Planning (Ch. 10)

Somehow, I don't think you thought your cunning plan all the way through.
Announcements

Writing 4: due today!

Writing 2:  
  Resubmission due Monday, April 18  
  Last day to talk to Gabriel, April 15
Backward search

When computing the “reverse action”, you apply the negated effects before preconditions:

Action( $\text{ Deliver}(m, x, y)$, 
Precondition: $\text{ At}(m, y) \land \text{ Package}(m, x)$,  
Effect: $\neg\text{ Package}(m, x) \land \text{ Package}(y, x) \land \text{ wedge At}(m, y)$)

\[ \text{ Action}( \text{ Deliver}^{-1}(m, x, y), \]
\[ \text{ Precondition: } \neg \text{ Package}(m, x) \land \text{ Package}(y, x) \land \text{ At}(m, y) \]
\[ \text{ Effect: } \text{ At}(m, y) \land \text{ Package}(m, x) \land \text{ Package}(m, x) \land \neg \text{ Package}(y, x) \land \neg \text{ At}(m, y) \]

First apply blue, then apply red
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 \( \Rightarrow \)), also NP-hard but P approximation
Ignore preconditions

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

\[
\text{Action( } GoTo(x, y, z), \\
\text{Precondition: } \neg At(x, y) \land Mobile(x), \\
\text{Effect: } \neg At(x, y) \land \neg 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( \textit{Deliver}(m, x, y),

Precondition:

Effect: \neg \textit{Package}(m, x) \land \textit{Package}(y, x))
Ignore negative effects

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

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

$$\text{Action}( \text{GoTo}(x, y, z),$$

Precondition: $At(x, y) \land Mobile(x)$,

Effect: $\neg At(x, y) \land At(x, z)$

$$\text{Action}( \text{GoTo}(x, y, z),$$

Precondition: $At(x, y) \land Mobile(x)$,

Effect: $At(x, z)$)
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!
At(Truck, UPSD) \land At(Truck, H1) \land At(Truck, H2) \\
\land Package(UPSD, P1) \land Package(UPSD, P2) \land Mobile(Truck)

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)

A 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. In 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)
A heuristic we will go over in detail is graph planning, which is almost a merge of the two action heuristics from earlier.

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).
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

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: $\neg Sleepy(x) \land Hungry(x)$)
Graph Plan

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 \( S_0 = \{H, S\} \)
\( A_0 = \{C, E, \text{all “no ops”}\} \)
\( S_1 = \{H, \neg H, S, \neg S\} \)
\( A_1 = \{C, E, Sl, \text{all “no ops”}\} \)
\( S_2 = \{H, \neg H, S, \neg S\} \)
Graph Plan

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

You would keep building the graph until the states, actions and mutexes converge (i.e. stop changing)

The mutexes represent what two actions cannot be done together within a level
Mutexes

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

This implies that having both of these actions/relations are impossible simultaneously.

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:

\[
\text{Action}(\text{Eat}(x), \text{Precondition: Hungry}(x), \text{Effect: } \neg\text{Hungry}(x))
\]

\[
A1 = \text{Eat} \\
\neg H \in Effect(A1) \\
H \in Pre(A1)
\]
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. $x \in Effect(A1) \land \neg x \in Effect(A2)$
2. $x \in Pre(A1) \land \neg x \in Effect(A2)$
3. $x \in Pre(A1) \land \neg x \in Pre(A2)$
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

You do it!
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($\text{School}(x)$),
Precondition: ,
Effect: $\text{Debt}(x) \land \text{Smart}(x)$
Action($\text{Job}(x)$),
Precondition: ,
Effect: $\text{Money}(x) \land \neg \text{Smart}(x)$
Action($\text{Pay}(x)$),
Precondition: $\text{Money}(x)$,
Effect: $\neg \text{Money}(x) \land \neg \text{Debt}(x)$
Mutexes: actions
Mutexes: states

There are 2 (easier) 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

Can rephrase second rule: All pairs of states start with a mutex, but remove mutex if there are un-mutexes actions that lead to state pair
Mutexes: 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 \( \neg x \))
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

A Graph plan converges when the next level:
1. Has the same relations/states
2. No change in mutexes

At this point, you have generated the full graph plan

You try adding mutexes to the previous in-class activity!
Mutexes: actions

Non-trivial mutexes:
(SC, P),
(J, P),
(SC, J),
(P, \lnot D & M),
(SC, \lnot D & \lnot S),
(J, \lnot M & S)