32002: AI (Minimax Search) **Spring 2014**

Lecture 34: April 09, 2014

Instructor: K.R. Chowdhary *: Professor of CS*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

34.1 Minimax Search

Consider the figure 34.1, where the two players (*maximizer* and *minimizer*) play alternately. The game tree is shown with three levels 0 (for root), 1, and 2. The maximizer might hope to get to the situation yielding score 9, he knows that minimizer can choose a move deflecting the play towards the score 2. In general, the maximizer (here at level 0) must take the choices available to the minimizer at next level, into cognizance. And, similarly, the minimizer must also take into cognizance the choices available to maximizer at the next level down.

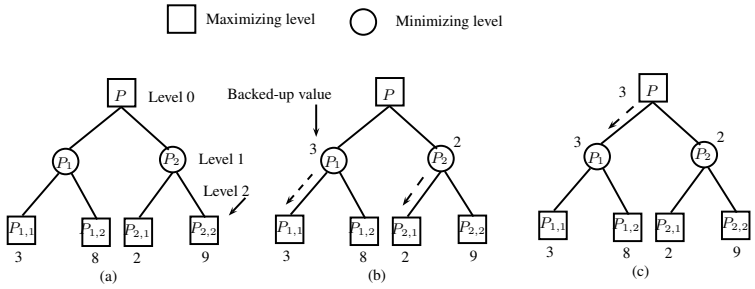


Figure 34.1: MINIMAX Search.

Eventually, the limits of exploration are reached and the static evaluator provides the direct basis for selecting among the alternatives. The minimizer may choose between the values 3 and 2, at the level just up from the static evaluations. Knowing these scores the maximizer at level one up makes the best choice between 2 and 3, at level zero.

The procedure by which the scoring information passes up the game tree is called MINIMAX algorithm, because the score at any level i is either minimum or the maximum of what is available at the level $i + 1$. The recursive form of the MINIMAX algorithm is shown as Algorithm 1.

The idea of *minimaxing* is to translate the board into a *static number*. However, the process may be expensive due to generation of large number of paths.

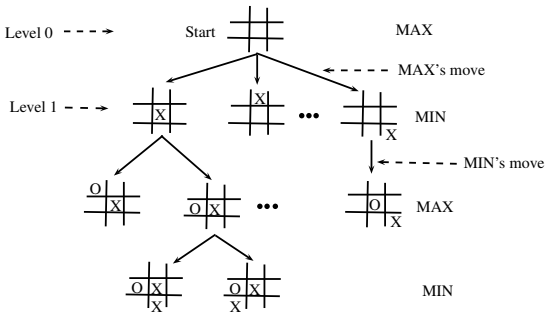


Figure 34.2: Tic-toc-toe with some initial moves.

Algorithm 1 MINIMAX-Algorithm

```
1: if limit of search has reached then
2:    compute the static value of current position relative to the appropriate player;
3:    Report the result;
4: else
5:    if level is minimizing level then
6:       call MINIMAX-Algorithm on the children of current position;
7:       Report minimum of the result;
8:    else
9:       (level is maximizing level.)
10:      call MINIMAX-Algorithm on the children of current position;
11:      Report maximum of the result;
12:   end if
13: end if
```

The *minimax* algorithm has the following properties:

1. The algorithm is *complete*, if tree is finite.
2. The Time complexity is $O(b^d)$, where b is branching factor, and d is total distance.
3. The Space complexity is $O(bd)$ equal to depth-first exploration.

Example 34.1 *Tic-tac-toe Game*

The tic-tac-toe is a board game, with $3 \times 3 = 9$ positions, and played between two players through alternate moves. The game has total $9! = 362880$ configurations, and many of them collapses into single configurations due to symmetry, hence the actual number of configurations are far less. Initial configuration is empty board. The one player, called as MAX makes the first move with 'X', and the other player, called MIN player, makes a 'O' move. There are total three 'X' and three 'O' markers. The player which is first in arranging all 'X' or 'O' array as row or column or diagonal wins the game. Figure 34.2 shows some initial configurations and moves of this game.

The game of tic-tac-toe generates all the moves at level i before generating the moves of level $i+1$, hence, it is BFS. We note that while in start configuration (level 0) having all empty positions, the player MAX can make moves to level 1. The MAX will choose moves in such a way that it leads to a configuration ultimately

such that all the X's are continuous and are in a line, or column or diagonal. At the same time the MAX will choose such a move that it becomes difficult for MIN later to make a move to a configuration with all O's in a line, or column or diagonal. Thus, each of MAX and MIN makes the moves so that it is winning for them and blocking for the other player.

When all the six objects (X, and O) are on board, the subsequent moves by MAX or MIN are shifting of these X or O to lead them to corresponding winning configurations.

To indicate which configuration is superior to move into, we associate a weight function that will be maximized by the MAX player, and the MIN player will try to minimize it. If a configuration is indicated by c, then this function is defined as:

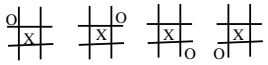
$$f(c) = m - n \tag{34.1}$$

where $m =$ number of complete rows, columns, and diagonals that are till open for the next move of MAX, and

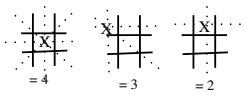
$n =$ number of complete rows, columns, and diagonals that are till open for the next move of MIN.

If c is a winning configuration for MAX, then $f(c) = \infty$, if c is win for MIN then $f(c) = -\infty$ if a configuration is,

then $f(c) = 2 - 6 = -4$. we make use of symmetries in generating successor positions, then the following game states are identical.



For efficiency reasons, early in the game, the branching factor is kept small by symmetries, but late in the game, it is kept small by number of open positions available. The first move is always important. The three initial moves for X in the following configurations, leaves different alternatives for next move for X.



The figure 34.3 demonstrates the computation of this weight function at each of the node. To win the game, the MAX at level 0, should make move to such a node at level 1 that produces maximum value of function $f(c)$ at level 1. It is this maximum value that is backup at level 0. This value is as a result of maximum value of the corresponding nodes that are under the MAX node at level say i, (that is, those at level $i + 1$). The MIN node at level $i + 1$ should make a move which produces minimum value at level $i + 2$, from its expanded nodes at level $i + 3$. This process goes on until the *maximizer* sees the *static values* at the last level (bottom) of the tree. Similarly, the *minimizer* will also, choose the next move by attempting to see the static values at the last node.

The static values at the bottom nodes in this example are shown as 3, 2, ..., 3. The values at the upper levels, i.e, parents of static nodes, and their parents are called *backup values.* The backup values are based

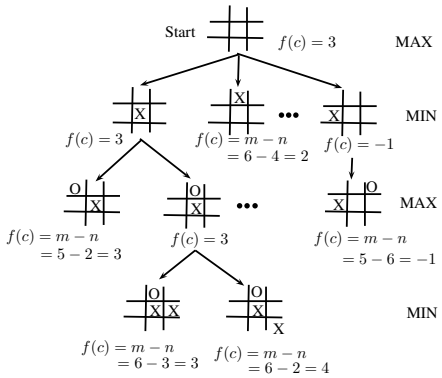


Figure 34.3: Tic-tac-toe with backup of static values.

on the "look-ahead" of the game-tree, and this depends on the lower levels nodes, in fact even those that are near the end of the game-tree.

Some of the nodes may represent win for MIN, hence they may have value $-\infty$. When evaluations are backed-up, the MAX's best move is that one which avoids his immediate defeat. In the case of MAX, the win corresponds to node value of $+\infty$. □

References

[1] Chowdhary K.R. (2020) Logic and Reasoning Patterns. In: Fundamentals of Artificial Intelligence. Springer, New Delhi. https://doi.org/10.1007/978-81-322-3972-7_11