

Planning, Execution & Learning: GraphPlan

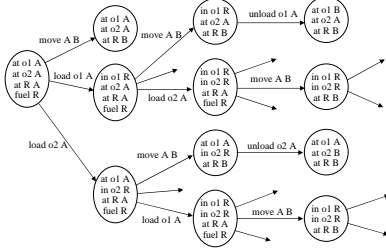
Reid Simmons

Planning in Graph-Space

- Graph is Factored State-Space
 - Nodes are *literals* or *instantiated actions*
 - Directed layered (leveled) graph
- Forward/Backward Search
 - Use (approximate) reachability analysis to constrain search
 - Backward search extracts valid plan
- Plan is Sequence of Sets of Actions
 - $\{a_1, a_2\}; \{a_3, a_4, a_5\}; \{a_6\}$
 - More general than state-space plan (sequence)
 - Less general than plan-space plan (partial order)

Reachability Analysis

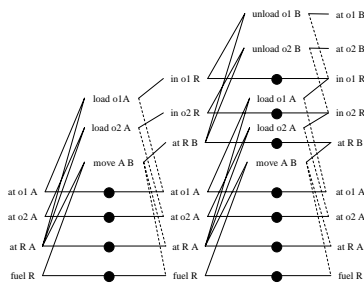
- What States are Reachable in N Steps
 - Can find *all* goals reachable from initial state
 - Exponential in time and memory



Plan Graph

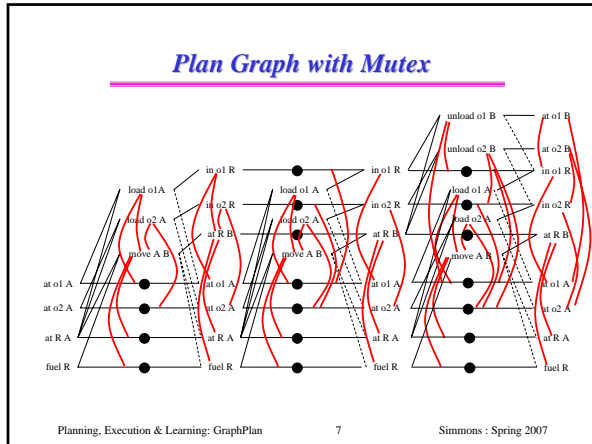
- Relaxes Reachability Analysis
 - Not *necessary* conditions, but *sufficient*
 - Trade off time spent computing reachability (polynomial time and space) for time spent extracting plan (exponential)
- Layered Graph
 - Factors state-space into propositions and actions
 - To create *action-level i*:
 - Add each instantiated operator preconditions are all present at proposition-level *i*
 - Add all the no-op actions
 - To create *proposition-level i+1*:
 - Add all effects of the actions at action-level *I*
 - Distinguish add and delete effects

Plan Graph (Naïve)

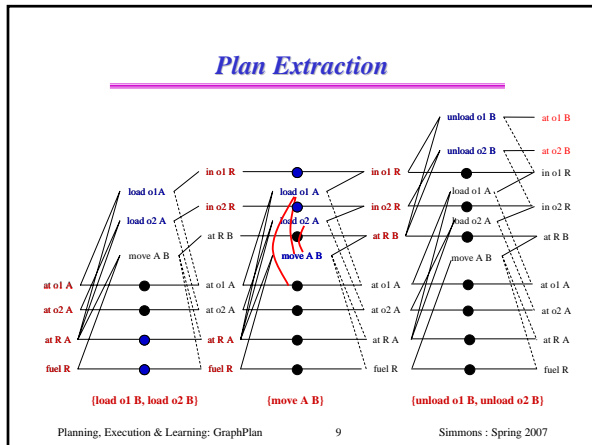


Mutual Exclusivity (Mutex) Constraints

- Actions A and B are *exclusive*, at action-level *i*, if:
 - Interference:** A (or B) deletes a precondition or an add-effect of B (or A)
 - Competing Needs:** *p* is a precondition of A and *q* is a precondition of B, and *p* and *q* are exclusive in proposition-level *i - 1*
- Propositions *p* and *q* are *exclusive* in a proposition-level *i* if:
 - All** actions that add *p* are exclusive of **all** actions that add *q*



- ### Plan Extraction
- GP (*plan-graph, level, goals*)
 - $i = \text{level}; \text{plan} = []; \text{goalset} = \text{goals};$
 - If some goal in *goalset* not present in *plan-graph[i]* or some goals in *goalset* are mutex in *plan-graph[i]* return NULL
 - Repeat until $i = 0$
 - $\text{opset} = []; \text{preconds} = [];$
 - For each goal g in *goalset*:
 - Choose action a at level $i - 1$ that achieves g and it is not exclusive with any other action in *opset*
 - add a to *opset*
 - add preconditions of a to *preconds*
 - $i = i - 1; \text{goalset} = \text{preconds}; \text{plan} = [\text{opset}; \text{plan}]$
 - Enhancement:**
 - Memoize (hash) sets of goals that are unsolvable at some level
 - Use to do immediate backtrack
- Planning, Execution & Learning: GraphPlan 8 Simmons : Spring 2007



- ### Graphplan Extensions
- Trivial:
 - Negated predicates
 - Disjunctive preconditions
 - Quantification
 - Conditional Effects
 - Full expansion (Kazen & Knoblock)
 - Partial expansion (Koehler)
 - Factored expansion (Anderson, Smith & Weld)
 - “In Place” Graph Expansion
 - Use monotonicity properties to maintain single bipartite graph
- Planning, Execution & Learning: GraphPlan 10 Simmons : Spring 2007

- ### Factored Expansion
- Basic Ideas:
 - Make all effects conditional
 - Extend the way mutexes are determined
 - Factor Operator Definitions
 - Pull preconditions into effects
 - Pickup(b)
 - if (Block(b), Handempty, Clear(b), On(b, x))
 - Add: Holding(b); Delete: Handempty, On(b, x)
 - if (Block(b), Handempty, Clear(b), On(b, x), Block(x))
 - Add: Clear(x)
 - Separate into “components”
- Planning, Execution & Learning: GraphPlan 11 Simmons : Spring 2007

- ### Factored Expansion
- Infer Mutex for **Induced Components**
 - C_m induces C_n if:
 - C_m and C_n are not mutex
 - The negation of C_n 's conditions cannot be satisfied if C_m occurs
 - None of the antecedents are deleted
 - A negated antecedent of C_n is mutex with some antecedent of C_m
 - C_m and C_k are **mutex** if:
 - C_m induces C_n and
 - C_n and C_k are mutex
-
- Planning, Execution & Learning: GraphPlan 12 Simmons : Spring 2007

Factored Expansion

- Search is Extended to Handle Equivalent of “Confrontation”
 - Ensure components of same action do not delete goals
 - Consider all pairs of components to see if any are mutex
 - May have to consider components that are not even in plan graph yet!
 - Prevent any way for undesired condition to occur (possible **choice point**)
 - Clever implementation: Add a no-op action that represents the negation of the antecedent