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>Search</th>
<th>Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>States</td>
<td>Logical sentences</td>
</tr>
<tr>
<td>Actions</td>
<td>Preconditions/outcomes</td>
</tr>
<tr>
<td>Goal</td>
<td>Logical sentence (conjunction)</td>
</tr>
<tr>
<td>Plan</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.)

Finish

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

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

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 POP(initial, goal, operators) returns plan

plan ← Make-Minimal-Plan(initial, goal)
loop do
  if Solution?(plan) then return plan
  Sneed, c ← Select-Subgoal(plan)
  Choose-Operator(plan, operators, Sneed, c)
  Resolve-Threats(plan)
end

function Select-Subgoal(plan) returns Sneed, c
pick a plan step Sneed from Steps(plan)
  with a precondition c that has not been achieved
return Sneed, c

POP algorithm contd.

procedure Choose-Operator(plan, operators, Sneed, c)
choose a step Sadd from operators or Steps(plan) that has c as an effect
if there is no such step then fail
add the causal link Sadd ←→ Sneed to Links(plan)
add the ordering constraint Sadd ≺ Sneed to Orderings(plan)
if Sadd is a newly added step from operators then
  add Sadd to Steps(plan)
  add Start ≺ Sadd ≺ Finish to Orderings(plan)

procedure Resolve-Threats(plan)
for each Sthreat that threatens a link Si ←→ Sj in Links(plan) do
  choose either
    Demotion: Add Sthreat ≺ Si to Orderings(plan)
    Promotion: Add Sj ≺ Sthreat 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., $Go(Home)$ clobbers $At(Supermarket)$:

![Diagram showing the clobbering process]

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

Clear(x) On(x,z) Clear(y)

PutOn(x,y)

~On(x,z) ~Clear(y)

Clear(z) On(x,y)

Clear(x) On(x,z)

PutOnTable(x)

~On(x,z) Clear(z) On(x,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)

FINISH
Example contd.

START

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

PutOn(B,C)

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

FINISH

Chapter 11 17

Example contd.

START

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

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

PutOn(A,B)

PutOn(B,C)

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

FINISH

Chapter 11 18
Example contd.

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

On(C,z) Cl(C)

PutOnTable(C)

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

PutOn(A,B)

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

PutOn(B,C)

PutOnTable(C)

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

FINISH