Lecture 8

CS 6260 Automated Planning and Learning

Heuristics for Planning 2

Department of Computer Science and Engineering

Indian Institute of Technology Madras

Some material based on lecture slides for ASU's CSE571 Artificial Intelligence by Siddharth Srivastava.

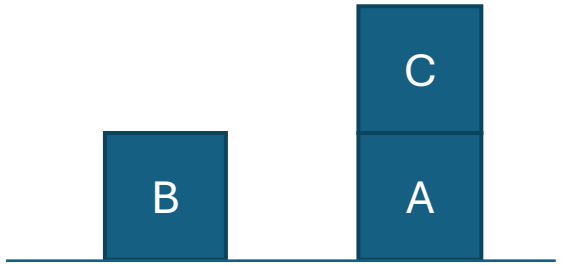
Recap: Heuristics leveraging PDDL

Ideas for Generating Heuristics and Search Control

- Goal decomposition (not quite)
- Planning graph
- Landmarks

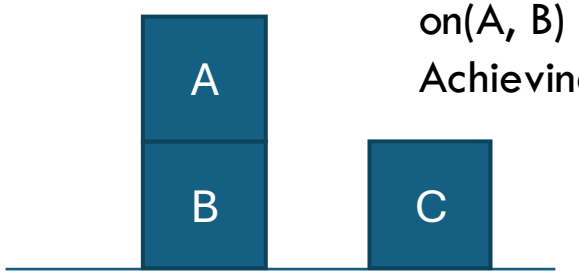
Goal Decomposition: Sussman's Anomaly

Goal requires: on(A, B); on(B,C)
What happens if we try to achieve them independently?

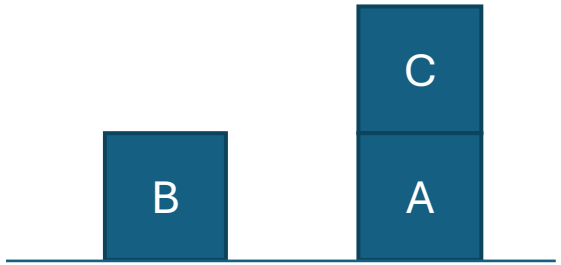


Try to achieve: on(A, B)

Move(C, table); Move(A, B)

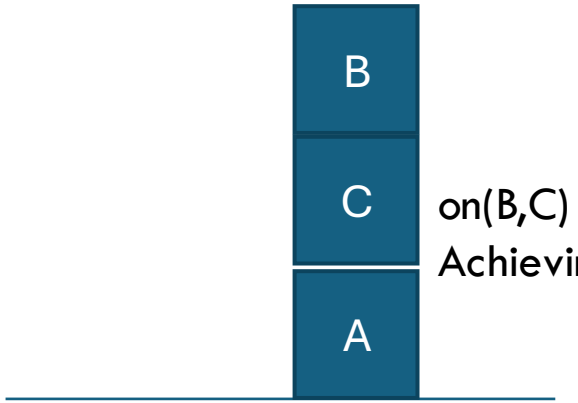


on(A, B) achieved ☺
Achieving on(B,C) will undo it ☹



Try to achieve: on(B, C)

Move(B, C)



on(B,C) achieved ☺
Achieving on(A,B) will undo it ☹

Planning Graph

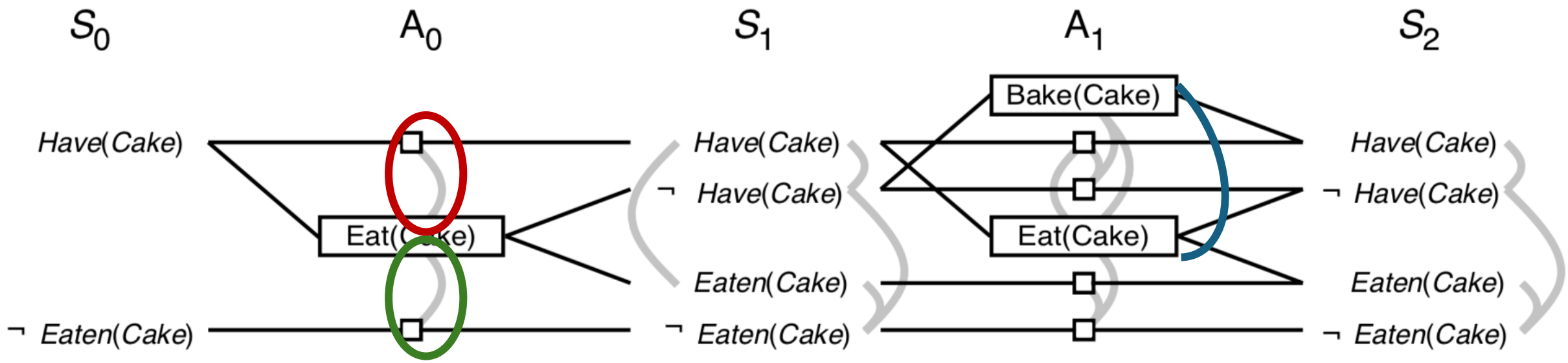
$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
PRECOND: $Have(Cake)$
EFFECT: $\neg\, Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
PRECOND: $\neg\, Have(Cake)$
EFFECT: $Have(Cake)$

Actions in a layer are *mutex* if:

Inconsistent Effects: one action's effect negates another's effect

Interference: one action negates another's precondition

Competing needs: one action requires negation of another's precondition



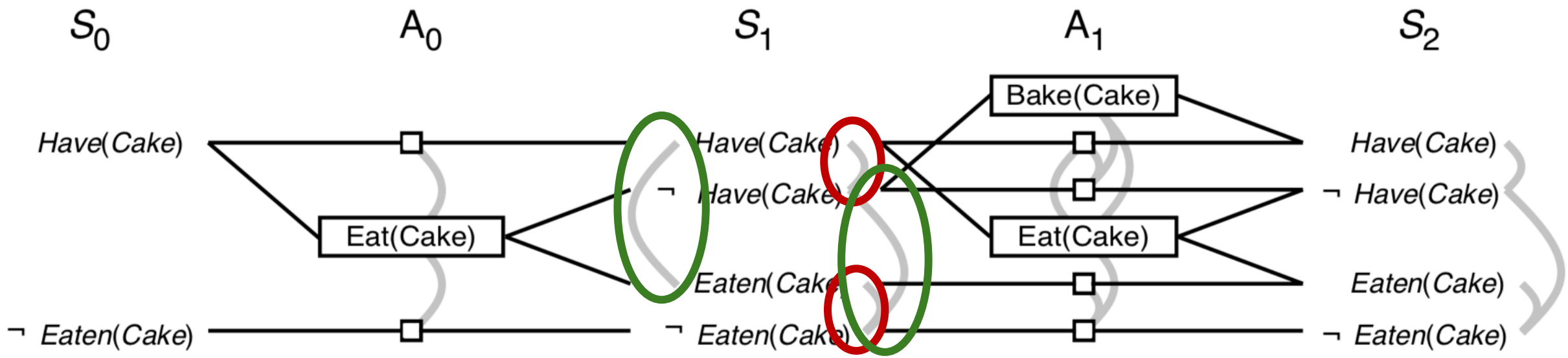
Planning Graph

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
PRECOND: $Have(Cake)$
EFFECT: $\neg\, Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
PRECOND: $\neg\, Have(Cake)$
EFFECT: $Have(Cake)$

Propositions in a layer are *mutex* if:

Negation: one negates the other

Inconsistent Support: all ways of achieving them are pairwise mutex

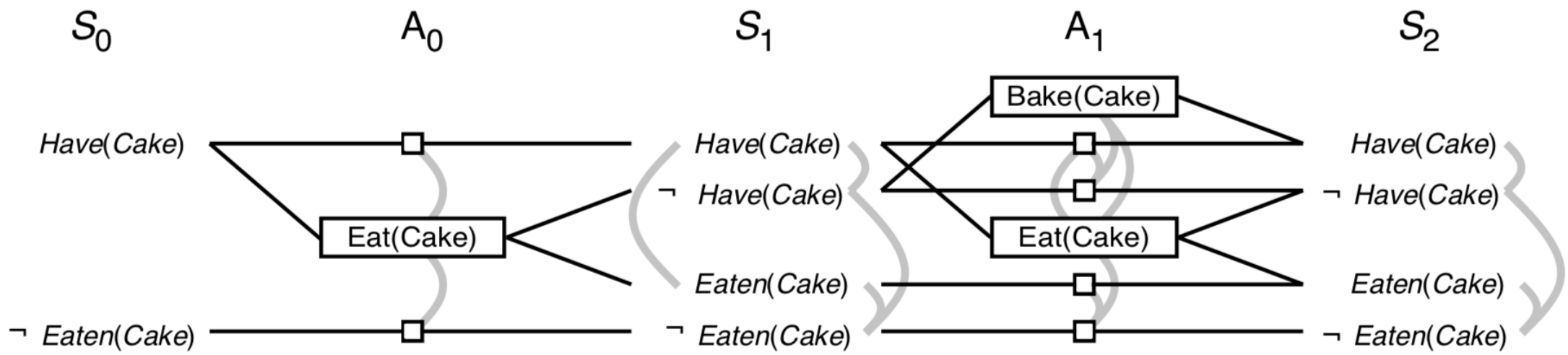


```
function GRAPHPLAN(problem) returns solution or failure

    graph ← INITIAL-PLANNING-GRAPH(problem)
    goals ← GOALS[problem]
    loop do
        if goals all non-mutex in last level of graph then do
            solution ← EXTRACT-SOLUTION(graph, goals, LENGTH(graph))
            if solution ≠ failure then return solution
            else if NO-SOLUTION-POSSIBLE(graph) then return failure
        graph ← EXPAND-GRAPH(graph, problem)
```

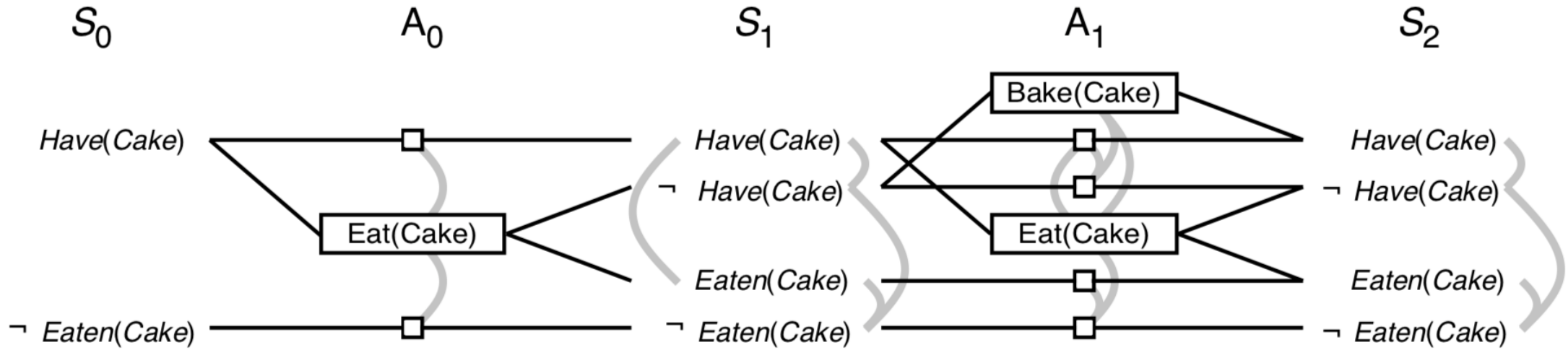
Extracting Heuristics From the Planning Graph

- Mutex relations can be used to try to extract plans that achieve the goal
- It is possible that a plan is not feasible within a given expansion of graphplan
  - Add another layer, try again
  - Until the graph levels off
  - If all the goal literals do not appear in the final level, goal is unreachable



Extracting Heuristics From the Planning Graph

- Expand the graph until you reach a fixed point
- If a literal doesn't occur in the last layer, it cannot be achieved
- Level-cost heuristic
  - Estimated cost of achieving a goal literal = level at which it first appears
  - Is this admissible?
- What about goals that include multiple literals?
  - Max-level heuristic
  - Level-sum heuristic
    - Is this optimistic (admissible) or pessimistic (non-admissible)?



Heuristic Search Planner (HSP)

- Consider only positive effects of actions (ignore negated literals in effects)
  - This creates a relaxed problem: it allows more actions to be applicable in a state than truly possible
  - Recall: To generate a heuristic, use cost of solution in relaxed problem…
  - But computing optimal solutions is too hard even for such relaxed problems
- Compute estimated cost of achieving an atom as

$$g_s(p) = \begin{cases} 0 & \text{if } p \in s, \\ \min\limits_{op \in O(p)} [1 + g_s(Prec(op))] & \text{otherwise,} \end{cases}$$

Operators that add p

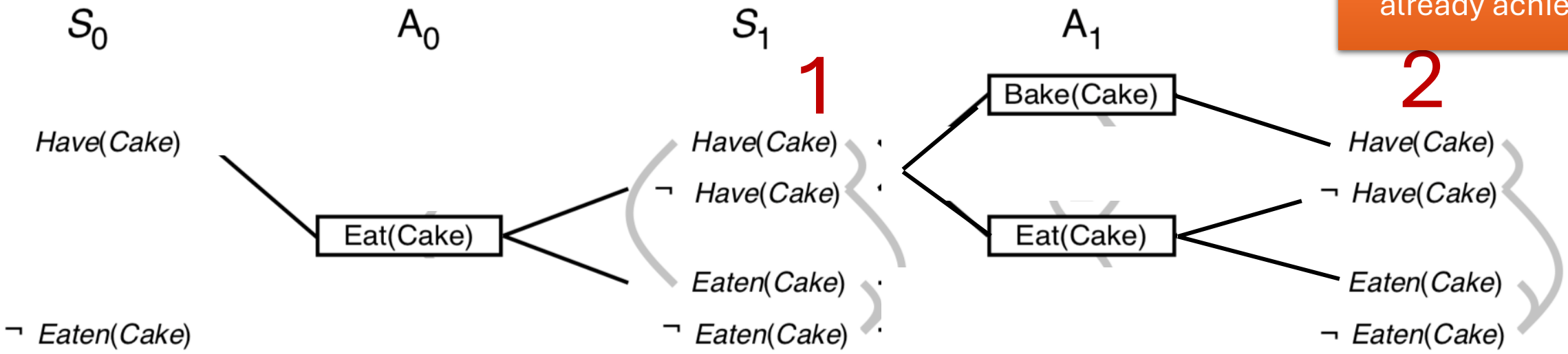
Estimated cost of achieving preconditions of op

Heuristic Search Planner (HSP)

$$g_s(p) = \begin{cases} 0 & \text{if } p \in s, \\ \min\limits_{op \in O(p)} [1 + g_s(Prec(op))] & \text{otherwise,} \end{cases}$$

Ignore mutexes, persistence actions

For atoms not already achieved!



Heuristic Search Planner (HSP)

- Compute estimated cost of achieving an atom as

$$g_s(p) = \begin{cases} 0 & \text{if } p \in s, \\ \min\limits_{op \in O(p)} [1 + g_s(Prec(op))] & \text{otherwise,} \end{cases}$$

Operators that add p

Estimated cost of achieving preconditions of op

Heuristic Search Planner (HSP)

- Compute estimated cost of achieving an atom as

$$g_s(p) = \begin{cases} 0 & \text{if } p \in s, \\ \min\limits_{op \in O(p)} [1 + g_s(Prec(op))] & \text{otherwise,} \end{cases}$$

Operators that add p

Estimated cost of achieving preconditions of op

Options for g_s of a set of atoms:

$$g_s^{min}(C) = \min_{r \in C} g_s(r) \qquad g_s^{max}(C) = \max_{r \in C} g_s(r) \qquad g_s^{+}(C) = \sum_{r \in C} g_s(r)$$

Heuristic Search Planner (HSP)

- Compute estimated cost of achieving an atom as

$$g_s(p) = \begin{cases} 0 & \text{if } p \in s, \\ \min\limits_{op \in O(p)} [1 + g_s(Prec(op))] & \text{otherwise,} \end{cases}$$

Operators that add p

Estimated cost of achieving preconditions of op

Use this to compute the heuristic estimate:

$$h(s) \coloneqq g_s(G)$$

Heuristic Search Planner (HSP)

- Compute estimated cost of achieving an atom as

$$g_s(p) = \begin{cases} 0 & \text{if } p \in s, \\ \min\limits_{op \in O(p)} [1 + g_s(Prec(op))] & \text{otherwise,} \end{cases}$$

Operators that add p

Estimated cost of achieving preconditions of op

Use this to compute the heuristic estimate:

$$h(s) := g_s(G)$$

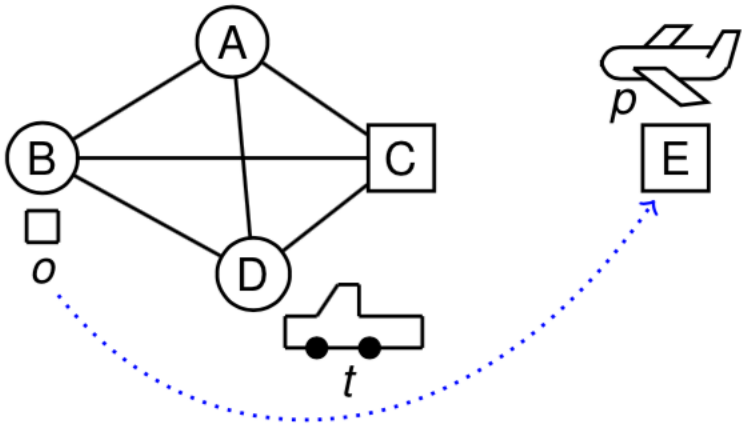
h_{min} uses g_s^{min} h_{max} uses g_s^{max} h_{add} uses g_s^{+}

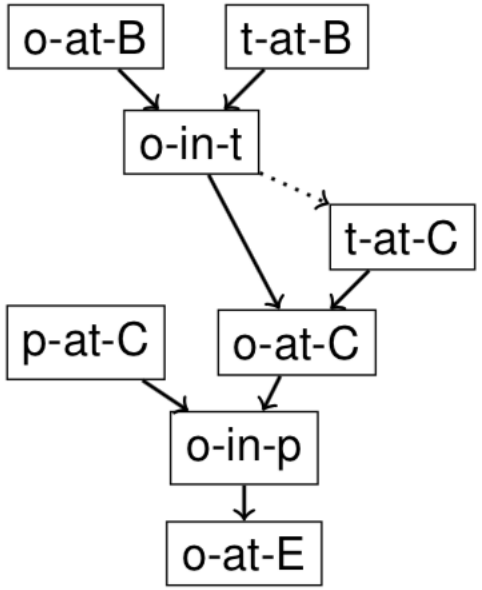
FF Planner

- https://arxiv.org/abs/1106.0675
- https://fai.cs.uni-saarland.de/hoffmann/ff.html
- Uses custom h_{FF} heuristic.
- h_{FF} is not admissible but practically better than h_{max}.
- $h_{min} \preccurlyeq h_{max} \preccurlyeq h_{FF} \preccurlyeq h_{add}$.

Landmarks in Planning

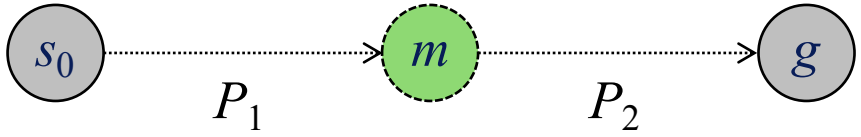
- A *landmark* is a formula that must become true at some point in every plan
- An *action landmark* is an action that must occur in every plan
- Landmarks and action landmarks can be ordered





Why are Landmarks Useful?

- Can break a problem down into smaller subproblems

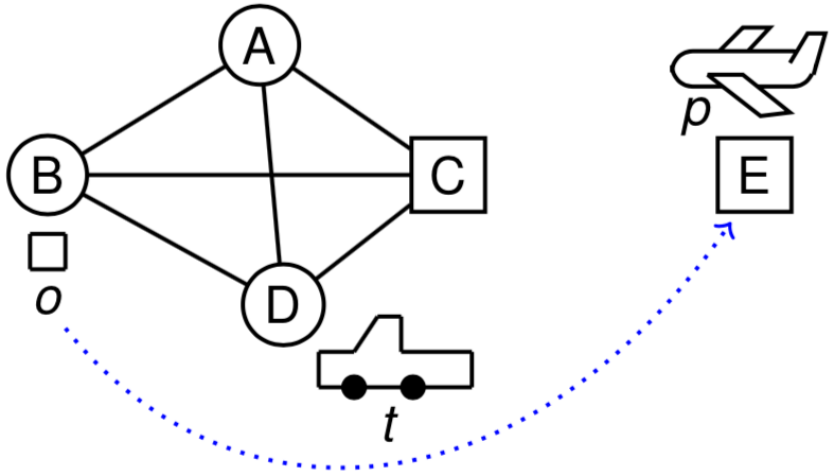


- Suppose m is a landmark
  - Every solution to P must achieve m
- Possible strategy:
  - find a plan to go from s_0 to any state s_1 that satisfies m
  - find a plan to go from s_1 to any state s_2 that satisfies g

- Suppose m_1, m_2, m_3 are landmarks
  - Every solution to P must achieve m_1, then m_2, then m_3
- Possible strategy:
  - find a plan to go from s_0 to any state s_1 that satisfies m_1
  - find a plan to go from s_1 to any state s_2 that satisfies m_2
  - …

Computing Landmarks

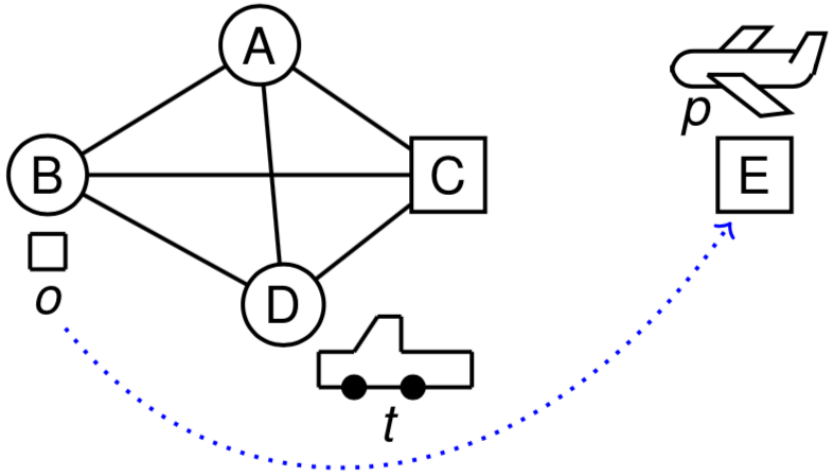
- Backchaining:
  - Every goal is a landmark
  - If B is a landmark and A is a common precondition in all actions that achieve B,
    - A is a landmark
    - $A \preccurlyeq B$

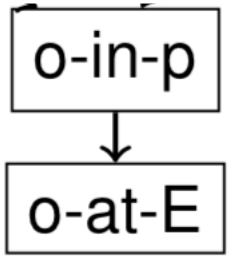


o-at-E

Computing Landmarks

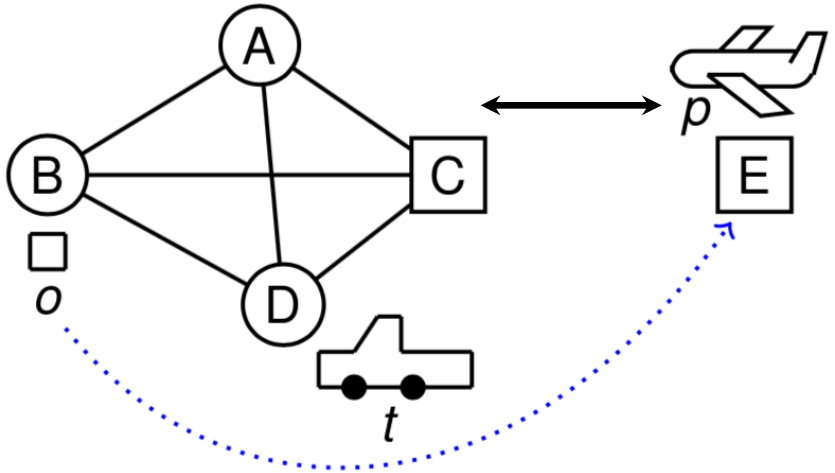
- Backchaining:
  - Every goal is a landmark
  - If B is a landmark and A is a common precondition in all actions that achieve B,
    - A is a landmark
    - $A \preccurlyeq B$

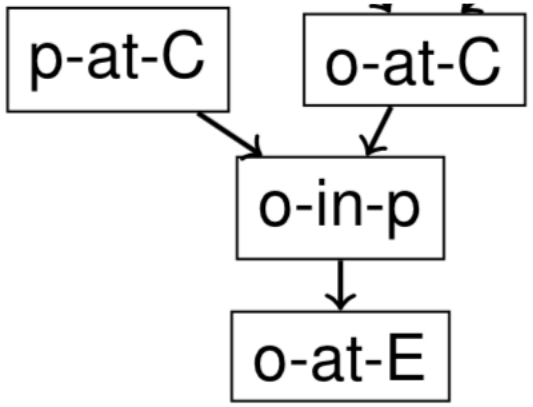




Computing Landmarks

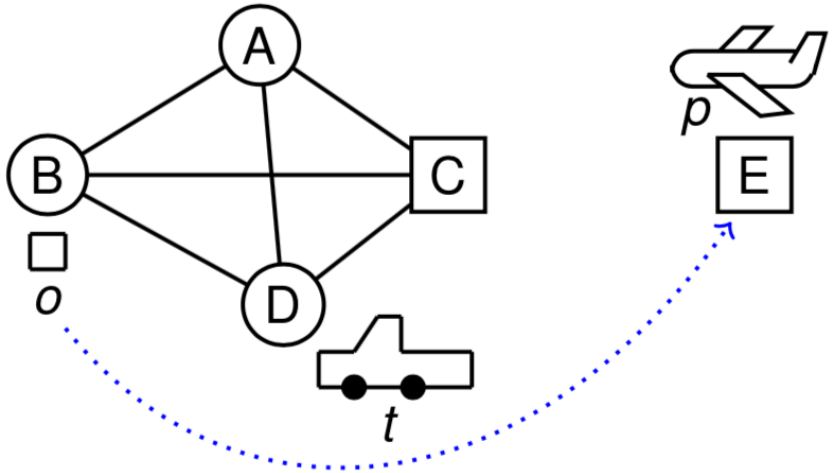
- Backchaining:
  - Every goal is a landmark
  - If B is a landmark and A is a common precondition in all actions that achieve B,
    - A is a landmark
    - $A \preccurlyeq B$

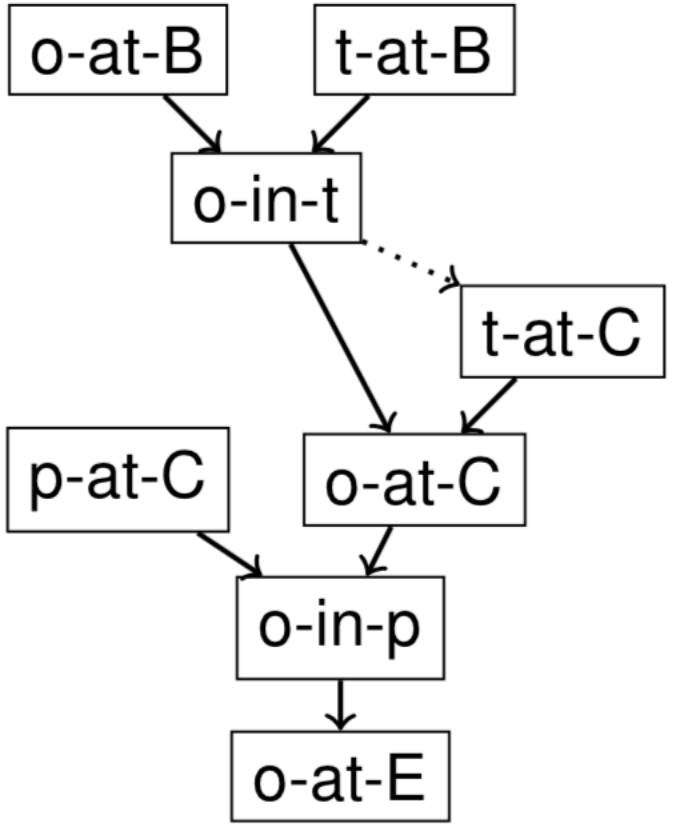




Computing Landmarks

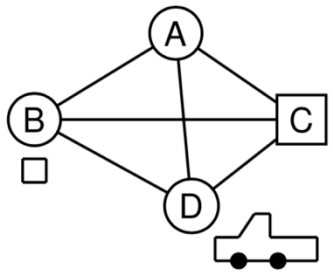
- Backchaining:
  - Every goal is a landmark
  - If B is a landmark and A is a common precondition in all actions that achieve B,
    - A is a landmark
    - $A \preccurlyeq B$

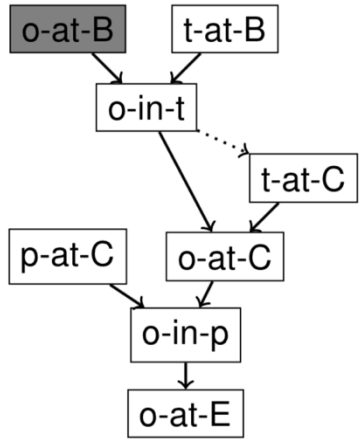




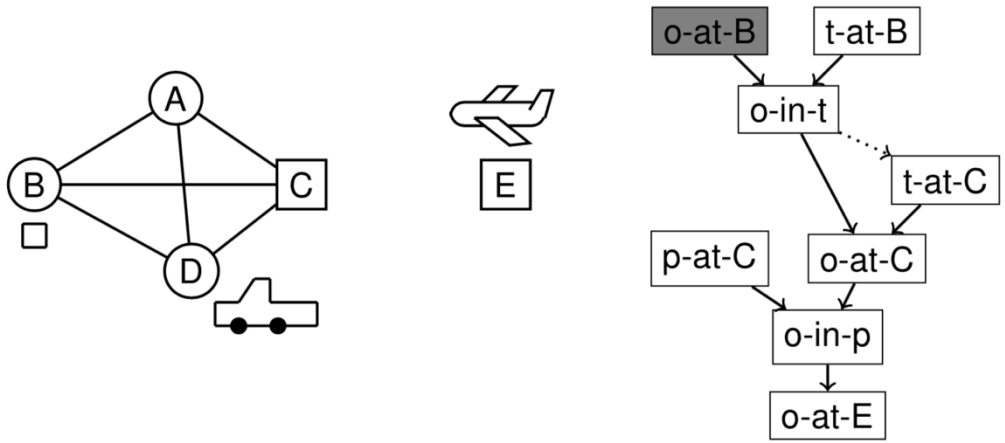
Using Landmarks as Heuristics

- Two ways of using landmarks: subgoals, components in heuristic estimate
- Subgoals:
  - Select earliest landmark(s)
  - Set as disjunctive goal for any planner
  - After plan is found, repeat

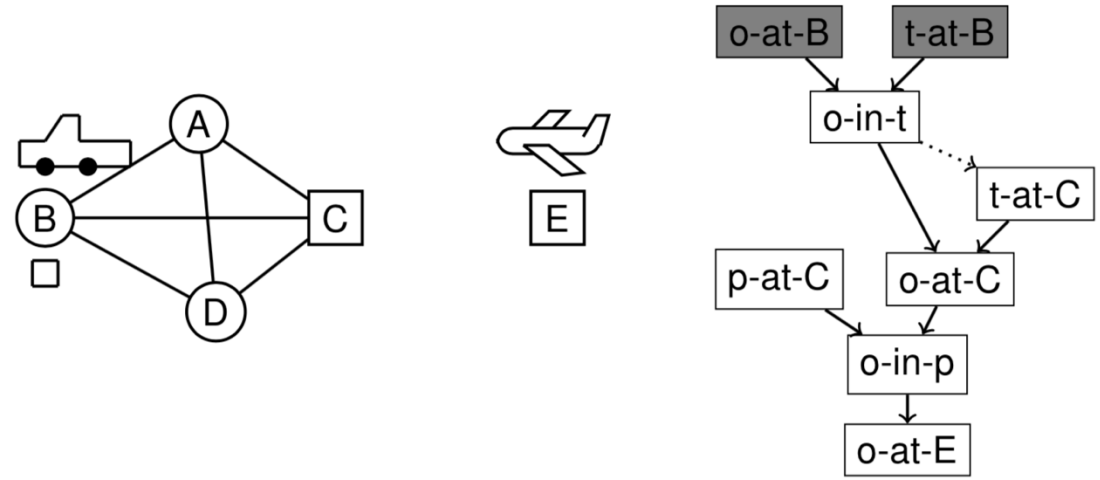




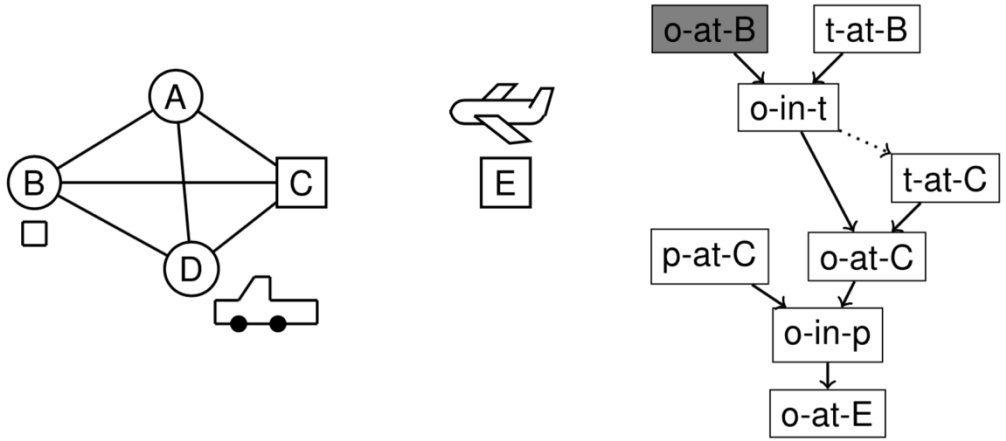
- Partial plan: $\emptyset$
- Goal: t-at-B $\vee$ p-at-C



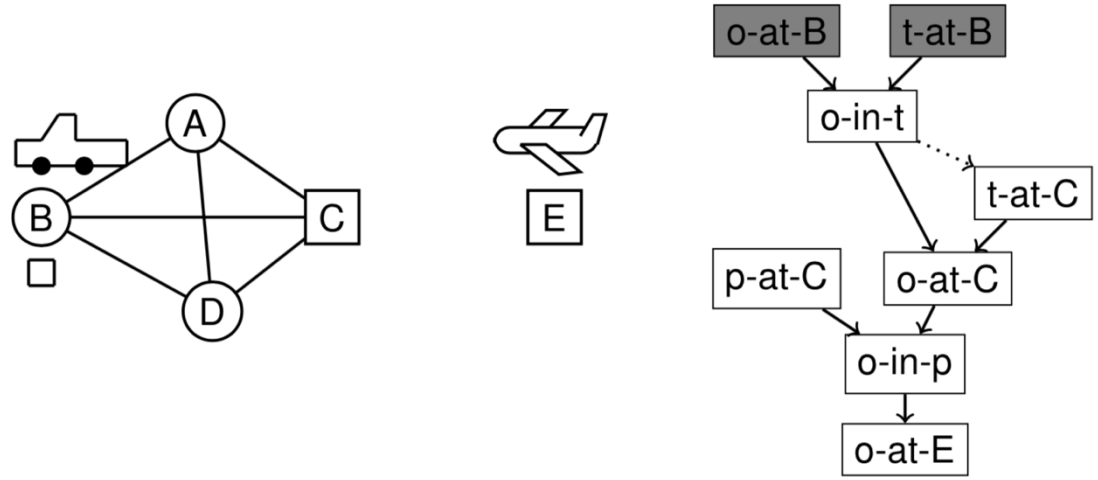
- Partial plan: ∅
- Goal: t-at-B ∨ p-at-C



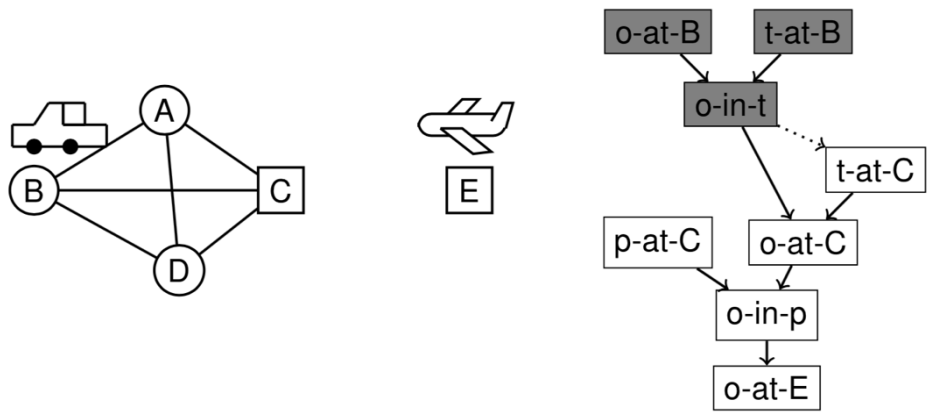
- Partial plan: Drive-t-B
- Goal: o-in-t ∨ p-at-C



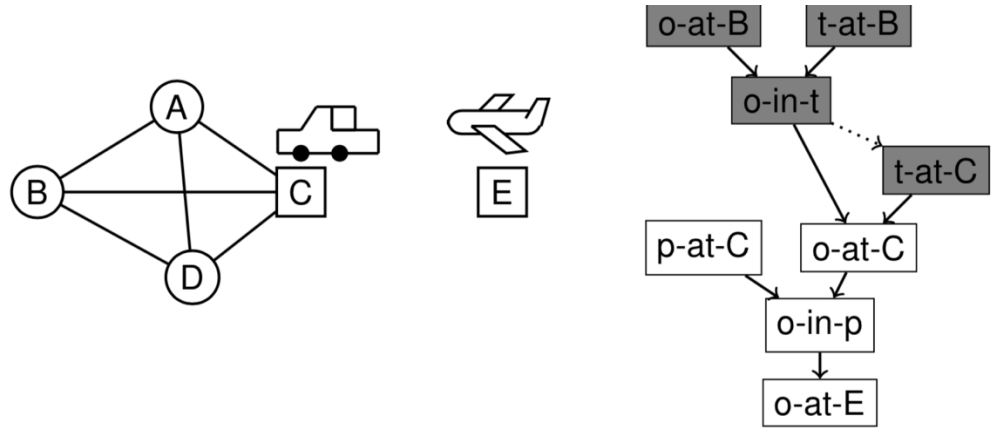
- Partial plan: ∅
- Goal: t-at-B ∨ p-at-C



- Partial plan: Drive-t-B
- Goal: o-in-t ∨ p-at-C



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



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

Using Landmarks

- Using landmarks as subgoals is not guaranteed to be efficient
  - Order of achieving mutually unordered landmarks can affect size of the plan
    - Variant of sussman's anomaly
  - May fail for problems with deadends
- Alternative:
  - Use # unachieved landmarks as a part of the heuristic estimate

Backward Search

- Forward search: forward from initial state
  - In state s, choose applicable action a
  - Compute state transition $s' = T(s,a)$
- Backward search: backward from the goal
  - For goal g, choose *relevant* action a
    - A possible "last action" before the goal
    - Sometimes this has a lower branching factor
- Compute *inverse* state transition $g' = T^{-1}(g,a)$
  - g' = properties a state s' should satisfy in order for $T(s',a)$ to satisfy g
- Equivalently, if S_g = {all states that satisfy g} then
  - $S_{g'}$ = {all states s such that $T(s,a) \in S_g$}