Game Tree Search

2-8-16

xkcd.com/832
Reading Quiz

This is an example of a(n) __________ representation of a __________ game.

a) normal form … perfectly competitive

b) normal form … perfectly cooperative

c) extensive form … perfectly competitive

d) extensive form … perfectly cooperative

e) none of these

<table>
<thead>
<tr>
<th></th>
<th>R</th>
<th>P</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>0,0</td>
<td>-1,1</td>
<td>1,-1</td>
</tr>
<tr>
<td>P</td>
<td>1,-1</td>
<td>0,0</td>
<td>-1,1</td>
</tr>
<tr>
<td>S</td>
<td>-1,1</td>
<td>1,-1</td>
<td>0,0</td>
</tr>
</tbody>
</table>
Recall the model from last time ... 

- Assign numerical utilities to all possible outcomes.

- Enumerate the choices an agent can make and what outcomes they lead to.

- Design an agent to maximize expected utility.
How can we account for other agents?

1) Assume that they are part of the environment.
   ● Model their actions as probabilistic.
     ○ How do we come up with a model for their actions?
     ○ Given a model, maximizing our own expected utility is (relatively) easy.

2) Assume they are intelligent.
   ● Model them as expected utility maximizers.
     ○ How do we determine their utilities?
     ○ We also assume that they will also assume that we behave optimally, and so on.

This is a hard problem! But we'll ignore it for now.
Game Representations

**Extensive Form Game**
- Agents take turns
- Represented as a tree.
- Internal nodes are agent decisions.
- Edges are actions.
- Leaf nodes are outcomes.

**Normal Form Game**
- Agents move simultaneously.
- Represented as a matrix.
- Each agent controls a dimension.
- Actions select a row/column.
- Matrix cells are outcomes.

---

**Normal Form Game**

<table>
<thead>
<tr>
<th></th>
<th>R</th>
<th>P</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0,0</td>
<td>-1,1</td>
<td>1,-1</td>
</tr>
<tr>
<td>P</td>
<td>1,-1</td>
<td>0,0</td>
<td>-1,1</td>
</tr>
</tbody>
</table>

**Extensive Form Game**

```
(3,1)     (1,2)     (2,1)     (0,0)
L          R          L          R
```

```
1
L         R
2
```

```
2
L          R
2
L          R
```
Extensive form games

- Each node in the tree identifies who makes a decision.
- The out-edges from a node correspond to the actions available at that decision point.
- Terminal nodes correspond to outcomes.
- Each agent has a utility for each outcome, represented as a tuple.

```
   1
  /   \
R 2   L 2
 /   /  \
R L  R L
(3,1) (1,2) (2,1) (0,0)
```
Example Game: Nimm

- There are initially $N$ pieces.
- Each turn a player must remove 1, 2, or 3 pieces.
- The player who removes the last piece loses.

Let’s play a game where $N=9$, you go first.
Exercise

Play games of Nimm with various values of N

- Who wins under optimal play?
- How does it depend on N?
<table>
<thead>
<tr>
<th>N</th>
<th>outcome for p1</th>
<th>first move</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>L</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>W</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>W</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>W</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>L</td>
<td>?</td>
</tr>
<tr>
<td>6</td>
<td>W</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>W</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>W</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>L</td>
<td>?</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
N=5
Backwards induction

Key idea: start from the outcomes and work your way up.

- At leaf nodes, return the outcome.
- At decision nodes, recursively determine the outcome of each action.
- Then return the best outcome for the current player.
N=5

The diagram shows a tree with branches labeled with pairs of values, (W, L) and (L, W), indicating two types of nodes. The numbers on the branches represent the order of the nodes. The tree structure is complex, with multiple branching paths and nodes.
Special Case: 2-player zero sum: min/max search

minimaxDecision(state) returns an action
    set values to an empty list
    for each action in possibleActions(state)
        nextState = successor(state, action)
        add minValue(nextState) to values list
    return action associated with the maximum of values
**Helper functions**

`maxValue(state) returns number representing value of state`
- `if terminalTest(state) return value(state)`
- `v = -maxint`
- `for each action in possibleActions(state)`
  - `nextState = successor(state, action)`
  - `v = max(v, minValue(nextState))`
- `return v`

`minValue(state) returns number representing value of state`
- `if terminalTest(state) return value(state)`
- `v = maxint`
- `for each action in possibleActions(state)`
  - `nextState = successor(state, action)`
  - `v = min(v, maxValue(nextState))`
- `return v`
Min-Max algorithm

- Performs a depth-first traversal of the game tree

- Minimax is optimal assuming that the leaf node evaluations are correct
Problem: game tree size

- For most interesting games the game tree is too large to search to the end and to find optimal moves.
- In chess, the branching factor is approximately 35 and games can last for 100 moves.
- This creates a game tree of 35 to the power of 100 nodes which is approximately $10^{154}$!
- Instead we will search to a limited depth and try to approximate the value of states.
Evaluation Function

- Look at a game state without knowing any context and try to assign it a value
- Performance of a game playing program is highly dependent on this evaluation
- Using a good evaluation function allows us to make informed decisions about which move now is likely to lead to good situations later
Features of a good evaluation function

- When a terminal state is reached, score it correctly
- Should be efficient to calculate since it will be called many, many times
- Should reflect the actual chances of winning
- Exactness is less important than trying to get the relative values correct