Comparison of State-Space and Plan-Space Planning

Manuela Veloso

Carnegie Mellon University
Computer Science Department

Planning - Fall 2001
Outline: Selecting ACTIONS

- Planning algorithms
- Comparison of planning algorithms
- Learning in planning
- Planning, execution, and learning
- Behavior-based (reactive) planning
- Action selection in multiagent systems
Several Planning Algorithms

- TWEAK (Chapman 87), SNLP (McAllester & Rosenblitt 91), UCPOP (Penberthy and Weld 92)
- NONLIN (Tate 76), O-PLAN (Tate), SIPE (Wilkins 88)
- Prodigy2.0 (Minton et al. 87), Prodigy4.0 (Veloso et al. 90)
- UNPOP, Planning and acting (McDermott 78)
- Reactive planning (Georgeff & Lansky 87), (Firby 87), (Hendler & Sanborn 87)
- Action and time (Allen 84) (Dean & McDermott 87)
Several Planning Algorithms

- Walksat, Satplan (Selman et al. 92, Kautz & Selman 92, 96)
- Flecs (Veloso & Stone 95)
- Graphplan (Blum & Furst 95)
- MBP (Cimatti, Roveri, Traverso 98)
- UMOP (Jensen & Veloso 00)
- More at planning competitions - AIPS’98, AIPS’00
Plan-Space Partial-Order Nonlinear Planning

SNLP Planning Algorithm McAllester & Rosenblitt 91

1. Terminate if the goal set is empty.

2. Select a goal $g$ from the goal set and identify the plan step that needs it, $S_{\text{need}}$.

3. Let $S_{\text{add}}$ be a step (operator) that adds $g$, either a new step or a step that is already in the plan. Add the causal link $S_{\text{add}} \xrightarrow{g} S_{\text{need}}$, constrain $S_{\text{add}}$ to come before $S_{\text{need}}$, and enforce bindings that make $S_{\text{add}}$ add $g$. 
4. Update the goal set with all the preconditions of the step $S_{add}$, and delete $g$.

5. Identify threats and resolve the conflicts by adding ordering or bindings constraints.

- A step $S_{k}$ threatens a causal link $S_{i} \xrightarrow{g} S_{j}$ when it occurs between $S_{i}$ and $S_{j}$, and it adds or deletes $p$.

- Resolve threats by using promotion, demotion, or separation.
Plan-space Planning

- Complete, sound, and optimal.
- Optimal handling of goal orderings.
Rocket Domain - Linking

Example – LINKING
Rocket Domain - Threats

Example – THREATS
Comparison of Planning Algorithms

- Complete nonlinear state-space planning
- Plan-space planning
- Graphplan
- Satplan
- And more

Is there a universally best planning algorithm?
State-space and Plan-space

- Planning is NP-hard.

- Two different planning approaches: state-space and plan-space planning

<table>
<thead>
<tr>
<th>Commitments in plan step orderings</th>
<th>State-space</th>
<th>Plan-space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Therefore, suffer with goal orderings</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Therefore, handle goal interactions</td>
<td>Poorly</td>
<td>Efficiently</td>
</tr>
</tbody>
</table>
Step Ordering Commitments

WHY?

Use of the STATE of the world while planning

In Prodigy4.0 advantages include:

- Means-ends analysis - plan for goals that reduce the differences between current and goal states.
- Informed selection of operators - select operators that need less planning work than others.
- State useful for learning, generation and match of conditions supporting informed decisions.
- Helpful for generating anytime planning - provide valid, executable, plans at any time.
Facts and Goals

● FACTS:
  – Partial-order planners are perceived as generally more efficient than total-order planners.
  – MANY results supporting this claim.

● HOWEVER:
  – Planning as search implies necessarily a series of commitments during search.
  – Partial-order planners do search.
Facts and Goals

• GOALS:
  – Identify commitments in a partial-order planner.
  – Understand the implications of such commitments.
  – Provide clear demonstration of exemplary domains where total-order planners perform better than partial-order planners.
## Parallel between Commitments

<table>
<thead>
<tr>
<th>Operator Polish</th>
<th>Operator Drill-Hole</th>
</tr>
</thead>
<tbody>
<tr>
<td>preconds: ()</td>
<td>preconds: ()</td>
</tr>
<tr>
<td>adds: polished</td>
<td>adds: has-hole</td>
</tr>
<tr>
<td>deletes: ()</td>
<td>deletes: polished</td>
</tr>
</tbody>
</table>

**Goal:** polished and has-hole  
**Initial state:** empty

- plan for goal polished  
  - select Polish  
    - order Polish as first step  
  - plan for goal has-hole  
  - select Drill-Hole  
    - order Drill-Hole ≻ Polish  
      - polished deleted, backtrack  
    - Polish ≻ Drill-Hole

**Goal:** polished and has-hole  
**Initial state:** polished

- plan for goal polished  
  - select Initial state  
    - link Initial to polished  
  - plan for goal has-hole  
  - select Drill-Hole  
    - link Drill-Hole to has-hole  
    - threat - relink polished  
      - select Polish  
        - link Polish to polished  
        - Polish ≻ Drill-Hole
Serializability and Linkability

• A set of subgoals is serializable (Korf):
  – If there exists some ordering whereby they can be solved sequentially,
  – without ever violating a previously solved subgoal.

• Easily serializable, laboriously serializable

• A set of subgoals is easily linkable:
  – If, independently of the order by which the planner links these subgoals to operators,
  – it never has to undo those links.
  – Otherwise it is laboriously linkable.
Easily Linkable Goals

operator $A_i$
preconds ()
adds $g_i$
deletes ()

operator $A_*$
preconds ()
adds $g_*$
deletes $g_i, \forall i$

Initial state: $g_1, g_2, g_3, g_4, g_5$
Goal statement: $g_2, g_5, g_4, g_*, g_3, g_1$
Plan: $A_*, A_2, A_5, A_4, A_3, A_1$
Easily Linkable Goals

![Graph showing the relationship between time in milliseconds and the number of goals for prodigy4.0 and snlp.](image)

- **prodigy4.0**
- **snlp**
**Laboriously Linkable Goals**

<table>
<thead>
<tr>
<th>Operator $A_i$</th>
<th>Operator $A_*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconds $g_*, g_{i-1}$</td>
<td>Preconds $()$</td>
</tr>
<tr>
<td>Adds $g_i$</td>
<td>Adds $g_*$</td>
</tr>
<tr>
<td>Deletes $g_*$</td>
<td>Deletes $()$</td>
</tr>
</tbody>
</table>

Initial state: $g_*$
Goal statement: $g_*, g_5$
Plan: $A_1, A_*, A_2, A_*, A_3, A_*, A_4, A_*, A_5, A_*$
Laboriously Linkable Goals

- prodigy4.0
- snlp

Time in msecs vs. Highest goal
Multiple Linking Alternatives

operator $A_i$
preconds $g_j, \forall j < i$
adds $g_i, g_j, \forall j < i - 1$
deletes $g_{i-1}$

operator $A_5$
pre $g_4, g_3, g_2, g_1$
add $g_5, g_3, g_2, g_1$
del $g_4$

operator $A_4$
pre $g_3, g_2, g_1$
add $g_4, g_2, g_1$
del $g_3$

operator $A_3$
pre $g_2, g_1$
add $g_3, g_1$
del $g_2$

Initial state: $g_1, g_2, g_3, g_4$
Goal statement: $g_2, g_5, g_4, g_3, g_1$
Plan: $A_5, A_4, A_3, A_2, A_1$
Empirical Results - Multiple Linking

![Graph showing the relationship between time in milliseconds and the number of goals for prodigy4.0 and snlp.](image)

Veloso, Carnegie Mellon

15-889 – Fall 2001
Summary – Comparison of Planners

• Similar empirical comparison results for other planning algorithms (we’ll see later).

• There is not a planning strategy that is universally better than the others.

• Even for a particular planning algorithm: There is no single domain-independent search heuristic that performs more efficiently than others for all problems or in all domains.

Learning is challenging and appropriate for ANY planner.