Learning
Probabilistic Belief
Probabilistic Reasoning
Adversarial Search
CSPs
Planning
Heuristic Search

CS121
Heuristic Search

First, you need to formulate your situation as a Search Problem:
- Where is the start? What is the goal?
- For each of those transitions, what is the cost?
- What is the successor function?
- From one state, what other states can you get to?
- What is a state?

Search Problem
Heuristic Search
Heuristic Search

- Easy to formulate for problems that are inherently discrete
  - Solve a Rubik's cube
  - Given all the flights of the airlines, figure out the best way (time/distance/cost) to get from city A to city B

- What about problems that have continuous spaces?
  - Maneuvering a robot through a building
  - Maneuvering a robot arm to do a task

Easy to formulate for problems that are inherently discrete
Heuristic Search
Heuristic Search
Heuristic Search

- No Heuristic
  - Uniform Cost, Iterative Deepening, BFS, DFS
- Heuristic
  - Consistency
  - Admissibility
  - Have fringe sorted by \( f = g + h \)
Planning

● Just a search problem!

● Use STRIPS to formulate the problem!

Plan

How do we get a heuristic?

(what is the new state) •

Add/Delete (which are allowed) •

Preconditions (which are allowed) •

Successor function given by actions •

IN(Robot, R1), HOLDING(Apple) •

A state is a set of propositions which are true •

A state is a set of propositions which are true •
Planning

- Given some state $s$, how many actions will it take to get to a state satisfying $g$?
- Planning Graph

```
Planning Graph

Find the first $S^1$ where the $g$ is met

Repeat until convergence
- Add the add lists of actions that apply to get $S^1$
- Initialize to $S^0$ all the proposition in $s$
```

Given some state $s$, how many actions will it take to get to a state satisfying $g$?
Planning

- Forward Planning
  - Start initial node as initial state
  - Find all successors by applying actions
  - For each successor, build a planning graph to determine heuristic value
  - Add to fringe, pop, repeat

Problems

- Multiple planning graphs
- Branching factor
Planning

Backward Planning

- Backward Planning
- Start initial node as goal
- Construct planning graph from initial state
  - Look up heuristic values in planning graph
  - Find successors by regressing through relevant actions
  - Add to fringe, pop, repeat
Constraint Satisfaction

- Goal: All variables assigned without violating anything
- Successor: Assign a value to next variable
- State: Partial assignment to variables

Again, just a search problem (Backtracking)

- Problem: Assign values to everything without violating any constraint
- Constraints between variables and their values
- Variables, each with some domain

Formulation
Constraint Satisfaction

- No sense of "optimal" path. We just want to cut down on search time.

- How to choose the next value?
  - Least constraining value
  - Most constraining variable
  - Most constrained variable

- How to choose the next value to assign next?
To benefit from these heuristics, should update domains

- **Forward Checking**
  After assigning a value to a variable, remove all conflicting values from other variables

- **AC3**
  Given a set of variables, look at pairs X,Y
  - If for a value of X, there is no value of Y that works, remove that value from X
Adversarial Search

- Order of evaluation does matter
  - Alpha-Beta pruning
  - Tells you which move to take
  - Propagate values up according to MIN/MAX
    - Use evaluation function
    - Values at "bottom" of the tree – end of game, or
      1. By MAX and MIN player
      2. Game tree from moves performed successfully

Adversarial Search
Probabilistic Reasoning

- Assume there is some state space.
- Now actions are probabilistic.
- Some states have rewards (positive or negative).

There is a probability associated with going into each state. They must sum to 1. If I do action A, there are several different possible states I may end up in. 

Now actions are probabilistic.
Assume there is some state space.

We would like to calculate utility for each state.

Some states have rewards (positive or negative) (they must sum to 1).
Probabilistic Reasoning
How do you calculate the Utilities?

- If no cycles, can back values up the tree
- Otherwise, can use Value Iteration
  - Start all utilities as 0, calculate new utilities
  - Repeat until convergence
- Or, Policy Iteration
  - Pick a random policy, solve utilities for it
  - Or, Policy Iteration
  - Calculate new policy, repeat until convergence

How do you calculate the Utilities?
Probabilistic Belief

- Say N variables, each with 2 values, joint probability table has $2^N$ entries.

<table>
<thead>
<tr>
<th></th>
<th>Cavity</th>
<th>Cavity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.576</td>
<td>0.144</td>
<td>0.064</td>
</tr>
<tr>
<td>0.008</td>
<td>0.072</td>
<td>0.012</td>
</tr>
<tr>
<td>Patch</td>
<td>Patch</td>
<td>Patch</td>
</tr>
<tr>
<td>Toothache</td>
<td>Toothache</td>
<td>Toothache</td>
</tr>
</tbody>
</table>

Probabilistic Belief
Table more compactly
If variables are independent, can represent this

Probabilistic Belief
We are given a bunch of examples, where each example has values $X_1$ to $X_N$ and $Y$.

The goal is that given some partial example $X_1 \ldots X_N$, we can use $H(X)$ to guess $Y$.

This should work well for $X$'s from the training set, but also for $X$'s never seen before!

We want to create some function $H(X)$, that will take all the $X$'s and output a single value.

We are given a bunch of examples, where each example has values $X_1 \ldots X_N$ and $Y$.

(Supervised) Learning
<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>Play Tennis</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>D2</td>
<td>Hot</td>
<td>Mild</td>
<td>Normal</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>D3</td>
<td>Overcast</td>
<td>Strong</td>
<td>Normal</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>D4</td>
<td>Rain</td>
<td>High</td>
<td>Normal</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>D5</td>
<td>Cool</td>
<td>Strong</td>
<td>Normal</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>D6</td>
<td>Cool</td>
<td>Weak</td>
<td>Normal</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>D7</td>
<td>Overcast</td>
<td>Weak</td>
<td>Normal</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>D8</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>D9</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>D10</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>D11</td>
<td>Overcast</td>
<td>High</td>
<td>Normal</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>D12</td>
<td>Sunn</td>
<td>Strong</td>
<td>Normal</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>D13</td>
<td>Overcast</td>
<td>Weak</td>
<td>Normal</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>D14</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Some types of functions we can use:

- Neural Net
- Decision Tree
- Linear Regression
- Data Cache

(Supervised) Learning
(Supervised) Learning

Decision Tree

- At each non-terminal node in tree, branch
- A leaf node should output a value for \( Y \)
- Look at all examples at current node
- Choose \( X_i \) to split on that will allow you to classify the most number of examples correctly
- Building the Tree (Greedy)
Supervised Learning

Neural Net

Unit (Neuron)
Supervised Learning

Neural Net