Planning (Ch. 10)

PLANNING
Somehow, I don't think you thought your cunning plan all the way through.
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)
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 will alternate between possible facts ("state level") and actions ("action level").

Initial state

Graph Plan

state level 0 =

Prop. Level 0  Action Level 1  Prop. Level 2  Action Level 3  Prop. Level 4

P
¬Q
R
S
Graph Plan

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

- **Action**($\text{Eat}(x)$),
  - **Precondition**: $\text{Hungry}(x)$,
  - **Effect**: $\neg\text{Hungry}(x)$

- **Action**($\text{Coffee}(x)$),
  - **Precondition**: ,
  - **Effect**: $\neg\text{Sleepy}(x)$

- **Action**($\text{Sleep}(x)$),
  - **Precondition**: $\text{Sleepy}(x) \land \neg\text{Hungry}(x)$,
  - **Effect**: $\neg\text{Sleepy}(x) \land \text{Hungry}(x)$
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 $S_0 = \{H, S\}$

$A_0 = \{C, E, \text{all “no ops”}\}$

$S_1 = \{H, \neg H, S, \neg S\}$

$A_1 = \{C, E, Sl, \text{all “no ops”}\}$

$S_2 = \{H, \neg H, S, \neg S\}$
Graph Plan

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

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

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

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: \( \neg Money(x) \land \neg Debt(x) \))
Graph Plan

\[
\begin{align*}
D & \rightarrow D & S & \rightarrow S & M & \rightarrow M \\
D & \rightarrow D & S & \rightarrow S & M & \rightarrow M \\
S & \rightarrow S & M & \rightarrow M & M & \rightarrow M
\end{align*}
\]
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 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)
Mutexes

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:

\[
\begin{align*}
\text{Action( } & \text{Eat}(x), \\
\text{Precondition: } & \text{Hungry}(x), \\
\text{Effect: } & \neg \text{Hungry}(x) \end{align*}
\]
\[
\begin{align*}
A1 = \text{Eat} \\
\neg H & \in \text{Effect}(A1) \\
H & \in \text{Pre}(A1)
\end{align*}
\]
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. $x \in Effect(A1) \land \neg x \in Effect(A2)$
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. \( x \in Effect(A1) \land \neg x \in Effect(A2) \)
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: 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.
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
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

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

\[ H \rightarrow S \]
\[ S \rightarrow H \]

\[ H \rightarrow E \]
\[ E \rightarrow H \]

\[ S \rightarrow \neg S \]
\[ \neg S \rightarrow S \]

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

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

D

Sc

S

P

J

M

D

Sc

S

P

J

M

M

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

Action: $(\text{MakeFood, Clean}, \text{Food})$
Action: $(\text{Takeout, Garbage, } \neg \text{Garbage} \land \neg \text{Clean})$

Action: $(\text{Wrap, Quiet, Present})$
Action: $(\text{Dolly, Garbage, } \neg \text{Garbage} \land \neg \text{Quiet})$
GraphPlan: states
Mutexes

Possible state pairs:

- F, C
- C, Q
- F, ┐C
- C, ┐Q
- F, C
- C, P
- F, ┐C
- C, ┐Q
- F, C
- C, G
- F, ┐Q
- ┐C, Q
- F, P
- ┐C, ┐Q
- C, ┐C
- ┐C, ┐Q
- C, G
- ┐C, ┐Q
- 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)
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, Garbage, Present}  
First appearance: F=1, G=1, P=1
GraphPlan: states

Level 0:
- C
- G
- Q
- M

Level 1:
- C
- G
- Q
- 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, ¬Garbage, Present)

So all pairs that need to have no mutex: (F, ¬G), (F, P), (¬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 CSP
(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

(a) Planning Graph

(b) DCSP from Do & Kambhampati

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

Init State: Active\{G_1, G_2, G_3, G_4\}
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 from last time...

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

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

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: \( \neg Money(x) \land \neg Debt(x) \) )
Ask: \( \neg D \wedge \neg S \wedge \neg M \)  
Find first no mutex...
Ask:

Ask: \( D \uparrow S \uparrow M \)

... then back search

Error!

actions in mutex

1. 2. 3.

4. 5. 6.
Ask: 

\[ \mathbb{D} \wedge \mathbb{S} \wedge \mathbb{M} \]

try different back path...

Graph Plan

Error states in mutex

1.

2.

3.

4.
Ask: \( D^S M \)

found solution!
Formally, the algorithm is:

graph = initial
oxGoods = empty table (hash)
for level = 0 to infinity
  if all goal pairs not in mutex
    solution = DFS with noGoods
    if success, return paths
  if graph & noGoods converged, return fail
graph = expand graph
Mutexes

You try it!