Nonlinear State-Space Planning:
Prodigy 4.0

Manuela Veloso

Carnegie Mellon University
Computer Science Department

Planning - Fall 2001
Planning - Problem Solving

Newell and Simon 1956

- Given the *actions* available in a task domain.

- Given a problem specified as:
  - an initial *state* of the world,
  - a set of *goals* to be achieved.

- Find a *solution* to the problem, i.e., a way to transform the initial state into a new state of the world where the goal statement is true.

Action Model, State, Goals
Classical Deterministic Planning

- Action Model: complete, deterministic, correct, rich representation
- State: single initial state, fully known
- Goals: complete satisfaction

Several different planning algorithms
Many Planning “Domains”

- Web management agents
- Robot planning
- Manufacturing planning
- Image processing management
- Logistics transportation
- Crisis management
- Bank risk management
- Blocksworld
- Puzzles
- Artificial domains
**Example - Action Model**

**DRILL PRESS**

- **<drill-bit>**
- **<part>**

**TYPE HIERARCHY**

- **PART**
- **DRILL-BIT**
- **SPOT-DRILL**
- **TWIST-DRILL**

---

**drill-spot (<part>, <drill-bit>)**
- **<part>: type PART**
- **<drill-bit>: type SPOT-DRILL**
- **Pre:** (holding-tool <drill-bit>)
  (holding-part <part>)
- **Add:** (has-spot <part>)

---

**put-drill-bit (<drill-bit>)**
- **<drill-bit>: type DRILL-BIT**
- **Pre:** tool-holder-empty
- **Add:** (holding-tool <drill-bit>)
- **Del:** tool-holder-empty

---

**put-part(<part>)**
- **<part>: type PART**
- **Pre:** part-holder-empty
- **Add:** (holding-part <drill-bit>)
- **Del:** part-holder-empty

---

**drill-hole(<part>, <drill-bit>)**
- **<part>: type PART**
- **<drill-bit>: type TWIST-DRILL**
- **Pre:** (has-spot <part>)
  (holding-tool <drill-bit>)
  (holding-part <part>)
- **Add:** (has-hole <part>)

---

**remove-drill-bit(<drill-bit>)**
- **<drill-bit>: type DRILL-BIT**
- **Pre:** (holding-tool <drill-bit>)
- **Add:** tool-holder-empty
- **Del:** (holding-tool <drill-bit>)

---

**remove-part(<part>)**
- **<part>: type PART**
- **Pre:** (holding-part <drill-bit>)
- **Add:** part-holder-empty
- **Del:** (holding-part <drill-bit>)

---

*Veloso, Carnegie Mellon*
## Example - Problem and Plan

**Initial State**

- part-holder-empty
- drill-holder-empty

**Set of Objects**

- part-1, part-2 : type PART
- drill-1: type SPOT-DRILL
- drill-2, drill-3 : type TWIST-DRILL

**Goal Statement**

- has-hole (part-1)

**Plan**

1. `put-part(part-1)`
2. `put-drill-bit(drill-1)`
3. `drill-spot(part-1, drill-1)`
4. `remove-drill-bit(drill-1)`
5. `put-drill-bit(drill-2)`
6. `drill-hole(part-1, drill-2)`
Generating a Solution Plan

- Linear planning – Planning with a goal stack.
- Nonlinear planning – Interleaving of goals
  - State-space search
  - Plan-space search
  - Graph-based search
  - Sat-based search
  - OBDD-based search
- Hierarchical planning
  - Emphasis on action decomposition/refinement
  - Knowledge engineering/acquisition
  - Very little search
Generating a Solution Plan

A complex process:

- Alternative operators to achieve a goal.
- Multiple goals that interact.
- Efficiency and plan quality.
Means-ends Analysis

(Newell and Simon 60s)

**GPS Algorithm** *(initial-state, goals)*

- If *goals* $\subseteq$ *initial-state*, then return *True*
- Choose a difference $d \in$ *goals* between *initial-state* and *goals*
- Choose an operator $o$ to reduce the difference $d$
- If no more operators, then return *False*
- $State = \text{GPS}(\text{initial-state}, \text{preconditions}(o))$
- If $State$, then return $\text{GPS}(\text{apply}(o, \text{initial-state}), \text{goals})$
Example: One-Way Rocket (Veloso 89)

(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))
Incompleteness of Linear Planning

Initial state:  
- (at obj1 locA)
- (at obj2 locA)
- (at ROCKET locA)
- (has-fuel ROCKET)

Goal statement:  
- (and
  - (at obj1 locB)
  - (at obj2 locB))

<table>
<thead>
<tr>
<th>Goal</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>(at obj1 locB)</td>
<td>(LOAD-ROCKET obj1 locA)</td>
</tr>
<tr>
<td></td>
<td>(MOVE-ROCKET)</td>
</tr>
<tr>
<td></td>
<td>(UNLOAD-ROCKET obj1 locB)</td>
</tr>
<tr>
<td>(at obj2 locB)</td>
<td>failure</td>
</tr>
</tbody>
</table>
State-Space Nonlinear Planning

Extend linear planning:

- From **stack** to **set** of goals.
- Include in the search space all possible interleaving of goals.

State-space nonlinear planning is **complete**.

<table>
<thead>
<tr>
<th>Goal</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>(at obj1 locB)</td>
<td>(LOAD-ROCKET obj1 locA)</td>
</tr>
<tr>
<td>(at obj2 locB)</td>
<td>(LOAD-ROCKET obj2 locA)</td>
</tr>
<tr>
<td>(at obj1 locB)</td>
<td>(MOVE-ROCKET)</td>
</tr>
<tr>
<td>(at obj2 locB)</td>
<td>(UNLOAD-ROCKET obj1 locB)</td>
</tr>
<tr>
<td>(at obj2 locB)</td>
<td>(UNLOAD-ROCKET obj2 locB)</td>
</tr>
</tbody>
</table>
1. Terminate if the goal statement is satisfied in the current state.

2. Compute the SET of pending goals $G$, and the set of applicable operators $A$.

- A goal is pending if it is a precondition, not satisfied in the current state, of an operator already in the plan.
- An operator is applicable when all its preconditions are satisfied in the state.
3. Choose a goal $G$ in $G$ or choose an operator $A$ in $A$.

4. If $G$ has been chosen, then

- **Expand goal $G$,**
  i.e., get the set $\mathcal{O}$ of *relevant instantiated operators* that could achieve the goal $G$,
- Choose an operator $O$ from $\mathcal{O}$,
- Go to step 1.

5. If an operator $A$ has been selected as directly applicable, then

- **Apply $A$,**
- Go to step 1.
Adding an operator to the tail-plan

Applying an operator (moving it to the head)

Adding an operator to the tail-plan
### The Need for “Apply/Subgoal”

<table>
<thead>
<tr>
<th>pre</th>
<th>OP1</th>
<th>OP2</th>
<th>OP3</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>g1</td>
<td>g11</td>
<td>g2</td>
</tr>
<tr>
<td>del</td>
<td>p</td>
<td>g2</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>prob1</th>
<th>prob2</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>g2, p</td>
</tr>
<tr>
<td>Goal</td>
<td>g1, g2</td>
</tr>
<tr>
<td>Plan</td>
<td>OP2, OP3, OP1</td>
</tr>
</tbody>
</table>

**USER(4): (run ’prob1)**

4 n4 <*finish*>  
5 n5 (g1)  
7 n7 <op1>  
8 n8 (g11)  
10 n10 <op2>  
11 n11 <OP2>  
12 n12 <OP1>  
13 n13 (g2)  
15 n15 <op3>  
16 n16 (p) ...no ops.  
11 n11 <OP2> ...no goals.  
#<PRODIGY: NIL, 0.0 s, 16 nodes>

**USER(4): (run ’prob2)**

4 n4 <*finish*>  
5 n5 (g1) [(g2)]  
7 n7 <op1>  
8 n8 (g11) [(g2)]  
10 n10 <op2>  
11 n11 <OP2> [(g2)]  
12 n12 <OP1>  
13 n13 (g2)  
15 n15 <op3>  
16 n16 (p) ...no ops.  
.......backtracking.......  
Solution: <op2> <op3> <op1>  
#<PRODIGY: T, 0.05 s, 43 nodes>
# Incompleteness of MEA in Prodigy

<table>
<thead>
<tr>
<th></th>
<th>OP1</th>
<th>OP2</th>
<th>OP3</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre</td>
<td>–</td>
<td>g3</td>
<td>g4</td>
</tr>
<tr>
<td>add</td>
<td>g1</td>
<td>g4</td>
<td>g2</td>
</tr>
<tr>
<td>del</td>
<td>g2, g3</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

**Problem:**

- **Initial state:** g2, g3
- **Goal:** g1, g2
- **Plan:** OP2, OP1, OP3

<table>
<thead>
<tr>
<th>Choice/action</th>
<th>Choice made</th>
<th>Other choices/result</th>
</tr>
</thead>
<tbody>
<tr>
<td>goal</td>
<td>g1</td>
<td>–; no other goals using MEA</td>
</tr>
<tr>
<td>op</td>
<td>OP1</td>
<td>–</td>
</tr>
<tr>
<td>ap/subg</td>
<td>ap</td>
<td>–; no pending goals (g2 in state)</td>
</tr>
<tr>
<td>APPLY</td>
<td>OP1</td>
<td>–; new state = g1</td>
</tr>
<tr>
<td>goal</td>
<td>g2</td>
<td>–</td>
</tr>
<tr>
<td>op</td>
<td>OP3</td>
<td>–</td>
</tr>
<tr>
<td>ap/subg</td>
<td>subg</td>
<td>–; no applicable op (OP3 needs g4)</td>
</tr>
<tr>
<td>goal</td>
<td>g4</td>
<td>–</td>
</tr>
<tr>
<td>op</td>
<td>OP2</td>
<td>–</td>
</tr>
<tr>
<td>ap/subg</td>
<td>subg</td>
<td>–; no applicable op (OP2 needs g3)</td>
</tr>
<tr>
<td>goal</td>
<td>g3</td>
<td>–; failure - end (no backtracking open)</td>
</tr>
<tr>
<td>op</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Veloso, Carnegie Mellon 15-889 – Fall 2001
Planning Choices

- Planning as search, i.e., a decision-making process – learn search heuristics
Control Rules - Heuristic to Guide Search

**Select Rule**

If (has-spot <part>) is the current goal
   and drill-spot (<part>, <drill>) is the current operator
   and (holding-drill-bit <drill-1>) holds in the current state
   and <drill-1> is of the type SPOT-DRILL
Then select instantiating <drill> with <drill-1>

**Prefer Rule**

If (has-hole <part-1>) is a candidate goal
   and (has-hole <part-2>) is a candidate goal
   and (holding-part <part-1>) holds in the current state
Then prefer the goal (has-hole <part-1>) to (has-hole <part-2>)
Why Ordering Commitments?

In PRODIGY:

Use of a unique world STATE while planning.

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 is useful for learning, generation and match of conditions supporting informed decisions.

• State is helpful for generating anytime planning - provide valid, executable plans at any time.

• Probabilistic planning - may be useful to reason about states, events that affect them, and eventual transitions.
The Importance of Step 3: Apply or Subgoal?

- **Step 3**: Prodigy’s main search can be captured by the regular expression \((\text{Subgoal Apply}^*)^*\).

- Prodigy uses **state** to determine . . .
  - if the goal state has been reached (**step 1**).
  - which goals still need to be achieved (**step 2**).
  - which operators are applicable (**step 2**).
  - which operators to try first while planning (**step 4**).
Two Heuristics: SAVTA, SABA

SAVTA: Eager application = **Eager** state changes

**Subgoal After eVery Try to Apply**

SABA: Eager subgoaling = **Delayed** state changes

**Subgoal Always Before Applying**

Veloso, Carnegie Mellon
1. Compute $G$, set of goals, to plan for:
   - $C$ - current state,
   - $O$ - operators selected,
   - $P$ - unplanned preconditions of operators in $O$,
   - then $G = P - C$ means-ends analysis

2. Succeed and terminate if $G$ is empty.

3. Choose a goal $g$ from $G$ to plan for.

4. Choose an instantiated operator $O$ to achieve $g$. Add $O$ to $O$.

5. Apply any applicable operator, i.e., $A$ in $O$ with preconditions satisfied in $C$. Update $C$. 

Veloso, Carnegie Mellon 15-889 – Fall 2001
1. Compute $G$ - set of goals to plan for:
   - $C$ - current state,
   - $O$ - operators selected,
   - $P$ - unplanned preconditions of operators in $O$,
   - then $G = P - C$ means-ends analysis

2. If $G$ is empty, then go to 5.

3. Choose a goal $g$ from $G$ to plan for.

4. Choose an instantiated operator $O$ to achieve $g$. Add $O$ to $O$.

5. If there are no applicable operators, succeed. Otherwise, compute the set of applicable operators, the operators $O$ with preconditions satisfied in $C$.

6. Select an applicable operator, taking into account the interactions among the preconditions and effects of the set of applicable operators. Go to step 1.
Eagerly Subgoaling Can Be Better

Operator: \( A_i \)

preconds: \( \{ I_i \} \)

adds: \( \{ G_i \} \)

deletes: \( \{ I_j | j < i \} \)

Example:

- Initial state: I1, I2, I3
- Goal: G2, G3, G1
- Plan: A1, A2, A3
### Eagerly Subgoaling Can Be Better

<table>
<thead>
<tr>
<th>Op:</th>
<th>paint-white &lt;obj&gt;</th>
<th>paint-yellow &lt;obj&gt;</th>
<th>...</th>
<th>paint-black &lt;obj&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre:</td>
<td>(usable white)</td>
<td>(usable yellow)</td>
<td>...</td>
<td>(usable black)</td>
</tr>
<tr>
<td>add:</td>
<td>(white &lt;obj&gt;)</td>
<td>(yellow &lt;obj&gt;)</td>
<td>...</td>
<td>(black &lt;obj&gt;)</td>
</tr>
<tr>
<td>del:</td>
<td></td>
<td>(usable white)</td>
<td>...</td>
<td>(usable white)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td>(usable yellow)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td>(usable brown)</td>
</tr>
</tbody>
</table>
Operator: $A_i$
preconds: $\{I_i\}$
adds: $\{< g >\}$
deletes: $\{I_i\}$

Note that each operator adds any goal ($\{< g >\}$ is a variable), but each operator can only be used ONCE.
Eagerly Applying Can Be Better

<table>
<thead>
<tr>
<th>Op:</th>
<th>paint-with-brush1</th>
<th>...</th>
<th>paint-with-brush8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;parts&gt; &lt;color&gt;</td>
<td>...</td>
<td>&lt;parts&gt; &lt;color&gt;</td>
</tr>
<tr>
<td>pre:</td>
<td>(unused brush1)</td>
<td>...</td>
<td>(unused brush8)</td>
</tr>
<tr>
<td>add:</td>
<td>(painted &lt;parts&gt; &lt;color&gt;)</td>
<td>...</td>
<td>(painted &lt;parts&gt; &lt;color&gt;)</td>
</tr>
<tr>
<td>del:</td>
<td>(unused brush1)</td>
<td>...</td>
<td>(unused brush8)</td>
</tr>
<tr>
<td>Op</td>
<td>Designate-Roller</td>
<td>Fill-Roller</td>
<td>Paint-Wall</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------</td>
<td>---------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td></td>
<td>$\langle wall \rangle \langle roller \rangle \langle color \rangle$</td>
<td>$\langle roller \rangle \langle color \rangle$</td>
<td>$\langle wall \rangle \langle roller \rangle \langle color \rangle$</td>
</tr>
<tr>
<td>pre:</td>
<td>(clean $\langle roller \rangle$)</td>
<td>(clean $\langle roller \rangle$)</td>
<td>(ready $\langle wall \rangle \langle roller \rangle \langle color \rangle$)</td>
</tr>
<tr>
<td></td>
<td>(needs-painting $\langle wall \rangle$)</td>
<td>(chosen $\langle roller \rangle \langle color \rangle$)</td>
<td>(filled-with-paint $\langle roller \rangle \langle color \rangle$)</td>
</tr>
<tr>
<td>add:</td>
<td>(ready $\langle wall \rangle \langle roller \rangle \langle color \rangle$)</td>
<td>(filled-with-paint $\langle roller \rangle \langle color \rangle$)</td>
<td>(painted $\langle wall \rangle \langle color \rangle$)</td>
</tr>
<tr>
<td></td>
<td>(chosen $\langle roller \rangle \langle color \rangle$)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>del:</td>
<td></td>
<td>(clean $\langle roller \rangle$)</td>
<td>(ready $\langle wall \rangle \langle roller \rangle \langle color \rangle$)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(needs-painting $\langle wall \rangle$)</td>
</tr>
</tbody>
</table>
FLECS: An Intermediate Heuristic

<table>
<thead>
<tr>
<th>Initial State</th>
<th>Goal Statement</th>
<th>An Optimal Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>(needs-painting wallA)</td>
<td>(painted wallA red)</td>
<td>&lt;Designate-Roller wallA roller1 red&gt;</td>
</tr>
<tr>
<td>(needs-painting wallB)</td>
<td>(painted wallB red)</td>
<td>&lt;Designate-Roller wallB roller1 red&gt;</td>
</tr>
<tr>
<td>(needs-painting wallC)</td>
<td>(painted wallC red)</td>
<td>&lt;Designate-Roller wallC roller1 red&gt;</td>
</tr>
<tr>
<td>(needs-painting wallD)</td>
<td>(painted wallD green)</td>
<td>&lt;Fill-Roller roller1 red&gt;</td>
</tr>
<tr>
<td>(needs-painting wallE)</td>
<td>(painted wallE green)</td>
<td>&lt;Paint-Wall wallA roller1 red&gt;</td>
</tr>
<tr>
<td>(clean roller1)</td>
<td></td>
<td>&lt;Paint-Wall wallB roller1 red&gt;</td>
</tr>
<tr>
<td>(clean roller2)</td>
<td></td>
<td>&lt;Paint-Wall wallC roller1 red&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Designate-Roller wallD roller2 green&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Designate-Roller wallE roller2 green&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Fill-Roller roller2 green&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Paint-Wall wallD roller2 green&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Paint-Wall wallE roller2 green&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>time(sec)</th>
<th>solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>eager applying</td>
<td>500</td>
<td>no</td>
</tr>
<tr>
<td>eager subgoaling</td>
<td>500</td>
<td>no</td>
</tr>
<tr>
<td>variable strategy</td>
<td>4</td>
<td>yes</td>
</tr>
</tbody>
</table>
**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.

- **Planning as search**: control rules to capture heuristics for efficient search; learning opportunities.