Graphplan/ SATPlan

Chapter 11.4-11.7

Some material adapted from slides by
Jean-Claude Latombe / Lise Getoor

GraphPlan

GraphPlan: Basic idea

- Construct a graph that encodes constraints on possible plans
- Use this "planning graph" to constrain search for a valid plan
- Planning graph can be built for each problem in a relatively short time

Planning graph

- Directed, leveled graph with alternating layers of nodes
- Odd layers ("**state levels**") represent candidate propositions that could possibly hold at step *i*
- Even layers ("**action levels**") represent candidate actions that could possibly be executed at step *i*, including maintenance actions [do nothing]
- **Arcs** represent preconditions, adds and deletes
- We can only execute one real action at any step, but the data structure keeps track of **all actions and states that are *possible***

GraphPlan properties

- STRIPS operators: conjunctive preconditions, no conditional or universal effects, no negations
  - Planning problem must be convertible to propositional representation
  - NO continuous variables, temporal constraints, …
  - Problem size grows exponentially
- Finds "shortest" plans (by some definition)
- Sound, complete, and will terminate with failure if there is no plan

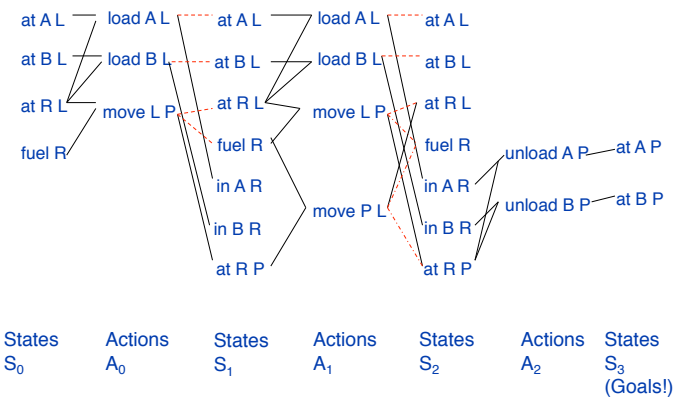
What actions and what literals?

- Add an action in level A_i if ***all*** of its preconditions are present in level S_i
- Add a literal in level S_i if it is the effect of ***some*** action in level A_{i-1} (*including no-ops*)
- Level S_0 has all of the literals from the initial state

Simple domain

- Literals:
  - at X Y — X is at location Y
  - fuel R — rocket R has fuel
  - in X R — X is in rocket R
- Actions:
  - load X L — load X (onto R) at location L (X and R must be at L)
  - unload X L — unload X (from R) at location L (R must be at L)
  - move X Y — move rocket R from X to Y (R must be at L and have fuel)
- Graph representation:
  - Solid black lines: preconditions/effects
  - Dotted red lines: negated preconditions/effects

Example planning graph



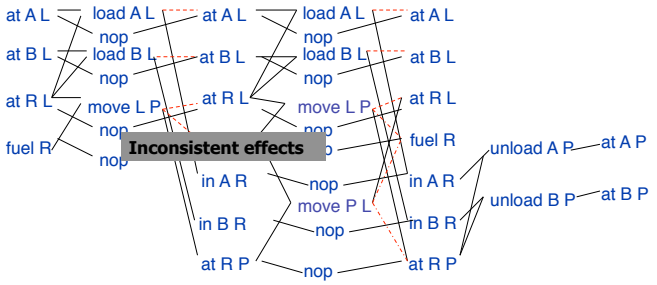
Valid plans

- A **valid** plan is a planning graph where:
  - Actions at the same level don't interfere (delete each other's preconditions or add effects)
  - Each action's preconditions are true at that point in the plan
  - Goals are satisfied at the end of the plan

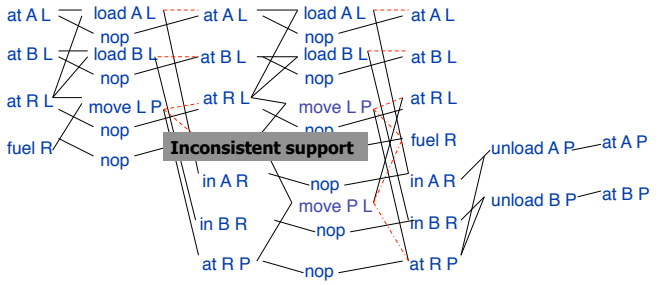
Exclusion relations (mutexes)

- Two actions (or literals) are **mutually exclusive ("mutex")** at step *i* if no valid plan could contain both.
- Can quickly find and mark *some* mutexes:
  - **Interference**: Two actions that interfere (the effect of one negates the precondition of another) are mutex
  - **Competing needs**: Two actions are mutex if any of their preconditions are mutex with each other
  - **Inconsistent support**: Two literals are mutex if all ways of creating them both are mutex

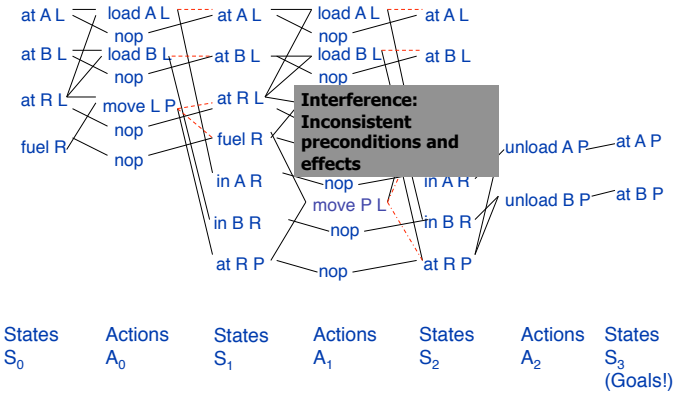
Example: Mutex constraints



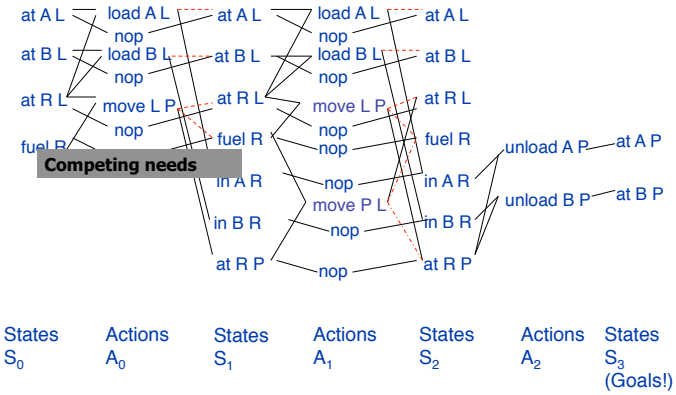
Example: Mutex constraints



Example: Mutex constraints



Example: Mutex constraints



Extending the planning graph

- **Action level A_i**:
  - Include all instantiations of all actions (including maintains (no-ops)) that have all of their **preconditions satisfied** at level S_i, with no two being mutex
  - Mark as mutex all **action-maintain pairs** that are incompatible
  - Mark as mutex all **action-action pairs** that have competing needs
- **State level S_{i+1}**:
  - Generate all propositions that are the **effect of some action** at level A_i
  - Mark as mutex all pairs of propositions that can only be generated by **mutex action pairs**

Basic GraphPlan algorithm

- **Grow** the planning graph (PG) until all goals are reachable and none are pairwise mutex. (If PG levels off [reaches a steady state] first, fail)
- **Search** the PG for a **valid plan**
- If none found, **add a level** to the PG and try again

Creating the planning graph is usually fast

- Theorem 1:

  The size of the t-level planning graph and the time to create it are polynomial in:
  - t (number of levels),
  - n (number of objects),
  - m (number of operators), and
  - p (number of propositions in the initial state)

Searching for a plan

- Backward chain on the planning graph
- Complete all goals at one level before going back
- At level *i*, pick a non-mutex subset of actions that achieve the goals at level *i+1*. The preconditions of these actions become the goals at level *i*.
- Build the action subset by iterating over goals, choosing an action that has the goal as an effect. Use an action that was already selected if possible. Do forward checking on remaining goals.

SATPlan

SATPlan

- Formulate the planning problem as a CSP
- Assume that the plan has k actions
- Create a binary variable for each possible action a:
  - Action(a,i) (TRUE if action a is used at step i)
- Create variables for each proposition that can hold at different points in time:
  - Proposition(p,i) (TRUE if proposition p holds at step i)

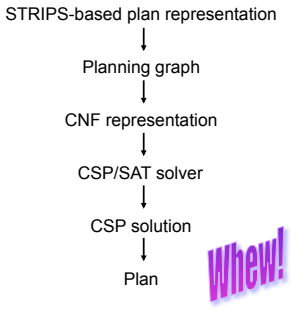
Constraints

- Only one action can be executed at each time step (XOR constraints)
- Constraints describing effects of actions
- Persistence: if an action does not change a proposition p, then p's value remains unchanged
- A proposition is true at step i only if some action (possibly a maintain action) made it true
- Constraints for initial state and goal state

Now apply our favorite CSP solver!

Still more variations…

- Blackbox:



Applications of Planning

- Military operations
- Autonomous space operations
- Most applied systems use extended representation languages, nonlinear planning techniques, and domain-specific heuristics
- Design of experiments in genetics
- Command sequences for satellite