CSC384
Introduction to Artificial Intelligence: Adversarial Search

September 23, 2014

Adversarial search

Introduction

- Also known as **"Game Tree search"**

Previous search problems assumed complete control of the environment. We will now introduce non-determinism:

- Actions beyond our control may occur (e.g., weather)
- Actions may be taken by agents with conflicting goals from us
- We no longer have strict control over the path we take

This material corresponds to chapters 5.1, 5.2, and 5.3 of the textbook.

Adversarial search

Introduction

Zero sum game: Total payoff to all players is always the same

- For two player games (e.g., chess, checkers, go, othello, tic-tac-toe) there is one winner and loser or a draw, so the possible payoffs are: $\{(1,0),(0,1),(\frac{1}{2},\frac{1}{2}))\}$

A key feature of games is that multiple agents act to influence the outcome.

- Zero sum games are inherently competitive.
- Some games can be cooperative (but won't be our immediate focus)
- An agent's action depends on the action take by other agents

Adversarial search

Introduction

Example: Rock Paper Scissors

	R	P	S
R	$(\frac{1}{2}, \frac{1}{2})$	$(0, 1)$	$(1, 0)$
P	$(1, 0)$	$(\frac{1}{2}, \frac{1}{2})$	$(0, 1)$
S	$(0, 1)$	$(1, 0)$	$(\frac{1}{2}, \frac{1}{2})$

Adversarial search

Definitions

For a given a set of game states S we have:

S_0: The initial state
$Player(s)$: The active player in state s
$Actions(s)$: Possible moves in state s
$Result(s, a)$: The state resulting from action a in state s
$Terminal(s)$: Determine if the game has ended
$Utility(s, p)$: Payoff for player p in terminal state s
In the two player case this can be reduced to $Utility(s)$

We can construct a search tree of the game states

Adversarial search

Minimax search

Assume we have two players (Min, Max) who play as best they can.
Min wins if $Utility(s) = 0$, Max if $Utility(s) = 1$.
If they play optimally we can calculate the optimal decisions for the players as follows:

$$Minimax(s) = \begin{cases} Utility(s) : if(Terminal(s) \\ max_{a \in Actions(s) Minimax(s,a)} : if(Player(s) = Max) \\ min_{a \in Actions(s) Minimax(s,a)} : if(Player(s) = Min) \end{cases}$$

This can be used to make a depth first search

Adversarial search

Alpha Beta pruning

The search space is too large for most games. We can prune out states that would not be chosen though.

α : Value of best choice seen so far for Max
β : Value of best choice seen so far for Min

Prune a search node if it is worse than α or β for Max or Min

Adversarial search

Move ordering

Alpha beta pruning alone is not enough for many hard problems.

- The order in which we evaluate nodes can effect pruning greatly
- For minimax we examine $O(b^m)$ nodes (b is the maximum branches, m is the depth of the problem)
- For a perfect ordering alpha beta pruning must examine $O(b^{m/2})$ nodes
- For a random ordering alpha beta pruning must examine $O(b^{3m/4})$ nodes with restrictions on b