Planning

Philipp Koehn

26 March 2019
Outline

- Search vs. planning
- STRIPS operators
- Partial-order planning
- The real world
- Conditional planning
- Monitoring and replanning
search vs. 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
Search vs. Planning

- Planning systems do the following
  1. improve action and goal representation to allow selection
  2. divide-and-conquer by **subgoaling**
  3. relax requirement for sequential construction of solutions

- Differences

<table>
<thead>
<tr>
<th></th>
<th>Search</th>
<th>Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>States</td>
<td>Data structures</td>
<td>Logical sentences</td>
</tr>
<tr>
<td>Actions</td>
<td>Program code</td>
<td>Preconditions/outcomes</td>
</tr>
<tr>
<td>Goal</td>
<td>Program 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
STRIPS Operators

- Tidily arranged actions descriptions, restricted language

\[
\begin{array}{c}
At(p) \quad Sells(p,x) \\
\hline
Buy(x) \\
\hline
Have(x)
\end{array}
\]

- **ACTION**: \(Buy(x)\)
  **PRECONDITION**: \(At(p), Sells(p,x)\)
  **EFFECT**: \(Have(x)\)

- Note: this abstracts away many important details!

- Restricted language \(\implies\) efficient algorithm
  - Precondition: conjunction of positive literals
  - Effect: conjunction of literals
partial-order planning
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

\[
\begin{align*}
&\text{Start} \\
&\text{At(Home) } \quad \text{Sells(HWS,Drill)} \quad \text{Sells(SM,Milk)} \quad \text{Sells(SM,Ban.)} \\
&\text{At(SM) } \quad \text{Sells(SM,Milk)} \\
&\text{Buy(Milk)} \\
&\text{Have(Milk) } \quad \text{At(Home)} \quad \text{Have(Ban.)} \quad \text{Have(Drill)} \\
&\text{Finish}
\end{align*}
\]
Example
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
function POP(initial, goal, operators) returns plan

   plan ← MAKE-MINIMAL-PLAN(initial, goal)

loop do
   if SOLUTION?(plan) then return plan
   S\textsubscript{need}, c ← SELECT-SUBGOAL(plan)
   CHOOSE-OPERATOR(plan, operators, S\textsubscript{need}, c)
   RESOLVE-THREATS(plan)
end

function SELECT-SUBGOAL(plan) returns S\textsubscript{need}, c

   pick a plan step S\textsubscript{need} from STEPS(plan)
   with a precondition c that has not been achieved

   return S\textsubscript{need}, c
**Partially Ordered Plans Algorithm**

**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 $S_{add} \xrightarrow{c} S_{need}$ to `LINKS(plan)`
add the ordering constraint $S_{add} < S_{need}$ to `ORDERINGS(plan)`
if $S_{add}$ is a newly added step from `operators` then
  add $S_{add}$ to `STEPS(plan)`
  add `Start < S_{add} < Finish` to `ORDERINGS(plan)`

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

for each $S_{threat}$ that threatens a link $S_i \xrightarrow{c} S_j$ in `LINKS(plan)` do
  choose either
  * **Demotion:** Add $S_{threat} < S_i$ to `ORDERINGS(plan)`
  * **Promotion:** Add $S_j < S_{threat}$ to `ORDERINGS(plan)`
  if not `CONSISTENT(plan)` then fail
end
A **clobberer** is a potentially intervening step that destroys the condition achieved by a causal link. E.g., \( \text{Go(Home)} \) clobbers \( \text{At(Supermarket)} \):

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

**Promotion:** put after \( \text{Buy(Milk)} \)
Properties of Partially Ordered Plans

- 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

- Partially Ordered Plans 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

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

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

+ several inequality constraints
Example

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

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

FINISH
Example

START
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
Example
the real world
The Real World

- Flat(Spare)
- Intact(Spare)
- Off(Spare)
- On(Tire1)
- Flat(Tire1)

START

- On(x)

Remove(x)

- Off(x)
- ClearHub

FINISH

- On(x)

Puton(x)

- Off(x)
- ClearHub

- Intact(x)

Inflate(x)

- On(x)
- ~ClearHub

- ~Flat(x)
Things Go Wrong

- **Incomplete information**
  - Unknown preconditions, e.g., $\text{Intact}(\text{Spare})$?
  - Disjunctive effects, e.g., $\text{Inflate}(x)$ causes $\text{Inflated}(x) \lor \text{SlowHiss}(x) \lor \text{Burst}(x) \lor \text{BrokenPump} \lor \ldots$

- **Incorrect information**
  - Current state incorrect, e.g., spare NOT intact
  - Missing/incorrect postconditions in operators

- **Qualification problem** can never finish listing all
  - required preconditions of actions
  - possible conditional outcomes of actions
Solutions

- **Conformant or sensorless planning**
  Devise a plan that works regardless of state or outcome
  *Such plans may not exist*

- **Conditional planning**
  Plan to obtain information (**observation actions**)
  Subplan for each contingency, e.g.,
  
  \[
  [Check(Tire_1), \text{if } Intact(Tire_1) \text{ then } Inflate(Tire_1) \text{ else } CallAAA]
  \]

  *Expensive because it plans for many unlikely cases*

- **Monitoring/Replanning**
  Assume normal states, outcomes
  Check progress *during execution*, replan if necessary
  *Unanticipated outcomes may lead to failure (e.g., no AAA card)*

⇒ Really need a combination; plan for likely/serious eventualities,
dead with others when they arise, as they must eventually.
Conformant Planning

- Search in space of belief states (sets of possible actual states)
conditional planning
Conditional Planning

- If the world is nondeterministic or partially observable then percepts usually provide information, i.e., split up the belief state.
Conditional Planning

- Conditional plans check (any consequence of KB +) percept

- \([\ldots, \text{if } C \text{ then } Plan_A \text{ else } Plan_B, \ldots]\)

- Execution: check \(C\) against current KB, execute “then” or “else”

- Need some plan for every possible percept
  - game playing: some response for every opponent move
  - backward chaining: some rule such that every premise satisfied

- AND–OR tree search (very similar to backward chaining algorithm)
Example

- Double Murphy: sucking or arriving may dirty a clean square
monitoring and replanning
Execution Monitoring

- Plan with Partially Ordered Plans algorithms
- Process plan, one step at a time
- Validate planned conditions against perceived reality
- “Failure” = preconditions of remaining plan not met
- Preconditions of remaining plan
  = all preconditions of remaining steps not achieved by remaining steps
  = all causal links crossing current time point
Responding to Failure

- Run Partially Ordered Plans algorithms again
- Resume Partially Ordered Plans to achieve open conditions from current state
- IPEM (Integrated Planning, Execution, and Monitoring)
  - keep updating *Start* to match current state
  - links from actions replaced by links from *Start* when done
Example

Start

\(\text{At(Home)}\)
\(\text{Go(HWS)}\)

\(\text{At(HWS)}\)
\(\text{Sells(HWS,Drill)}\)

Buy(Drill)

\(\text{At(HWS)}\)
\(\text{Go(SM)}\)

\(\text{At(SM)}\)
\(\text{Sells(SM,Milk)}\)

Buy(Milk)

\(\text{At(SM)}\)
\(\text{Sells(SM,Ban.)}\)

Buy(Ban.)

\(\text{At(SM)}\)
\(\text{Go(Home)}\)

Finish

\(\text{At(Home)}\)
\(\text{Sells(HWS,Drill)}\)
\(\text{Sells(SM,Ban.)}\)
\(\text{Sells(SM,Milk)}\)
Example
Example
Example
Example
Example
Emergent Behavior

预条件

START

\(\text{Color(Chair,Blue)} \rightarrow \neg \text{Have(Red)}\)

Get(Red)

\(\text{Have(Red)}\)

Paint(Red)

\(\text{Color(Chair,Red)}\)

FINISH

失败响应

取更多红色

红色
Emergent Behavior

PRECONDITIONS

START

Color(Chair, Blue) \land \neg Have(\text{Red})

Get(\text{Red})

Have(\text{Red})

Paint(\text{Red})

\text{Color}(\text{Chair}, \text{Red})

FINISH

FAILURE RESPONSE

Extra coat of paint
• “Loop until success” behavior **emerges** from interaction between monitor/replan agent design and uncooperative environment
Summary

• Planning
  – break down problem into subgoals
  – search for plans for subgoals
  – merge sub-plans

• Defined actions in terms of preconditions and effects

• Partially Ordered Plans algorithm

• Clobbering: need to deal with steps that destroy clausal link in plan

• Real world: incomplete and incorrect information

⇒ conformant or conditional planning, monitoring and replanning