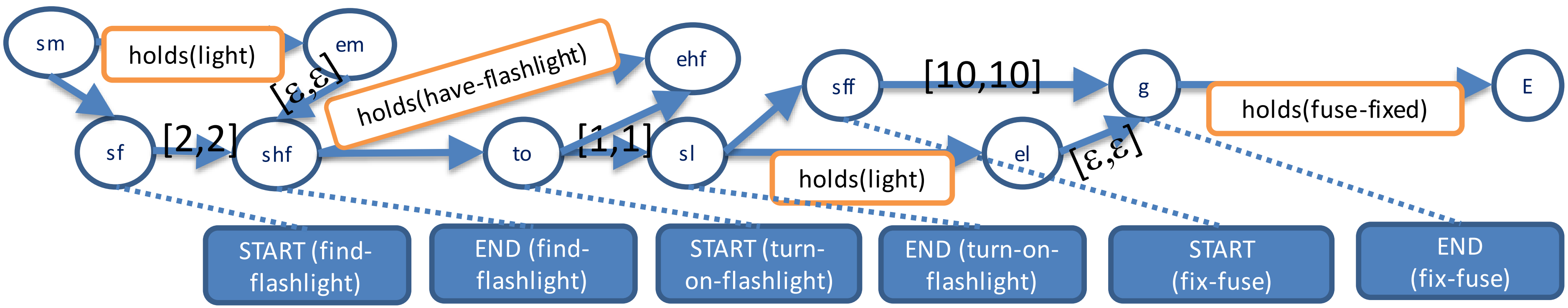
Temporal Landmarks for Flashlight Match Cellar

- Light must be achieved. This can be done by either END(turn-on-flashlight) or START(light-match). But we need light for 10-ε and light-match will only give us light for 5, so:
  - occurs_{sl}(END(turn-on-flashlight))
- As before, the start must precede the end
  - occurs_{to}(START(turn-on-flashlight)), with $sl - to = 1$
- To turn on the flashlight, we must have it, so
  - $\text{holds}_{shf:ehf}$(have-flashlight), with $shf \leq to$ and $to \leq ehf$



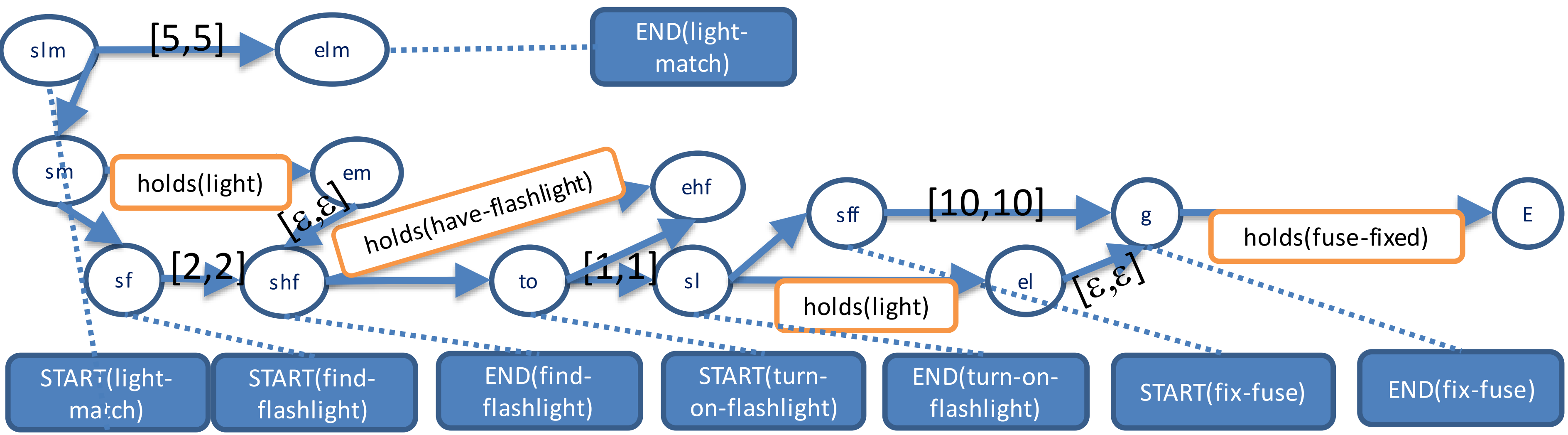
Temporal Landmarks for Flashlight Match Cellar

- The only possible achiever of have-flashlight is END(find-flashlight), so
  - $occurs_{shf}$(END(find-flashlight)
- The find-flashlight action must start
  - $occurs_{sf}$(START(find-flashlight), with $shf - sf = 2$
- Its invariant must hold, so
  - $holds_{sm:em}$(light), with $sm \leq sf$ and $em = shf - \varepsilon$



Temporal Landmarks for Flashlight Match Cellar

- We can now check what are the possible ***first time*** achievers of light, which is only START(light-match), so
  - occurs$_{slm}$(START(light-match)), with $slm \leq sm$
- Finally, the action must end, so
  - occurs$_{elm}$(END(light-match)), with $elm - slm = 5$



Temporal Landmarks Derivation Rules

- If fact *F* must hold for a duration of *d*, then:
  - *F* must be achieved. Furthermore, this must be by an action which does not delete *F* at the end, if its duration is shorter than d.
  - *F* must be achieved for the first time
- Every action must have a start and an end. Its invariant condition must hold between them.
- Every event must have its conditions hold when it happens
- Every event causes its effects to hold when it happens

Using Temporal Landmarks

- We have some temporal landmarks, now what?
  - Plan “skeleton”
  - Use underlying STN as heuristic to estimate makespan
  - Enriching the Problem: “Compile” landmarks into the problem

Some Results

- Compilation approach
  - In the compilation, we limit the size of a disjunction to 1, 4, or ∞
- Comparing performance of planners with and without temporal landmarks on benchmarks from IPC 2011 and 2014
- The planners
  - POPF (IPC-2011)
  - Temporal Fast Downward (IPC-2014)
  - YAHSP3-MT (IPC-2014)

Results on Temporally Expressive Domains

POPF	orig	e1	e4	e∞
matchcellar (2011)	20	20	20	20
matchcellar (2014)	20	20	20	20
tms (2011)	5	**11**	4	4
tms (2014)	0	**6**	0	0
turnandopen (2011)	**9**	8	9	8
turnandopen (2014)	0	0	0	0
TOTAL	54	**65**	53	52

TFD	orig	e1	e4	e∞
matchcellar (2011)	**20**	**20**	18	0
matchcellar (2014)	**20**	**20**	**20**	0
tms (2011)	0	**2**	0	0
tms (2014)	0	0	0	0
turnandopen (2011)	**19**	**19**	0	0
turnandopen (2014)	**7**	1	0	0
TOTAL	**66**	62	38	0

Number of solved problems

(Interesting) Results on Non-Temporally Expressive Domains

POPF	orig	e1	e4	e∞
crew (2011)	**20**	**20**	16	16
elevators (2011)	**3**	0	1	0
floortile (2011)	1	0	**2**	**2**
parcprinter (2011)	0	0	1	**5**
parking (2011)	**20**	19	19	18
parking (2014)	12	12	12	**17**
pegsol (2011)	**19**	**19**	10	3
satellite (2014)	**4**	**4**	2	2
sokoban (2011)	**3**	**3**	2	0
TOTAL	**102**	97	85	83

TFD	orig	e1	e4	e∞
crew (2011)	**20**	**20**	6	6
elevators (2011)	**20**	19	5	6
floortile (2011)	**5**	**5**	0	0
mapanalyser (2014)	**17**	**17**	0	0
openstacks (2011)	**20**	**20**	**20**	0
parcprinter (2011)	**10**	0	0	0
parking (2011)	**20**	10	10	0
parking (2014)	**20**	**20**	19	0
pegsol (2011)	**19**	**19**	0	0
satellite (2014)	**17**	8	1	0
sokoban (2011)	**5**	1	0	0
TOTAL	**173**	139	61	12

YAHSP3-MT	orig	e1	e4	e ∞
driverlog (2014)	**3**	**3**	0	2
elevators (2011)	**20**	10	9	8
floortile (2011)	**11**	10	2	2
floortile (2014)	**6**	5	1	0
parcprinter (2011)	1	3	**5**	3
parking (2011)	**20**	**20**	18	15
pegsol (2011)	**20**	**20**	17	13
sokoban (2011)	**10**	5	6	1
storage (2011)	7	**8**	7	0
storage (2014)	**9**	**9**	4	0
TOTAL	**188**	174	150	125

Summary

- Landmarks provide a way to utilize the implicit structure of a planning problem
- Landmarks work well in
  - Classical planning
  - Partially observable planning with sensing (Maliah et al, 2014)
  - Oversubscription Planning (Mirkis & Domshlak, 2014)
  - Temporal planning
    - At least, when the problems are temporally expressive

…