Planning (Ch. 10)

Don't bother me.

I'm trying to remember Plan B.
Announcements

Homework 5 due date is Friday

Writing 3 done:
Talk to Gabriel by April 29
Resubmission due May 2
GraphPlan

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 $\mathcal{P}$ time.

The basic idea of graph plan is to track all the statements that could be true at any time and allow multiple actions.
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
We start with just the initial state in GraphPlan. Any actions that are possible (preconditions met) have their effects added. We also allow a “persistence effect”, which indicates the state simply did not change. Repeatedly applying this results in a list of all achievable states.
Today we will consider this problem:

**Initial:** $\text{Clean} \land \text{Garbage} \land \text{Quiet}$

**Goal:** $\text{Food} \land \neg \text{Garbage} \land \text{Present}$

**Action:** $(\text{MakeFood}, \text{Clean}, \text{Food})$

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

**Action:** $(\text{Wrap}, \text{Quiet}, \text{Present})$

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

C → M → F → C
G → T → G
Q → D → Q
W → P → Q
 Mutexes

Between actions:
If any combination of preconditions/effects result in opposite states (or preconditions in mutex already)

Between states:
Always between opposite states (i.e. P and \( \neg P \))
Also when the only way to reach the states are in mutex already (by other conditions)
Mutexes

Possible state pairs:

F, C  C, Q
F, ┐C  C, ┐Q
F, G  C, P
F, ┐G  ┐C, G
F, Q  ┐C, ┐G
F, P  ┐C, ┐Q
C, ┐C  ┐C, ┐Q
C, G  ┐C, P
C, ┐G  ... (more)
GraphPlan: convergence

This process is continued until both the mutexes and states no longer change.

Once this happens, we have enough to both:
1. Compute a heuristic (next)
2. Try and generate a solution (after next)

First, compute one more level of the previous problem.
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, Garbage, Present}  
First appearance: F=1, Garbage=1, P=1
GraphPlan: states

Level 0:
- C
- G
- Q
- W
- D

Level 1:
- M
- T
- G
- D
- Q

Heuristic (1):
$\text{Max}(1,1,1) = 1$

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

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 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:

\[D \land S \land M\]

Find first

no mutex...

Graph Plan
Ask: \[D \lor S \lor M\] ... then back search.

Error! actions in mutex.
Ask:

\[ D \wedge S \wedge M \]

try different back path...

1. Error states in mutex

2. 2.

3. 3.

4. 4.
Ask: \[ D \circlearrowleft S \circlearrowleft M \]
found solution!
Finding a solution

Formally, the algorithm is:

graph = initial
noGoods = 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!