CS 4649/7649 RIP
Robot Intelligence: Planning

Partial Order Planning, Graphplan

Sungmoon Joo
School of Interactive Computing
College of Computing
Georgia Institute of Technology

Course Info.

- Course Website: joosm.github.io/RIP2014
  - add your contact info, start grouping/filling in project ideas, etc.
  - github invitation sent (if you didn't get one, let me know)
  *announcements on t-square
- Email me(sungmoon.joo@cc.gatech.edu)
  - Introduce yourself, your github id
  - Project ideas, etc.
Regression Planning

Possible previous states

Regression Planning (Variables Show Up)

C would already be on B

\[ x = ? \]
Regression Planning

- Just as with forward (progression) planning this algorithm would be complete. However:
- Still large branching factor (potentially larger)
- When do we instantiate the variables?

Regression Planning

- Still large branching factor (potentially larger)
- When do we instantiate the variables?
- Introduces new concept:

  “Least Commitment Planning”

  Make choices only when they are relevant to solving
  The current part of the problem
Partial Order Planning

- Based on the concept of “Least Commitment”
- Nodes are partial plans
- Arcs/Transitions are plan refinements
- Solution is a node, not a path (search in plan space!!)

Partial Order Planning

- A plan consists of:
  - $A$: Set of actions
  - $O$: Set of orderings for actions (a < b)
  - $Q$: Set of causal links
- A causal link $q \in Q$ is defined as follows:
  - Action $b$ has a precondition that is established by Action $w$

\[ w = Work \quad q_1 = \text{Have($\$')} \quad b = \text{Buy(Mercedes)} \]
Partial Order Planning: Threats

- A threat to \((a, q, b)\) is defined as an action \(t\) that:
  - Has \(\neg q\) as an effect
    \[-q \in t_{\text{add}}\]
  - Could occur between \(a\) and \(b\)
    \[O \cup a < t < b\] is inconsistent
- What action \(t\) would be a threat to causal link \(q_i\)?

\[
\begin{align*}
\text{w = Work} & \quad q_1 = \text{Have(\$)} & \quad b = \text{Buy(Mercedes)}
\end{align*}
\]

Partial Order Planning: Initialization

Since we’re only talking about actions, let’s turn states into actions:

- \(A_{\text{II}}\)
  - No preconditions
  - Initial state as effects
  - Must be the first step in the plan (all actions \(> A_{\text{II}}\) in \(O\))

- \(A_{\text{NI}}\)
  - No effects
  - Goals as preconditions
  - Must be the last step in the plan (all actions \(< A_{\text{NI}}\) in \(O\))
**POP Algorithm: Simplified UCPOP’ (Weld)**

**UCPOP (Universal, Conditional Partial-Order Planner)**

S. Joo (sungmoon.joo@cc.gatech.edu) 9/2/2014

---

**POP Algorithm**

```
POP ((A, O, Q), agenda, actions)
If agenda is empty then return (A, O, Q)
(q, a<sub>need</sub>) = Choose (pair) from agenda
a<sub>next</sub> = Choose (actions) s.t. q ∈ Add (a<sub>next</sub>)
If no such action a<sub>next</sub> exists, Fail
Q<sup>'</sup> = Q ∪ (a<sub>next</sub>, q, a<sub>next</sub>)
Q<sup>'</sup> = Q ∪ (a<sub>next</sub>, Q, a<sub>next</sub>)
agenda<sup>'</sup> = agenda - (q, a<sub>need</sub>)

If a<sub>next</sub> is new, then A = A ∪ a<sub>next</sub> and
∀ p ∈ PC(a<sub>next</sub>) add (p, a<sub>next</sub>) to agenda<sup>'</sup>

For every action a<sub>i</sub> that threatens any causal link (a<sub>i</sub>, Q, a<sub>j</sub>) in Q<sup>'</sup>
Choose to add a<sub>i</sub> ≤ a<sub>j</sub> or a<sub>i</sub> ≤ a<sub>j</sub> to Q<sup>'</sup>
If neither choice is consistent, Fail

POP ((A, O, Q), agenda, actions)
```

The magic "Choose" enables backtracking.

---

**Example: Agenda = ( Have($), buy(Merc) )**

**Q = ( (buy(Merc), Have(Merc), Finish) )**

**O = ( buy(Merc) < Finish )**

---

**PC: Have ($)**

**D: Have ($)**

**A: Have(x)**

**buy(Mercedes)**

**α<sub>N</sub> = Finish**

**PC: Have(Mercedes)**
**POP Algorithm**

```
POP ((A, O, Q), agenda, actions)
If agenda = ∅ then return (A, O, Q)
(q, a_{next}) = Choose (pair) from agenda
a_{next} = Choose (action) s.t. q ∈ Add(a_{next})
If no such action a_{next} exists, Fail

Q′ = Q ∪ (q, a_{next})
O′ = O ∪ (q, a_{next} < a_{next})
agenda′ = agenda ∖ (q, a_{next})

If a_{next} is new, then A = A ∪ a_{next} and
∀ p ∈ PC(a_{next}) add (p, a_{next}) to agenda'

For every action a_i that threatens any causal link (a_i, q, a_j) in Q' 
Choose to add a_i < a_j or a_j < a_i to O
if neither choice is consistent, Fail

POP((A′, O′, Q′), agenda, actions)
```

**Agenda** =

```
{}
```

**Q** =

```
{(buy(Mercedes), Have(Mercedes), Finish)
(work, Have($), buy(Mercedes))
}
```

**O** =

```
{(buy(Mercedes) < Finish)
work < buy(Mercedes)
}
```

---

**DONE!**

```
work
```

```
buy(Mercedes)
```

```
a_N = Finish
PC : Have(Mercedes)
```

---

**POP Algorithm**

```
POP ((A, O, Q), agenda, actions)
If agenda = ∅ then return (A, O, Q)
(q, a_{next}) = Choose (pair) from agenda
a_{next} = Choose (action) s.t. q ∈ Add(a_{next})
If no such action a_{next} exists, Fail

Q′ = Q ∪ (q, a_{next})
O′ = O ∪ (q, a_{next} < a_{next})
agenda′ = agenda ∖ (q, a_{next})

If a_{next} is new, then A = A ∪ a_{next} and
∀ p ∈ PC(a_{next}) add (p, a_{next}) to agenda'

For every action a_i that threatens any causal link (a_i, q, a_j) in Q' 
Choose to add a_i < a_j or a_j < a_i to O
if neither choice is consistent, Fail

POP((A′, O′, Q′), agenda, actions)
```

**Agenda** =

```
{}
```

**Q** =

```
{(buy(Mercedes), Have(Mercedes), Finish)
(work, Have($), buy(Mercedes))
}
```

**O** =

```
{(buy(Mercedes) < Finish)
work < buy(Mercedes)
}
```

---

**DONE!**

```
work
```

```
buy(Mercedes)
```

```
a_N = Finish
PC : Have(Mercedes)
```
**POP Algorithm Details**

POP solves Sussman Anomaly

\[ A_0 \]

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

Init

\[ \begin{array}{ccc} A & B & C \end{array} \]

Goal

\[ \begin{array}{ccc} A & B & C \end{array} \]

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

\[ A_n \]
POP solves Sussman Anomaly

A

\(\text{On}(C,A)\) \(\text{On}(A,\text{Table})\) \(\text{On}(B,\text{Table})\) \(\text{Clear}(C)\) \(\text{Clear}(B)\) \(\text{Clear}(\text{Table})\)

Causal Links \(\in Q\)

\(\text{A}_1: \text{move}(B,\text{Table},C)\)

\(\sim \text{On}(B,\text{Table})\) \(\sim \text{Clear}(C)\) \(\text{On}(B,\text{C})\)

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

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

\(\text{A}_N\)

S. Joo (sungmoon.joo@cc.gatech.edu) 9/2/2014 17
POP Algorithm Details

\[
\text{POP}((A,O,Q), \text{agenda}, \text{actions})
\]

If agenda = ∅ then return \((A,O,Q)\)

\(a_{\text{new}} = \text{Choose}(\text{pair}) \text{ from agenda}
\)

\(a_{\text{new}} = \text{Choose}(\text{actions}) \text{ s.t. } q \in \text{Add}(a_{\text{new}})\)

If no such action \(a_{\text{new}}\) exists, \text{Fail}

\(Q' = Q \cup \langle a_{\text{new}}, q_{\text{new}} \rangle\)

\(O' = O \cup \langle a_{\text{new}}, q_{\text{new}} \rangle\)

\text{agenda}' = \text{agenda} - \langle q, a_{\text{new}} \rangle

If \(a_{\text{new}}\) is new, then \(A = A \cup a_{\text{new}}\) and

\(\forall p \in \text{PC}(a_{\text{new}}), \text{add } (p, a_{\text{new}}) \text{ to agenda}'\)

For every action \(a_{i}\) that \text{threatens} any causal link \((a_i, Q, a_j)\) in \(O'\)

\text{Choose} to add \(a_i < a_j\) or \(a_j < a_i\) to \(O\)

If neither choice is consistent, \text{Fail}

\(\text{POP}((A', O', Q'), \text{agenda}, \text{actions})\)

POP solves Sussman Anomaly

\[\begin{align*}
A_0 & \quad \text{On}(C,A) \quad \text{On}(A,Table) \quad \text{On}(B,Table) \quad \text{Clear}(C) \quad \text{Clear}(B) \quad \text{Clear}(Table) \\
A_1 & \quad \text{move}(B,Table,C) \quad \text{On}(B,Table) \quad \text{Clear}(C) \quad \text{Clear}(B) \\
A_2 & \quad \text{move}(A,Table,B) \quad \text{On}(A,Table) \quad \text{Clear}(A) \quad \text{On}(A,Table) \\
A_3 & \quad \text{move}(A,Table,B) \quad \text{On}(A,B) \quad \text{On}(A,B) \quad \text{On}(B,C) \\
A_4 & \quad \text{move}(B,Table,C) \quad \text{On}(B,Table) \quad \text{Clear}(C) \quad \text{Clear}(B) \\
A_5 & \quad \text{move}(A,Table,B) \quad \text{On}(A,Table) \quad \text{Clear}(A) \quad \text{On}(A,Table) \\
A_6 & \quad \text{move}(A,Table,B) \quad \text{On}(A,B) \quad \text{On}(A,B) \quad \text{On}(B,C)
\end{align*}\]
POP solves Sussman Anomaly

(A₀, Clear(B), A₁) threatened by A₂

PROMOTE

What's Left on our Agenda?
POP solves Sussman Anomaly

A₀

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

On(C, A)  Clear(C)  Clear(Table)

A₁: move(C, A, Table)

¬On(C, A)  On(C, Table)  Clear(A)

Clear(B)  Clear(A)

A₂: move(A, Table, B)

¬On(A, Table)  ¬Clear(B)  On(A, B)

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

Aₙ

A₀, Clear(C), A₂ threatened by A₁
POP solves Sussman Anomaly

Properties of a Planner

1) Sound: The planner produces valid plans

2) Optimal: The planner produces optimal (shortest) plans

3) Complete: The planner finds a solution when there is one or returns that the solution is not possible.
**POP Algorithm Details**

Add Helpful Actions

POP \((A, \Omega, Q, \text{agenda}, \text{actions})\)

1. If agenda=B then return \((A, \Omega, Q)\)
2. \((q, \text{agenda}) \in \text{Choose}(\text{paren})\) from agenda
3. If no such action \(\text{agenda}\) exists, Fail
4. \(Q' = Q \cup (\text{agenda} \cup \Omega_{\text{goal}})\)
5. \(\Omega' = \Omega \cup (\text{agenda} \leq \Omega_{\text{goal}})\)
6. agenda' = agenda - \((q, \text{agenda})\)

If \(\text{agenda}'\) is now, then \(A = A \cup \text{agenda}'\) and
7. \(\forall p \in \text{PC}(\text{agenda}), \text{add} (p, \text{agenda})\) to agenda'

For every action \(a_t\) that threatens any causal link \((a_i, Q, a_j)\) in \(Q'\)
8. Choose to add \(a_i < a_t\) or \(a_t < a_j\) to \(Q'\)
9. If neither choice is consistent, Fail

POP \((\text{agenda}', \Omega', Q'), \text{agenda}, \text{actions})\)

Recursion

Protect Causal Links

use \(Q\):

Demotion:
\[ a_t < a_i \]

Promotion:
\[ a_t > a_j \]

---

**Summary**

- Important Properties of Planners
  - Soundness
  - Completeness
  - Optimality

Two types of planning:

- **State Space**
  - Non-deterministic Choices: \(n = |\text{actions}|\)
  - Backtracking for goal ordering
  - Simplicity!

- **Plan Space**
  - Least Commitment
  - Non-deterministic Choices: \(n = |\text{preconditions}| + |\text{link protection}|\)
  - Smaller branching factor (goal ordering not relevant)
  - Typically more optimal
State Space vs. Plan Space

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>State Space</th>
<th>Plan Space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Progression</td>
<td>Partial Order Planning</td>
</tr>
<tr>
<td></td>
<td>Regression</td>
<td></td>
</tr>
<tr>
<td>Nodes</td>
<td>World States</td>
<td>Partial Plans</td>
</tr>
<tr>
<td>Transitions</td>
<td>Actions</td>
<td>Plan Refinement Ops</td>
</tr>
<tr>
<td></td>
<td>-Move(x,y,z)</td>
<td>-Adding Steps</td>
</tr>
<tr>
<td></td>
<td>-Load(x,y)</td>
<td>-Promotion</td>
</tr>
<tr>
<td></td>
<td>-Open(r)</td>
<td>-Demotion</td>
</tr>
</tbody>
</table>

Last Major Classical Planner: Graphplan*

- A form of state space planning: actually "Factored State Space"
- Nodes are literals and instantiated actions
- Uses Least Commitment principle to constrain search
- Simpler than Plan Space Planning
- More general than State Space Planning

http://www.cs.cmu.edu/~avrim/graphplan.html

Avrim Blum
Merrick Furst
1997
Graphplan Basics

- Graph has levels that represent time steps

- Levels alternate between preconditions and actions
  \[ P_0 \quad A_0 \quad P_1 \quad A_1 \quad P_2 \quad A_2 \ldots \]

- Precondition level \( N \) reflects what \textbf{could} be true in \( N \) steps

- Mutex Links indicate that actions / preconditions \textbf{cannot} occur together.
  Mutual Exclusion encodes the constraints on the action space.

Graph Nodes: \( P_i \) – Literals, \( A_i \) Actions

Add an action in level \( A_i \) if all of its preconditions are in \( P_i \)

Add a precondition in \( P_{i+1} \) if it is the effect of some action in \( A_i \)

\textbf{Mutex Actions:}
- One "clobbers" the other’s effects or preconditions
- They have Mutex preconditions

\textbf{Mutex Preconditions:}
- All ways of achieving them are mutex
  Mutual Exclusion encodes the constraints on the action space.
**Example: Dinner Date**

- **Init:** Garbage  CleanHands  Quiet
- **Goal:** Dinner  Present  ~ Garbage
- **Actions:**
  - cook  PC: CleanHands  EF: Dinner
  - wrap  PC: Quiet  EF: Present
  - carry  PC:  EF: ~ Garbage  ~ CleanHands
  - dolly  PC:  EF: ~ Garbage  ~ Quiet

---

**Initial Actions:** $A_0$

All actions whose preconditions are satisfied
**A₀ Mutex: CARRY Action**

 Mutex Actions:
- One “clobbers” the other’s effects or preconditions
- They have Mutex preconditions

**A₀ Mutex: All**

 Mutex Actions:
- One “clobbers” the other’s effects or preconditions
- They have Mutex preconditions
**A₀ Mutex: Complete**

Mutex Actions:
- One “clobbers” the other’s effects or preconditions
- They have Mutex preconditions

**A₁: Action  P₂: Precondition**
When can we make a plan?

- All goal conditions are satisfied
- Actions at the same level don’t interfere
- Each action’s preconditions are made true
- Is this sufficient?
  - No, but it’s necessary
- Regression search from goals to check for Mutex Conditions

Solution at Level 1?

- Build a graph
- Extract a solution (Backward Search)

Tow possible sets of actions at level 1:
- \{\text{wrap, cook, dolly}\}
- \{\text{wrap, cook, carry}\}

Neither set works – why?
Add new layer

Solution at Level 2?

Some actions sets look OK at layer 2
Solution at Level 2?

Possible action at level 1: \{cook\};\{carry, wrap\}

Possible action at level 2: \{cook\};\{carry\}

Another possible action sets:

\{cook, wrap\};\{carry\}
Observations

- Actions monotonically increase
- Propositions monotonically increase
- Mutex Conditions monotonically decrease

Graphplan Conclusions

- Combines simplicity of state-space planning with constraints of partial order planning
- Very popular method for extending and evaluating improvements
State Space vs. Plan Space vs. Graph Space

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>State Space (60s - )</th>
<th>Plan Space (70s - )</th>
<th>Graph Space (90s - )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Progression Regression</td>
<td>Partial Order Planning</td>
<td>Iterates Graph Building Regression Search</td>
<td></td>
</tr>
<tr>
<td>Nodes</td>
<td>World States</td>
<td>Partial Plans</td>
<td>Graph Levels</td>
</tr>
<tr>
<td>Transitions</td>
<td>Actions</td>
<td>Plan Refinement Operations</td>
<td>Sets of Actions</td>
</tr>
<tr>
<td>- Move(x,y,z)</td>
<td>- Adding Steps</td>
<td>- Constraint Coding</td>
<td></td>
</tr>
<tr>
<td>- Load(x,y)</td>
<td>- Promotion</td>
<td>- Mutual Exclusion</td>
<td></td>
</tr>
<tr>
<td>- Open(r)</td>
<td>- Demotion</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Modern Action Representation

Move ?b from ?x to ?y
parameters: ?b, ?x, ?y
preconditions: (and (on ?b ?x) (clear ?b) (clear ?y))
(= ?y Table)
effects: (and (on ?b ?y) (not (on ?b ?x)) (clear ?x) (not (clear ?y)))

PDDL: Planning Domain Definition Language*

S. Joo (sungmoon.joo@cc.gatech.edu) 9/2/2014 48
Expressive Actions

- Actions with variables
- Disjunctive preconditions
- Conditional effects
- Universal quantification

UCPOP [Penberthy and Weld'92]

Action Representation

Move ?b from ?x to ?y
parameters: ?b, ?x, ?y
preconditions: (and (on ?d ?x) (clear ?b) (clear ?y))
(= ?y Table)

effects: (and (on ?d ?y) (not (on ?d ?x)) (clear ?x) (not (clear ?y)))

\[ VS \]

Move ?b from ?x to ?y
parameters: ?b, ?x, ?y
preconditions: (and (on ?d ?x) (clear ?b) (clear ?y))
effects: (and (on ?d ?y) (not (on ?d ?x)) (clear ?x)
(when (= ?y Table) (not (clear ?y)))

PDDL: Planning Domain Definition Language*
Conditional Planning

- Conditional Effects
  - Effects that depend on state
  - Require special attention but same planning concept

- Disjunctive Effects
  - Different concept entirely
  - We don’t know the outcome of an action