Planning (Ch. 10)

Artificial Intelligence

what people think it is

what amateur programmers think it is

what actually it is
GraphPlan: states

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

Action: (MakeFood,
Precondition: Clean,
Effects: Food)

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

Action: (Wrap,
Precondition: Quiet,
Effects: Present)

Action: (Dolly,
Precondition: Garbage,
Effects: $\neg\text{Garbage} \land \neg\text{Quiet}$)
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, ¬G=1, P=1
GraphPlan: states

Level 0:

C → M
G → T
Q → W

Level 1:

C → C
G → G
Q → Q

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

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).

Start by finding actions that satisfy all goal conditions, then recursively try to satisfy all of the selected actions’ preconditions.

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

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

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

Action( Pay, 
Precondition: \( Money \), 
Effect: \( \neg Money \land \neg Debt \) )
Ask:

\[ D^S \land M \]

Find first

no mutex...
Ask: \( \downarrow D \wedge S \wedge \downarrow M \)

... then back search

Graph Plan

Error! States of 1&4 in mutex

Error: States of 1&4 in mutex
Ask: \[ D \wedge S \wedge M \]

try different back path...

Error, actions 3&4 in mutex

1. try different back path...

2. try different back path...

3. try different back path...

4. try different back path...
Ask: \( D \wedge S \wedge M \)

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 = recursive search with noGoods
        if success, return paths
    if graph & noGoods converged, return fail
graaph = expand graph
```
Initial: $\text{Clean} \land \text{Garbage} \land \text{Quiet}$
Goal: $\text{Food} \land \neg \text{Garbage} \land \text{Present}$

You try it!
Neural networks (Ch. 12)
Computer science is fundamentally a creative process: building new & interesting algorithms.

As with other creative processes, this involves mixing ideas together from various places.

Neural networks get their inspiration from how brains work at a fundamental level (simplification... of course).
Biology: brains

(Disclaimer: I am not a neuroscience-person) Brains receive small chemical signals at the “input” side, if there are enough inputs to “activate” it signals an “output”
An analogy is sleeping: when you are asleep, minor sounds will not wake you up.

However, specific sounds in combination with their volume will wake you up.
Other sounds might help you go to sleep (my majestic voice?)

Many babies tend to sleep better with “white noise” and some people like the TV/radio on
Neural network: basics

Neural networks are connected nodes, which can be arranged into layers (more on this later).

First is an example of a perceptron, the most simple NN; a single node on a single layer.
Neural network: basics

Neural networks are connected nodes, which can be arranged into layers (more on this later)

First is an example of a perceptron, the most simple NN; a single node on a single layer

\[ \sigma \left( b + \sum_{i=1}^{n} a_i w_i \right) \]
Mammals

Let's do an example with mammals...

First the definition of a mammal (wikipedia):

Mammals [posses]:
(1) a neocortex (a region of the brain),
(2) hair,
(3) three middle ear bones,
(4) and mammary glands
Mammals

Common mammal misconceptions:
(1) Warm-blooded
(2) Does not lay eggs

Let's talk dolphins for one second.

Dolphins have hair (technically) for the first week after birth, then lose it for the rest of life ...
... I will count this as “not covered in hair”
Perceptrons

Consider this example: we want to classify whether or not an animal is mammal via a perceptron (weighted evaluation)

We will evaluate on:
1. Warm blooded? (WB) Weight = 2
2. Lays eggs? (LE) Weight = -2
3. Covered hair? (CH) Weight = 3

If \(2 \cdot WB + (-2) \cdot LE + 3 \cdot CH > 1\) \(\Rightarrow\) Mammal
Consider the following animals:
Humans \{WB=y, LE=n, CH=y\}, mam=y
\[2(1) + (-2)(-1) + 3(1) = 7 > 1 \ldots \text{Correct!}\]
Bat \{WB=\text{sorta}, LE=n, CH=y\}, mam=y
\[2(0.5) + (-2)(-1) + 3(1) = 6 > 1 \ldots \text{Correct!}\]
What about these?
Platypus \{WB=y, LE=y, CH=y\}, mam=y
Dolphin \{WB=y, LE=n, CH=n\}, mam=y
Fish \{WB=n, LE=y, CH=n\}, mam=n
Birds \{WB=y, LE=y, CH=n\}, mam=n
Perceptrons

But wait... what is the general form of:

\[ w_1 x + w_2 \cdot y + w_3 \cdot z > c \]
Perceptrons

But wait... what is the general form of:

\[ w_1 x + w_2 \cdot y + w_3 \cdot z \geq c \]

This is simply one side of a plane in 3D, so this is trying to classify all possible points using a single plane...
Perceptrons

If we had only 2 inputs, it would be everything above a line in 2D, but consider XOR on right

There is no way a line can possibly classify this (limitation of perceptron)
Neural network: feed-forward

Today we will look at feed-forward NN, where information flows in a single direction.

Recurrent networks can have outputs of one node loop back to inputs as previous.

This can cause the NN to not converge on an answer (ask it the same question and it will respond differently) and also has to maintain some “initial state” (all around messy).
Neural network: feed-forward

Let's expand our mammal classification to 5 nodes in 3 layers (weights on edges):

if Output(Node 5) > 0, guess mammal
Neural network: feed-forward

You try Bat on this: \{WB=0, LE=-1, CH=1\}

Assume (for now) output = sum input

If Output(Node 5) > 0, guess mammal
Neural network: feed-forward

Output is -7, so bats are not mammal... Oops...

if Output(Node 5) > 0, guess mammal
Neural network: feed-forward

In fact, this is no better than our 1 node NN

This is because we simply output a linear combination of weights into a linear function (i.e. if $f(x)$ and $g(x)$ are linear... then $g(x)+f(x)$ is also linear)

Ideally, we want a activation function that has a limited range so large signals do not always dominate
Neural network: feed-forward

One commonly used function is the sigmoid:

\[ S(x) = \frac{1}{1 + e^{-x}} \]
Back-propagation

The neural network is as good as it's structure and weights on edges

Structure we will ignore (more complex), but there is an automated way to learn weights

Whenever a NN incorrectly answer a problem, the weights play a “blame game”...
- Weights that have a big impact to the wrong answer are reduced
Back-propagation

To do this blaming, we have to find how much each weight influenced the final answer.

Steps:
1. Find total error
2. Find derivative of error w.r.t. weights
3. Penalize each weight by an amount proportional to this derivative
Back-propagation

Consider this example: 4 nodes, 2 layers

This node as a constant bias of 1
Node 1: \(0.15 \times 0.05 + 0.2 \times 0.1 = 0.35\) as input thus it outputs (all edges) \(S(0.35) = 0.59327\).
Eventually we get: \( \text{out}_1 = 0.606, \text{out}_2 = 0.630 \)

Suppose wanted: \( \text{out}_1 = 0.01, \text{out}_2 = 0.99 \)
Back-propagation

We will define the error as: $\frac{1}{2} \sum_i (\text{correct}_i - \text{output}_i)^2$
(you will see why shortly)

Suppose we want to find how much $w_5$ is to blame for our incorrectness

We then need to find: $\frac{\partial \text{Error}}{\partial w_5}$

Apply the chain rule:

$$\frac{\partial \text{Error}}{\partial \text{out}_1} \cdot \frac{\partial S(\text{In}(N_3))}{\partial \text{In}(N_3)} \cdot \frac{\partial \text{In}(N_3)}{\partial w_5}$$
Back-propagation

\[
\text{Error} = \sum_i (\text{correct}_i - \text{output}_i)^2 \\
\frac{\partial \text{Error}}{\partial \text{out}_1} = -(\text{correct}_1 - \text{out}_1) \\
= -(0.01 - 0.606) = 0.596
\]

As \( S'(x) = S(x) \cdot (1 - S(x)) \)

\[
\frac{\partial S(\text{In}(N_3))}{\partial \text{In}(N_3)} = S(\text{In}(N_3)) \cdot (1 - S(\text{In}(N_3))) \\
= 0.606 \cdot (1 - 0.606) = 0.214
\]

\[
\frac{\partial \text{In}(N_3)}{\partial w_5} = \frac{\partial \text{Out}(N_1)+w_6\cdot\text{Out}(N_2)+b_2\cdot1}{\partial w_5} \\
= \text{Out}(N_1) = 0.506
\]

Thus, \( \frac{\partial \text{Error}}{\partial w_5} = 0.596 \cdot 0.214 \cdot 0.506 = 0.0645 \)
Back-propagation

In a picture we did this:

Now that we know $w_5$ is 0.08217 part responsible, we update the weight by:

$$w_5 \leftarrow w_5 - \alpha \times 0.0645 = 0.374 \text{ (from 0.4)}$$

$\alpha$ is learning rate, set to 0.5
Back-propagation

Updating this $w_5$ to $w_8$ gives:

$w_5 = 0.3589$
$w_6 = 0.4067$
$w_7 = 0.5113$
$w_8 = 0.5614$

For other weights, you need to consider all possible ways in which they contribute
Back-propagation

For \( w_1 \) it would look like:

\[
\frac{\partial E_{\text{total}}}{\partial w_1} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{h1}} \cdot \frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}} \cdot \frac{\partial \text{net}_{h1}}{\partial w_1}
\]

\[
\frac{\partial E_{\text{total}}}{\partial \text{out}_{h1}} = \frac{\partial E_{o1}}{\partial \text{out}_{h1}} + \frac{\partial E_{o2}}{\partial \text{out}_{h1}}
\]

\[
E_{\text{total}} = E_{o1} + E_{o2}
\]

(book describes how to dynamic program this)
Back-propagation

Specifically for $w_1$ you would get:

$$\frac{\partial \text{Error}}{\partial S(\text{In}(N_1))} = \frac{\partial \text{Error}_1}{\partial S(\text{In}(N_1))} + \frac{\partial \text{Error}_2}{\partial S(\text{In}(N_1))}$$

$$\frac{\partial S(\text{In}(N_1))}{\partial \text{In}(N_1)} = S(\text{In}(N_1)) \cdot (1 - S(\text{In}(N_1))) = 0.506 \cdot (1 - 0.506) = 0.250$$

$$\frac{\partial \text{In}(N_3)}{\partial w_5} = \frac{\partial w_1 \cdot \text{In}_1 + w_2 \cdot \text{In}_2 + b_1 \cdot 1}{\partial w_5} = \text{In}_1 = 0.05$$

Next we have to break down the top equation...
Back-propagation

\[
\frac{\partial \text{Error}}{\partial S(In(N_1))} = \frac{\partial \text{Error}_1}{\partial S(In(N_1))} + \frac{\partial \text{Error}_2}{\partial S(In(N_1))}
\]

\[
\frac{\partial \text{Error}_1}{\partial S(In(N_1))} = \frac{\partial \text{Error}_1}{\partial S(In(N_3))} \cdot \frac{\partial S(In(N_3))}{\partial In(N_3)} \cdot \frac{\partial In(N_3)}{\partial S(In(N_1))}
\]

From before...

\[
\frac{\partial \text{Error}_1}{\partial S(In(N_3))} \cdot \frac{\partial S(In(N_3))}{\partial In(N_3)} = 0.596 \cdot 0.214 = 0.128
\]

\[
\frac{\partial In(N_3)}{\partial S(In(N_1))} = \frac{\partial w_5 \cdot S(In(N_1)) + w_6 \cdot S(In(N_2)) + b_1 \cdot 1}{\partial S(In(N_1))}
\]

\[
= w_5 = 0.4
\]

Thus, \[
\frac{\partial \text{Error}_1}{\partial S(In(N_1))} = 0.128 \cdot 0.4 = 0.0510
\]
Back-propagation

Similarly for Error$_2$ we get:

\[
\frac{\partial \text{Error}}{\partial S(\text{In}(N_1))} = \frac{\partial \text{Error}_1}{\partial S(\text{In}(N_1))} + \frac{\partial \text{Error}_2}{\partial S(\text{In}(N_1))}
\]

\[
= 0.0510 + -0.0190 = 0.0320
\]

Thus, \[
\frac{\partial \text{Error}}{\partial w_1} = 0.0320 \cdot 0.250 \cdot 0.05 = 0.000400
\]

Update \( w_1 \leftarrow w_1 - \alpha \frac{\partial \text{Error}}{\partial w_1} = 0.15 - 0.5 \cdot 0.0004 = 0.1498 \)

You might notice this is small...
This is an issue with neural networks, deeper the network the less earlier nodes update...