“STRIPS” Planning

- Set of operators, where each operator has
  - Set of parameters
  - Set of preconditions
  - Set of effects, consisting of
    add effects and
    delete effects.

- Set of objects to instantiate operator’s parameters
  fully instantiated operator \( \equiv \) action

- Set of propositions representing initial state

- Set of propositions representing goals

Planning problem: Find sequence of actions that, starting in initial state, achieve all the goals
Approaches to STRIPS planning

• Search through space of world states
  – forward search,
  – regression search
  – bi-directional search
  – means-ends analysis
  – ...

• Search through space of plans
  – total order planning
  – partial order planning

• Search through planning graph
GraphPlan Approach

1. Construct a “PlanGraph” that contains all valid plans + other stuff (invalid plans)
   up to a maximum depth

2. Search PlanGraph for valid plan
   ... then return that plan
Simple Cake-Eating Domain

- **Initial**: HaveCake \( \land \neg \text{EatenCake} \)

- **Goal**: HaveCake \( \land \text{EatenCake} \)

- **Actions**:
  
  \[
  \begin{align*}
  \text{Op}_0 \quad & \text{Eat} \\
  \text{PreC}: & \text{HaveCake} \\
  \text{Eff}: & \neg \text{HaveCake} \land \text{EatenCake}
  \end{align*}
  \]

  \[
  \begin{align*}
  \text{Op}_0 \quad & \text{Bake} \\
  \text{PreC}: & \neg \text{HaveCake} \\
  \text{Eff}: & \text{HaveCake}
  \end{align*}
  \]

- **PlanGraph**

---

Graph-Plan
**Parts of a PlanGraph**

<table>
<thead>
<tr>
<th>$S_0$</th>
<th>$A_0$</th>
<th>$S_1$</th>
<th>$A_1$</th>
<th>$S_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Have(Cake)</td>
<td></td>
<td>Have(Cake)</td>
<td>Bake(Cake)</td>
<td>Have(Cake)</td>
</tr>
<tr>
<td>Eat(Cake)</td>
<td>Eaten(Cake)</td>
<td>Eat(Cake)</td>
<td></td>
<td>Eaten(Cake)</td>
</tr>
<tr>
<td>$\neg$Eaten(Cake)</td>
<td>$\neg$Eaten(Cake)</td>
<td>$\neg$Eaten(Cake)</td>
<td>$\neg$Eaten(Cake)</td>
<td>$\neg$Eaten(Cake)</td>
</tr>
</tbody>
</table>

"2-leveled" Graph $\langle S_0, A_0, S_1, A_1, \ldots \rangle$

- $S_0$: propositions in initial state
- $A_i$: each action whose preconditions all occur in level $S_{i-1}$
- $S_i$: each prop’n that is ADDED/DELETED by
  * an action in level $A_i$
  * a “No-Op” (persistence)

- **Mutex** links
  * between actions in level $A_i$
  * between propositions in level $S_i$
  "mutually exclusive"
  "cannot occur in same plan"
Mutex Conditions #1: Actions

Between 2 actions $O_1$ and $O_2$, same level $A_i$:

- **Inconsistent effects**
  
  $O_1:Eff$ negates $O_2:Eff$

  \[
  \text{EatenCake, NoOp(HaveCake)} \text{ disagree wrt "HaveCake"}
  \begin{align*}
  \text{EatenCake}:Eff & = \neg \text{HaveCake} \\
  \text{NoOp(HaveCake)}:Eff & = \text{HaveCake}
  \end{align*}
  \]

- **Interference**
  
  $O_1:Eff$ negates $O_2:PreC$

  \[
  \text{EatenCake interferes with NoOp(HaveCake)}: \\
  \begin{align*}
  \text{EatenCake}:Eff & = \neg \text{HaveCake} \\
  \text{NoOp(HaveCake)}:PreC & = \text{HaveCake}
  \end{align*}
  \]

- **Competing Needs**
  
  $O_1:PreC$ negates $O_2:PreC$

  \[
  \begin{align*}
  \text{Bake}:PreC & = \neg \text{HaveCake} \\
  \text{Eat}:PreC & = \text{HaveCake}
  \end{align*}
  \]
Mutex Conditions\#2: Propositions

Between 2 propositions $\rho_1$ and $\rho_2$, same level $S_i$:

- **Negation**
  \[ \rho_1 = \neg \rho_2 \]

- **Inconsistent Support**
  Every action achieving $\rho_1$ (from $S_{i-1}$) is mutex with every action achieving $\rho_2$

In $S_1$: 

- **HaveCake** mutex **EatenCake** as only way to achieve **HaveCake**: 
  - NoOp(HaveCake)
- is mutex with only way to achieve **EatenCake**: 
  - Eat

*N.b.:* Not mutex at $S_2$!
A valid plan is “2-leveled” graph
- two kinds of nodes
  (propositions, actions)
  alternates: proposition level, action level
- 5 kinds of edges
  * precondition \((S_i \rightarrow A_i)\)
  * add effect \((A_i \rightarrow S_{i+1})\)
  * delete effect \((A_i \rightarrow S_{i+1})\)
  * mutex-action \((A_i \leftrightarrow A_i)\)
  * mutex-prop \((S_i \leftrightarrow S_i)\)

- Include action \(O\) at action-level \(A_i\)
  if all preconditions at proposition-level \(S_i\)

- Include proposition \(\rho\) at proposition-level \(S_i\)
  if it is add/delete effect of action \(O \in A_{i-1}\)
  (including no-op actions)

**Restriction:**
Allow actions \(O_1, O_2\) at same time \(t\)
ONLY if don’t interfere with each other

- **PlanningGraph** \(\approx\) valid plan but
  **without** no-interfere restriction
function Graphplan( problem ) returns solution or failure
graph ← Initial-Planning-Graph(problem)
goals ← Goals[problem]
loop do
  if goals all non-mutex in last level of graph then do
    solution ← Extract-Solution(graph, goals, Length(graph))
    if solution ≠ failure then return solution
  else if No-Solution-Possible(graph) then return failure
  graph ← Expand-Graph(graph, problem)
end
Flat-Tire Domain

Fl= Flat; Sp= Spare; Ax= Axel; Tr= Trunk; Gr= Ground

- **Initial:** $\text{At}(\text{Fl}, \text{Ax}) \land \text{At}(\text{Sp}, \text{Tr})$

- **Goal:** $\text{At}(\text{Sp}, \text{Ax})$

- **Actions:**

  - **TakeOutSpare**
    
    $\text{Op} \left( \begin{array}{l}
    \text{TakeOutSpare} \\
    \text{PreC: } \text{At}(\text{Sp}, \text{Tr}) \\
    \text{Eff: } \neg \text{At}(\text{Sp}, \text{Tr}) \land \text{At}(\text{Sp}, \text{Gr})
    \end{array} \right)$

  - **RemoveFlat**
    
    $\text{Op} \left( \begin{array}{l}
    \text{RemoveFlat} \\
    \text{PreC: } \text{At}(\text{Fl}, \text{Ax}) \\
    \text{Eff: } \neg \text{At}(\text{Fl}, \text{Ax}) \land \text{At}(\text{Fl}, \text{Gr})
    \end{array} \right)$

  - **PutOnSpare**
    
    $\text{Op} \left( \begin{array}{l}
    \text{PutOnSpare} \\
    \text{PreC: } \text{At}(\text{Sp}, \text{Gr}) \land \neg \text{At}(\text{Fl}, \text{Ax}) \\
    \text{Eff: } \neg \text{At}(\text{Sp}, \text{Gr}) \land \text{At}(\text{Sp}, \text{Ax})
    \end{array} \right)$

  - **LeaveOverNight**
    
    $\text{Op} \left( \begin{array}{l}
    \text{LeaveOverNight} \\
    \text{PreC: } \{\} \\
    \text{Eff: } \neg \text{At}(\text{Sp}, \text{Gr}) \land \neg \text{At}(\text{Sp}, \text{Ax}) \land \neg \text{At}(\text{Sp}, \text{Tr}) \\
    \land \neg \text{At}(\text{Fl}, \text{Gr}) \land \neg \text{At}(\text{Fl}, \text{Ax})
    \end{array} \right)$
Flat-Tire in GraphPlan
Trace of GraphPlan Algorithm #1

- $S_0$: initial facts (include $\neg$-facts)

- As $\text{At}(Sp, Ax) \notin S_0$
do not call Extract-Solution

- Expand-Graph forms $A_0$ with
  * 3 “real” actions
  * 5 no-op actions;
  $S_1$ is effects

  Expand-Graph then finds
  * 4 action-mutex within $A_0$
  * 4 prop-mutex within $S_1$

- As $\text{At}(Sp, Ax) \notin S_1$
do not call Extract-Solution

- Expand-Graph forms $A_1$ with
  * 4 “real” actions
  * 7 no-op actions
  $S_2$ is effects
Mutex wrt FlatTire

- **Inconsistent Effects**
  \[
  \text{RemoveSpare} + \text{LeaveOvernight} \\
  \text{RemoveSpare: Eff} = \text{At(Sp, Gr)} \\
  \text{LeaveOvernight: Eff} = \neg \text{At(Sp, Gr)}
  \]

- **Interference**
  \[
  \text{RemoveFlat} + \text{LeaveOvernight} \\
  \text{RemoveFlat: PreC} = \text{At(Sp, Ax)} \\
  \text{LeaveOvernight: Eff} = \neg \text{At(Sp, Ax)}
  \]

- **Competing Needs**
  \[
  \text{RemoveFlat} + \text{PutOnSpare} \\
  \text{RemoveFlat: PreC} = \text{At(Fl, Ax)} \\
  \text{PutOnSpare: Eff} = \neg \text{At(Fl, Ax)}
  \]

- **Inconsistent Support**
  \[
  \text{At(Sp, Ax)} + \text{At(Fl, Ax)} \text{ in } S_2 \\
  \text{At(Sp, Ax) by } \text{PutOnSpare} \\
  \text{At(Fl, Ax) by } \text{NoOp[At(Fl, Ax)]} \\
  \text{and} \\
  \text{PutOnSpare mutex NoOp[At(Fl, Ax)]}
  \]

  (Can’t put 2 objects in same place at same time)
Trace of GraphPlan Algorithm #2

- “All” goal literals, \([\text{At}(Sp, Ax)]\), in \(S_2\)
  none are mutex . . .

- So there MAY be solution
  . . . call Extract-Solution

Extract-Solution(. . .)
  Let \(G_n\) be the GOAL at last level, \(S_n\)
  For each \(i = n..1\)
    * Let \(H_i\) be a conflict-free subset of \(A_{i-1}\),
      that covers \(G_i\) (in \(S_i\))
    * Let \(G_{i-1}\) be preconditions of \(H_i\)
  . . . until reach state in \(S_0\) satisfying all goals

Action-set \(H\) is “conflict-free”

\[\equiv\]
no pair of \(H\) are mutex, and
no pair of preconditions (in \(G\)) are mutex
Trace of Extract-Solution

- $G_2 = \{ \text{At}(\text{Sp}, \text{Ax}) \}$
  $H_2 = \{ \text{PutOnSpare} \}$

- $G_1 = \{ \text{At}(\text{Sp}, \text{Gr}), \neg \text{At}(\text{Fl}, \text{Ax}) \}$

What is $H_1$?

- Achieve $\text{At}(\text{Sp}, \text{Gr})$ by \boxed{\text{TakeOutSpare}}

- Achieve $\neg \text{At}(\text{Fl}, \text{Ax})$ by
  #1. $\text{LeaveOvernight}$
  #2. $\text{RemoveFlat}$

  But not #1, as $\text{LeaveOvernight}$ is mutex with $\text{TakeOutSpare}$

$\Rightarrow H_1 = \{ \text{TakeOutSpare}, \text{RemoveFlat} \}$

- $G_0 = \{ \text{At}(\text{Sp}, \text{Tr}), \text{At}(\text{Fl}, \text{Ax}) \}$

  As in $G_0 \subseteq S_0$, DONE!
Extending PlanGraph

Add action level $A_i$:

**ForEach** action(*) $O$

If $O$’s preconditions all true in prop-level $S_i$, and NOT mut-ex,

Then add $O$ to level $A_i$

include precondition-links
create mutex ($O$:actions-I-am-exclusive-of)

Add prop-level $S_{i+1}$:

**ForEach** effect $\rho$ of each action in action-level $A_i$

Add $\rho$ to prop-level $S_{i+1}$

Add $S \leftarrow \rho$ add- or delete- links

Mark $\rho_1, \rho_2$ as mutex if

each way of generating $\rho_1$ is mutex to
each way of generating $\rho_2$

(*) each instantiation of each operator; including “no-op”s
**Correctness**

**Graphplan is sound and complete:**
* any plan Graphplan finds is a legal plan
* if ∃ legal plan then Graphplan will find one.

| Theorem: If ∃ valid plan using \( \leq t \) time steps, then plan is subgraph of (depth-\( t \)) Planning Graph. |

If Goals not satisfiable by any valid plan, then GraphPlan will halt, w/failure, in finite time.

(extends most partial-order planners)
Leveling Off

• GraphPlan ≈ Iterative deepening
  When to stop??

• **Lemma:** If no valid plan exists, then
  ∃ a prop-level $S_n$ s.t. all future proposition
  levels are identical to $S_n$
    – Identical $\equiv$ same propositions, mutual exclusions
    – graph has “leveled off after $S_n$”

• **Corollary:** No solution exists if
  – a goal does not appear in $S_n$ or
  – $S_n$ has mutually exclusive goals

• **Subtlety:**
  \{ on(A,B), on(B,C), on(C,A) \}
Termination Condition

- Let $S_t^i$ denote set of memoized goal sets at level $i$ after an unsuccessful stage $t$

- **Theorem:** If the graph has leveled off at level $n$ and stage $t$ has passed in which $|S_{n-1}^t| = |S_n^t|$, then no valid plan exists
Termination Proof

- As PlanGraph gets deeper...
  - Literals increase monotonically
  - Actions increase monotonically
  - Mutex decrease monotonically
    * If $O_1$ and $O_2$ are mutex in $A_k$, then mutex in $A_i$ $i = 1..k$ provided $O_1, O_2 \in A_i$
    * If $\rho_1$ and $\rho_2$ are mutex in $S_k$, then mutex in $S_i$ $i = 1..k$ provided $\rho_1, \rho_2 \in S_i$

- Only finite # of actions/literals, planning graph must eventually “level off”
Experimental Results

"2 Rockets Problem"

"Link-Repeat Problem"
Accounting for Graphplan’s Efficiency

- Mutual exclusions
  (Most constraints are pair-wise mut-ex’s;
   Propagating constraints prunes large part of space.)

- Consideration of parallel plans
  (Valid parallel plans are short, wrt total plan
   \( \Rightarrow \) reduces cost of constructing pgraph, search)

- Memoizing
  (Many goal-sets appear > 1)

- Low-level costs
  (Graphplan avoid cost of instantiation during search)
Efficiency
Size of Planning Graph

**Theorem:** Consider planning problem with
- \( n \) objects,
- \( p \) propositions in initial state,
- \( m \) operators,
  each w/constant number of parameters
Let \( l \) be length of longest add list.
Then size of a \( t \)-level planning graph, and
time needed to create the graph,
are polynomial in \( n, m, p, l, \) and \( t \).

- Empirically: exclusion relations most expensive part of graph creation
  Graph creation only significant in simple problems

⇒ As graph is small,
“finding mut-ex” is hard as planning... PSpace-hard
**Comments**

- PlanGraph $\neq$ StateGraph
  
  plan $\equiv$ path in StateGraph but
  plan $\equiv$ flow in PlanGraph

- Like “Traditional TotalOrder Planner”:
  considers action at $FIXED$ time
  
  Like “Partial Order Planner”
  generates partially-ordered plans

- **Parallel Plan**: can execute many actions at once
  if no conflicts
  (eg, load all items at once)

- Guaranteed to find **SHORTEST** plan

- $\approx$ Not sensitive to given order of goals
Final Comments

• Planning $\equiv$ Searching

  $\Rightarrow$ GraphPlan

  \ldots a new approach to Planning

• Future work
  \begin{itemize}
  \item Learning (from one plan to next)
  \item Two-way search (fact$\rightarrow$goal, goal$\rightarrow$fact)
  \item beyond “Strips”-like domains
    creating objects, $\forall$, \ldots
  \item incorporating other types of constraints
  \item Why guarantee SHORTEST path?
  \end{itemize}

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

- Convert plan-situation
  (Operators, Initial/Final Conditions, ...)
  to SAT
  (Up to fixed length)

- Run WalkSat to find
  satisfying assignment \( \equiv \) plan...

  ...iterative deepening

- Plays to SatPlan’s strength,
  as \( \exists \) satisfying assignment...