Planning (Ch. 10)

WHY DID STAR WARS EPISODES 4,5,6 COME BEFORE 1,2,3?

BECAUSE IN CHARGE OF PLANNING I WAS
Forward search

Action( GoTo(x, y, z),
Precondition: At(x, y) \land Mobile(x),
Effect: \neg At(x, y) \land At(x, z))

Last time...

Initial: At(Truck, UPSD) \land Package(UPSD, P1) \land Package(UPSD, P2) \land Mobile(Truck)

Goal: Package(H1, P1) \land Package(H2, P2)

Action( Load(m, x, y),
Precondition: At(m, y) \land Package(y, x),
Effect: \neg Package(y, x) \land Package(m, x) \land At(m, y))

Action( Deliver(m, x, y),
Precondition: At(m, y) \land Package(m, x),
Effect: \neg Package(m, x) \land Package(y, x) \land At(m, y))
Heuristics for planning

Backwards search has a smaller branching factor in general, but it is hard to use heuristics. This is due to it looking at sets of states, and not a single state for the next action. For this reason, it is often better to apply a good heuristic to the dumb forward search.
Heuristics for planning

Reformulate our grilling problem as actions:

Action( MakeSandwitch(x),
Precondition: Meat(x) \land Make(Bread, x, Bread))
Effect: \neg Meat(x) \land Sandwitch(Bread)

If our goal is just “Sandwitch(Bread)”,
backtracking search would try to solve:

Meat(x) \land Make(Bread, x, Bread))
... but since “x” is still a variable, this
represents a set of states rather than one
Heuristics for planning

In “search” we had no generalize-able heuristics as each problem could be different.

Heuristics in planning are found the same way, we (1) relax the problem (2) solve it optimally.

Two generic ways to always do this are:
1. Add more actions
2. Reduce number of states
Heuristics: add actions

Multiple ways to add actions (to goal faster):

1. Ignore preconditions completely - also ignore any effects not related to goals

   This becomes set-covering problem, which is NP-hard but has P approximations

2. Ignore any deletions in effects (i.e. anything with $\implies$), also NP-hard but P approximation
Ignore preconditions

By simply removing preconditions, we allow every action to happen at every state

Action( GoTo(x, y, z),
Precondition: \( At(x, y) \land Mobile(x) \),
Effect: \(!At(x, y) \land At(x, z)\) )
Ignore preconditions

More importantly for the solution is how the Delivery action changes

The USPD can now just directly deliver to houses, so goal is:
Deliver(USPD, P1, H1) and then
Deliver(USPD, P2, H2)

Action( Deliver(m, x, y),
Precondition:
Effect: \neg Package(m, x) \land Package(y, x))
Ignore negative effects

To use this heuristic, the goal cannot have negative functions/literals (i.e. $\neg \text{At}(\text{Truck}, \text{H2})$).

This can always be rewritten to something else (for above $\text{At}(\text{Truck}, \text{USPD}) \lor \text{At}(\text{Truck}, \text{H1}) \lor \text{At}(\text{Truck}, \text{Truck})$).

Action( $\text{GoTo}(x, y, z)$,
Precondition: $\text{At}(x, y) \land \text{Mobile}(x)$,
Effect: $\neg \text{At}(x, y) \land \text{At}(x, z)$)

Action( $\text{GoTo}(x, y, z)$,
Precondition: $\text{At}(x, y) \land \text{Mobile}(x)$,
Effect: $\text{At}(x, z)$)
Ignore negative effects

For the UPS delivery example, it does not help us find a solution faster (min is 6 still)

However, there are many more solutions as every action “copies” instead of “moves”

For example, a solution could be: Move, Move, Load, Load, Deliver, Deliver

This is possible as truck exists at all 3 spots!
After 2 moves... then load...
After 2 moves... then load...
Heuristics: group states

Group similar states together into “super states” and solve the problem within “super states” separately (divide & conquer)

An admissible but bad heuristic would be the maximum of all “super states” individual solutions (but this is often poor)

A possibly non-admissible would be the sum of all “super states” (need independence)
Heuristics: group states

These “super states” can be created in many ways:

1. Delete relations/fluents (e.g. no more “At”)
2. Merge objects/literals (e.g. merge UPSD and Truck)

You then need to solve two problems:
1. Between the abstract “super states”
2. Within each “super state”
Heuristics: group states

Consider if there were 3 houses, but only two needed packages

We could remove all “At”s for this third house, as we can easily abstract it away

In this case the “super state” solution is the actual solution as there is no need to add back in a third house
Heuristics: group states

For example, if we were instead delivering 3 packages, 1 to H1 and 2 to H2...

We combine the two packages for H2 into a single “super package” with only one load and deliver (overall “super state” solution)

We then can simply see that each load/deliver corresponds to two individual loads/delivers (within super state solution)
CHANGE THE TOPIC
A heuristic we will go over in detail is graph planning, which tries to do all possible actions at each step.

The graph plan heuristic is nice because it is always admissible and computable in P time.

The basic idea of graph plan is to track all the statements that could be true at any time.
Graph Plan

Graph plan is an underestimate because once a relation/literal is added, it is never removed.

Unlike the “remove negative effects” heuristic, we allow both negative and positive effects.

But we can also use any preconditions that have been found anytime before (not quite as open as completely removing them).
Graph Plan

These simplifications/relaxations probably make the problem too easy

So we also track pairs of both actions and literals that are in conflict (called mutexes)

First, let's go over how to convert actions and relations into graph plan, then later we will add in the mutexes
Graph plan will alternate between possible facts ("state level") and actions ("action level").

Initial state:

- P
- \neg Q
- R
- S
Graph Plan

You start with the relations of the initial state on the left (now explicitly stating negatives).

Then you add “no actions” which simply keep all the relationships the same but move them to the right.

Then you add actions, which you do by linking preconditions on the left to resulting effects on the right (adding any new ones).
Graph Plan

Consider this problem:
Initial: $Sleepy(me) \land Hungry(me)$
Goal: $\neg Sleepy(me) \land \neg Hungry(me)$

Action($Eat(x)$),
Precondition: $Hungry(x)$,
Effect: $\neg Hungry(x)$)

Action($Coffee(x)$),
Precondition: ,
Effect: $\neg Sleepy(x)$)

Action($Sleep(x)$),
Precondition: $Sleepy(x) \land \neg Hungry(x)$,
Effect: $\neg Sleepy(x) \land Hungry(x)$)
Consider this problem:
Graph Plan

Each set of relations/literals are what we call levels of the graph plan, $S = \text{states}, \ A = \text{actions}$

State level 0 is $S_0 = \{H, S\}$
$A_0 = \{C, E, \text{all “no ops”}\}$
$S_1 = \{H, \neg H, S, \neg S\}$
$A_1 = \{C, E, S\!l, \text{all “no ops”}\}$
$S_2 = \{H, \neg H, S, \neg S\}$
Graph Plan

You do it! (show 3 state and 2 action levels)

Initial: \( \neg \text{Money}(me) \land \neg \text{Smart}(me) \land \neg \text{Debt}(me) \)

Goal: \( \text{Money}(me) \land \text{Smart}(me) \land \neg \text{Debt}(me) \)

Action( School\((x)\),  
Precondition: ,  
Effect: Debt\((x) \land \text{Smart}(x)\))  

Action( Job\((x)\),  
Precondition: ,  
Effect: Money\((x) \land \neg \text{Smart}(x)\))  

Action( Pay\((x)\),  
Precondition: Money\((x)\),  
Effect: \( \neg \text{Money}(x) \land \neg \text{Debt}(x) \))
Graph Plan

The graph plan allows multiple actions to be done in a single turn, which is why $S_1$ has both $\neg\text{Sleepy}(\text{me})$ and $\neg\text{Hungry}(\text{me})$

You keep building the graph until either:
(1) You find your goal (more on this later)
(2) The graph converges (i.e. states, actions and mutexes stop changing)
A mutex are two things that cannot be together (i.e. cannot happen or be true simultaneously)

You can put mutexes:
1. Between two relationships/literals
2. Between actions

There are different rules for doing mutexes between actions vs. relations
Mutexes: actions

For all of these cases I will assume actions two actions: A1 and A2

These actions have preconditions and effects: Pre(A1) and Effect(A1), respectively

For example, I will abbreviate below as:

Action( Eat(x),
Precondition: Hungry(x),
Effect: ¬Hungry(x))

A1 = Eat
H ∈ Pre(A1)
¬H ∈ Effect(A1)
Mutexes: actions

**Mutex Action rules:**

1. \( x \in Effect(A1) \land \neg x \in Effect(A2) \)
2. \( x \in Pre(A1) \land \neg x \in Effect(A2) \)
3. \( x \in Pre(A1) \land \neg x \in Pre(A2) \)
Mutexes: actions

Mutex Action rules:

1. $x \in Effect(A1) \land \neg x \in Effect(A2)$
2. $x \in Pre(A1) \land \neg x \in Effect(A2)$
3. $x \in Pre(A1) \land \neg x \in Pre(A2)$
Mutexes: actions

Mutex Action rules:

1. $x \in \text{Effect}(A1) \land \neg x \in \text{Effect}(A2)$
2. $x \in \text{Pre}(A1) \land \neg x \in \text{Effect}(A2)$
3. $x \in \text{Pre}(A1) \land \neg x \in \text{Pre}(A2)$
Mutexes: actions

Mutex Action rules:

1. $x \in \text{Effect}(A1) \land \neg x \in \text{Effect}(A2)$
2. $x \in \text{Pre}(A1) \land \neg x \in \text{Effect}(A2)$
3. $x \in \text{Pre}(A1) \land \neg x \in \text{Pre}(A2)$
Mutexes: actions

Mutex Action rules:

1. \( x \in \text{Effect}(A1) \land \neg x \in \text{Effect}(A2) \)
2. \( x \in \text{Pre}(A1) \land \neg x \in \text{Effect}(A2) \)
3. \( x \in \text{Pre}(A1) \land \neg x \in \text{Pre}(A2) \)
Mutex Action rules:

1. $x \in \text{Effect}(A1) \land \neg x \in \text{Effect}(A2)$
2. $x \in \text{Pre}(A1) \land \neg x \in \text{Effect}(A2)$
3. $x \in \text{Pre}(A1) \land \neg x \in \text{Pre}(A2)$
Mutexes: actions

Mutex Action rules:

1. \( x \in \text{Effect}(A1) \land \neg x \in \text{Effect}(A2) \)
2. \( x \in \text{Pre}(A1) \land \neg x \in \text{Effect}(A2) \)
3. \( x \in \text{Pre}(A1) \land x \in \text{Pre}(A2) \)

\[ 
\begin{align*}
H & \quad \text{E} & \quad H \\
\text{E} & \quad \text{S} & \quad \text{E} \\
\text{C} & \quad \text{S} & \quad \text{C} \\
S & \quad \text{S} & \quad S
\end{align*} 
\]
 Mutexes: states

There are 2 rules for states, but unlike action-mutexes they can change across levels

1. Opposite relations are mutexes ($x$ and $\neg x$)
2. If there are mutexes between all possible actions that “lead” to a pair of states...

Two ways that “leading” can be in mutex:
1. Actions are in mutex
2. Preconditions of action pair are in mutex
Another way to compute state mutexes:

1. Add mutexes between all pairs in state
2. If any pair of actions can lead to this pair of relationships, un-mutex them

Recap:
If any valid pair of actions = no mutex
All ways of reaching invalid = mutex
Mutexes: states

1. Opposite relations are mutexes (x and ¬x)
2. If there are mutexes between all possible actions that lead to a pair of states
1. Opposite relations are mutexes ($x$ and $\neg x$)
2. If there are mutexes between all possible actions that lead to a pair of states
Mutexes: states

1. Opposite relations are mutexes (x and \( \lnot x \))
2. If there are mutexes between all possible actions that lead to a pair of states

None... but if we remove coffee...
1. Opposite relations are mutexes (x and \( \neg x \))

2. If there are mutexes between all possible actions that lead to a pair of states

Sl has mutex with both E and NoOp(\( \neg H \))

This mutex will be gone on the next level (as you can eat again)
1. Opposite relations are mutexes (x and \( \neg x \))
2. If there are mutexes between all possible actions that lead to a pair of states
Mutexes: actions

You do it!

Initial: $\neg Money(me) \land \neg Smart(me) \land \neg Debt(me)$

Goal: $(me) \land Smart(me) \land \neg Debt(me)$

Action( School($x$),
Precondition: ,
Effect: Debt($x$) $\land$ Smart($x$))

Action( Job($x$),
Precondition: ,
Effect: Money($x$) $\land$ $\neg$Smart($x$))

Action( Pay($x$),
Precondition: Money($x$),
Effect: $\neg$Money($x$) $\land$ $\neg$Debt($x$))
Mutexes: actions
Mutexes: actions

Non-trivial mutexes:
(SC, P),
(J, P),
(SC, J),
(P, D&M),
(SC, D&M),
(J, M&S)
GraphPlan can be computed in $O(n(a+l)^2)$, where $n =$ levels before convergence, $a =$ number of actions, $l =$ number of relations/literals/states (square is due to needing to check all pairs).

The original planning problem is PSPACE, which is known to be harder than NP.
GraphPlan: states

Let's consider this problem:
Initial: \(\text{Clean} \land \text{Garbage} \land \text{Quiet}\)
Goal: \(\text{Food} \land \neg \text{Garbage} \land \text{Present}\)

<table>
<thead>
<tr>
<th>Action</th>
<th>Precondition</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\text{MakeFood, Clean, Food}))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>((\text{Wrap, Quiet, Present}))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>((\text{Takeout, Garbage, } \neg \text{Garbage} \land \neg \text{Clean}))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>((\text{Dolly, Garbage, } \neg \text{Garbage} \land \neg \text{Quiet}))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
GraphPlan: states
Mutexes

Possible state pairs:

- F, C
- C, Q
- F, ┐C
- C, ┐Q
- F, C
- C, ┐Q
- F, G
- C, P
- F, ┐G
- ┐C, G
- F, Q
- ┐C, ┐G
- F, ┐Q
- ┐C, Q
- F, P
- ┐C, ┐Q
- C, ┐C
- ┐C, P
- C, G
- ┐C, ┐G
- C, G
- ... (more)
Mutexes

Make one more level here!
Blue mutexes disappear

Mutexes

Pink = new mutex
GraphPlan as heuristic

GraphPlan is optimistic, so if any pair of goal states are in mutex, the goal is impossible.

3 basic ways to use GraphPlan as heuristic:
(1) Maximum level of all goals
(2) Sum of level of all goals (not admissible)
(3) Level where no pair of goals is in mutex

(1) and (2) do not require any mutexes, but are less accurate (quick 'n' dirty)
GraphPlan as heuristic

For heuristics (1) and (2), we relax as such:
1. Multiple actions per step, so can only take fewer steps to reach same result
2. Never remove any states, so the number of possible states only increases

This is a valid simplification of the problem, but it is often too simplistic directly
GraphPlan as heuristic

Heuristic (1) directly uses this relaxation and finds the first time when all 3 goals appear at a state level

(2) tries to sum the levels of each individual first appearance, which is not admissible (but works well if they are independent parts)

Our problem: goal=\{Food, \neg Garbage, Present\}
First appearance: F=1, \neg G=1, P=1
GraphPlan: states

Level 0:
- C
- G
- Q
- W

Level 1:
- M
- T
- D
- F

Heuristic (1):
Max(1,1,1) = 1

Heuristic (2):
1+1+1=3
GraphPlan as heuristic

Often the problem is too trivial with just those two simplifications

So we add in mutexes to keep track of invalid pairs of states/actions

This is still a simplification, as only impossible state/action pairs in the original problem are in mutex in the relaxation
GraphPlan as heuristic

Heuristic (3) looks to find the first time none of the goal pairs are in mutex

For our problem, the goal states are: (Food, \( \neg \) Garbage, Present)

So all pairs that need to have no mutex: (F, \( \neg \) G), (F, P), (\( \neg \) G, P)
None of the pairs are in mutex at level 1

This is our heuristic estimate
Finding a solution

GraphPlan can also be used to find a solution:
(1) Converting to a Constraint Sat. Problem
(2) Backwards search

Both of these ways can be run once GraphPlan has all goal pairs not in mutex (or converges)

Additionally, you might need to extend it out a few more levels further to find a solution (as GraphPlan underestimates)
GraphPlan as CSP

Variables = states, Domains = actions out of
Constraints = mutexes & preconditions

Variables: $G_1, \ldots, G_4, P_1 \ldots P_6$

Domains:
- $G_1: \{A_1\}$, $G_2: \{A_2\}$
- $G_3: \{A_3\}$, $G_4: \{A_4\}$
- $P_1: \{A_5\}$, $P_2: \{A_6, A_{11}\}$
- $P_3: \{A_7\}$, $P_4: \{A_8, A_9\}$
- $P_5: \{A_{10}\}$, $P_6: \{A_{10}\}$

Constraints (normal):
- $P_1 = A_5 \Rightarrow P_4 \neq A_9$
- $P_2 = A_6 \Rightarrow P_4 \neq A_8$
- $P_2 = A_{11} \Rightarrow P_3 \neq A_7$

Constraints (Activity):
- $G_1 = A_1 \Rightarrow Active\{P_1, P_2, P_3\}$
- $G_2 = A_2 \Rightarrow Active\{P_4\}$
- $G_3 = A_3 \Rightarrow Active\{P_5\}$
- $G_4 = A_4 \Rightarrow Active\{P_1, P_6\}$

Init State: $Active\{G_1, G_2, G_3, G_4\}$

(a) Planning Graph

(b) DCSP

from Do & Kambhampati
Finding a solution

For backward search, attempt to find arrows back to the initial state (without conflict/mutex).

This backwards search is similar to backward chaining in first-order logic (depth first search).

If this fails to find a solution, mark this level and all the goals not satisfied as: (level, goals).

(level, goals) stops changing, no solution.
Graph Plan

Remember this...
Initial: \( \neg \text{Money}(me) \land \neg \text{Smart}(me) \land \neg \text{Debt}(me) \)
Goal: \( (me) \land \text{Smart}(me) \land \neg \text{Debt}(me) \)

Action( School\((x)\),
Precondition: ,
Effect: \( \text{Debt}(x) \land \text{Smart}(x) \) )

Action( Job\((x)\),
Precondition: ,
Effect: \( \text{Money}(x) \land \neg \text{Smart}(x) \) )

Action( Pay\((x)\),
Precondition: \( \text{Money}(x) \),
Effect: \( \neg \text{Money}(x) \land \neg \text{Debt}(x) \) )
Ask: \( D^\wedge S^\wedge M \)
Find first no mutex...
Ask: D^S^M
... then back search

Error! Actions 1&4 in mutex

1. 2. 3. 4.
Ask: try different back path...

Graph Plan

Error states in mutex

1. try different back path...

2. Error states in mutex

3. Error states in mutex

4. Error states in mutex
Ask: \[ D^S M \]

found solution!
Finding a solution

Formally, the algorithm is:

\[
\begin{align*}
&\text{graph} = \text{initial} \\
&\text{noGoods} = \text{empty table (hash)} \\
&\text{for level} = 0 \text{ to infinity} \\
&\quad \text{if all goal pairs not in mutex} \\
&\quad \quad \text{solution} = \text{DFS with noGoods} \\
&\quad \quad \text{if success, return paths} \\
&\quad \text{if graph} \& \text{noGoods converged, return fail} \\
&\text{graph} = \text{expand graph}
\end{align*}
\]
Mutexes

You try it!