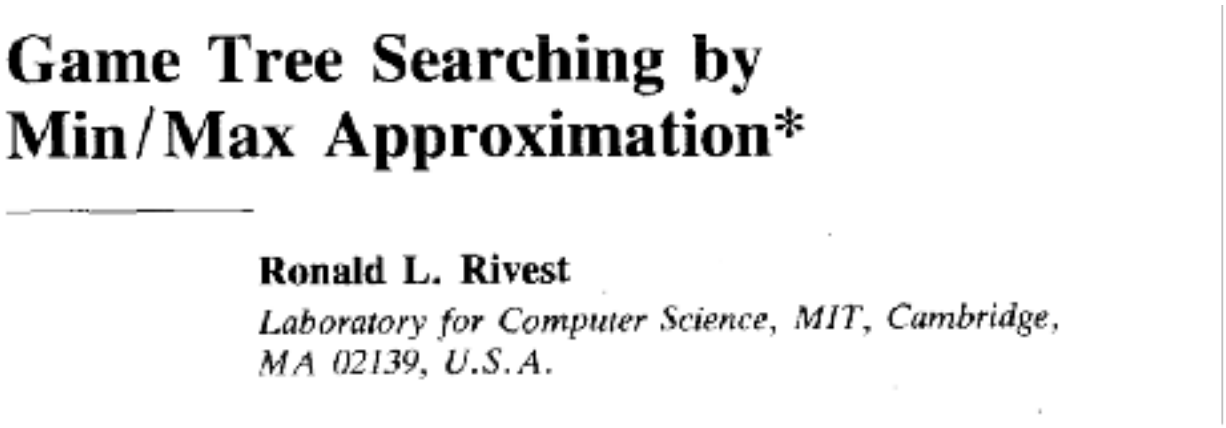
Seth Weidman
Udacity Artificial Intelligence Nanodegree
Project 2: Build a Game-Playing Agent
July 16, 2017

Game Tree Searching by Min/Max Approximation



Game Tree Searching by Min/Max Approximation*

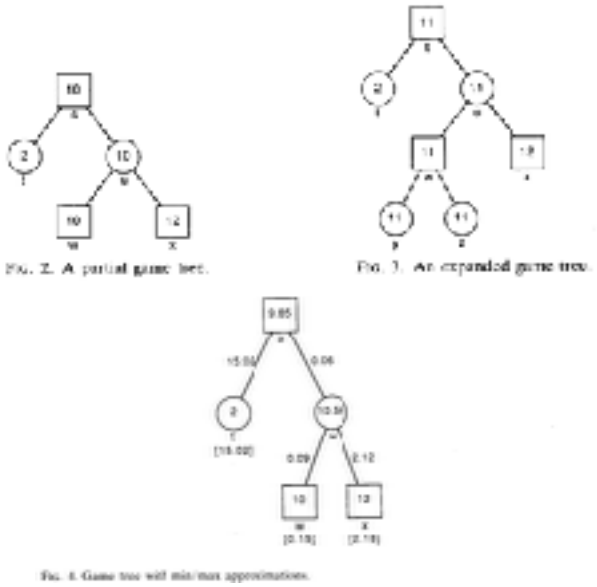
Ronald L. Rivest
Laboratory for Computer Science, MIT, Cambridge, MA 02139, U.S.A.

We learned two classical methods for finding the best moves in games during the course of this project. The first was minimax search. The idea behind minimax search is simple. You suppose that, looking forward in the game, any move you make will try to optimize your "payoff". This could be simply "positive infinity" if you win, "negative infinity" if you lose, or some payoff in between (like the total number of pieces you have left on the board in chess), but roughly speaking, you want to maximize your expected payoff. You also know that your opponent will try to minimize your expected payoff at every chance. So, you'll try to find your best move in the following way:

- You'll say "which move maximizes the payoff to me?"
- To answer this, you'll consider "For each move I make, my opponent can make several moves. I know he will choose the move to *minimize* my payoff for each move I make. So, the payoff of each of my moves to me is equal to the *minimum* payoff each of those moves gives my opponent the opportunity to make."

“[This algorithm] makes it possible for us to expand and examining one node deeper into the tree than others.

-Ronald Rivest



To determine the payoffs of your moves to your opponent, you could use similar reasoning, saying that the payoffs of your moves to him would be determined by you trying to *maximize* your payoffs in *two* moves, and so on, to arbitrary depth. In this way, you could traverse the tree of possible game moves, one move at a time, eventually figuring out the best move for you.

There is an improvement on this. Let’s say that in that first step, we find that the first move we evaluate gives our opponent very few good options, so that we know it will be hard to choose a move that results in a small payoff for *us*. We can define the payoff of this particular move as “alpha”. Then, for every move we subsequently evaluate, we know that as soon as we see our opponent having just *one* move that is has a more negative payoff for us than the move we have already evaluated, we can stop evaluating the rest of the possibilities from this move since it is clear that it will result in a lower payoff to us than choosing the first move. This procedure is called “Alpha-Beta Pruning” since alpha and beta refer to the coefficients that are used in the algorithm.

A final method that has been shown to improve on both of these in the case of Connect Four is *Min/Max Approximation*. Here, at each node, instead of just taking the minimum / maximum value of the nodes below it to determine the payoff for that node, we take the product of all possible moves of that node and raise it to an exponent - a negative number when we are trying to approximate the minimum function and a positive number to approximate the maximum function. This is a geometric mean of sorts. Then, for each move, we compute the potential impact of “expanding the tree” at that node to look further down by taking the product of the partial derivatives of all of the nodes leading to that node from the top of the tree. This makes it possible for us to expand and examining one node deeper into the tree than others. Intuitively, if several lines of play “aren’t that interesting”, but one line could potentially have a large impact on that game’s final payoff, then that one line is examined more closely

than the others. By contrast, both minimax search and alpha beta search examine all nodes at each level before moving on to the next level. This could potentially make them miss opportunities that depend on deeply examining a particular path of the tree to find as well as spending too much time examining unpromising sections of the game tree.