KI-Programmierung

Planning

Bernhard Beckert

Universität Koblenz-Landau

Winter Term 2007/2008
Outline

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

Consider the following task

Get milk, bananas, and a cordless drill

Standard search algorithms seem to fail miserably
Search vs. Planning

Actions have requirements & consequences that should constrain applicability in a given state

⇒ stronger interaction between actions and states needed
Search vs. Planning

Actions have requirements & consequences that should constrain applicability in a given state

⇒ stronger interaction between actions and states needed

Most parts of the world are independent of most other parts

⇒ solve subgoals independently
Search vs. Planning

- Actions have requirements & consequences that should constrain applicability in a given state
  - stronger interaction between actions and states needed

- Most parts of the world are independent of most other parts
  - solve subgoals independently

- Human beings plan goal-directed; they construct important intermediate solutions first
  - flexible sequence for construction of solution
Planning systems do the following

- Unify action and goal representation to allow selection (use logical language for both)
- Divide-and-conquer by subgoaling
- Relax requirement for sequential construction of solutions
STRIPS

STandford Research Institute Problem Solver

- Tidily arranged actions descriptions
- Restricted language (function-free literals)
- Efficient algorithms
STRIPS: States

States represented by:

Conjunction of ground (function-free) atoms

Example

\[\text{At(Home)}, \text{Have(Bread)}\]
STRIPS: States

States represented by:

Conjunction of ground (function-free) atoms

Example

\[ \text{At(Home), Have(Bread)} \]

Closed world assumption

Atoms that are not present are assumed to be false

Example

State: \[ \text{At(Home), Have(Bread)} \]
Implicitly: \[ \neg\text{Have(Milk), \neg Have(Bananas), \neg Have(Drill)} \]
STRIPS: Operators

Operator description consists of:

<table>
<thead>
<tr>
<th>Action name</th>
<th>Positive literal</th>
<th>Precondition</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive literal</td>
<td>$At(Shop) \land Sells(Shop, Milk)$</td>
<td>$Have(Milk)$</td>
</tr>
</tbody>
</table>

Buy($Milk$)
STRIPS: Operators

Operator description consists of:

- **Action name**: Positive literal
- **Precondition**: Conjunction of positive literals
- **Effect**: Conjunction of literals

Operator schema

Operator containing variables

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

Operator \(o\) applicable in state \(s\) if:
there is substitution \(Subst\) of the free variables such that

\[
Subst(precond(o)) \subseteq s
\]
STRIPS: Operator Application

Operator applicability

Operator \( o \) applicable in state \( s \) if:
there is substitution \( Subst \) of the free variables such that

\[
Subst(precond(o)) \subseteq s
\]

Example

\( \text{Buy}(x) \) is applicable in state

\[
\text{At}(\text{Shop}) \land \text{Sells}(\text{Shop}, \text{Milk}) \land \text{Have}(\text{Bread})
\]

with substitution

\[
Subst = \{ p/\text{Shop}, x/\text{Milk} \}
\]
STRIPS: Operator Application

**Resulting state**

Computed from old state and literals in $\text{Subst}(\text{effect})$

- Positive literals are added to the state
- Negative literals are removed from the state
- All other literals remain unchanged
  (avoids the frame problem)
STRIPS: Operator Application

**Resulting state**

Computed from old state and literals in \( Subst(\text{effect}) \)

- Positive literals are added to the state
- Negative literals are removed from the state
- All other literals remain unchanged (avoids the frame problem)

**Formally**

\[
s' = (s \cup \{ P \mid P \text{ a positive atom, } P \in Subst(\text{effect}(o)) \}) \\
\setminus \{ P \mid P \text{ a positive atom, } \neg P \in Subst(\text{effect}(o)) \}
\]
STRIPS: Operator Application

Example

Application of $\text{Drive}(a,b)$

precond: $\text{At}(a), \text{Road}(a,b)$

effect: $\text{At}(b), \neg \text{At}(a)$
Example

Application of

\[ \text{Drive}(a,b) \]

precond:  \( \text{At}(a), \text{Road}(a,b) \)

effect:  \( \text{At}(b), \neg\text{At}(a) \)

to state

\[ \text{At(Koblenz)}, \text{Road(Koblenz, Landau)} \]
Example

Application of

\[ \text{Drive}(a,b) \]

\text{precond:} \quad \text{At}(a), \text{Road}(a, b) \\
\text{effect:} \quad \text{At}(b), \neg \text{At}(a) \\

to state

\text{At(Koblenz)}, \text{Road(Koblenz, Landau)} \\

results in

\text{At(Landau)}, \text{Road(Koblenz, Landau)} \\

Planning problem

Find a sequence of actions that make instance of the goal true
State Space vs. Plan Space

**Planning problem**

Find a sequence of actions that make instance of the goal true

**Nodes in search space**

*Standard search:* node = concrete world state

*Planning search:* node = partial plan
State Space vs. Plan Space

Planning problem

Find a sequence of actions that make instance of the goal true

Nodes in search space

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

(Partial) Plan consists of

- Set of operator applications $S_i$
- Partial (temporal) order constraints $S_i \prec S_j$
- Causal links $S_i \xrightarrow{c} S_j$

Meaning: “$S_i$ achieves $c \in \text{precond}(S_j)$” (record purpose of steps)
State Space vs. Plan Space

Operators on partial plans

- add an action and a causal link to achieve an open condition
- add a causal link from an existing action to an open condition
- add an order constraint to order one step w.r.t. another

Open condition

A precondition of an action not yet causally linked
State Space vs. Plan Space

Operators on partial plans

- add an action and a causal link to achieve an open condition
- add a causal link from an existing action to an open condition
- add an order constraint to order one step w.r.t. another

Open condition

A precondition of an action not yet causally linked

Note

We move from incomplete/vague plans to complete, correct plans
**Partially Ordered Plans**

![Diagram of partially ordered plans]

**Special steps with empty action**

- **Start**: no precond, initial assumptions as effect
- **Finish**: goal as precond, no effect
**Partially Ordered Plans**

Special steps with empty action

*Start*  no precond, initial assumptions as effect)

*Finish*  goal as precond, no effect

Note

Different paths in partial plan are *not* alternative plans, but alternative sequences of actions
Partially Ordered Plans

**Complete plan**

A plan is complete if and only if every precondition is achieved.
Partially Ordered Plans

**Complete plan**

A plan is complete iff every precondition is achieved

A precondition $c$ of a step $S_j$ is achieved (by $S_i$) if

- $S_i \prec S_j$
- $c \in \text{effect}(S_i)$
- there is no $S_k$ with $S_i \prec S_k \prec S_j$ and $\neg c \in \text{effect}(S_k)$ (otherwise $S_k$ is called a clobberer or threat)
Partially Ordered Plans

**Complete plan**

A plan is complete iff every precondition is achieved

A precondition $c$ of a step $S_j$ is achieved (by $S_i$) if

- $S_i \prec S_j$
- $c \in \text{effect}(S_i)$
- there is no $S_k$ with $S_i \prec S_k \prec S_j$ and $\neg c \in \text{effect}(S_k)$

(otherwise $S_k$ is called a clobberer or threat)

**Clobberer / threat**

A potentially intervening step that destroys the condition achieved by a causal link
Clobbering and Promotion/Demotion

Example

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

**Demotion**

Put before \( \text{Go(HWS)} \)

**Promotion**

Put after \( \text{Buy(Drill)} \)
Example: Blocks world

"Sussman anomaly" problem

Start State

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

Goal State

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

+ several inequality constraints
Example: Blocks World

START

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

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

FINISH

B. Beckert: KI-Programmierung – p.18
Example: Blocks World

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}(B) \quad \text{On}(B,z) \quad \text{Cl}(C)

\text{PutOn}(B,C)

\text{On}(A,B) \quad \text{On}(B,C)

FINISH
Example: Blocks World

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

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

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

PutOnTable(C)

On(C, z) Cl(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)

START

FINISH

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

PutOn(B, C) clobbers Cl(C) => order after PutOnTable(C)
function POP(initial, goal, operators) returns plan

\[ plan \leftarrow \text{MAKE-MINIMAL-PLAN}(\text{initial}, \text{goal}) \]

loop do
  if SOLUTION?(\text{plan}) then return \text{plan} \quad \% \text{complete and consistent} \\
  \text{S}_{\text{need}}, \, c \leftarrow \text{SELECT-SUBGOAL}(\text{plan}) \\
  \text{CHOOSE-OPERATOR}(\text{plan}, \text{operators}, \text{S}_{\text{need}}, \, c) \\
  \text{RESOLVE-THREATS}(\text{plan})
end

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

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

return \text{S}_{\text{need}}, \, c
procedure \texttt{CHOOSE-OPERATOR}(plan, operators, S_{need}, c)\

\textbf{choose} a step $S_{add}$ from operators or STEPS(\texttt{plan}) that has $c$ as an effect\n
\textbf{if} there is no such step \textbf{then fail}\n
add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS(\texttt{plan})\n
add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS(\texttt{plan})\n
\textbf{if} $S_{add}$ is a newly added step from operators \textbf{then}\n
\hspace{1em} add $S_{add}$ to STEPS(\texttt{plan})\n
\hspace{1em} add $Start \prec S_{add} \prec Finish$ to ORDERINGS(\texttt{plan})
PO POP Algorithm (Cont’d)

\[
\text{procedure } \text{RESOLVE-THREATS}(plan) \\
\text{for each } S_{\text{threat}} \text{ that threatens a link } S_i \xrightarrow{c} S_j \text{ in } \text{LINKS}(plan) \text{ do} \\
\quad \text{choose either} \\
\quad \text{Demotion: Add } S_{\text{threat}} \prec S_i \text{ to } \text{ORDERINGS}(plan) \\
\quad \text{Promotion: Add } S_j \prec S_{\text{threat}} \text{ to } \text{ORDERINGS}(plan) \\
\quad \text{if not } \text{CONSISTENT}(plan) \text{ then fail} \\
\text{end}
\]
Non-deterministic search for plan, backtracks over choicepoints on failure:

- choice of $S_{add}$ to achieve $S_{need}$
- choice of promotion or demotion for clobberer
Properties of POP

- Non-deterministic search for plan, backtracks over choicepoints on failure:
  - choice of $S_{add}$ to achieve $S_{need}$
  - choice of promotion or demotion for clobberer

- Sound and complete
Properties of POP

- Non-deterministic search for plan, backtracks over choicepoints on failure:
  - choice of $S_{add}$ to achieve $S_{need}$
  - choice of promotion or demotion for clobberer

- Sound and complete

- There are extensions for:
  disjunction, universal quantification, negation, conditionals
Properties of POP

- Non-deterministic search for plan, backtracks over choicepoints on failure:
  - choice of $S_{add}$ to achieve $S_{need}$
  - choice of promotion or demotion for clobberer

- Sound and complete

- There are extensions for:
  disjunction, universal quantification, negation, conditionals

- Efficient with good heuristics from problem description
  But: very sensitive to subgoal ordering
Properties of POP

- Non-deterministic search for plan, backtracks over choicepoints on failure:
  - choice of $S_{\text{add}}$ to achieve $S_{\text{need}}$
  - choice of promotion or demotion for clobberer

- Sound and complete

- There are extensions for:
  - disjunction, universal quantification, negation, conditionals

- Efficient with good heuristics from problem description
  But: very sensitive to subgoal ordering

- Good for problems with loosely related subgoals
The Real World

START

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

On(x)  ~Flat(x)

FINISH

On(x)
Remove(x)

Off(x)  ClearHub

Off(x)  ClearHub

Intact(x)  Flat(x)

Puton(x)

On(x)  ~ClearHub

Inflate(x)

~Flat(x)
Things Go Wrong

Incomplete information

- Unknown preconditions

Example: $\text{Intact}(\text{Spare})$?

- Disjunctive effects

Example: $\text{Inflate}(x)$ causes

$\text{Inflated}(x) \lor \text{SlowHiss}(x) \lor \text{Burst}(x) \lor \text{BrokenPump} \lor \ldots$
Things Go Wrong

**Incomplete information**

- **Unknown preconditions**  
  Example: $\text{Intact}(\text{Spare})$?

- **Disjunctive effects**  
  Example: $\text{Inflate}(x)$ causes $\text{Inflated}(x) \lor \text{SlowHiss}(x) \lor \text{Burst}(x) \lor \text{BrokenPump} \lor \ldots$

**Incorrect information**

- **Current state incorrect**  
  Example: spare NOT intact

- **Missing/incorrect postconditions in operators**
Things Go Wrong

Incomplete information

- Unknown preconditions  
  Example:  \( \text{Intact}(Spare) \)?

- Disjunctive effects
  
  Example:  \( \text{Inflate}(x) \) causes  
  \( \text{Inflated}(x) \lor \text{SlowHiss}(x) \lor \text{Burst}(x) \lor \text{BrokenPump} \lor \ldots \)

Incorrect information

- Current state incorrect  
  Example:  \( \text{spare NOT intact} \)

- Missing/incorrect postconditions in operators

Qualification problem

- Can never finish listing all the required preconditions and possible conditional outcomes of actions
Conditional planning

Plan to obtain information (observation actions)

Subplan for each contingency

Example: \[\text{Check}(Tire1), \text{If} \left( \text{Intact}(Tire1), \text{Inflate}(Tire1) \right), \text{CallHelp} \]

Disadvantage: Expensive because it plans for many unlikely cases
Solutions

Conditional planning

- Plan to obtain information  (observation actions)
- Subplan for each contingency

  Example:  \[\text{Check} (\text{Tire1}), \text{If} (\text{Intact} (\text{Tire1}), [\text{Inflate} (\text{Tire1})], [\text{CallHelp}])\]

  Disadvantage:  Expensive because it plans for many unlikely cases

Monitoring/Replanning

- Assume normal states / outcomes
- Check progress **during execution**, replan if necessary

  Disadvantage:  Unanticipated outcomes may lead to failure
Conditional Planning

Execution of conditional plan

\[\ldots, \textbf{If}(p, \textbf{thenPlan}, \textbf{elsePlan}), \ldots\]

Check $p$ against current knowledge base, execute $\textbf{thenPlan}$ or $\textbf{elsePlan}$
Conditional Planning

Execution of conditional plan

\[
\ldots, \text{If}(p, \text{thenPlan}, \text{elsePlan}), \ldots
\]

Check \(p\) against current knowledge base, execute \(\text{thenPlan}\) or \(\text{elsePlan}\)

Conditional planning

Just like POP except:
If an open condition can be established by observation action

- add the action to the plan
- complete plan for each possible observation outcome
- insert conditional step with these subplans

\[
\text{CheckTire}(x) \quad \text{KnowsIf}(\text{Intact}(x))
\]
Conditional Planning Example

Start

\begin{align*}
on(Tire1) \\
n Flat(Tire1) \\
Inflated(Spare)
\end{align*}

Finish

\begin{align*}
on(x) \\
Inflated(x) \\
(True)
\end{align*}
Conditional Planning Example
Conditional Planning Example
Conditional Planning Example
Conditional Planning Example

Start
On(Tire1)
Flat(Tire1)
Inflated(Spare)

Check(Tire1)

Flat(Tire1)
Intact(Tire1)

Inflate(Tire1)
(Intact(Tire1))

Finish
On(Tire1)
Inflated(Spare)

Finish
On(Tire1)
Inflated(Spare)

(Intact(Tire1))

(^Intact(Tire1))
Conditional Planning Example
Monitoring

Execution monitoring

Failure: Preconditions of remaining plan not met
Monitoring

**Execution monitoring**

**Failure:** Preconditions of remaining plan not met

**Action monitoring**

**Failure:** Preconditions of next action not met

(or action itself fails, e.g., robot bump sensor)
Monitoring

Execution monitoring

Failure: Preconditions of remaining plan not met

Action monitoring

Failure: Preconditions of next action not met
(or action itself fails, e.g., robot bump sensor)

Consequence of failure

Need to replan
Preconditions for Remaining Plan

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

At(HWS)

Have(Drill)

Sells(SM,Ban.)

Sells(SM,Milk)
Replanning

**Simplest**

On failure, replan from scratch
Replanning

**Simplest**

On failure, replan from scratch

**Better**

Plan to get back on track by reconnecting to best continuation
Replanning: Example

PRECONDITIONS

<table>
<thead>
<tr>
<th>Action</th>
<th>Precondition</th>
<th>Failure Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color(Chair,Red)</td>
<td>none</td>
<td>N/A</td>
</tr>
<tr>
<td>Have(Chair,Red)</td>
<td>Have(Chair,Red)</td>
<td>Fetch more red</td>
</tr>
<tr>
<td>Color(Chair,Red)</td>
<td>Color(Chair,Red)</td>
<td>Repaint</td>
</tr>
</tbody>
</table>

START

Color(Chair,Blue) ~Have(Chair,Red)

Get(Chair,Red)

Have(Chair,Red)

Paint(Chair,Red)

Finish
Summary Planning

- Differs from general problem search; subgoals solved independently

- STRIPS: restricted format for actions, logic-based

- Nodes in search space are partial plans

- POP algorithm

- Standard planning cannot cope with incomplete/incorrect information

- Conditional planning with sensing actions to complete information; expensive at planning stage

- Replanning based on monitoring of plan execution; expensive at execution stage