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)  Sells(HWS,Drill)  Go(HWS)  At(HWS)  Buy(Drill)  At(HWS)  Go(SM)  Sells(SM,Milk)  At(SM)  Buy(Milk)  Sells(SM,Ban.)  At(SM)  Buy(Ban.)  At(SM)  Go(Home)  Have(Milk)  Finish

Chapter 11  9
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
POP algorithm sketch

function \text{POP}(\text{initial}, \text{goal}, \text{operators}) \text{ returns } \text{plan} \\
\text{plan} \leftarrow \text{Make-Minimal-Plan}(\text{initial}, \text{goal}) \\
\text{loop do} \\
\quad \text{if } \text{Solution?}(\text{plan}) \text{ then return } \text{plan} \\
\quad \text{S}_{\text{need}}, \text{c} \leftarrow \text{Select-Subgoal}(\text{plan}) \\
\quad \text{Choose-Operator}(\text{plan}, \text{operators}, \text{S}_{\text{need}}, \text{c}) \\
\quad \text{Resolve-Threats}(\text{plan}) \\
\text{end} \\

function \text{Select-Subgoal}(\text{plan}) \text{ returns } \text{S}_{\text{need}}, \text{c} \\
pick a plan step \text{S}_{\text{need}} from \text{Steps}(\text{plan}) \\
\quad \text{with a precondition } \text{c} \text{ that has not been achieved} \\
\text{return } \text{S}_{\text{need}}, \text{c}
**POP algorithm contd.**

**procedure** `CHOOSE-OPERATOR(plan, operators, S_{need}, c)`

1. choose a step $S_{add}$ from `operators` or `STEPS(plan)` that has $c$ as an effect
2. if there is no such step then fail
3. add the causal link $S_{add} \rightarrow c \rightarrow S_{need}$ to `LINKS(plan)`
4. add the ordering constraint $S_{add} \prec S_{need}$ to `ORDERINGS(plan)`
5. if $S_{add}$ is a newly added step from `operators` then
   1. add $S_{add}$ to `STEPS(plan)`
   2. add $Start \prec S_{add} \prec Finish$ to `ORDERINGS(plan)`

**procedure** `RESOLVE-THREATS(plan)`

1. for each $S_{threat}$ that threatens a link $S_i \rightarrow c \rightarrow S_j$ in `LINKS(plan)` do
   1. choose either
      1. Demotion: Add $S_{threat} \prec S_i$ to `ORDERINGS(plan)`
      2. Promotion: Add $S_j \prec S_{threat}$ to `ORDERINGS(plan)`
   2. 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., \(Go(Home)\) clobbers \(At(Supermarket)\):

\[
\text{Go(Supermarket)} \rightarrow \text{Go(Home)} \rightarrow \text{At(Home)} \rightarrow \text{Buy(Milk)} \rightarrow \text{At(Home)} \rightarrow \text{Finish}
\]

**Demotion**: put before \(Go(Supermarket)\)

**Promotion**: put after \(Buy(Milk)\)
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

Goal State

\[
\begin{align*}
\text{Clear}(x) \; \text{On}(x,z) \; \text{Clear}(y) \\
\text{PutOn}(x,y) \\
\sim\text{On}(x,z) \; \sim\text{Clear}(y) \\
\text{Clear}(z) \; \text{On}(x,y)
\end{align*}
\]

\[
\begin{align*}
\text{Clear}(x) \; \text{On}(x,z) \\
\text{PutOnTable}(x) \\
\sim\text{On}(x,z) \; \text{Clear}(z) \; \text{On}(x,\text{Table})
\end{align*}
\]

+ several inequality constraints
Example contd.

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

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

START

FINISH
Example contd.

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

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

PutOn(B, C)

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

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

PutOn(A,B)

PutOn(B,C)

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

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

FINISH
Example contd.

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

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

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

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

PutOnTable(C)