

The Minimax Algorithm

Jessica Billings
CS 330, Spring 2008
Bryn Mawr College

The minimax algorithm is a specialized search algorithm which returns the optimal sequence of moves for a player in an zero-sum game. On its own, it is too complex to be practical for a game-playing program, but there are several extensions that, taken together, make it an effective algorithm.

1 Use

The optimal game for a minimax algorithm is a deterministic, turn-taking, two-player, zero-sum game of perfect information. This describes many common games, including chess. In this type of game, two players take turns to achieve a winning state. Both players know all the information about the game environment, including the actions of the other player. Both players cannot win: there must be either a loss or a draw. This means that the players' utility values at the end of the game are always equal and opposite.

The minimax function computes an optimal decision, which is a special type of search problem with specific components. The *initial state* is the board position and information on which player moves first. The *successor function* returns a list of (move, state) pairs which each indicate a legal move and the resulting state. The *terminal test* determines when the game is over. Ending states are called *terminal states*. The *utility function* assigns a numeric value to the terminal states. For example, if the options are win, lose, or draw, the terminal values might be +1, -1, and 0. These states and legal moves form the game's *game tree*.

2 How It Works

2.1 Overview

In a normal search, the optimal solution is simply a sequence of moves that leads to a goal state. Search in a multi-player game is different because the moves of the other player must be taken into account when deciding a sequence of moves. Therefore, Max must develop a strategy, something which specifies Max's move in the initial state, Max's moves resulting from all possible responses Min might make to its move, Max's moves in the states resulting from every possible response by Min to those moves, and so on. An optimal strategy in this case leads to outcomes at least as good as any other strategy when playing against an infallible opponent.

In the game tree that results from the Minimax algorithm, each level represents a move by either Max or Min, called a *ply*. Given a tree, the optimal

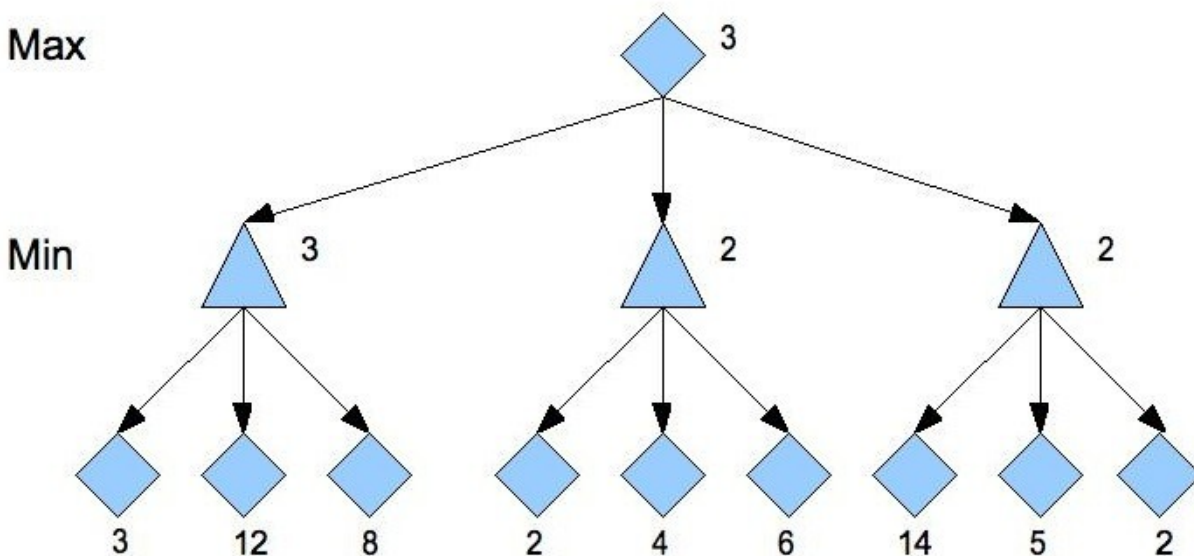
strategy can be determined by examining the *minimax value* of each node. This value is the utility (for Max) of being in a certain state, assuming both players play optimally. The minimax value of the terminal state is its own utility. Given a choice, Max chooses to move to a state of maximum value and Min chooses a state of minimum value.

$$\begin{aligned}
 \text{MINIMAX-VALUE}(n) = & \\
 & \text{UTILITY}(n) && \text{if } n \text{ is a terminal state} \\
 & \max_{s \in \text{Successors}(n)} \text{MINIMAX-VALUE}(s) && \text{if } n \text{ is a MAX node} \\
 & \min_{s \in \text{Successors}(n)} \text{MINIMAX-VALUE}(s) && \text{if } n \text{ is a MIN node}
 \end{aligned}$$

The minimax algorithm computes the minimax decision for the current state by recursively computing the minimax values of each successor state. The recursion proceeds to the leaves of the game tree, then the minimax values are backed up up through the tree to give the final value.

2.2 Example

In this example, the minimax algorithm has calculated the utility values for one complete move. The terminal values for Max are shown on level three. Max assumes that Min is playing optimally, which means that Min will choose the lowest values for all possible moves. This means that of the three moves Max can make, the highest utility values it can choose from are 3, 2, or 2. Max will therefore choose 3, the highest of those values.



3 Pseudocode

Returns the action corresponding to the best possible move (i.e. the move that leads to the outcome with the best utility value). The functions *Min-Value* and *Max-Value* go through the entire game tree to determine the backed-up value of the state.

```
function Minmax Decision(state) returns some_action
  inputs: state, current_state
  v ← Max-Value(state)
  return the action in Successors(state) with value n
```

```
function Max-Value(state) returns some_utility_value
  if Terminal-Test(state) then return Utility(state)
  v ← ∞
  for a, s in Successors(state) do
    v ← Max(v, Min-Value(s))
  return v
```

```
function Min-Value(state) returns some_utility_value
  if Terminal-Test(state) then return Utility(state)
  v ← ∞
  for a in Successors(state) do
    v ← Min(v, Max-Value(s))
  return v
```

4 Complexity

The minimax algorithm explores the entire game tree using a depth-first approach. If the maximum depth of the tree is m and there are b legal moves at each point, then the time complexity for the algorithm is $O(b^m)$. The space complexity is $O(bm)$ if the algorithm generates all successors at once and $O(m)$ if it generates successors one at a time.

5 Extensions

The minimax algorithm, in its basic form, is useful as a rubric against which other similar algorithms can be judged, but its complexity is far too great for it to be used on its own. A number of strategies have been developed to make the algorithm more practical.

5.1 Alpha-Beta Pruning

5.1.1 Overview

Although Minimax is an effective algorithm for calculating optimal strategies, the number of states it has to examine is exponential with regards to moves. While the exponent cannot be eliminated, the application of alpha-beta pruning can decrease it by half. It does so by only looking at certain nodes in the game tree. Using alpha-beta pruning, it is often possible to prune entire subtrees.

The alpha-beta pruning algorithm is used to examine a node such that the player has a choice of moving to that node. If the player has a better choice m either at the parent node of n or at any choice point further up, then n will never be reached in actual play. Therefore, n can be pruned from the game tree. Alpha-beta pruning gets its name from the parameters that describe values it uses to decide what to prune:

- α = the value for the best (i.e. highest value) choice found so far at any choice point along the path for Max
- β = the value of the best (i.e. lowest value) choice found so far at any choice point along the path for Min

As it moves depth-first down the game tree, the alpha-beta algorithm updates the values of α and β . It prunes the remaining branches at the node as soon as the value of the current node is found to be worse than the current α or β value for Max or Min, respectively.

5.1.3 Complexity

The effectiveness of alpha-beta pruning is highly dependent on the order in which the successors are examined. If the worst successors are generated first, no branches will be pruned. If successors are examined in random order, the total number of nodes examined will be roughly $O(b^{3m/4})$ for moderate b .

Because of this weakness, it is useful to try to first examine successors that are likely to be best. Assuming this can be done, alpha-beta pruning only needs to examine about $O(b^{m/2})$ nodes to pick out the best move, instead of $O(b^m)$. The effective branching factor becomes \sqrt{b} instead of b .

5.2 Real-time decisions

While the use of alpha-beta pruning does decrease how much of the game tree the algorithm needs to search, the algorithm must still search all the way to the terminal states. This is not practical for games where turns must be decided in a

reasonable amount of time. To get around this, the algorithm should cut off the search before the end and apply a *heuristic evaluation function* to the leaves, turning them into terminal leaves. To do this, the algorithm's evaluation function is replaced by a heuristic evaluation function, which gives an estimate of the node's utility, and the terminal test is replaced by a *cutoff test*, which decides when to apply the heuristic evaluation function.

5.2.1 Evaluation Functions

An evaluation function is imprecise by nature. It returns an estimate of the expected utility of the game from a given position, not an absolute value. Obviously, the performance of the algorithm therefore becomes dependent on the quality of its evaluation function. A good evaluation function should order the terminal states in the same way as the true utility function and, for nonterminal states, should be strongly correlated with the actual chances of winning.

Most evaluation functions work by calculating various features of a given state. Taken together, these features define categories of states that have the same values for all features. Any given category will have states that lead to each of the various outcomes. The evaluation function returns a value that reflects the proportion of states with each outcome. The weighted value of these proportions is called the *expected value* and is used for evaluation purposes.

5.2.2 Cutoff tests

When a certain depth is reached, the cutoff test is called. This depth can be pre-defined or can be defined through iterative deepening, but both approaches can lead to errors. To avoid these error, the cutoff test should only be called when no wild swings of utility are predicted in the near future. These are called quiescent moves. Nonquiescent moves can be expanded until quiescent moves are found.

5.3 Multiple Players

To change the minimax algorithm to accept more than one player, the value of each node must be changed to a vector of values. For example, in a three player game with players A, B, and C, a vector $\langle v_A, v_B, v_C \rangle$ is associated with each node. For terminal states, this gives the utility for each state from each player's point of view. The utility function therefore returns a vector of utilities instead of a single value. In this case, each player is effectively Max and wants to choose the move with the highest value in its element of the vector.

Bibliography

Mayefsky, Eric, Francine Anene, and Marina Sirota. "Algorithms - Minimax." *Strategies and Tactics for Intelligent Search*.

<http://www-cs-faculty.stanford.edu/~eroberts/courses/soco/projects/2003-04/intelligent-search/minimax.html>

"Minimax." *Wikipedia: the Free Encyclopedia*.

<http://en.wikipedia.org/wiki/Minimax>

"Opening Moves: Origins of Computer Chess." *Mastering the Game: A History of Computer Chess*. Computer History Museum.

<http://www.computerhistory.org/chess/main.php?sec=thm-42b86c2029762&sel=thm-42b86c6ab9811>

Russel, Stuart J., and Peter Norvig. *Artificial Intelligence, A Modern Approach*.

2nd Edition. Upper Saddle River, New Jersey: Pearson Education, Inc., 2003.