Classical Planning: instantiated actions, state search, heuristics

Manuela M. Veloso

Carnegie Mellon University
Computer Science Department

15-887 – Planning, Execution, and Learning – Fall 2016

Outline

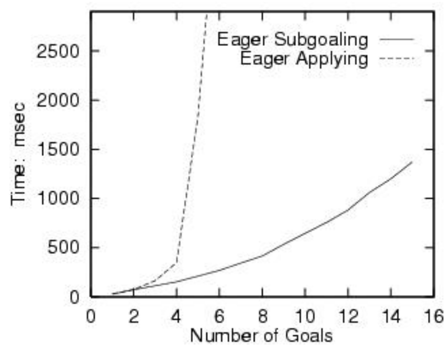
- State-space search
- GraphPlan
  - A type of state-space search
  - Fully instantiated operators
- (Satplan, FF)

Impact of State-Space Search and MEA

$$
\begin{aligned}
\text{Operator}: &\ A_i \\
\text{preconds}: &\ \{I_i\} \\
\text{adds}: &\ \{G_i\} \\
\text{deletes}: &\ \{I_j \mid j < i\}
\end{aligned}
$$

Example:

- Initial state: I_1, I_2, I_3
- Goal: G_2, G_3, G_1
- Plan: A_1, A_2, A_3



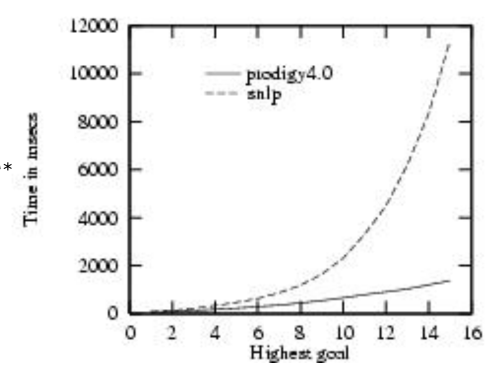
Impact of State-Space and MEA

```
operator Ai              operator A*
  preconds g*, gi-1        preconds ()
  adds     gi              adds     g*
  deletes  g*              deletes  ()
```

Initial state: g_*

Goal statement: g_*, g_5

Plan: $A_1, A_*, A_2, A_*, A_3, A_*\ A_4, A_*$



Example: One-Way Rocket Domain

```
(OPERATOR LOAD-ROCKET
 :preconds
  ?roc ROCKET
  ?obj OBJECT
  ?loc LOCATION
 (and (at ?obj ?loc)
      (at ?roc ?loc))
 :effects
  add (inside ?obj ?roc)
  del (at ?obj ?loc))
```

```
(OPERATOR UNLOAD-ROCKET
 :preconds
  ?roc ROCKET
  ?obj OBJECT
  ?loc LOCATION
 (and (inside ?obj ?roc)
      (at ?roc ?loc))
 :effects
  add (at ?obj ?loc)
  del (inside ?obj ?roc))
```

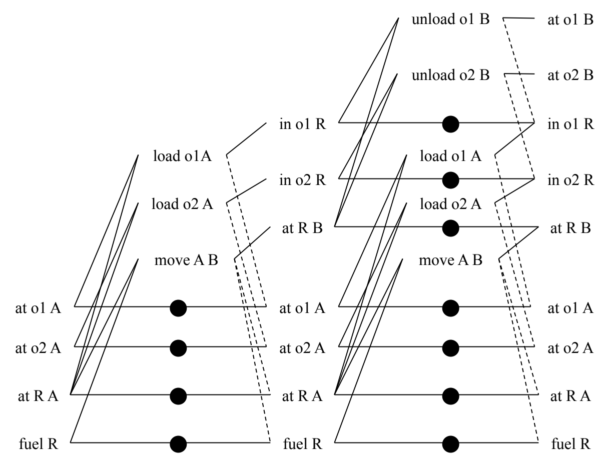
```
(OPERATOR MOVE-ROCKET
 :preconds
  ?roc ROCKET
  ?from-l LOCATION
  ?to-l LOCATION
 (and (at ?roc ?from-l)
      (has-fuel ?roc))
 :effects
  add (at ?roc ?to-l)
  del (at ?roc ?from-l)
  del (has-fuel ?roc))
```

Graphplan

Blum & Furst 95

- Preprocessing before engaging in search.
- Forward search combined with backward search.
- Construct a *planning graph* to reveal constraints
- Two stages:
  - Extend: One time step in the planning graph.
  - Search: Find a valid plan in the planning graph.
- Graphplan finds a plan or proves that no plan has fewer "time steps."

Plan Graph
One-Way Rocket Example



Extending a Planning Graph - Actions

- To create an action-level *i*:
  - Add each instantiated operator, for which all of its preconditions are present at proposition-level *i* AND *no two of its preconditions are exclusive*.
  - Add all the no-op actions.
- Determine the exclusive actions.

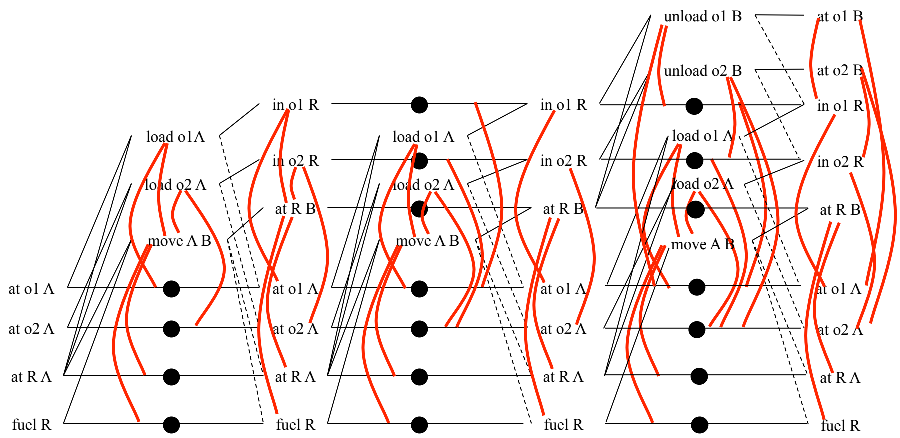
Extending a Planning Graph – Propositions

- To create a proposition-level *i* + 1:
  - Add all the effects of the inserted actions at action-level *i* - distinguishing add and delete effects.
- Determine the exclusive actions.

Planning Graphs

- A literal may exist at level *i* + 1 if it is an Add-Effect of some action in level *i*.
- Two propositions *p* and *q* are *exclusive* in a proposition-level if ALL actions that add *p* are exclusive of ALL actions that add *q*.
- 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.

Mutex Exclusivity Relations
One-Way Rocket Example



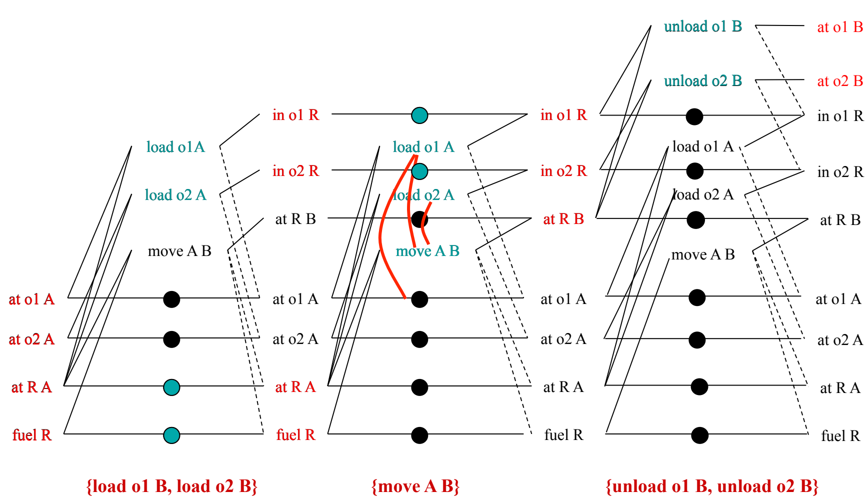
Exclusivity Examples

- Exclusive Actions: (Move A B) deletes a precondition of (Load o1 A). Therefore exclusive (existence of threats).
- Exclusive Propositions: (at R A) and (at R B) at time 2 are exclusive. (at R A) is added by a no-op and (at R B) is added by (Move A B) and no-op and (Move A B) are exclusive actions.
- Exclusive Actions: Then (Load o1 A) and (Load o2 B) are exclusive because (at R A) and (at R B) are exclusive.
- Propositions can be exclusive in some time step and not in others: If (at o1 A) and (at R A) at time 1, then (in o1 A) and (at R B) are exclusive at time 2, but not at time 3.

Searching a Planning Graph

- Level-by-level backward-chaining approach to use the exclusivity constraints.
- Given a set of goals at time *t*, identify all the sets of actions (including no-ops) at time *t* - 1 who add those goals and are not exclusive. The preconditions of these actions are new goals for *t* - 1.

Searching a Planning Graph



Recursive Search

- For each goal at time *t* in some arbitrary order:
  - Select some action at time *t* - 1 that achieves that goal and it is not exclusive with any other action already selected.
  - Do this recursively for all the goals at time *t* - do not add new action, but use the ones already selected if they add another goal.
  - If recursion returns failure, then select a different action.
- The new goal set is the set of all the preconditions of the selected actions.

Enhancements

- Forward-checking - for the goals ahead, check if all the actions that add it are exclusive with the selected action.
- Memoization - when a set of goals is not solvable at some time *t*, then this is recorded and hashed. If back at time *t*, the hash table is checked and search proceeds backing up right away.

Planning as Satisfiability

- One interpretation: ``first-order deductive theorem-proving does not scale well.' '
- One solution: ``propositional satisfiability' '
- Uniform clausal representation for goals and operators.
- Stochastic local search is a powerful technique for planning.

SatPlan

- Assume the plan has *n* (time-parallel) steps. *(strong assumption)*
- Initial state: completely specified at time 0.
  $\text{at-o1-A}_0 \wedge \text{at-o2-A}_0 \wedge \text{at-R-A}_0$
- Goal: specified at time 2*n*.
  $\text{at-o1-B}_6 \wedge \text{at-o2-B}_6$
- Actions: specified at *odd* times; An action implies its preconditions and effects.
  $(\neg\text{load-o1-A}_1 \vee \text{at-o1-A}_0) \wedge (\neg\text{load-o1-A}_1 \vee \text{at-R-A}_0) \wedge$
  $(\neg\text{load-o1-A}_1 \vee \text{in-R-A}_2) \wedge (\neg\text{load-o1-A}_1 \vee \neg\text{at-o1-A}_2)$

"FF"

- A* search with heuristic values from:
  - *Relaxed* planning graph – only add effects

Summary

- **Planning:** selecting one sequence of actions (operators) that transform (apply to) an initial state to a final state where the goal statement is true.
- **Means-ends analysis:** identify and reduce, as soon as possible, *differences* between state and goals.
- **Linear planning:** backward chaining with means-ends analysis using a stack of goals - potentially efficient, possibly unoptimal, incomplete; GPS, STRIPS.
- **Nonlinear planning with means-ends analysis:** backward chaining using a set of goals; reason about *when* "to reduce the differences;" Prodigy4.0.
- **Graphplan**
  - **Expand (forward) and search (backwards)**
- **SATPlan, FF**