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:

![Diagram showing various tasks and their connections, including:
- Start
- Go To Supermarket
- Go To School
- Go To Pet Store
- Go To Sleep
- Read A Book
- Sit in Chair
- Etc. Etc. ...
- Buy Tuna Fish
- Buy Arugula
- Buy Milk
- Buy a Dog
- Talk to Parrot
- Go To Class
- Go To Pet Store
- Go To School
- Go To Supermarket
- Sit Some More
- Read A Book
- Finish]

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.)

Sells(HWS,Drill)

Buy(Drill)

Sells(HWS,Drill)

At(HWS)

At(x)

Go(SM)

Sells(SM,Milk)

Buy(Milk)

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

Finish
Example

Start

At(Home)
Go(HWS)

At(HWS) Sells(HWS,Drill)
Buy(Drill)

At(HWS)
Go(SM)

At(SM) Sells(SM,Milk)
Buy(Milk)

At(SM) Sells(SM,Ban.)
Buy(Ban.)

At(SM)
Go(Home)

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

Finish
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
    S_{need}, c ← Select-Subgoal(plan)
    Choose-Operator(plan, operators, S_{need}, c)
    Resolve-Threats(plan)
end

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

pick a plan step S_{need} from Steps(plan)
    with a precondition c that has not been achieved
return S_{need}, c
```
procedure **CHOOSE-OPERATOR**(*plan*, *operators*, *S*<sub>need</sub>, *c*)

choose a step *S*<sub>add</sub> from *operators* or STEPS(*plan*) that has *c* as an effect
if there is no such step then fail
add the causal link *S*<sub>add</sub> → *S*<sub>need</sub> to LINKS(*plan*)
add the ordering constraint *S*<sub>add</sub> ≺ *S*<sub>need</sub> to ORDERINGS(*plan*)
if *S*<sub>add</sub> is a newly added step from *operators* then
add *S*<sub>add</sub> to STEPS(*plan*)
add *Start* ≺ *S*<sub>add</sub> ≺ *Finish* to ORDERINGS(*plan*)

---

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

for each *S*<sub>threat</sub> that threatens a link *S*<sub>i</sub> → *S*<sub>j</sub> in LINKS(*plan*) do
choose either
Demotion: Add *S*<sub>threat</sub> ≺ *S*<sub>i</sub> to ORDERINGS(*plan*)
Promotion: Add *S*<sub>j</sub> ≺ *S*<sub>threat</sub> 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 $At(Supermarket)$:

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

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

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

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

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

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

Chapter 11 15
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)

PutOn(B,C)

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

FINISH
Example contd.

\[ \text{On}(C, A) \quad \text{On}(A, \text{Table}) \quad \text{Cl}(B) \quad \text{On}(B, \text{Table}) \quad \text{Cl}(C) \]

\[ \text{PutOn}(A, B) \quad \text{PutOn}(B, C) \]

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

Chapter 11 18
On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

PutOnTable(C)

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

PutOn(A,B)

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

PutOn(B,C)

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

FINISH