Linear and Nonlinear State-Space Planning

Manuela Veloso
Reid Simmons

PEL, Fall 2010

September 20, 2010

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.

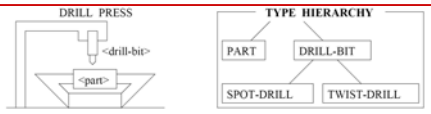
Action Model, State, Goals

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

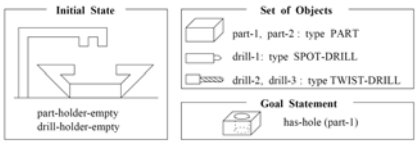
put-part(<part>)
<part>: type PART
Pre: part-holder-empty
Add: (holding-part <drill-bit>)
Del: part-holder-empty

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)

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

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

Goal

Initial State

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)

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

[Pick_Block(C)]

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)]

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)
Put_Block(C, B)]

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)]

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)]

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)]

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)

State

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

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

18. Search Stack

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

State

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

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

19. Search Stack

State

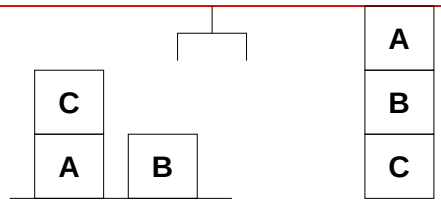
On(B, Table)
Clear(A)
On(C, B)
Handempty
On(A, C)

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

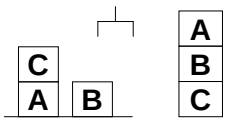
Properties of Planning Algorithms

- **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



The Sussman Anomaly



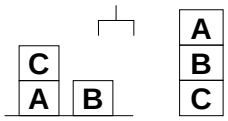
Linear Solution:

- *(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)*

Linear Planning – Goal Stack

- **Advantages**
  - Reduced search space, since goals are solved one at a time, and not all possible goal orderings are considered
  - 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)
  - Planner's efficiency is sensitive to goal orderings
    - Control knowledge for the "right" ordering
    - Random restarts
    - Iterative deepening
- Completeness?

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*

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

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 obj1 locB)`

Planning with Backward-Chaining (*aka Subgoaling*) and State-Reduction (*aka Operator Application*)

- "Plain" GPS and possible extensions
  - Linear stack of goals
  - Commitment on a chosen goal
    - Could consider all orderings of goals, at some stack level
  - Commitment on a chosen operator
    - Could consider multiple operators relevant to a goal
  - Commitment to apply an operator as soon as it is chosen and becomes applicable in the current state, and multiple applicable operators (if any) are applied in the order in which they are in the stack.

Prodigy Planner

- Extension to GPS
  - Set of goals, instead of stack of goals
  - Means-ends analysis for selection of "pending goals"
  - Choice point for applying an operator when applicable and continue backward-chaining (subgoaling)

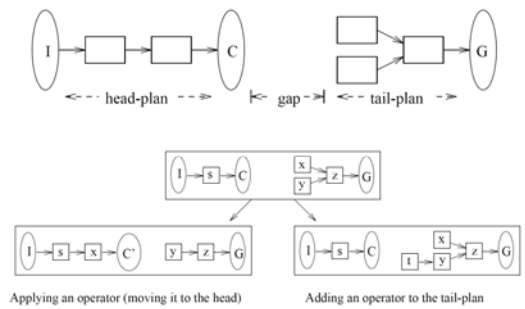
Prodigy4.0 (Veloso et al. 90)

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



Next Class

- Analysis of Prodigy
  - Heuristics for guiding choices
- Comparison of plan-space and state-space planning
- Graph-based planning
  - The basics

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.