Nonlinear State-Space Planning: Prodigy4.0

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

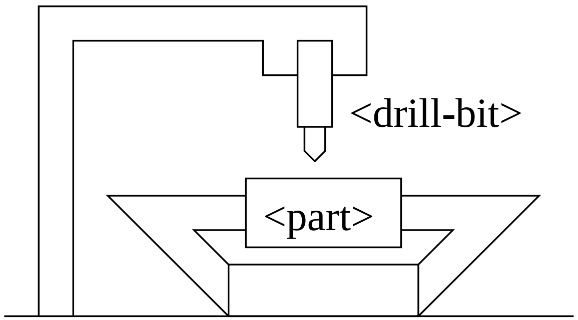
Several different planning algorithms

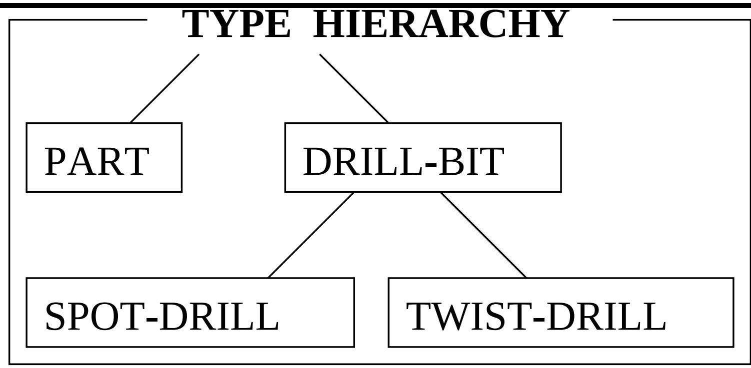
Many Planning "Domains"

- 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-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

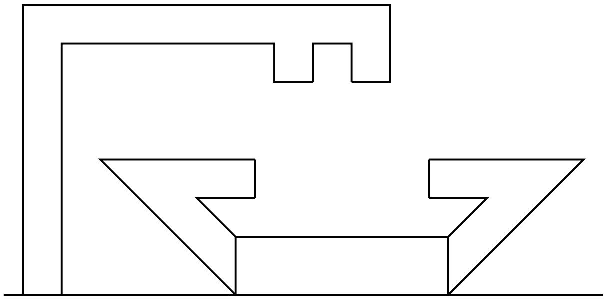
drill-hole(<part>, <drill-bit>)
<part>: type PART
<drill-bit>: type TWIST-DRILL
Pre: (has-spot <part>)
(holding-tool <drill-bit>)
(holding-part <part>)
Add: (has-hole <part>)

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>)

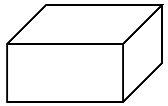
Example - Problem and Plan

Initial State



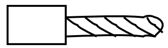
part-holder-empty
drill-holder-empty

Set of Objects



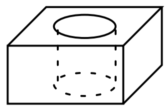
part-1, part-2 : type PART

drill-1: type SPOT-DRILL



drill-2, drill-3 : type TWIST-DRILL

Goal Statement



has-hole (part-1)

```
put-part(part-1)
put-drill-bit(drill-1)
drill-spot(part-1, drill-1)
remove-drill-bit(drill-1)
put-drill-bit(drill-2)
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*=**GPS**(*initial-state*, preconditions(o))
- If *State*, then return **GPS**(apply(o,*initial-state*), *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))

Goal	*Plan*
(at obj1 locB)	(LOAD-ROCKET obj1 locA) (MOVE-ROCKET) (UNLOAD-ROCKET obj1 locB)
(at obj2 locB)	*failure*

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**.

Goal	*Plan*
(at obj1 locB)	(LOAD-ROCKET obj1 locA)
(at obj2 locB)	(LOAD-ROCKET obj2 locA)
 (at obj1 locB)	(MOVE-ROCKET) (UNLOAD-ROCKET obj1 locB)
(at obj2 locB)	(UNLOAD-ROCKET obj2 locB)

Prodigy4.0 *(Veloso et al. 90)*

1. Terminate if the goal statement is satisfied
   in the current state.

2. Compute the **SET** of *pending goals* $\mathcal{G}$,
   and the **set** of *applicable operators* $\mathcal{A}$.

   - 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 $\mathcal{G}$ or choose an operator A in $\mathcal{A}$.

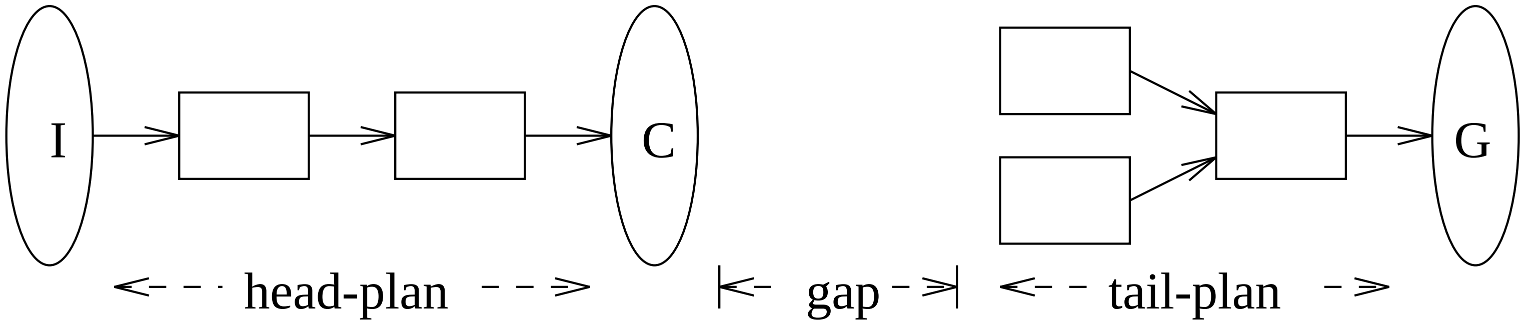
4. If G has been chosen, then

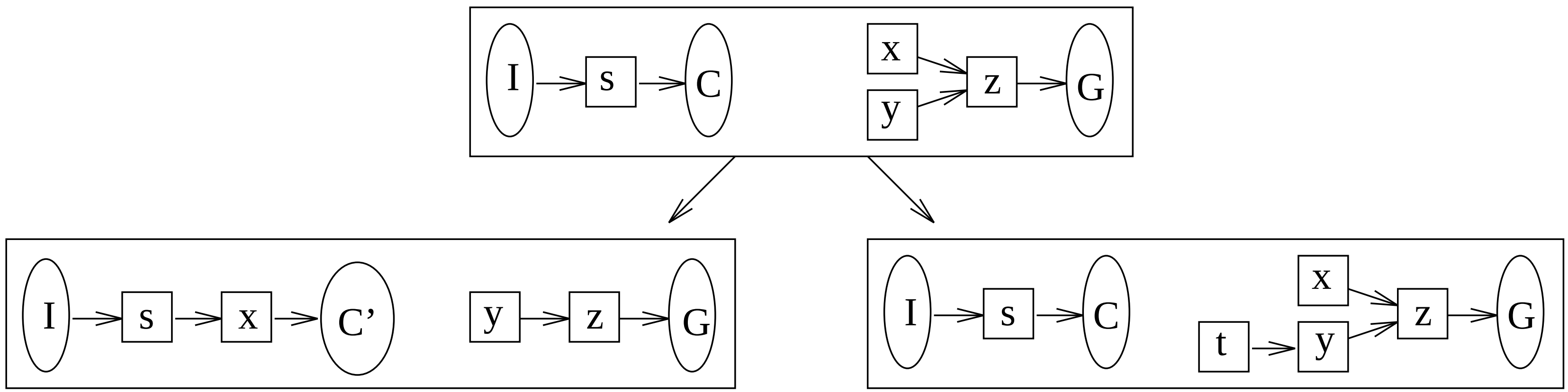
   - *Expand goal* G,
     i.e., get the set $\mathcal{O}$ of *relevant instantiated operators* that could achieve the goal G,
   - Choose an operator O from $\mathcal{O}$,
   - Go to step 1.

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

   - *Apply* A,
   - Go to step 1.

Prodigy4.0 - Search Representation





Applying an operator (moving it to the head)

Adding an operator to the tail-plan

The Need for "Apply/Subgoal"

	OP1	OP2	OP3
pre	g11	–	p
add	g1	g11	g2
del	p	g2	–

	prob1	prob2
State	g2, p	p
Goal	g1, g2	g1, g2
Plan	OP2, OP3, OP1	

```
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

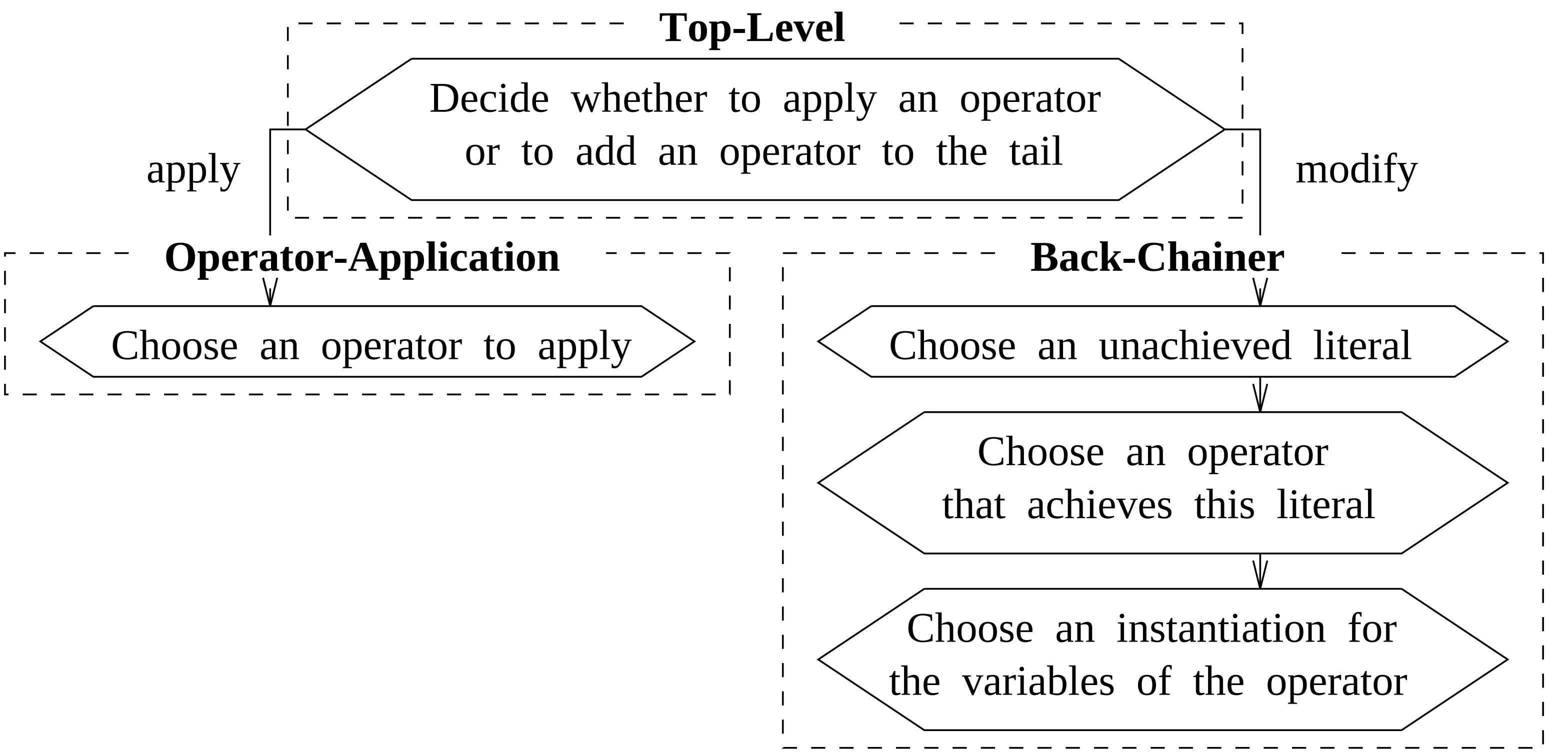
	OP1	OP2	OP3
pre	–	g3	g4
add	g1	g4	g2
del	g2, g3	–	–

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

Prodigy4.0 search with MEA on:

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

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 instantianting <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 (**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**).

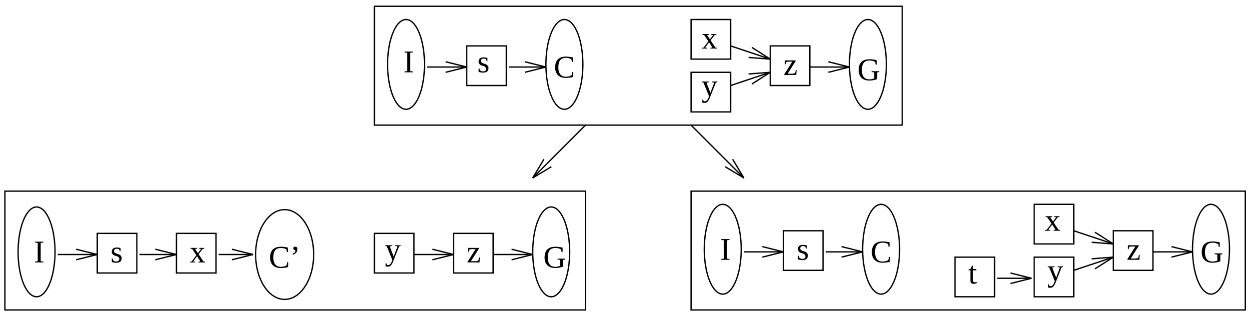
Two Heuristics: SAVTA, SABA

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

Subgoal **A**fter e**V**ery **T**ry to **A**pply

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

Subgoal **A**lways **B**efore **A**pplying



APPLYING - CHANGING THE STATE

SUBGOALING - CONTINUE PLANNING

SAVTA - Eager Applying

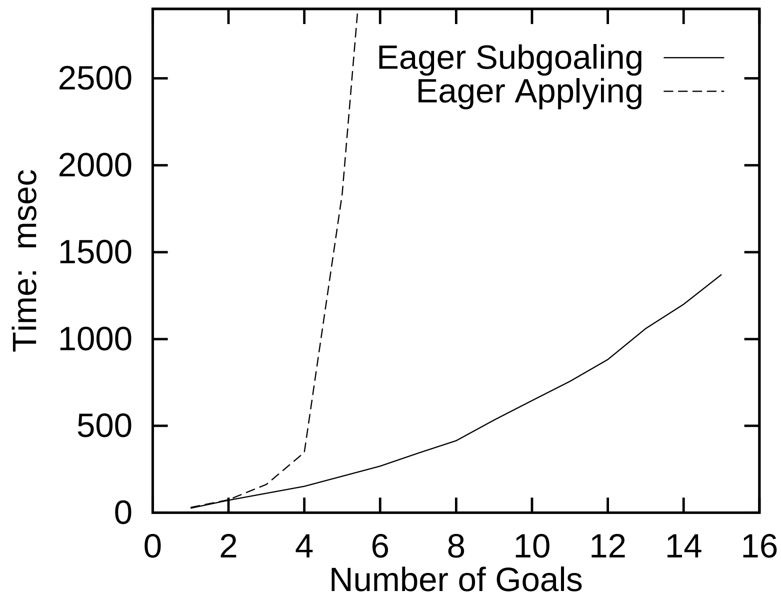
1. Compute $\mathcal{G}$, set of goals, to plan for:
   - $\mathcal{C}$ - current state,
   - $\mathcal{O}$ - operators selected,
   - $\mathcal{P}$ - unplanned preconditions of operators in $\mathcal{O}$,
   - then $\mathcal{G} = \mathcal{P} - \mathcal{C}$ *means-ends analysis*
2. Succeed and terminate if $\mathcal{G}$ is empty.
3. Choose a goal g from $\mathcal{G}$ to plan for.
4. Choose an instantiated operator O to achieve g. Add O to $\mathcal{O}$.
5. Apply any applicable operator, i.e., A in $\mathcal{O}$ with preconditions satisfied in $\mathcal{C}$. Update $\mathcal{C}$.

SABA - Eager Subgoaling

1. Compute $\mathcal{G}$ - set of goals to plan for:
   - $\mathcal{C}$ - current state,
   - $\mathcal{O}$ - operators selected,
   - $\mathcal{P}$ - unplanned preconditions of operators in $\mathcal{O}$,
   - then $\mathcal{G} = \mathcal{P} - \mathcal{C}$ *means-ends analysis*
2. If $\mathcal{G}$ is empty, then go to 5.
3. Choose a goal g from $\mathcal{G}$ to plan for.
4. Choose an instantiated operator O to achieve g. Add O to $\mathcal{O}$.
5. If there are no applicable operators, succeed. Otherwise, compute the set of applicable operators, the operators $\mathcal{O}$ with preconditions satisfied in $\mathcal{C}$.
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

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



Example:

- Initial state: I1, I2, I3
- Goal: G2, G3, G1
- Plan: A1, A2, A3

Eagerly Subgoaling Can Be Better

Op:	**paint-white** <obj>	**paint-yellow** <obj>	· · ·	**paint-black** <obj>
pre:	(usable white)	(usable yellow)	· · ·	(usable black)
add:	(white <obj>)	(yellow <obj>)	· · ·	(black <obj>)
del:		(usable white)	· · · · · · ⋮	(usable white) (usable yellow) ⋮ (usable brown)

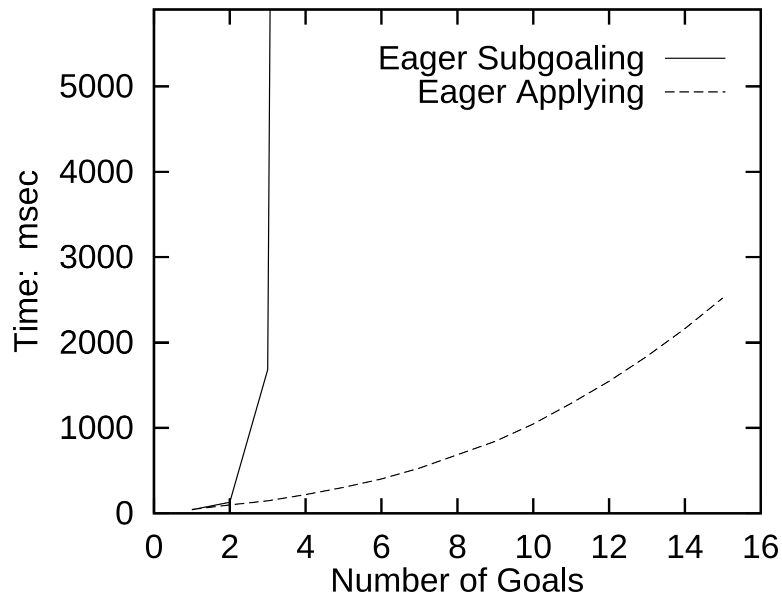
Eagerly Applying Can Be Better

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

Op:	**paint-with-brush1** <parts> <color>	. . . . . .	**paint-with-brush8** <parts> <color>
pre:	(unused brush1)	. . .	(unused brush8)
add:	(painted <parts> <color>)	. . .	(painted <parts> <color>)
del:	(unused brush1)	. . .	(unused brush8)

FLECS: An Intermediate Heuristic

Op:	**Designate-Roller** <wall> <roller> <color>	**Fill-Roller** <roller> <color>	**Paint-Wall** <wall> <roller> <color>
pre:	(clean <roller>) (needs-painting <wall>)	(clean <roller>) (chosen <roller> <color>)	(ready <wall> <roller> <color>) (filled-with-paint <roller> <color>)
add:	(ready <wall> <roller> <color>) (chosen <roller> <color>)	(filled-with-paint <roller> <color>)	(painted <wall> <color>)
del:		(clean <roller>)	(ready <wall> <roller> <color>) (needs-painting <wall>)

FLECS: An Intermediate Heuristic

Initial State

(needs-painting wallA)
(needs-painting wallB)
(needs-painting wallC)
(needs-painting wallD)
(needs-painting wallE)
(clean roller1)
(clean roller2)

Goal Statement

(painted wallA red)
(painted wallB red)
(painted wallC red)
(painted wallD green)
(painted wallE green)

An Optimal Solution

<Designate-Roller wallA roller1 red>
<Designate-Roller wallB roller1 red>
<Designate-Roller wallC roller1 red>
<Fill-Roller roller1 red>
<Paint-Wall wallA roller1 red>
<Paint-Wall wallB roller1 red>
<Paint-Wall wallC roller1 red>
<Designate-Roller wallD roller2 green>
<Designate-Roller wallE roller2 green>
<Fill-Roller roller2 green>
<Paint-Wall wallD roller2 green>
<Paint-Wall wallE roller2 green>

	time(sec)	solution
eager applying	500	no
eager subgoaling	500	no
variable strategy	4	yes

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.