

Linear and Nonlinear State-Space Planning

Manuela Veloso
Reid Simmons

PEL, Fall 2008

September 17, 2008

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.

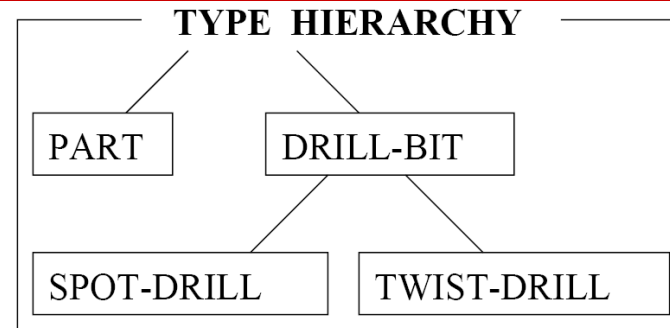
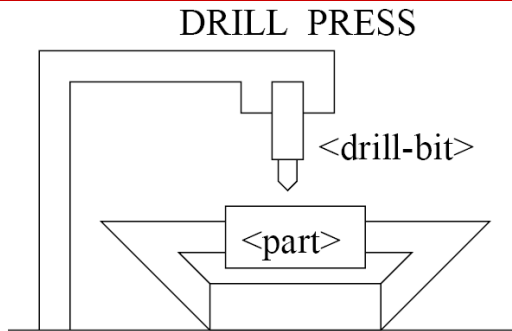
Action Model, State, Goals

Classical Deterministic Planning

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

Several different planning algorithms

Example – Action Model



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

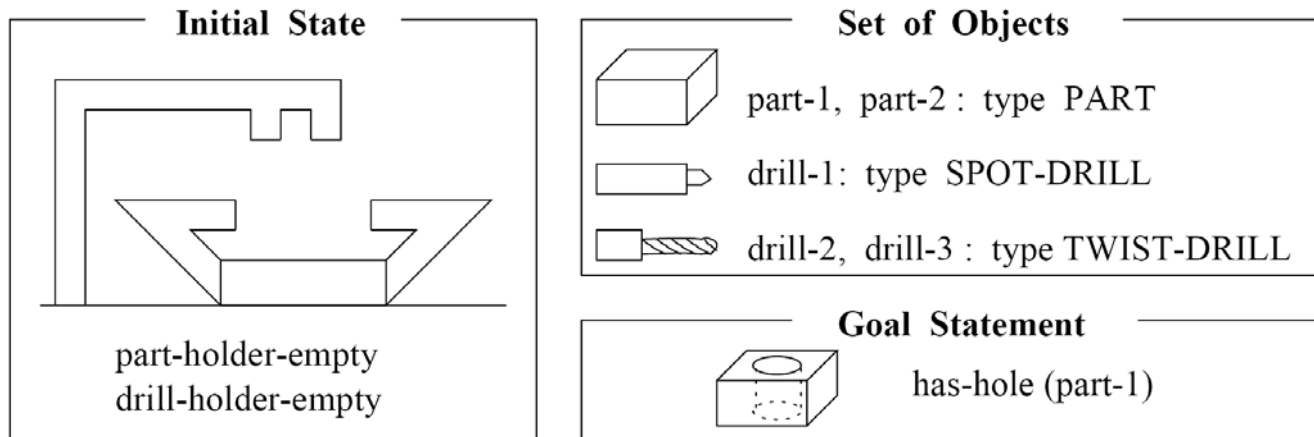
put-part(<part>)
 <part>: type PART
Pre: part-holder-empty
Add: (holding-part <drill-bit>)
Del: part-holder-empty

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



```
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

A complex process:

- **Alternative operators to achieve a goal**
- **Multiple goals that interact**
- **Efficiency and plan quality**

Linear Planning

- **Basic Idea – Goal stack**
 - *Work on one goal until completely solved before moving on to the next goal*

Means-Ends Analysis

- Basic Idea
 - *Search by reducing the difference between the state and the goals*
 - What **means** (operators) are available to achieve the desired **ends** (goal)

Means-ends Analysis in Linear Planning

(Newell and Simon 60s)

GPS Algorithm (*state*, *goals*, *plan*)

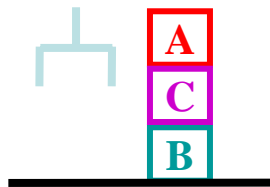
- If $goals \subseteq state$, then return (*state*, *plan*)
- Choose a difference $d \in goals$ between *state* and *goals*
- Choose an operator o to reduce the difference d
- If no applicable operators, then return *False*
- (*state*, *plan*) = **GPS** (*state*, $preconditions(o)$, *plan*)
- If *state*, then return **GPS** ($apply(o, state)$, *goals*, [*plan*, o])

Initial call: GPS (*initial-state*, *initial-goals*, [])

GPS Blocks-World Example

1. Search Stack

On(A, C) On(C, B)



Goal

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty



Initial State

2. Search Stack

On(A, C) On(C, B)

On(A, C)

On(C, B)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty

3. Search Stack

On(A, C) On(C, B)

On(A, C)

Put_Block(C, B)

Holding(C) Clear(B)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty

4. Search Stack

On(A, C) On(C, B)

On(A, C)

Put_Block(C, B)

Holding(C) Clear(B)

Holding(C)

Clear(B)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty

GPS Blocks-World Example

5. Search Stack

On(A, C) On(C, B)

On(A, C)

Put_Block(C, B)

Holding(C) Clear(B)

Holding(C)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty

6. Search Stack

On(A, C) On(C, B)

On(A, C)

Put_Block(C, B)

Holding(C) Clear(B)

Pick_Block(C)

Handempty Clear(C) On(C, ?b)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty

7. Search Stack

On(A, C) On(C, B)

On(A, C)

Put_Block(C, B)

Holding(C) Clear(B)

Pick_Block(C)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty

8. Search Stack

On(A, C) On(C, B)

On(A, C)

Put_Block(C, B)

Holding(C) Clear(B)

[Pick_Block(C)]

State

Clear(B)
Clear(C)
On(A, Table)
On(B, Table)
Holding(C)
Clear(A)

GPS Blocks-World Example

9. Search Stack

On(A, C) On(C, B)

On(A, C)

Put_Block(C, B)

State

Clear(B)
 Clear(C)
 On(A, Table)
 On(B, Table)
Holding(C)
 Clear(A)

[Pick_Block(C)]

10. Search Stack

On(A, C) On(C, B)

On(A, C)

State

Clear(C)
 On(A, Table)
 On(B, Table)
 Clear(A)
 Handempty
 On(C, B)

[Pick_Block(C); Put_Block(C, B)]

11. Search Stack

On(A, C) On(C, B)

Put_Block(A, C)

Holding(A) Clear(C)

State

Clear(C)
 On(A, Table)
 On(B, Table)
 Clear(A)
 Handempty
 On(C, B)

[Pick_Block(C)

Put_Block(C, B)]

Simmons, Veloso, Carnegie Mellon

12. Search Stack

On(A, C) On(C, B)

Put_Block(A, C)

Holding(A) Clear(C)

Holding(A)

Clear(C)

State

Clear(C)
 On(A, Table)
 On(B, Table)
 Clear(A)
 Handempty
 On(C, B)

[Pick_Block(C)

12 Put_Block(C, B)]

15-887/16-830

Fall 2008

GPS Blocks-World Example

13. Search Stack

On(A, C) On(C, B)
 Put_Block(A, C)
 Holding(A) Clear(C)
 Holding(A)

State

Clear(C)
 On(A, Table)
 On(B, Table)
 Clear(A)
 Handempty
 On(C, B)

[Pick_Block(C);
 Put_Block(C, B)]

15. Search Stack

On(A, C) On(C, B)
 Put_Block(A, C)
 Holding(A) Clear(C)
 Pick_Table(A)

State

Clear(C)
On(A, Table)
 On(B, Table)
 Clear(A)
Handempty
 On(C, B)

[Pick_Block(C);
 Put_Block(C, B)]

14. Search Stack

On(A, C) On(C, B)
 Put_Block(A, C)
 Holding(A) Clear(C)
 Pick_Table(A)
 Handempty Clear(A)
 On(A, Table)

State

Clear(C)
On(A, Table)
 On(B, Table)
 Clear(A)
Handempty
 On(C, B)

[Pick_Block(C); Put_Block(C, B)]

16. Search Stack

On(A, C) On(C, B)
 Put_Block(A, C)
 Holding(A) Clear(C)

State

Clear(C)
 On(B, Table)
 Clear(A)
 On(C, B)
Holding(A)

[Pick_Block(C);
 Put_Block(C, B);
 Pick_Table(A)]

GPS Blocks-World Example

17. Search Stack

On(A, C) On(C, B)

Put_Block(A, C)

[Pick_Block(C);
Put_Block(C, B);
Pick_Table(A)]

State

Clear(C)
On(B, Table)
Clear(A)
On(C, B)
Holding(A)

18. Search Stack

On(A, C) On(C, B)

[Pick_Block(C);
Put_Block(C, B);
Pick_Table(A);
Put_Block(A, C)]

State

On(B, Table)
Clear(A)
On(C, B)
Handempty
On(A, C)

19. Search Stack

[Pick_Block(C);
Put_Block(C, B);
Pick_Table(A);
Put_Block(A, C)]

State

On(B, Table)
Clear(A)
On(C, B)
Handempty
On(A, C)

Properties of Planning Algorithms

- **Soundness**

- A planning algorithm is **sound** if all solutions found are legal plans
 - All preconditions and goals are satisfied
 - No constraints are violated (temporal, variable binding)

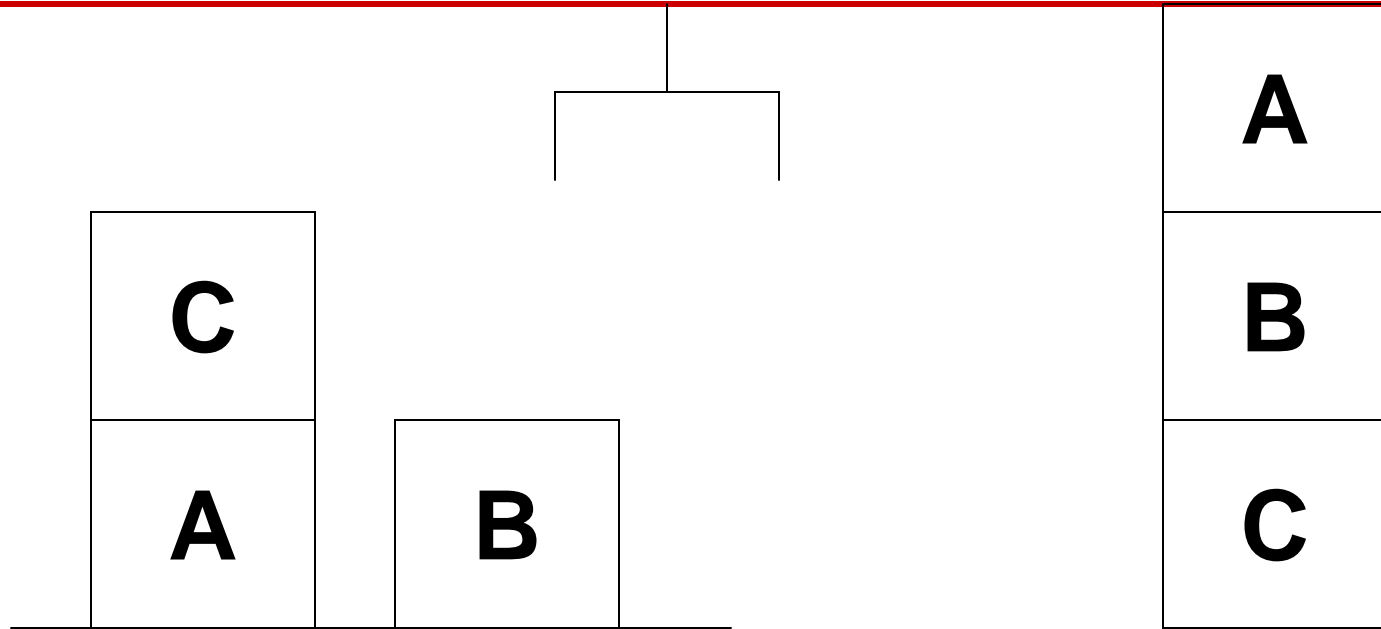
- **Completeness**

- A planning algorithm is **complete** if a solution can be found whenever one actually exists
- A planning algorithm is **strictly complete** if all solutions are included in the search space

- **Optimality**

- A planning algorithm is **optimal** if the order in which solutions are found is consistent with some measure of plan quality

The Sussman Anomaly



Linear Planning: Discussion

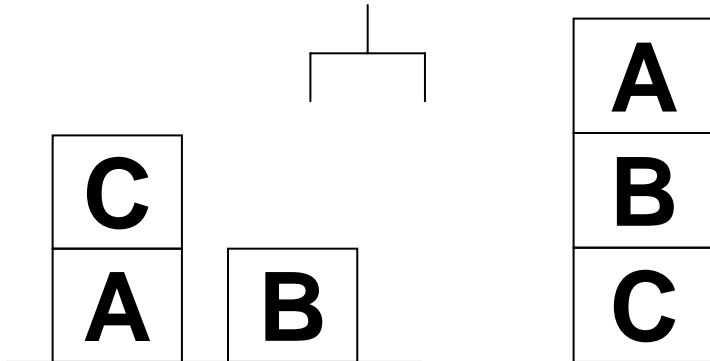
- **Advantages**

- Reduced search space, since goals are solved one at a time
- Advantageous if goals are (mainly) independent
- Linear planning is **sound**

- **Disadvantages**

- Linear planning may produce **suboptimal** solutions (based on the number of operators in the plan)

The Sussman Anomaly



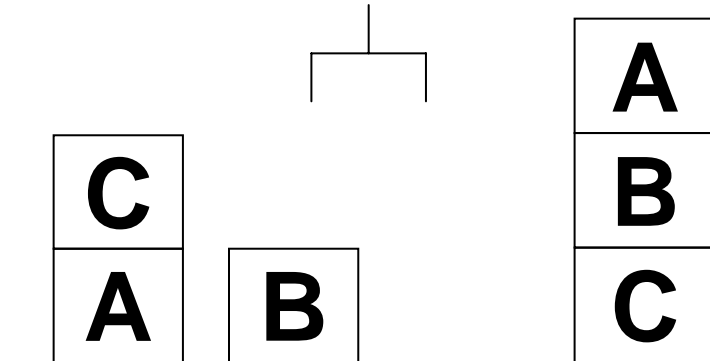
Linear Solution:

- (on B C)
 - Pickup (B)
 - Stack (B, C)
- (on A B)
 - Unstack (B, C)
 - Putdown (B)
 - Unstack (C, A)
 - Putdown (C)
 - Stack (A, B)
- (on B C)
 - Unstack (A, B)
 - Putdown (A)
 - Pickup (B)
 - Stack (B, C)
- (on A B)
 - Pickup (A)
 - Stack (A, B)

Linear Solution:

- (on A B)
 - Unstack (C, A)
 - Putdown (C)
 - Stack (A, B)
- (on B C)
 - Unstack (A, B)
 - Putdown (A)
 - Pickup (B)
 - Stack (B, C)
- (on A B)
 - Pickup (A)
 - Stack (A, B)

NonLinear Solution – Optimal



NonLinear Solution:

- *(on A B)*
 - *Unstack (C, A)*
 - *Putdown (C)*
- *(on B C)*
 - *Pickup (B)*
 - *Stack (B, C)*
- *(on A B)*
 - *Pickup (A)*
 - *Stack (A, B)*

Why is Planning Hard?

Planning involves a complex search:

- Alternative operators to achieve a goal
- Multiple goals that interact
- Solution optimality, quality
- Planning efficiency, soundness, completeness

Linear Planning – Goal Stack

- Planner can be unoptimal
- Planner's efficiency is sensitive to goal orderings
 - Control knowledge for the “right” ordering
 - Random restarts
 - Iterative deepening
- Planner keeps a small search space by not considering all the possible goal orderings.
- Any other problems/features?

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))
```

<i>Goal</i>	<i>Plan</i>
(at obj1 locB)	(LOAD-ROCKET obj1 locA) (MOVE-ROCKET) (UNLOAD-ROCKET obj1 locB)
(at obj2 locB)	<i>failure</i>

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**.

<i>Goal</i>	<i>Plan</i>
(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 obj1 locB)

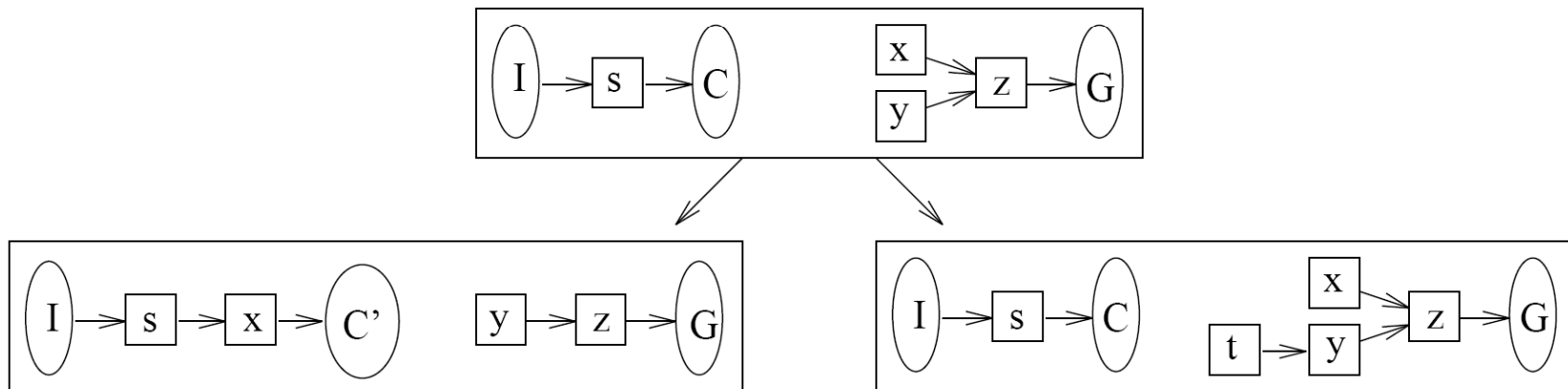
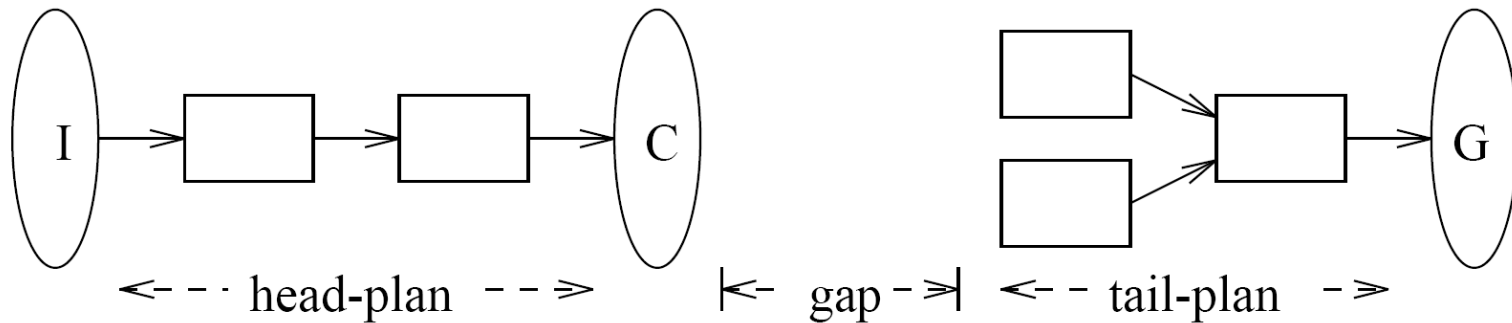
Prodigy4.0 (Veloso et al. 90)

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.
1. Choose a goal G in \mathcal{G} or choose an operator A in \mathcal{A} .

Prodigy4.0 Planning Algorithm

4. If G has been chosen, then
 - *Expand goal G ,*
i.e., get the set \mathcal{O} of *relevant instantiated operators*
that could achieve 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



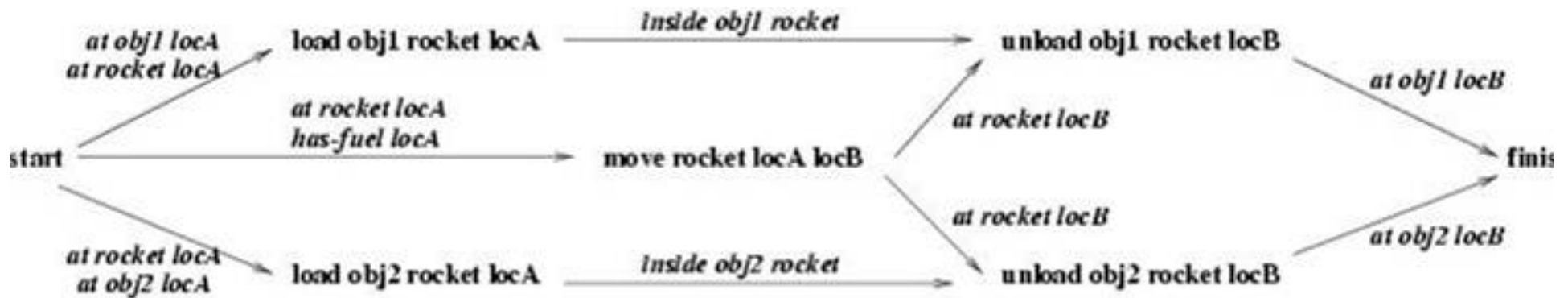
Applying an operator (moving it to the head)

Adding an operator to the tail-plan

Plan-Space Planning

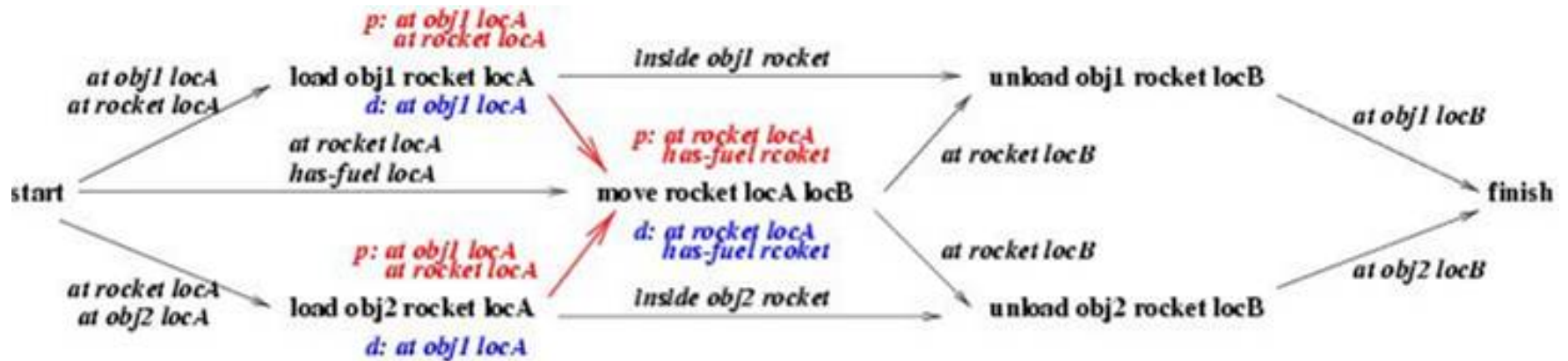
- Complete, sound, and “optimal” in terms of number of steps in the plan and within the operator choices made
- Optimal handling of goal orderings

Rocket Domain - Linking



Example - LINKING

Rocket Domain – Solving Threats



Example - THREATS

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.

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

	<i>State-space</i>	<i>Plan-space</i>
Commitments in plan step orderings	Yes	No
Therefore, suffer with goal orderings	Yes	No
Therefore, handle goal interactions	Poorly	Efficiently

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.

Eagerly Subgoaling Can Be Better

Operator: A_i

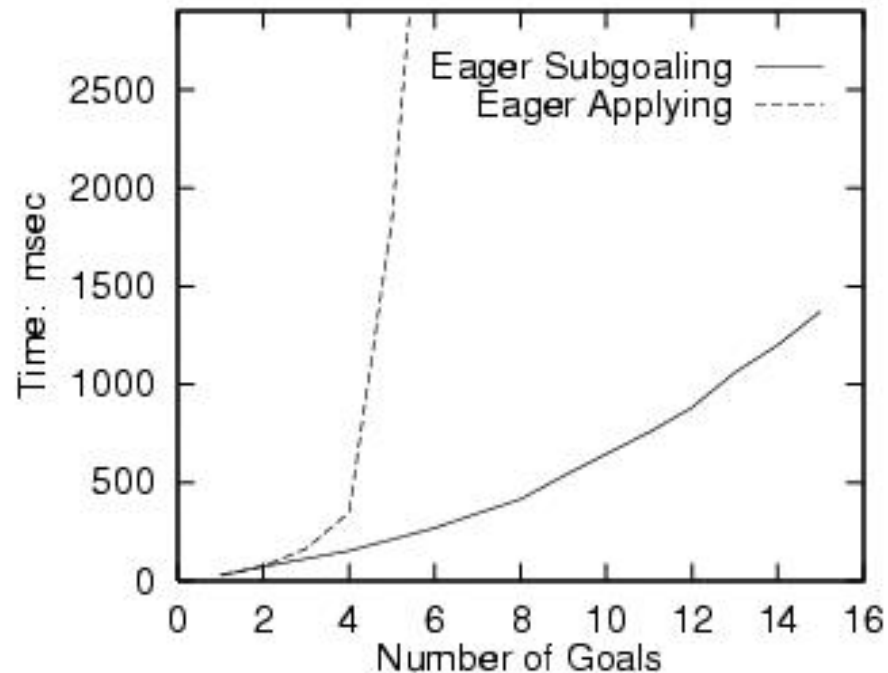
preconds: $\{I_i\}$

adds: $\{G_i\}$

deletes: $\{I_j \mid j < i\}$

Example:

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



Parallel between Commitments

Operator Polish
 preconds: ()
 adds: polished
 deletes: ()

Operator Drill-Hole
 preconds: ()
 adds: has-hole
 deletes: polished

Goal: *polished* and *has-hole*
 Initial state: empty
 Prodigy4.0

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

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

- 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* \Rightarrow *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_*

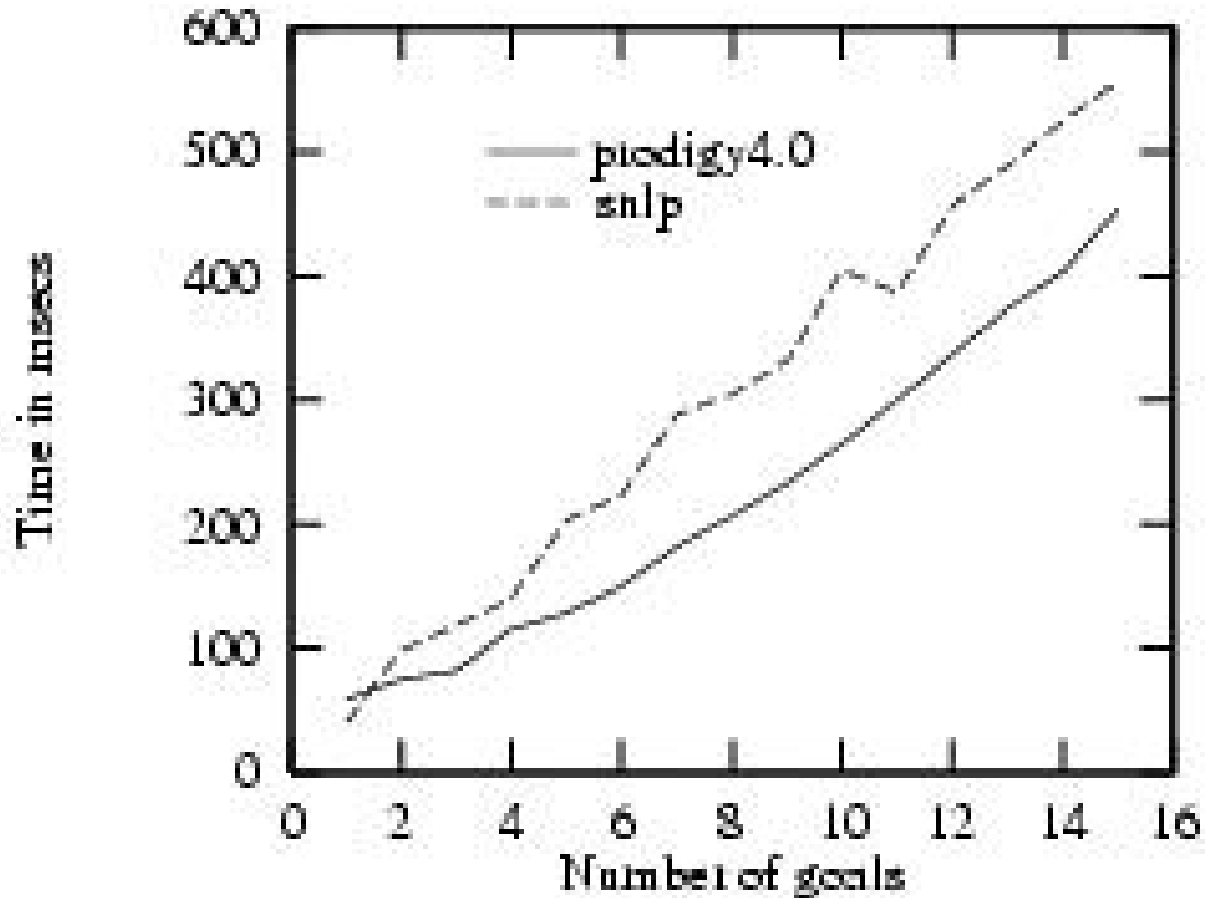
deletes $g_i, \forall i$

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



Laboriously Linkable Goals

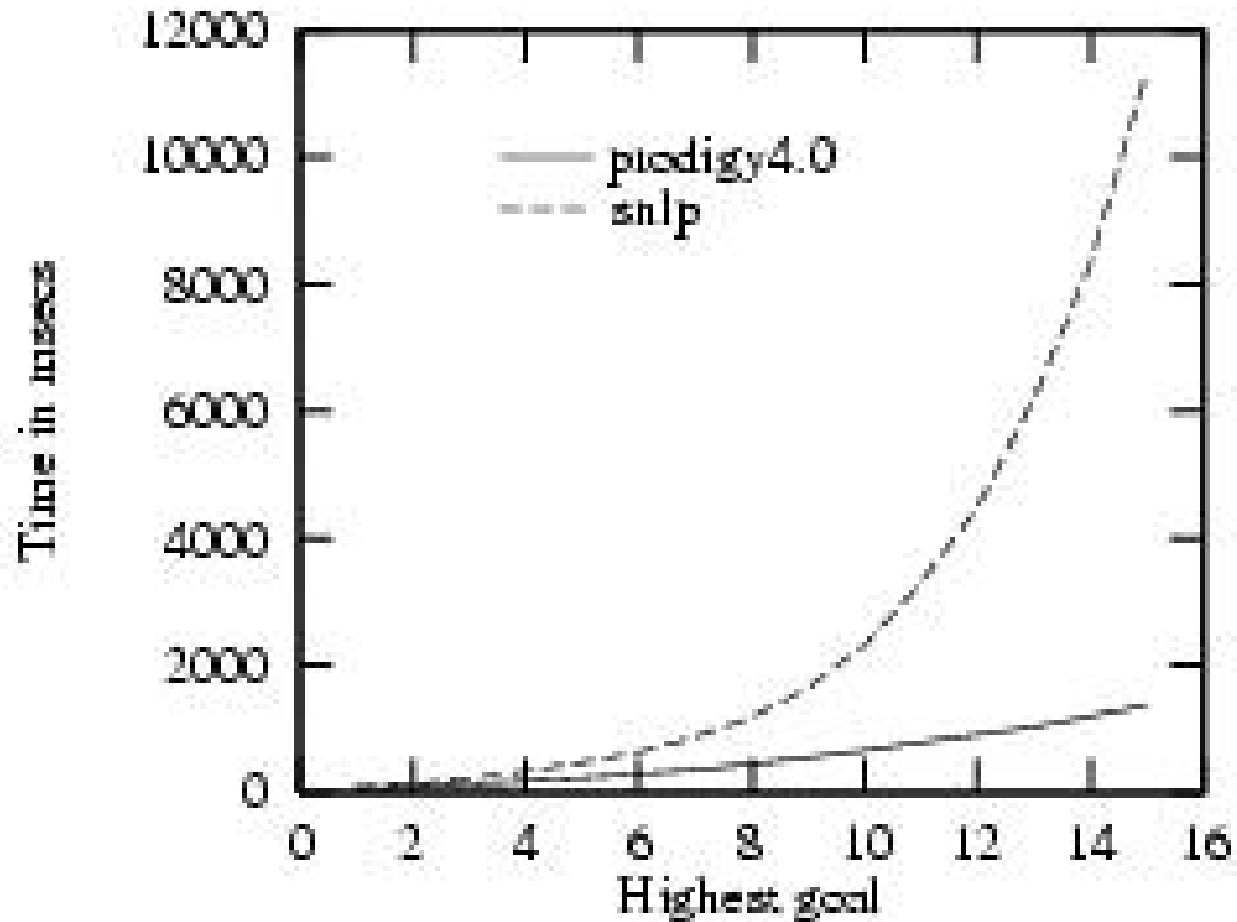
operator A_i		operator A_*	
preconds	g_*, g_{i-1}	preconds	()
adds	g_i	adds	g_*
deletes	g_*	deletes	()

Initial state: g_*

Goal statement: g_*, g_5

Plan: $A_1, A_*, A_2, A_*, A_3, A_* A_4, A_* A_5, A_*$

Laboriously Linkable Goals



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

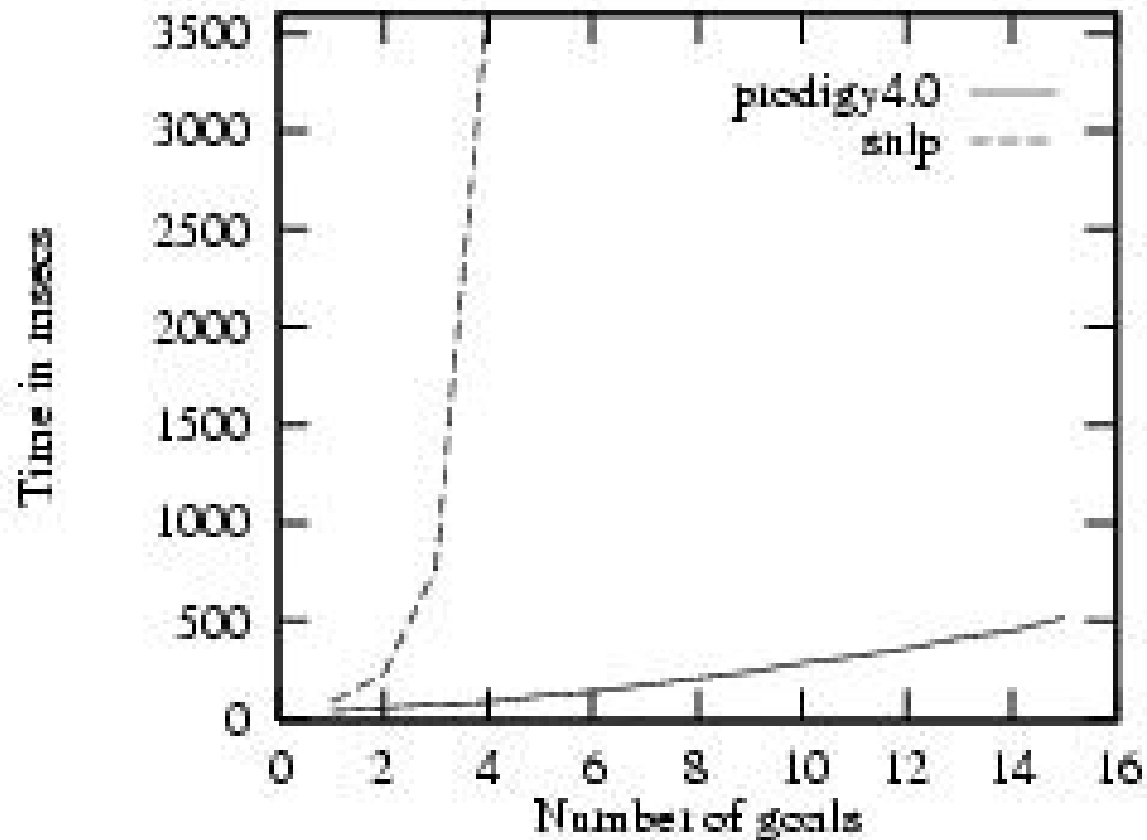
operator A_3
pre g_2, g_1
add g_3, g_1
del g_2

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



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.

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

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.