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: \(At(\text{Truck}, \text{UPSD}) \land Package(\text{UPSD}, \text{P1})\) ^ \(\land Package(\text{UPSD}, \text{P2}) \land Mobile(\text{Truck})\)

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

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: \(\neg Package(m, x) \land Package(y, x) \land At(m, y)\))
At(Truck, UPSD) \land Package(UPSD, P1) \land Package(UPSD, P2) \land Mobile(Truck)
\( \text{At(Truck, UPSD) \land Package(UPSD, P1)} \land \text{Package(UPSD, P2) \land Mobile(Truck)} \)

\[
\text{Action( Load(m, x, y),}
\]

Precondition: \( \text{At(m, y) \land Package(y, x),} \)

Effect: \( \neg \text{Package(y, x) \land Package(m, x)} \)
At(Truck, UPSD) \land Package(UPSD, P1) \\
\land Package(UPSD, P2) \land Mobile(Truck)

Find match m/Truck x/P1, y/UPSD

Action( Load(m, x, y), \\
Precondition: At(m, y) \land Package(y, x), \\
Effect: \neg Package(y, x) \land Package(m, x))
\( \text{At(Truck, UPSD)} \land \text{Package(UPSD, P1)} \land \text{Package(UPSD, P2)} \land \text{Mobile(Truck)} \land \text{Package(Truck, P1)} \)
At(Truck, UPSD) \land Package(Truck, P1) \land Package(UPSD, P2) \land Mobile(Truck)

Action( Load(Truck, P2, UPSD),
Precondition: At(Truck, UPSD) \land Package(UPSD, P2),
Effect: \neg Package(UPSD, P2) \land Package(Truck, P1))
\(\text{At(Truck, UPSD)} \land \text{Package(Truck, P1)} \land \text{Package(Truck, P2)} \land \text{Mobile(Truck)}\)

\[
\text{Action( GoTo(Truck, UPSD, H1), Precondition: At(Truck, UPSD) } \land \text{Mobile(Truck), Effect: } \neg \text{At(Truck, UPSD) } \land \text{At(Truck, H1))}
\]
At(Truck, H1) \land Package(Truck, P1) \\
\land Package(Truck, P2) \land Mobile(Truck)

Action\(\) Deliver(Truck, P1, H1),
Precondition: At(Truck, H1) \land Package(Truck, P1),
Effect: \(\neg\)Package(Truck, P1) \land Package(H1, P1)
Action( GoTo(Truck, H1, H2),
Precondition: At(Truck, H1) \land Mobile(Truck),
Effect: \neg At(Truck, H1) \land At(Truck, H2))
\[ \text{At}(\text{Truck}, H2) \land \text{Package}(H1, P1) \land \text{Package}(\text{Truck}, P2) \land \text{Mobile}(\text{Truck}) \]

**Action** (Deliver(Truck, P2, H2),

**Precondition:** \( \text{At}(\text{Truck}, H2) \land \text{Package}(\text{Truck}, P2), \)

**Effect:** \( \neg \text{Package}(\text{Truck}, P2) \land \text{Package}(H2, P2) \)
At(Truck, H2) \land \text{Package}(H1, P1) \\
\land \text{Package}(H2, P2) \land \text{Mobile}(\text{Truck})
While the solution might seem obvious to us, the search space is (surprisingly) quite large. The brute force way (forward search) simply looks at all valid actions from the current state. We can then search it in using BFS (or iterative deepening) to find fewest action cost goal.
Forward search

GoTo(Truck, USPD)

At(USPD)

At(H1)

At(H2)

At(USPD) ^ Package(P1)

Load(USPD, P1, USPD)

can ignore
Forward search

Actions: 3 (2 unique ones, as Deliver = Load)
Objects: 6 (Truck, USPD, H1, H2, P1, P2)
Min moves to goal: 6 (L, L, G, D, G, D)

Despite this problem being simplistic, the branching factor is about 4 to 5 (even with removing redundant actions)

This means we could search around 10,000 states before we found the goal
Forward search

This search is actually much more than the number of states due to redundant paths

Package() can be: UPSD, Truck, H1, H2
At() can be: UPSD, Truck, H1, H2, P1, P2

There are 2 packages for Package()
There is 1 truck for Truck()

So total states = $4^2 * 6 = 96$
Backward search

Backward search is also similar to FO's backward search

Start at goal and do actions in reverse (swap effect and precondition), except substitute:

| Action( GoTo(x, y),  
|  | Precondition:  At(x),  
|  | Effect: ¬At(x) ∧ At(y)) |
| Action⁻¹( GoTo(x, y),  
|  | Precondition: ¬At(x) ∧ At(y),  
|  | Substitute: At(x)) |

Example:

Goal = At(Home)  
Initial = At(Class)  
Unify: {x/Home,  
  y/Class} ... done
If applying the substitution is more difficult, you can convert by:
1. Apply effect to precondition
2. Negate effects, add original precondition

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

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

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

Unify: 
\begin{align*}
\text{m/Truck,} \\
\text{x/P2, y/H2}
\end{align*}
\[
\text{Action}(\text{Deliver}(m, x, y)^{-1}, \\
\text{Precondition: } \text{At}(m, y) \land \text{Package}(y, x)) \\
\text{Effect: } \text{At}(m, y) \land \text{Package}(m, x) \land \neg \text{Package}(y, x)
\]
At(Truck, H2) ∧ Package(H1, P1) ∧ Package(Truck, P2) ∧ Mobile(Truck)

Try to continue from here!

Action( Deliver(m, x, y),
Precondition: At(m, y) ∧ Package(m, x),
Effect: ¬Package(m, x) ∧ Package(y, x) ∧ At(m, y))

Action( Load(m, x, y),
Precondition: At(m, y) ∧ Package(y, x),
Effect: ¬Package(y, x) ∧ Package(m, x) ∧ At(m, y))

Action( GoTo(x, y, z),
Precondition: At(x, y) ∧ Mobile(x),
Effect: ¬At(x, y) ∧ At(x, z))
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:

\[
\text{Action( } Make\text{Sandwich}(x), \\
\text{Precondition: } Meat(x) \land Make(Bread, x, Bread)) \\
\text{Effect: } \neg Meat(x) \land Sandwitch(Bread)
\]

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

Action( $GoTo(x, y, z)$,
Precondition:
Effect: $\neg At(x, y) \land At(x, z)$)
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. $\neg \text{At(Truck, H2)}$)

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

Action( 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( 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)

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. 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)