Planning

Chapter 11
Outline

♦ Search vs. planning
♦ STRIPS operators
♦ Partial-order planning
Search vs. planning

Consider the task *get milk, bananas, and a cordless drill*
Standard search algorithms seem to fail miserably:

After-the-fact heuristic/goal test inadequate
Planning systems do the following:
1) open up action and goal representation to allow selection
2) divide-and-conquer by subgoaling
3) relax requirement for sequential construction of solutions

<table>
<thead>
<tr>
<th></th>
<th>Search</th>
<th>Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>States</td>
<td>Lisp data structures</td>
<td>Logical sentences</td>
</tr>
<tr>
<td>Actions</td>
<td>Lisp code</td>
<td>Preconditions/outcomes</td>
</tr>
<tr>
<td>Goal</td>
<td>Lisp code</td>
<td>Logical sentence (conjunction)</td>
</tr>
<tr>
<td>Plan</td>
<td>Sequence from $S_0$</td>
<td>Constraints on actions</td>
</tr>
</tbody>
</table>
STRIPS operators

Tidily arranged actions descriptions, restricted language

**ACTION:** $Buy(x)$

**PRECONDITION:** $At(p), Sells(p, x)$

**EFFECT:** $Have(x)$

[Note: this abstracts away many important details!]

Restricted language $\Rightarrow$ efficient algorithm

- Precondition: conjunction of positive literals
- Effect: conjunction of literals

A complete set of STRIPS operators can be translated into a set of successor-state axioms
Partially ordered plans

*Partially ordered* collection of steps with
- *Start* step has the initial state description as its effect
- *Finish* step has the goal description as its precondition
- *causal links* from outcome of one step to precondition of another
- *temporal ordering* between pairs of steps

**Open condition** = precondition of a step not yet causally linked

A plan is *complete* iff every precondition is achieved

A precondition is *achieved* iff it is the effect of an earlier step
and no possibly intervening step undoes it
Example

Start

At(Home)  Sells(HWS,Drill)  Sells(SM,Milk)  Sells(SM,Ban.)

Have(Milk)  At(Home)  Have(Ban.)  Have(Drill)

Finish
Example

Start

At(Home)  Sells(HWS,Drill)  Sells(SM,Milk)  Sells(SM,Ban.)

At(HWS)  Sells(HWS,Drill)

Buy(Drill)

At(x)

Go(SM)

At(SM)  Sells(SM,Milk)

Buy(Milk)

Have(Milk)  At(Home)  Have(Ban.)  Have(Drill)

Finish
Example

- At(Home)
- Go(HWS)
- Buy(Drill)
- Go(SM)
- Buy(Milk)
- Buy(Ban.)
- Go(Home)

Finish

- At(Home)
- Have(Ban.)
- Have(Drill)

Start

- At(HWS)
- Sells(HWS, Drill)
- Sells(SM, Milk)
- Sells(SM, Ban.)
Planning process

Operators on partial plans:
  * add a link from an existing action to an open condition
  * add a step to fulfill an open condition
  * order one step wrt another to remove possible conflicts

Gradually move from incomplete/vague plans to complete, correct plans

Backtrack if an open condition is unachievable or if a conflict is unresolvable
function POP($\text{initial, goal, operators}$) returns $\text{plan}$

$\text{plan} \leftarrow \text{Make-Minimal-Plan}(\text{initial, goal})$

loop do
  if $\text{Solution?}(\text{plan})$ then return plan
  $S_{\text{need}}, c \leftarrow \text{Select-Subgoal}(\text{plan})$
  $\text{Choose-Operator}(\text{plan, operators, } S_{\text{need}}, c)$
  $\text{Resolve-Threats}(\text{plan})$
end

function $\text{Select-Subgoal}(\text{plan})$ returns $S_{\text{need}}, c$

pick a plan step $S_{\text{need}}$ from $\text{Steps}(\text{plan})$
  with a precondition $c$ that has not been achieved

return $S_{\text{need}}, c$
**POP algorithm contd.**

procedure **`Choose-Operator(plan, operators, S_{need}, c)`**

- choose a step $S_{add}$ from `operators` or `Steps(plan)` that has $c$ as an effect
- if there is no such step then fail
- add the causal link $S_{add} \rightarrow c \rightarrow S_{need}$ to `Links(plan)`
- add the ordering constraint $S_{add} \prec S_{need}$ to `Orderings(plan)`
- if $S_{add}$ is a newly added step from `operators` then
  - add $S_{add}$ to `Steps(plan)`
  - add `Start` $\prec$ $S_{add}$ $\prec$ `Finish` to `Orderings(plan)`

procedure **`Resolve-Threats(plan)`**

- for each $S_{threat}$ that threatens a link $S_i \rightarrow c \rightarrow S_j$ in `Links(plan)` do
  - choose either
    - *Demotion*: Add $S_{threat} \prec S_i$ to `Orderings(plan)`
    - *Promotion*: Add $S_j \prec S_{threat}$ to `Orderings(plan)`
  - if not `Consistent(plan)` then fail
- end
Clobbering and promotion/demotion

A **clobberer** is a potentially intervening step that destroys the condition achieved by a causal link. E.g., \( Go(\text{Home}) \) clobbers \( At(\text{Supermarket}) \):

**Demotion**: put before \( Go(\text{Supermarket}) \)

**Promotion**: put after \( Buy(\text{Milk}) \)
Properties of POP

Nondeterministic algorithm: backtracks at choice points on failure:
- choice of $S_{add}$ to achieve $S_{need}$
- choice of demotion or promotion for clobberer
- selection of $S_{need}$ is irrevocable

POP is sound, complete, and systematic (no repetition)

Extensions for disjunction, universals, negation, conditionals

Can be made efficient with good heuristics derived from problem description

Particularly good for problems with many loosely related subgoals
Example: Blocks world

"Sussman anomaly" problem

Start State

\[
\text{Clear}(x) \ \text{On}(x,z) \ \text{Clear}(y)
\]

\[
\text{PutOn}(x,y)
\]

\[
\sim \text{On}(x,z) \ \sim \text{Clear}(y)
\]

\[
\text{Clear}(z) \ \text{On}(x,y)
\]

Goal State

\[
\text{Clear}(x) \ \text{On}(x,z)
\]

\[
\text{PutOnTable}(x)
\]

\[
\sim \text{On}(x,z) \ \text{Clear}(z) \ \text{On}(x,\text{Table})
\]

+ several inequality constraints
Example contd.

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

On(A,B) On(B,C)
Example contd.

On(C,A)  On(A,Table)  Cl(B)  On(B,Table)  Cl(C)

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

PutOn(B,C)

FINISH
Example contd.

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)

PutOn(A,B)

On(A,B) On(A,z) Cl(B)

Fin(1)

PutOn(B,C)

On(B,z) Cl(C)

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

START

FINISH
Example contd.

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

On(C, A)  On(A, Table)  Cl(B)  On(B, Table)  Cl(C)

PutOn(C, B)

Cl(B)  On(B, z)  Cl(C)

PutOn(A, B)  clobbers  Cl(B) => order after  PutOn(B, C)

PutOn(A, B)

PutOn(B, C)

Cl(B)  On(B, z)  Cl(C)

PutOn(B, C)  clobbers  Cl(C) => order after  PutOnTable(C)

PutOn(A, B)

PutOnTable(C)

On(C, z)

Cl(C)

FINISH

Chapter 11  19