
 This repository Search


Pull requestsIssuesGist

 lucoby / CS6601notes

Unwatch 1Star 0Fork 0

[Code](#) [Issues 0](#) [Pull requests 0](#) [Wiki](#) [Pulse](#) [Graphs](#) [Settings](#)

Branch: master CS6601notes / 1 - Game Playing.mdFind fileCopy path

 lucoby format!7b8ca66 on Jan 29

1 contributor

102 lines (57 sloc)6.5 KBRawBlameHistory

# Lesson 1 - Game Playing

## Adversarial Search

Building a game tree. You can build a tree where each node of the tree represents a state of the game board. In practice a tree data structure isn't necessarily used rather scores, moves, and board states are passed recursively through function calls and returns.

A naive evaluation function in a fully evaluated game tree can be +1 for wins and -1 for losses. This knowledge needs to be back propagated through the game tree.

A full tree that's stored in memory can be the basis of an opening or closing book. However generally for the algorithm to discover good and bad moves on its own a minimax algorithm is used.

## Minimax Algorithm

Assumes that the opponent is smart and will attempt to minimize scores. The player attempts to maximize their score. Hence the minimax algorithm. Layers of the trees alternate minimizing and maximizing layers. maximizing layers represents the players opportunity to move and the maximum value is propagated up. Minimizing layers represent the opponent turn to move and the minimal value is propagated up.

## Branching factor

Even in trivial game boards this can result in very large game trees making searching very infeasible.

For example a 5x5 game board an extreme upper bound estimate is 25! moves

However the maximum branching factor is 12. However this can be refined. An "AI" making random moves can estimate the average branching factor. Even in 5x5 isolation this estimate leaves us with  $8^{25}$  or approx.  $10^{22}$  nodes. Which is infeasible to search.

## Depth Limited Search

You can estimate the depth by

$$\text{nodes} / \text{sec} (n) * \text{sec} (t) = \text{nodes you can search in allotted time} (N)$$
$$\text{branching factor} (b) ^ \text{depth} (d) < N$$

$$\log_b (b^d) < \log_b (N)$$

log laws:

$$\log_a x = \log_b(x) / \log_b(a)$$

## Evaluation Functions

Evaluation function is a function to use to evaluate a board in the case where you can't reach a terminal node due to depth limitation. A simple evaluation function is `#my_moves()`.

It's important to test whether your evaluation function actually predicts a win. However just because if at a particular turn the evaluation function doesn't correlate to winning it doesn't necessarily mean that it's a bad evaluation function. The recommended move can possibly vary wildly between different depths of search if a *critical* decision is being made.

**Quiescence** describes the state where the recommended move doesn't vary wildly between search depths. Depth of Quiescence can vary wildly at different stages of the game.

## Iterative Deepening

Iterative deepening is the process of searching a low depth and determining the "best" move to make and storing that move. After start searching the next layer of the node. Surprisingly this isn't that inefficient due to the exponential nature of the search tree. For low branching factor there is a higher proportional overhead. For branching factor of 2 the iterative deepening nodes is less than double the number of nodes you would search in a normal depth limited search

In general number of nodes in the tree:

$$b=k$$

$$n = (k^{(d_1)} - 1) / (k - 1)$$

Compared to depth limited search, Iterative deepening is useful in the end game where lower branching factors can allow an AI to search deeper to more precisely determine good moves. Furthermore in the early game it can be handy for preparing an "okay" move when the branching factor is normally high and a fixed depth limited search might not return a move in a reasonable amount of time.

## Good Evaluation Functions

There can be stages in the game where a human player can more easily see that the game is about to be decided but would require many more layers of search for a depth oriented search to determine end game. This is know as **horizon effect**.

A more complicated evaluation function can in some cases be better. However increase the time spent on an evaluation function is multiplied exponentially across the entire search tree. So it ultimately depends on the game to balance the trade-offs between a more complicated eval function being able account for horizon effects or a simpler eval function that simply allows you to search deeper. It's always good to experiment with many evaluation functions to determine which one is best!

## Alpha-Beta Pruning

Alpha-beta is a pruning technique that allows you to ignore whole sections of the game tree but still get the same answer as with minimax. Often it is many times more efficient.

Algorithm	Time
Minimax	$b^d$

Minimax + alpha-Beta with optimal order	$b^{(d/2)}$
Minimax + alpha-Beta with random order	$b^{(3d/4)}$

AI is the study of finding clever hacks NP Hard problems. Once a clever hack is found or processing is good enough. That problem no longer belongs to artificial intelligence.

AI is all the NP hard problems that haven't been solved.

## Isolation Game Player

Minimax with Alpha-Beta, and Iterative Deepening

Additional Takeaways:

1. Symmetry. There is horizontal, vertical, and diagonal symmetry to the board. A board with with 1 move at (0,0) is equivalent to boards with 1 move at (0,4); (4,0); and (4,4). There is symmetry. This can be helpful in turns 1-3 where there is high branching factor and the low number of turns make symmetry more likely to be found.
2. Partitions: You don't need to search to end so long as a partition is found.
3. Reflection: If player 1 starts at (0,0) and player 2 moves to a reflectable position i.e. (0,1). Then player 1 wins 100% by making the move that is the 180 degree rotation of player 2's move

## 3 Player Games

For 3 player games, minimax no longer is effective. Rather use MaxN function to evaluate the board from the perspective of each player. Propagate the value up the tree that is most optimal for the player whose turn corresponds to the layer of the tree.

**3 player alpha beta** can work when there are upper and lower bounds for the score at each level of the tree. Furthermore only immediate and shallow pruning is possible. Deep pruning isn't possible.

## Probabilistic Games

It's possible to make a game tree that takes into account the probabilities of moves and then use ExpectedMax algorithm to make the best choice decision for a move.

Terminal nodes' score is a the probability of a board outcome multiplied by the utility of the board. Min and Max nodes propagate based on the expected score of the nodes. Pruning is possible if there are known bounds on the expected value of the board.

