Artificial Intelligence

Topic 9

Planning

- Search vs. planning
- Planning Languages and STRIPS
- State Space vs. Plan Space
- Partial-order Planning

Reading: Russell & Norvig, Chapter 11
1. 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
1. Search vs. Planning

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><strong>States</strong></td>
<td>internal state of Java objects</td>
<td>descriptive (logical) sentences</td>
</tr>
<tr>
<td><strong>Actions</strong></td>
<td>encoded in Java methods</td>
<td>preconditions/outcomes</td>
</tr>
<tr>
<td><strong>Goal</strong></td>
<td>encoded in Java methods</td>
<td>descriptive sentence</td>
</tr>
<tr>
<td><strong>Plan</strong></td>
<td>sequence from $s_0$</td>
<td>constraints on actions</td>
</tr>
</tbody>
</table>

$\Rightarrow$ *implicit*                                          $\Rightarrow$ *explicit*  
$\Rightarrow$ *hard to decompose*                                $\Rightarrow$ *easier to decompose*
2. Planning Languages and STRIPS

Require *declarative language* — *declarations* or *statements* about world.

Range of logics have been proposed — best descriptive languages we have, but can be difficult to use in practice.

\[
\text{more descriptive power} \rightarrow \text{more difficult to compute (reason)}
\]

\[
\text{automatically}
\]
2. Planning Languages and STRIPS

Require *declarative language* — *declarations or statements* about world.

Range of logics have been proposed — best descriptive languages we have, but can be difficult to use in practice.

*more descriptive power* $\rightarrow$ *more difficult to compute (reason)*


to automatically

STRIPS (STanford Research Institute Problem Solver) first to suggest suitable compromise

- restricted form of logic
- restricted language $\Rightarrow$ efficient algorithm

Basis of many subsequent languages and planners.

**States**

\[
\text{At(Home), } \neg \text{Have(Milk), } \neg \text{Have(Bananas), } \neg \text{Have(Drill)}
\]

(conjunctions of function-free ground literals)
2. Planning Languages and STRIPS

Goals

\[ \text{At(Home), Have(Milk), Have(Bananas), Have(Drill)} \]

Can have variables

\[ \text{At(x), Sells(x,Milk)} \]

(conjunctions of function-free literals)
2. Planning Languages and STRIPS

Goals

At(Home), Have(Milk), Have(Bananas), Have(Drill)

Can have variables

At(x), Sells(x, Milk)

(conjunctions of function-free literals)

Actions

ACTION (NAME): Buy(x)
PRECONDITION: At(p), Sells(p, x)
EFFECT: Have(x)

(Precondition: conjunction of positive literals
Effect: conjunction of literals)
3. State Space vs. Plan Space

Standard search: node = concrete world state
Planning search: node = partial plan

Definition: open condition is a precondition of a step not yet fulfilled

Operators on partial plans, eg:

- add a step to fulfill an open condition
- order one step wrt another
- instantiate an unbound variable

Gradually move from incomplete/vague plans to complete, correct plans
4. Partial-order planning

Example

Goal: RightShoeOn, LeftShoeOn

Operators:

\[ Op(\text{Action: RightShoe, Precond: RightSockOn, Effect: RightShoeOn}) \]
\[ Op(\text{Action: RightSock, Effect: RightSockOn}) \]
\[ Op(\text{Action: LeftShoe, Precond: LeftSockOn, Effect: LeftShoeOn}) \]
\[ Op(\text{Action: LeftShoe, Effect: LeftShoeOn}) \]

Consider partial plans:

1. LeftShoe, RightShoe — ordering unimportant
2. RightSock, RightShoe — ordering important
3. RightSock, LeftShoe, RightShoe — ordering between some actions important

*partial order planner* \(\Rightarrow\) planner that can represent steps in which some are ordered (in sequence) and others not (in “parallel”)

\[ \text{© Cara MacNish. Includes material © S. Russell & P. Norvig 1995,2003 with permission.} \]

CITS4211 Planning Slide 9
4. Partial-order planning

least commitment planner — partial order planner that delays commitment to order between steps for as long as possible

⇒ less backtracking

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
4. Partial-order planning

*linearisation* — obtaining a totally ordered plan from a partially ordered plan by imposing ordering constraints
4. Partial-order planning

In addition to orderings we must record

- variable bindings: eg. \( x = \text{LocalStore} \)
- causal links: \( S_i \xrightarrow{c} S_j \) (\( S_i \) achieves precondition \( c \) for \( S_j \))

Thus our initial plan might be:

\[
\text{Plan(Steps):}\{ \begin{align*}
S_1 &: \text{Op(Action: Start)}, \\
S_2 &: \text{Op(Action: Finish,} \\
& \quad \text{Precond: RightShoeOn, LeftShoeOn)}
\end{align*}\}
\[
\text{Orderings:}\{ S_1 \prec S_2 \},
\text{Bindings:}\{\},
\text{Links:}\{\}
\]

Algorithm \( \cdots \rightarrow \)
4.1 POP algorithm sketch

```plaintext
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
```

continued...
4.1 POP algorithm sketch

procedure \textsc{Choose-Operator}(\textit{plan}, \textit{operators}, \textit{S}_{\text{need}}, \textit{c})

choose a step \( S_{\text{add}} \) from \textit{operators} or \textsc{Steps}(\textit{plan}) that has \textit{c} as an effect

\textbf{if} there is no such step \textbf{then} \textbf{fail}

add the causal link \( S_{\text{add}} \rightarrow c \rightarrow S_{\text{need}} \) to \textsc{Links}(\textit{plan})

add the ordering constraint \( S_{\text{add}} \prec S_{\text{need}} \) to \textsc{Orderings}(\textit{plan})

\textbf{if} \( S_{\text{add}} \) is a newly added step from \textit{operators} \textbf{then}

\hspace{1em} add \( S_{\text{add}} \) to \textsc{Steps}(\textit{plan})

\hspace{1em} add \( \text{Start} \prec S_{\text{add}} \prec \text{Finish} \) to \textsc{Orderings}(\textit{plan})

procedure \textsc{Resolve-Threats}(\textit{plan})

\textbf{for each} \( S_{\text{threat}} \) that threatens a link \( S_i \rightarrow c \rightarrow S_j \) in \textsc{Links}(\textit{plan}) \textbf{do}

\hspace{1em} choose either

\hspace{2em} \textit{Demotion:} Add \( S_{\text{threat}} \prec S_i \) to \textsc{Orderings}(\textit{plan})

\hspace{2em} \textit{Promotion:} Add \( S_j \prec S_{\text{threat}} \) to \textsc{Orderings}(\textit{plan})

\hspace{1em} \textbf{if not} \textsc{Consistent}(\textit{plan}) \textbf{then} \textbf{fail}

\textbf{end}
4.1 POP algorithm sketch

**procedure** \textsc{Choose-Operator}(plan, operators, $S_{\text{need}}, c$)

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

**procedure** \textsc{Resolve-Threats}(plan)

- for each $S_{\text{threat}}$ that threatens a link $S_i \xrightarrow{c} S_j$ in $\text{Links}(plan)$ do
  - choose either
    - \textit{Demotion}: Add $S_{\text{threat}} \prec S_i$ to $\text{Orderings}(plan)$
    - \textit{Promotion}: Add $S_j \prec S_{\text{threat}}$ to $\text{Orderings}(plan)$
  - if not \textsc{Consistent}(plan) then fail

POP is sound, complete, and \underline{systematic} (no repetition)

Extensions for more expressive languages (eg disjunction, etc)
A *clobberer* is a potentially intervening step that destroys the condition achieved by a causal link. E.g., \( Go(Home) \) clobbers \( At(HWS) \):

[Diagram]

- **Demotion**: put before \( Go(HWS) \)
- **Promotion**: put after \( Buy(Drill) \)
4.3 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
4.3 Example: Blocks world

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

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

4.3 Example: Blocks world

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

PutOn(B,C)

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

FINISH
4.3 Example: Blocks world

\[
\begin{align*}
\text{START} & \\
\text{On}(C,A) & \quad \text{On}(A,\text{Table}) & \quad \text{Cl}(B) & \quad \text{On}(B,\text{Table}) & \quad \text{Cl}(C) \\
\text{Cl}(A) & \quad \text{On}(A,z) & \quad \text{Cl}(B) & \quad \text{PutOn}(A,B) & \quad \text{PutOn}(B,C) & \quad \text{Fin}\text{ISH} \\
\text{On}(A,B) & \quad \text{On}(B,C) \\
\end{align*}
\]

PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)
4.3 Example: Blocks world

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

PutOn(A,B)
PutOn(B,C)
PutOnTable(C)

PutOnTable(C) clobbers Cl(C)
PutOn(B,C) clobbers Cl(C)

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

The End