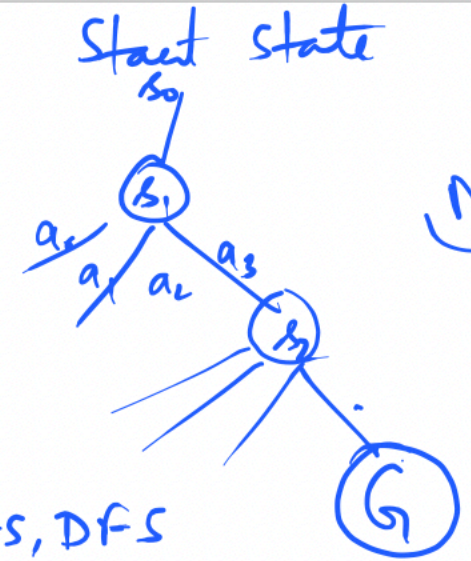
AI Search

Plan sequence of actions.

Planning



No percepts

→ Uninformed Search → BFS, DFS

→ Informed Search $h(n)$ admissible → never overestimate

Planning as a search problem

- Initial State S_0
- Actions available in a state
- Result of applying an action
- Goal Test

factored → take advantage of certain heuristics

atomic → BFS, DFS

Background

- **Atom** (Atomic Sentence)
  Can take a value True/False.
  `hot, humid, snowy, rainy, famous,`
  `At(Tajmahal,Agra)`
  - may or may not contain variables

- **Literal** can be a positive atom or a negated atom

- State elements (aka **Fluents**)
  Atoms which are
  - ground - do not have variables
  - functionless - do not have functions. `Inventor(Steam_Engine)`

- A **State**
  - Represented as a **conjunction of fluents**

`hot ∧ humid`
`Studies (Joe, Class1) ∧ Studies (Jack, Class2)`
`At (Truck1, Pune) ∧ At (Truck2, Mumbai)`

- Why do we want to describe a state as a conjunction of fluents?
  - so that a state can be manipulated using logical inference

Actions

- ACTIONS(s)
  Set of actions that are applicable in a state s
- RESULT (s, a)
  The result of applying an action a to state s

derived if the action schema is given.

- **Action Schema**

template

plane, airport 'from', airport 'to'

```
Action ( Fly(p, from, to),
     PRECOND: At (p, from) ∧ Plane (p) ∧ Airport
(from) ∧ Airport (to)
     EFFECT: ¬At(p, from) ∧ At (p, to)
)
```

Fly (P1, DEL, BOM)

action instance of the schema

PDDL: Planning domain definition language

- Precondition defines the states in which the action can be executed
- Effect defines the result of executing the action.

- From the action schema, we can obtain(instantiate) a **ground** action

```
Action ( Fly (P1, DEL, BOM),
      PRECOND: At (P1, DEL) ∧ Plane (P1) ∧ Airport
(DEL) ∧ Airport (BOM)
      EFFECT: ¬At (P1, DEL) ∧ At (P1, BOM)
)
```

- (a ∈ ACTIONS (s)) ⇔ s ⊨ PRECOND(a)

To apply action a in state s, the state must entail the preconditions for action a

⊨: stands for entailment

Computing the Successor State

- RESULT(s, a) = (s - DELETE(a)) ∪ ADD(a)

s: state a: action

DELETE(a): set of fluents that appear as negative literals in the action's effects

ADD(a): set of fluents that appear as positive literals in the action's effects

```
Action ( Fly (P1, DEL, BOM),
   PRECOND: At (P1, DEL) ∧ Plane (P1) ∧ Airport (DEL) ∧ Airport (BOM)
   EFFECT: ¬At (P1, DEL) ∧ At (P1, BOM)
)
```

The above action will remove At(P1, DEL)

and add At(P1, BOM)

Planning Domain Definition **Language** (PDDL)

- Action Schema (template)
  Can be used to represent a collection of actions.
  Represents a state in a factored way — conjunction of fluents
- A **set of action schemas** serves as a definition of a planning *domain*.

- A specific problem within the domain is defined with –
  - an **initial state** (conjunction of ground atoms, with closed world assumption)
  - **goal** (conjunction of literals, may contain variables)

$$At(p, DEL) \wedge Plane(p)$$

RESULTS(s, a)
ACTIONS(s)

The Planning Problem

$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \rightarrow \rightarrow s_f \rightarrow G$

Find a sequence of actions that end in a state that entails the goal.

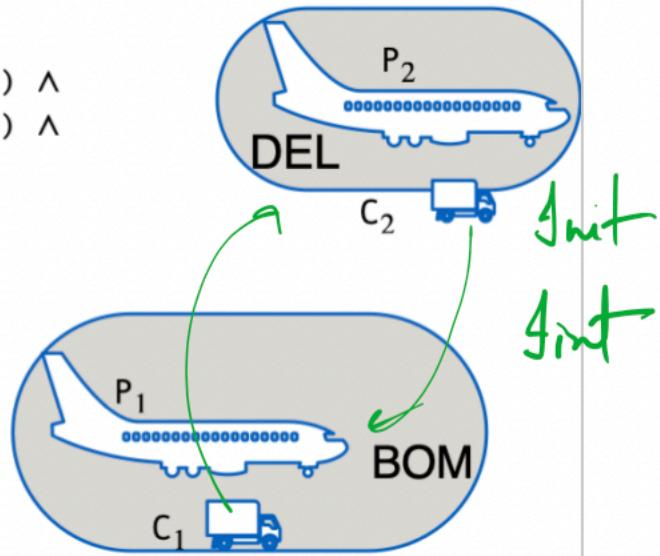
$s_f \models G$

Example Planning Problem: Air Cargo Transport

Initial state

- Init (At(C_1, BOM) ∧ At(C_2, DEL) ∧
  At(P_1, BOM) ∧ At(P_2, DEL) ∧
  Cargo (C_1),
  Cargo (C_2),
  Plane (P_1),
  Plane (P_2),
  Airport (BOM),
  Airport (DEL)
)
- Goal (At (C_1, DEL) ∧ At (C_2, BOM))

  Actions: Load, Unload, Fly



Example Planning Problem: Air Cargo Transport

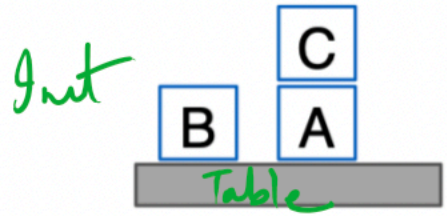
- Action (**UnLoad** (c, p, a),
  PRECOND: In (c, p) ∧
  At (p, a) ∧
  Cargo (c) ∧
  Plane (p) ∧
  Airport (a)

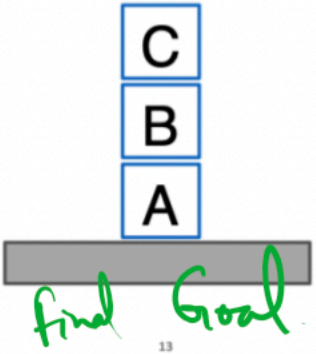
  EFFECT: At(c, a) ∧ ¬In (c, p)
)

- Action (**Fly** (p, from, to),
  PRECOND: At (p, from) ∧
  Plane (p) ∧
  Airport (from) ∧
  Airport (to)

  EFFECT: ¬At(p, from) ∧ At(p, to)
)

Example Planning Problem: **The blocks world**





- Goal (On (A, B) ∧
  On (B, C)

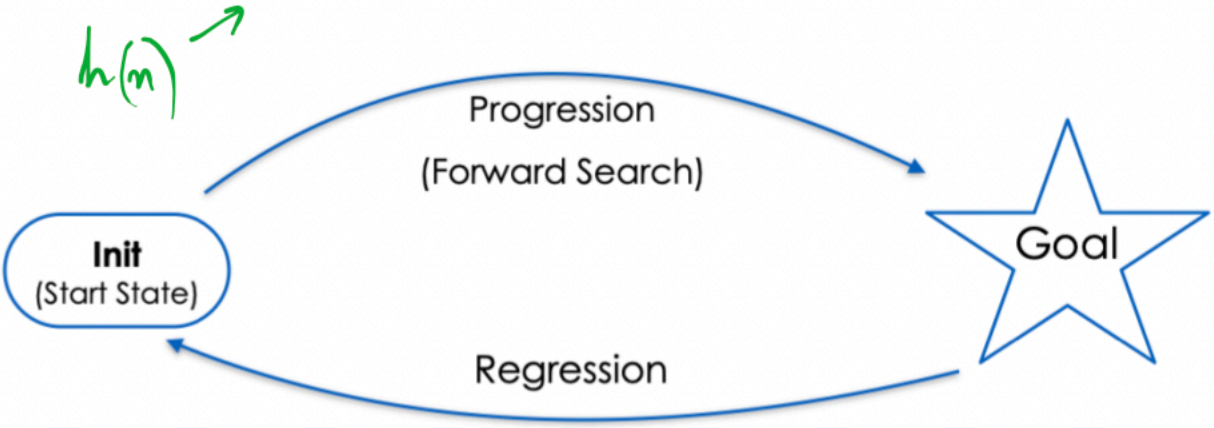
- Action (Move (b,x,y),

PRECOND:
On (b, x) ∧
Clear (b) ∧
Clear (y) ∧
Block (b) ∧
Block (y) ∧
(b ≠ x) ∧
(b ≠ y) ∧
(x ≠ y),

EFFECT:
On (b, y) ∧
Clear (x) ∧
¬On (b, x) ∧
¬Clear (y)
)

Algorithms for Planning

- We have been able to map the planning problem to a State Space Search setting.
- We can use heuristic search algorithms or local search algorithms



Large State Spaces in Planning Problems

single Airport

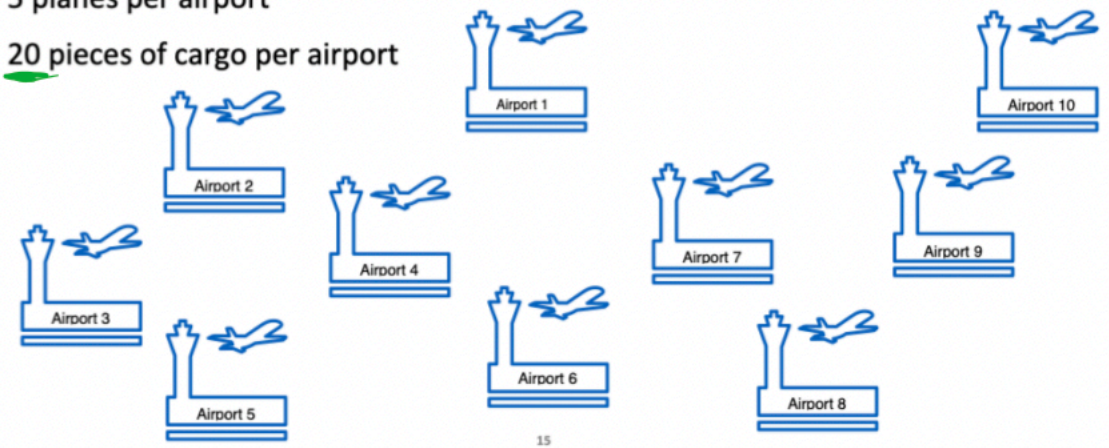
- Air Cargo Problem

  10 airports

  5 planes per airport

  20 pieces of cargo per airport

20 pcs.

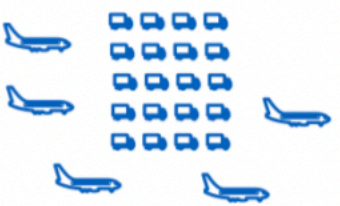


20 pcs.

- The goal is to move all the cargo from Airport A to Airport B.

  This should take 41 actions (steps),

  - loading 20 pieces,
  - flying the plane and
  - unloading the 20 pieces.



Large State Spaces in Planning Problems

- Consider the state space:

  The number of actions available in a state varies with the number of states.

  If all the planes and all the cargo are at a single airport, then:

  Total 200 pieces of cargo can load to 50 planes:

  giving 200 x 50 = 10000 actions load actions

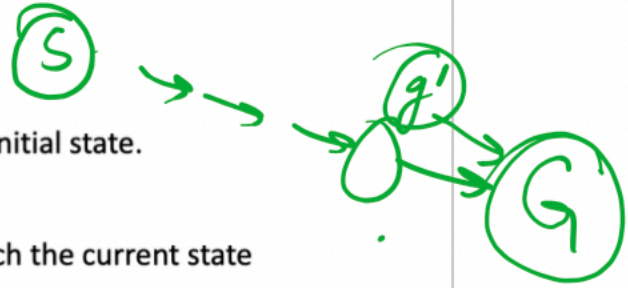
  450 destinations for airplanes to fly

  Total 10450 actions possible

q Destination

- If all Cargo is at airport with no planes, then 450 actions are possible.
- Average number of actions available in any state: 2000

Backward Search



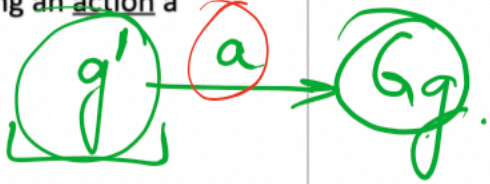
- We begin searching from the goal description to the initial state.
- We search for the relevant predecessor states.
- That is, find out the actions which are relevant to reach the current state
- One thing to note here is that the goal is a state description
  - Goal is specified by a set of literals.

    There are other literals which are unspecified.
  - We should know how to move one step backwards from a state description.

Backward Search

- Regressing (i.e. moving backwards) from a goal description g using an action a gives us a state description

$$g' = (g - ADD(a)) \cup Precond(a)$$



last action in the plan.

i.e. remove from the goal, any effects that were added by the action a

And also add the pre-conditions that make the action a applicable.

Backward Search

- Goal Description

$g \equiv At\ (C_{15},\ DEL)$

Consider action

Unload $(C_{15},\ p,\ DEL)$

$ADD(a) = At(C_{15},\ DEL)$

$PRECOND(a) = In\ (C_{15},\ p)\ \wedge$
$At\ (p,\ DEL)\ \wedge$
$Cargo\ (C_{15})\ \wedge$
$Plane\ (p)\ \wedge$
$Airport\ (DEL)$

- Action (**UnLoad** $(C_{15},\ p,\ DEL)$,
  PRECOND: $In\ (C_{15},\ p)\ \wedge$
  $At\ (p,\ DEL)\ \wedge$
  $Cargo\ (C_{15})\ \wedge$
  $Plane\ (p)\ \wedge$
  $Airport\ (DEL)$

  EFFECT: $At(C_{15},\ DEL)\ \wedge\ \neg In\ (C_{15},\ p)$
)

remove

Note that the variable p remains uninstantiated

$In(C_{15}, P) \wedge$.

$g' \xrightarrow{UNLOAD} g$

$In(P, DEL) \wedge Cargo(\ ) \wedge Plane(\ )$

Backward Search

Backward Search

- What is a relevant action for a goal description ?

  - A relevant action is the one that could be **the last action in a plan** leading up to the goal description or current state description

  - i.e. at least one of the action's effect should unify with an element of the goal description.

Unify means the literals become equivalent after variable substitution

The variable substitution should be the most general unifier.

Backward Search

ISBN

- Goal
  Own (0135042597)

  Action:
  Buy (i) with
  substitution
  {i/0135042597} θ

ground action.

- Action (Buy(i),

  PRECOND:
  ISBN (i)
  EFFECT:
  Own (i)
)

Example

- The goal is to deliver a specific piece of cargo at SFO

Goal $At(C_2, SFO)$

- This suggests the action $Unload(C_2, p', SFO)$

$Action(Unload(C_2, p', SFO),$
$\text{PRECOND:} In(C_2, p') \wedge At(p', SFO) \wedge Cargo(C_2) \wedge Plane(p') \wedge Airport(SFO)$
$\text{EFFECT:} At(C_2, SFO) \wedge \neg In(C_2, p')\,.$

- The regressed state description is

$$g' = In(C_2, p') \wedge At(p', SFO) \wedge Cargo(C_2) \wedge Plane(p') \wedge Airport(SFO)$$

Collection of states

$g' = (g - ADD(a)) \cup Precond(a)$

Backward Search

- Which action candidates to regress over?
  - Forward search choses actions that are applicable
  - Backward search choses actions that are relevant

    The relevant actions are defined as the ones that could be the last step in the plan leading up to the current goal state.
- For example, if the goal is $A \wedge B \wedge C$ then an action having the effect $A \wedge B \wedge \neg C$ cannot be the final step of a solution.

Seeking the most general unification with the action's effects

- For the goal `At(C2, DEL)`, one can use several instances of the `unload` action.
- Specifying a separate action for every plan will lead to a very large branching factor.
- Instead of using specific instances of the action, we use the generalised action `Unload(C2, p', DEL)` which is obtained by substituting the most general unifier into the (standardised) action schema.

- Assume a goal description g which contains a goal literal g_i and an action schema A that is standardised to produce A'
- A' has an effect literal e'_j which unifies with g_i using the substitution θ

  $\texttt{Unify}(g_i, e_j) = \theta$

- We apply the same substitution θ to the action A' to obtain $a' = \text{SUBST}(\theta, A')$
- a' is a relevant action towards goal g if there is no effect in a' that is the negation of a literal in g

- However, the fact that backward search uses state sets rather than individual states makes it harder to come up with good heuristics.

50 planes
10 airports
200 pieces of cargo
In
At

50 axioms for Ap1. At (p, A)
50 . . . A2
50 . . A3

Heuristics for Planning

Domain-independent Heuristics

50^{10}

$(200)^{50}$ | 200 axioms about the cargo in the 1st plane
200 . . .
2^{4}

$50^{10} \times 200^{50} \times 200^{10} = 10^{155}$ states

$(200)^{10}$ | 200 axioms about cargo in an airport A1
200 A2

Heuristics for Planning

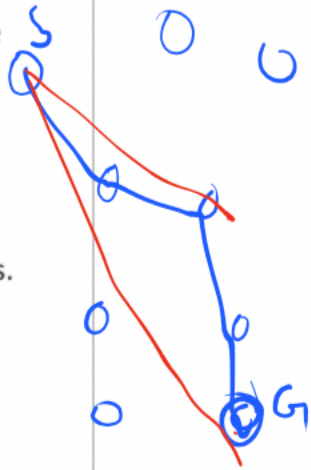
- A heuristic function is quite useful to guide the state space search.
- Heuristic function $h(s)$ is a function of the state s and estimates the distance of the state s to the goal state.
- A heuristic function should have a property called admissibility.
- That is, the estimated distance should always be less than the actual distance. If this property does not hold, the search will not yield an optimal solution.

To derive an admissible heuristic:

- Formulate a problem that is easier to solve it to get the exact cost.
  The exact cost of the solution to the relaxed problem then becomes the heuristic for the original problem.
- The factored representation of states in a planning problem helps in getting good relaxed problems by exploiting good domain independent heuristics.

Planning as a Search Problem

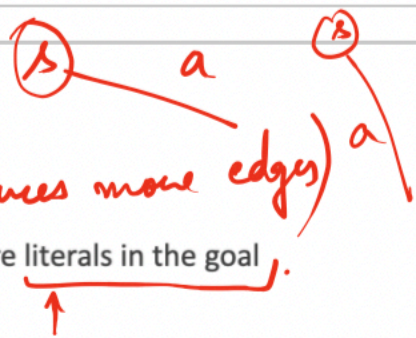
- The search problem is to find a path (in a graph) connecting the initial state to the goal state.



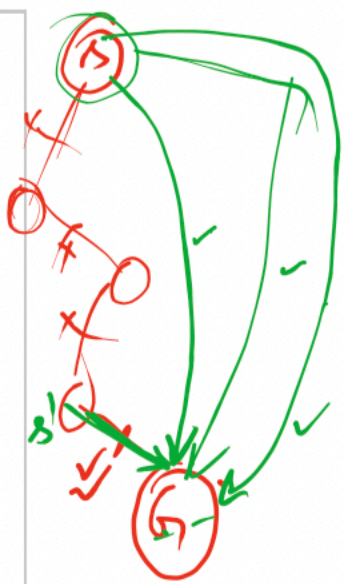
- To relax (i.e. to make it easier) a search problem,
  - we can add more edges to the graph - This makes it easier to find the paths.
  - We can group multiple nodes together, forming an abstraction of the state space. With fewer nodes, it is easier to search.
- Such relaxations are possible by manipulating the schemas.

- First, we relax the actions by
  - removing all preconditions
  - removing all effects except those that are literals in the goal.

(introduces more edges)



- Then we count the minimum number of actions required such that the union of those actions' effects satisfies the goal. (Set Cover Problem)
- Another way is to remove only selected preconditions of actions.



Relaxed Problem - Adding more edges

- Drop all preconditions from actions
- Every action becomes applicable in every state.
- This implies that every single goal fluent can be achieved in one step.
- So the number of steps required to solve the problem is the number of unsatisfied goals. However,
  - Some actions may achieve multiple goal fluents.
  - Some actions may undo the effects of others.

Ignore some preconditions

$$Action(Slide(t, s_1, s_2),$$
$$\text{PRECOND}: On(t, s_1) \wedge Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1, s_2)$$
$$\text{EFFECT}: On(t, s_2) \wedge Blank(s_1) \wedge \neg On(t, s_1) \wedge \neg Blank(s_2))$$

- If we remove the preconditions $Blank\ (s_2) \wedge Adjacent\ (s_1, s_2)$ then any tile can move in one action to any space and we get the number of misplaced tiles heuristic.
- If we remove $Blank\ (s_2)$ then we get the Manhattan distance heuristic.

Ignore delete lists

- Assume that all goals and preconditions contain only positive literals.
- The relaxed version of the problem is obtained by removing the delete lists from all actions (i.e. removing all negative literals from effects).
- Thus an action can never undo the progress made by another action.
- Ensures monotonic progress towards the goal.

Is relaxing the actions enough?

- Many planning problems have 10^{100} states.
- Relaxing the actions adds edges, but does not reduce the number of states.
- Many states can be combined by using state abstractions.

  This reduces the number of states.
- A solution in the abstract space will be shorter than a solution in the original space.

Axiom Cargo for destination 1 has been loaded onto the plane
2
3
⋮
5

5
5

Axiom Plane for destination 1 at A1.
Axiom " " 2 at A1
⋮
5 at A1

50 P
10 A
200 C
10^{155}

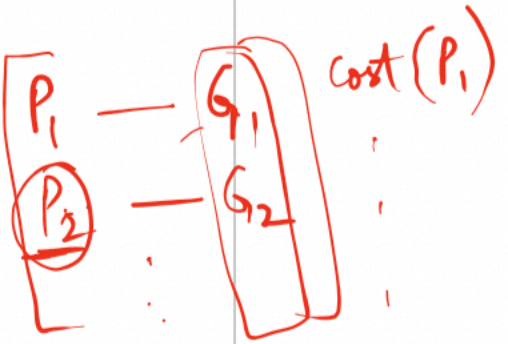
50P
10A → 5A
→C
$5^{10}\ 5^{5}\ 5^{10} \simeq 10^{17}$ state

Cargo for destination 1 at A1.
2 at A1.
⋮
5

5^{10}
$(5)^{10}$

Decomposition of the goal

- Decomposition: Dividing a problem into parts, solving each part independently, and then combining the parts.
- Suppose the goal is a set of fluents G, which we divide into disjoint subsets $G_1,\ldots,G_n$.
- We then find plans $P_1, \ldots, P_n$ that solve the respective subgoals.
- $\max_i \text{cost}(P_i)$ always yields an admissible heuristic
- If we combine the component costs using addition $\text{cost}(P_1) + \text{cost}(P_2) + \ldots$. then is the resulting estimate admissible?



too Optimistic

Subgoal Independence Assumption

- Subgoal independence assumption: says that the cost of solving a conjunction of subgoal is approximated by the sum of the costs of solving each subgoal independently.
  - If the assumption holds then the additive cost will be more accurate than $\max_i \text{cost}(P_i)$
- Without the assumption holding the estimates are
  - Optimistic when there are negative interactions between the subplans
  - Pessimistic when the subplans contain redundant actions.

Real h(n) → solve the real

- Pessimistic when the subplans contain redundant actions.

Real → h(n) → solve the real

Ensure that the heuristics indeed save the costs

- Computational costs are involved in
  - Defining the abstraction
  - Doing an abstract search
  - Mapping the abstraction back to the original problem
- We must ensure that the total cost of solving the problem after using the abstraction is less than the cost of solving the original problem.

Abstract solving (exact)

Planning Graphs

domain-independent heuristic

So ↓ ??

G.

Search tree → Planning Graph

→ Belief states (state levels)

Clubbing of actions

- A planning graph is a special data structure which can be used to obtain better heuristic estimates.
- Consider a tree with all possible actions from the initial state to the successor states and further actions to their successors and so on.
- This tree is exponential in size. So even a simple lookup to this tree to check if the goal state G is reachable from the start state S_0 can take exponential time.

Planning Graph

- A planning graph is a polynomial size approximation to this tree.
- A planning graph cannot answer definitively whether G is reachable from S_0 but it can estimate the number of steps to reach G.
- If it reports that the goal is not reachable, its absolutely correct.
- So it never over-estimates the number of steps required to reach the goal.

Planning Graph

- A planning graph is organized into levels, alternating between fluents and actions.
- The first level contains a set of nodes, each one representing a fluent in the initial state S_0
- Next level A_0 has nodes for each ground action or any choice of subsets of actions that might be applicable in S_0

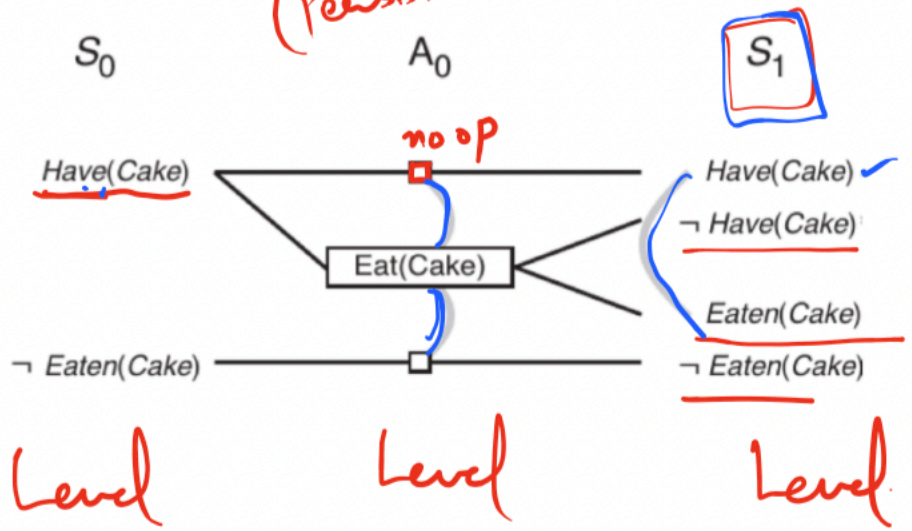
  (Note that we do not actually choose among actions) Instead we bring in all actions which are applicable and mention about their mutex links.
- This is followed by levels $S_1, A_1, S_2, A_2, \ldots$ till a termination condition is reached.

- For the set of fluents S_i at time i we execute the applicable actions A_i. This gives the resulting set of possible fluents S_{i+1}
- If P or $\neg P$ could hold, then both will be represented in S_{i+1}
- The applicable actions A_i are all the actions which have their preconditions satisfied at time i.
- We write possible fluents because there might be negative interactions between the fluents and the planning graph records only a restrict subset of the possible negative interactions.

- A literal may show up as true at level S_j when actually it would become true only at a later level.
- Still, the level j where the literal first appears is a good estimate of the number of steps it would take to achieve that literal from S_0

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

Eat, Bake.

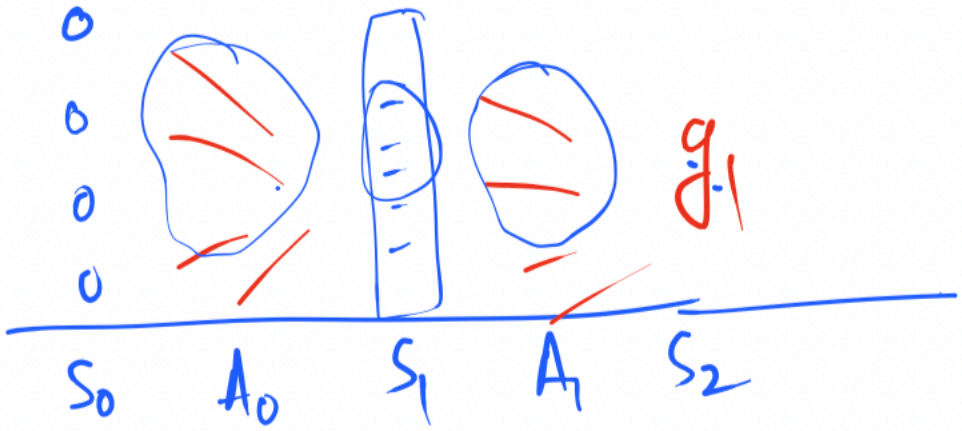


- Each action at level A_i is connected to its preconditions at S_i and its effects at S_{i+1}.
- So a literal appears because an (real) action caused it.
- A literal persists if no action negates it.
  Or we can assume that there is a no-op action which causes the literal to persist.

- The gray lines indicate mutual exclusion, i.e. the two actions cannot occur together because they disagree on their effects.
- Eat (Cake) has a mutex with Have(Cake) or not Eaten (Cake).
- Mutex links can be for action pairs as well as for fluent pairs.
- For example Have (Cake) and Eaten (Cake) are mutex. One of them can appear, but not both.

- A state level S_i represents a belief state. A possible state in this belief state can be a subset of literals such that no mutex links exist between the members of the subset.
- As we alternate between state level S_i and action level A_i, eventually the graph levels off - the consecutive levels are identical.

Mutex links for actions

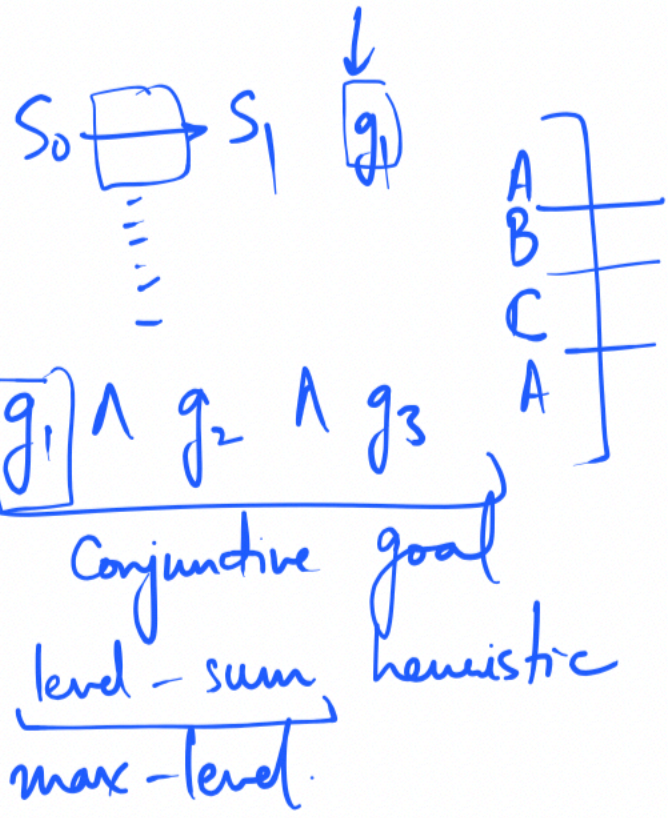


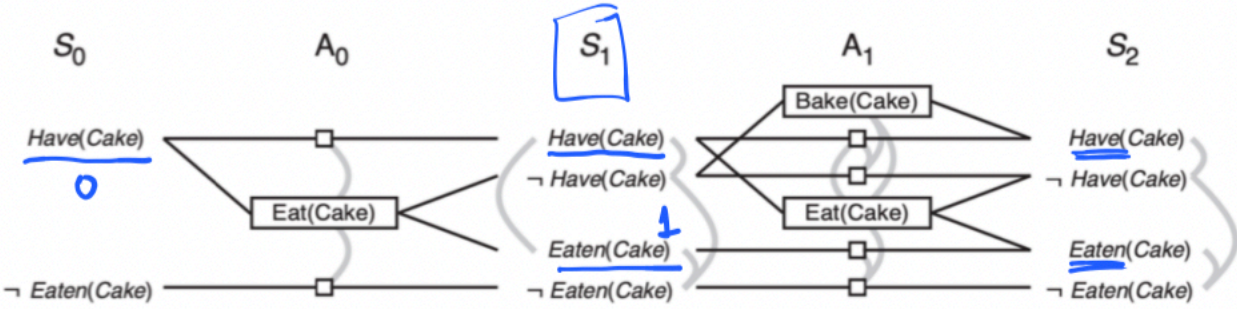
Planning Graphs for heuristic estimation

- If any goal literal fails to appear in the final level of the graph, then the problem is unsolvable.

- We can estimate the cost of achieving any goal literal g_i from state s as the level at which g_i first appears in the planning graph.
  This cost is called as the level cost of g_i

g_1





- Have(Cake) has level cost 0
- Eaten(Cake) has level cost 1

$0 + 1 = 1 < 2$

- Level cost is an admissible heuristic
- But it is not always accurate because planning graphs allow several actions at each level, whereas the heuristic counts just the level and not the number of actions.
- A serial planning graph can give more reasonable heuristic because it allows only one action to occur at any given time step.

$g_1 \wedge g_2 \wedge g_3$

- Cost of a conjunction of goals is estimated in 3 simple ways:
  - Max-level heuristic
  - Level sum heuristic
  - Set-level heuristic

level at which all the goal literals are achieved.

- If the goal g appears in the planning graph, then the planning graph can only promise that there is a plan that possibly achieves g and has no obvious flaws.

involves just 2 literals/actions

The GRAPHPLAN algorithm

- So far we have seen that a planning graph can be used to provide a heuristic.
- Now we shall see that a planning graph can also be used to directly extract a plan.

→ CSP
→ Backward search

(level, goal) pairs that are not achievable.

(1, g_2) nogood

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

graph ← INITIAL-PLANNING-GRAPH(*problem*) S_0 A_0 S_1 A_1 - -

goals ← CONJUNCTS(*problem*.GOAL) g_1 g_2 g_3

nogoods ← an empty hash table

for $tl = 0$ **to** ∞ **do**

  if *goals* all non-mutex in S_t of *graph* **then**

    solution ← EXTRACT-SOLUTION(*graph*, *goals*, NUMLEVELS(*graph*), *nogoods*)

    if *solution* ≠ *failure* **then return** *solution*

  if *graph* and *nogoods* have both leveled off **then return** *failure*

  graph ← EXPAND-GRAPH(*graph*, *problem*)

A_0 S_1 → create the next level of the planning graph

57

Spare Tire Problem

$Init(Tire(Flat) \land Tire(Spare) \land At(Flat, Axle) \land At(Spare, Trunk))$

$Goal(At(Spare, Axle))$

$Action(Remove(obj, loc),$

  PRECOND: $At(obj, loc)$

  EFFECT: $\neg At(obj, loc) \land At(obj, Ground))$

$Action(PutOn(t, Axle),$

  PRECOND: $Tire(t) \land At(t, Ground) \land \neg At(Flat, Axle)$

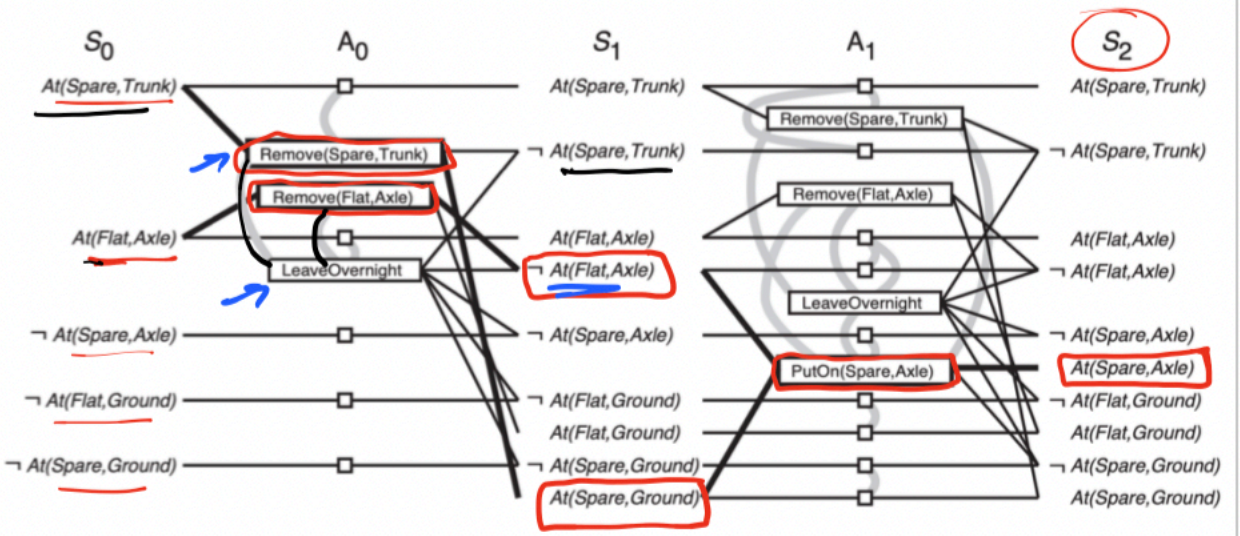
  EFFECT: $\neg At(t, Ground) \land At(t, Axle))$

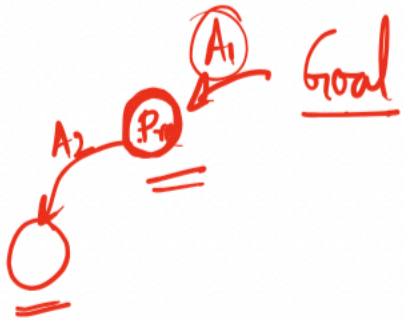
$Action(LeaveOvernight,$

  PRECOND:

  EFFECT: $\neg At(Spare, Ground) \land \neg At(Spare, Axle) \land \neg At(Spare, Trunk)$
  $\land \neg At(Flat, Ground) \land \neg At(Flat, Axle) \land \neg At(Flat, Trunk))$

58





Mutex for Actions

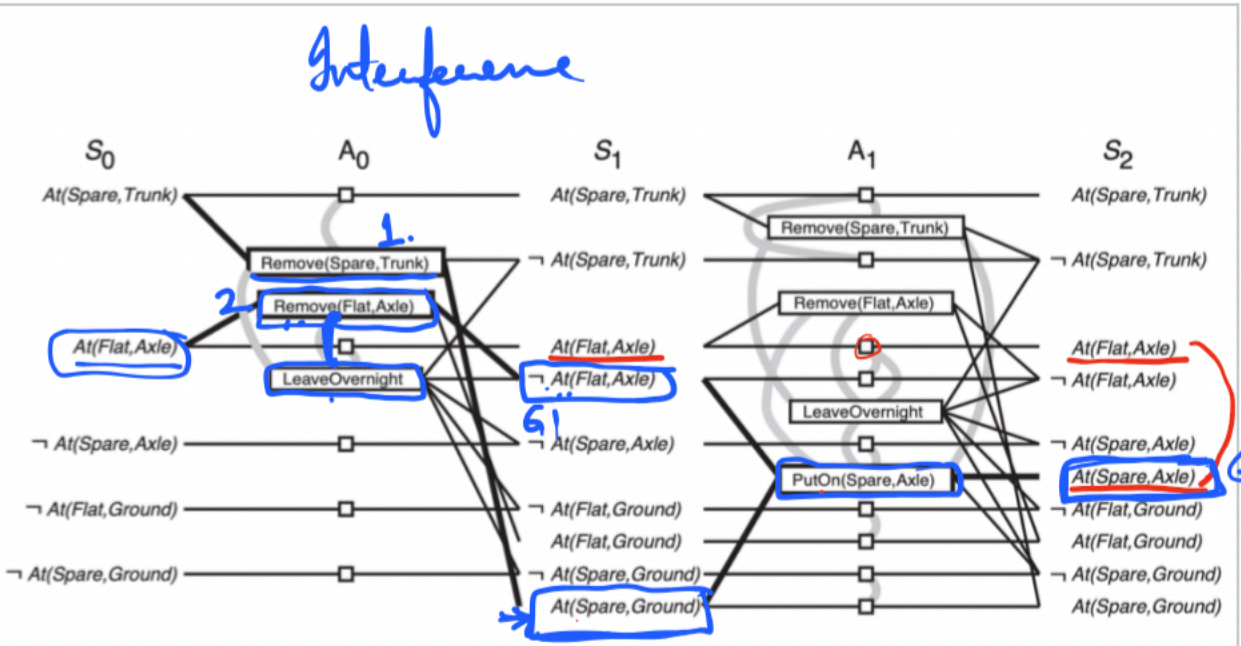
- *Inconsistent effects:* $Remove(Spare, Trunk)$ is mutex with $LeaveOvernight$ because one has the effect $At(Spare, Ground)$ and the other has its negation.

- *Interference:* $Remove(Flat, Axle)$ is mutex with $LeaveOvernight$ because one has the precondition $At(Flat, Axle)$ and the other has its negation as an effect.
- *Competing needs:* $PutOn(Spare, Axle)$ is mutex with $Remove(Flat, Axle)$ because one has $At(Flat, Axle)$ as a precondition and the other has its negation.

Mutex for Literals

- $At(Spare, Axle)$ is mutex with $At(Flat, Axle)$ in S_2

61



62

Interference

Level 0

No change

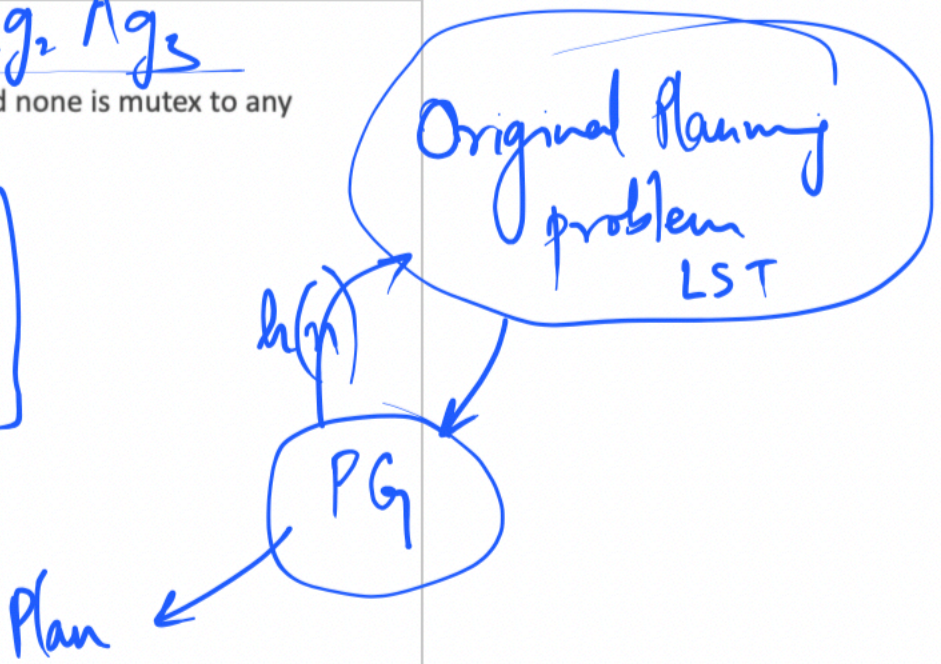
Actions

Literals

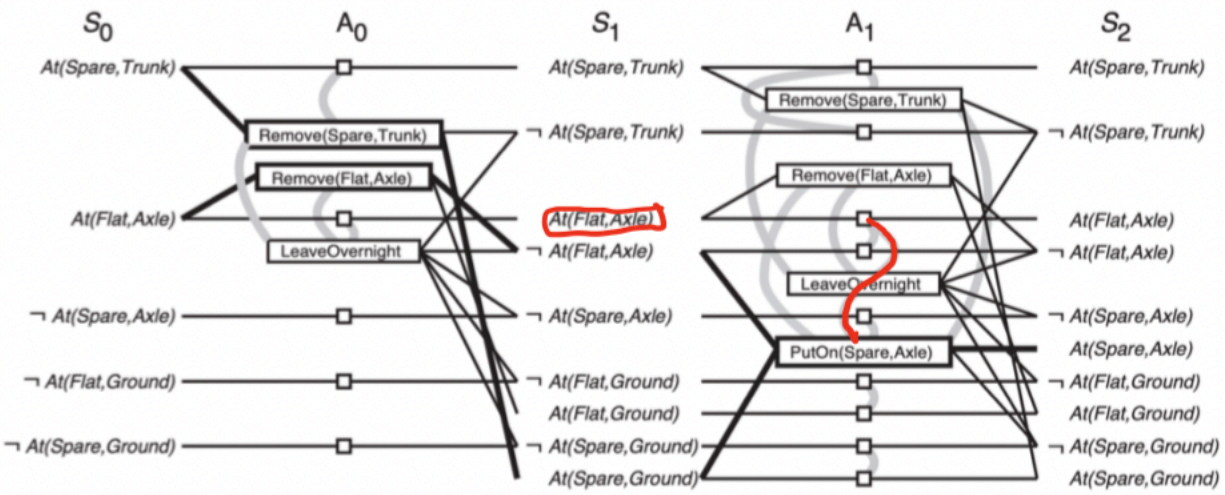
increase monotonically.

Goal

- At level S2 all the literals from the goal are present and none is mutex to any other.
- We can now call EXTRACT-SOLUTION to find a plan.
- We can formulate EXTRACT-SOLUTION
  - as a Boolean CSP
  - as a Backward Search Problem



- The initial state is the last level of the planning graph, S_n, along with the set of goals from the planning problem.
- The actions available in a state at level S_i are to select any conflict-free subset of the actions in A_{i-1} whose effects cover the goals in the state.
- The goal is to reach a state at level S_0 such that all the goals are satisfied.



65

fill
→ the (# no-goods) becomes constants

- In case of failure, the (level, goal) pairs can be marked as "no-good."

Literals ↑ # Actions ↑ # Mutexes ↓

no-goods ↓

Mutex between { Actions, Literals → Complementary }

Conflicting needs

Inconsistent effects, Interference, Helpless

66

Cargo Transport

g_1 g_2 g_3 g_4

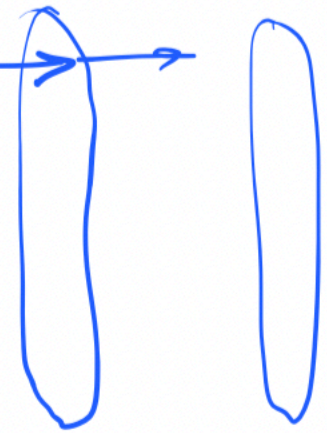
?

A → B

PGraph

5 Level of

g_1



Goal is not achievable

no-good $(g_2, 5)$ gets removed in the next level.

no-good $(g_3, 5)$

no-good $(g_4, 5)$

Precondition → Action → Effect

(A) Interference

(A) Competing needs

(A) Inconsistent effects

(L) Inconsistent Support — L1 — L2