CS540 Introduction to Artificial Intelligence

Lecture 21

Young Wu

Based on lecture slides by Jerry Zhu, Yingyu Liang, and Charles Dyer

August 9, 2022

Game Tree

Motivation

- The initial state is the beginning of the game.
- There are no goal states, but there are multiple terminal states in which the game ends.
- Each successor of a state represents a feasible action (or a move) in the game.
- The search problem is to find the terminal state with the lowest cost (or usually the highest reward).

Adversarial Search

Motivation

- The main difference between finding solutions of games and standard search problems or local search problems is that part of the search is performed by an opponent adversarially.
- Usually, the opponent wants to maximize the cost or minimize the reward from the search. This type of search problems is called adversarial search.
- In game theory, the solution of a game is called an equilibrium. It is a path in which both players do not want to change actions.

Backward Induction

Motivation

- Games are usually solved backward starting from the terminal states.
- Each player chooses the best action (successor) given the (already solved) optimal actions of all players in the subtrees (called subgames).

Zero-Sum Games

Motivation

- If the sum of the reward or cost over all players at each terminal state is 0, the game is called a zero-sum game.
- Usually, for games with one winner: the reward for winning and the cost of losing are both 1. If the game ends with a tie, both players get 0.

Nim Game Example Diagram

Motivation

Minimax Algorithm

Description

- Use DFS on the game tree.

Minimax Algorithm

Algorithm

- Input: a game tree (V, E, c), and the current state s.
- Output: the value of the game at s.

- If s is a terminal state, return $c(s)$.
- If the player is MAX, return the maximum value over all successors.

$$\alpha(s) = \max_{s' \in s'(s)} \beta(s')$$

- If the player is MIN, return the minimum value over all successors.

$$\beta(s) = \min_{s' \in s'(s)} \alpha(s')$$

Backtracking

Discussion

- The optimal actions (solution paths) can be found by backtracking from all terminal states as in DFS.

$$s^{\star}\left(s\right)=\underset{s'\in s'\left(s\right)}{\operatorname{argmax}}\,\beta\left(s'\right)\ \text{ for }\ \text{MAX}$$

$$s^{\star}\left(s\right)=\underset{s'\in s'\left(s\right)}{\operatorname{argmin}}\,\alpha\left(s'\right)\ \text{ for }\ \text{MIN}$$

Minimax Performance

Discussion

- The time and space complexity is the same as DFS. Note that $D = d$ is the maximum depth of the terminal states.

$$T = 1 + b + b^2 + ... + b^d$$
$$S = (b - 1) \cdot d$$

Non-deterministic Game

Discussion

- For non-deterministic games in which chance can make a move (dice roll or coin flip), use expected reward or cost instead.
- The algorithm is also called expectiminimax.

Pruning

Motivation

- Time complexity is a problem because the computer usually has a limited amount of time to "think" and make a move.
- It is possible to reduce the time complexity by removing the branches that will not lead the current player to win. It is called the Alpha-Beta pruning.

Alpha Beta Pruning

Description

- During DFS, keep track of both α and β for each vertex.
- Prune the subtree with $\alpha \geqslant \beta$.

Alpha Beta Pruning Algorithm, Part I

Algorithm

- Input: a game tree (V, E, c), and the current state s.
- Output: the value of the game at s.
- If s is a terminal state, return $c(s)$.

Alpha Beta Pruning Algorithm, Part II

Algorithm

- If the player is MAX, return the maximum value over all successors.

$$\alpha(s) = \max_{s' \in s'(s)} \beta(s')$$

$$\beta(s) = \beta(\text{ parent } (s))$$

- Stop and return β if $\alpha \geqslant \beta$.
- If the player is MIN, return the minimum value over all successors.

$$\beta(s) = \min_{s' \in s'(s)} \alpha(s')$$

$$\alpha(s) = \alpha(\text{ parent } (s))$$

- Stop and return α if $\alpha \geqslant \beta$.

Alpha Beta Performance

Discussion

- In the best case, the best action of each player is the leftmost child.
- In the worst case, Alpha Beta is the same as minimax.

Static Evaluation Function

Definition

- A static board evaluation function is a heuristics to estimate the value of non-terminal states.
- It should reflect the player's chances of winning from that vertex.
- It should be easy to compute from the board configuration.

Evaluation Function Properties

Definition

- If the SBE for one player is x, then the SBE for the other player should be $-x$.
- The SBE should agree with the cost or reward at terminal vertices.

Linear Evaluation Function Example

Definition

- For Chess, an example of an evaluation function can be a linear combination of the following variables.

1. Material.
2. Mobility.
3. King safety.
4. Center control.

- These are called the features of the board.

Iterative Deepening Search

Discussion

- IDS could be used with SBE.
- In iteration d, the depth is limited to d, and the SBE of the non-terminal vertices are used as their cost or reward.

Non Linear Evaluation Function

Discussion

- The SBE can be estimated given the features using a neural network.
- The features are constructed using domain knowledge, or a possibly a convolutional neural network.
- The training data are obtained from games between professional players.

Monte Carlo Tree Search

Discussion

- Simulate random games by selecting random moves for both players.
- Exploitation by keeping track of average win rate for each successor from previous searches and picking the successors that lead to more wins.
- Exploration by allowing random choices of unvisited successors.

Upper Confidence Bound

Discussion

- Combine exploitation and exploration by picking successors using upper confidence bound for tree.

$$\frac{w_s}{n_s} + c\sqrt{\frac{\log t}{n_s}}$$

- w_s is the number of wins after successor s, and n_s the number of simulations after successor s, and t is the total number of simulations.
- Similar to the UCB algorithm for MAB.

Alpha GO Example

Discussion

- MCTS with $> 10^5$ play-outs.
- Convolutional neural network to compute SBE.