MIT EECS
6.034: Artificial Intelligence
(Fall 2006)
Instructor: Luis E. Ortiz

Notes on Search in Games [1]

Minimax Search

This section describes the pseudocode for the MINIMAX procedure. Given a node n in the game tree, MINIMAX returns the *minimax value* of n. i.e., the value that the player who has the choice to move at n can achieve assuming the player and its opponent play *optimally* thereafter. MINIMAX is initially called with n as some node of the full game tree for which a player needs to make a choice. Considering n as the root of the game subtree we want to evaluate, recall that MINIMAX usually runs only up to a certain depth down from n. A *terminal node* of the game subtree rooted at the starting node n is either a leaf node of the full game tree, which corresponds to an actual terminal state of the game, or a node at a given depth from n at which search is cut off. Only terminal nodes are *statically evaluated*.

```
MINIMAX(n)
if n is a terminal node then
  return value(n)
end if
if n is a maximizer node then
  v ← −∞
  for all children s of n do
    v ← max(v, MINIMAX(s))
  end for
  return v
end if
if n is a minimizer node then
  v ← ∞
  for all children s of n do
    v ← min(v, MINIMAX(s))
  end for
  return v
end if
```

Alpha-Beta Pruning

This section describes the pseudocode for the Minimax-with-alpha-beta-pruning procedure ALPHA-BETA. Given as inputs a node n and the upper and lower bound α and β, respectively, it returns the value assigned to n conditioned on α and β. In particular, if n is a *maximizer* node and v is the *maximum* value of only the children of node n that ALPHA-BETA has evaluated, then if at any time $v \geq \beta$, ALPHA-BETA returns β; otherwise, we have $v < \beta$ for all the children of n and therefore ALPHA-BETA returns v. Similarly, if n is a *minimizer* node and v is the *minimum* value of only

[1] These notes are primarily based on the recitation notes of Kimberle Koile and the slides of Tomás Lozano-Perez from previous terms, as well as the books *Artificial Intelligence (Third Edition)* by Patrick Henry Winston and *Artificial Intelligence: A Modern Approach (Second Edition)* by Stuart Russell and Peter Norvig. Original date: October 6, 2004; Last updated: October 22, 2006.

the children of node n that ALPHA-BETA has evaluated, then if at any time $v \leq \alpha$, ALPHA-BETA returns β; otherwise, we have $v > \alpha$ for all the children of n and therefore ALPHA-BETA returns v.

ALPHA-BETA is initially called with n as the root of the game subtree, i.e., the node at which a player needs to make a move. Also, ALPHA-BETA is initially called with the bound-parameters values $\alpha = -\infty$ and $\beta = \infty$. (Read the next section of this note on progressive deepening for other possible initializations of α and β.)

ALPHA-BETA(n, α, β)
if n is a terminal node **then**
  return value(n)
end if
if n is a **maximizer** node **then**
  $v \leftarrow -\infty$
  for all children s of n **do**
    $v \leftarrow \max(v,$ ALPHA-BETA$(s, \alpha, \beta))$
    if $v \geq \beta$ **then**
      Play will never reach n! Because at some node on the path to n the **minimizer** *has at least as good a move as the one that leads to n.*
      return v
    end if
    $\alpha \leftarrow \max(v, \alpha)$
  end for
  return v
end if
if n is a **minimizer** node **then**
  $v \leftarrow \infty$
  for all children s of n **do**
    $v \leftarrow \min(v,$ ALPHA-BETA$(s, \alpha, \beta))$
    if $v \leq \alpha$ **then**
      Play will never reach n! Because at some node on the path to n the **maximizer** *has at least as good a move as the one that leads to n.*
      return v
    end if
    $\beta \leftarrow \max(v, \beta)$
  end for
  return v
end if

The parameter α can be interpreted as the *maximum* value that the *maximizer* player can guarantee itself from any node (i.e., choice point) in the path from the root of the game subtree under consideration to the node n. Similarly, β corresponds to the *minimum* value that the *minimizer* player can guarantee itself from any node in the path to n; in other words, the maximum value that the maximizer can achieve, or equivalently, that the minimizer can force the maximizer to obtain, if the maximizer ever moves to n.

The effectiveness of ALPHA-BETA is largely dependent on the order in which the procedure evaluates the nodes in the game tree. ALPHA-BETA is guaranteed to be most effective when it evaluates the children of the *maximizer* nodes in the order of *the highest to the lowest* valued, while

it evaluates the children of the *minimizer* nodes in the order of *the lowest to the highest* valued.

Progressive deepening

Progressive deepening, also known as iterative deepening, solves search problems by systematically increasing the search depth or horizon of evaluation, i.e., the cutoff depth, of the game tree. Progressive deepening is a very useful search technique in general and can also applied along with any of the traditional search methods. It allows us to have an answer ready *at any time* if one is needed by first solving the problem up to some small depth. In the context of games, for instance, the longer we run the algorithm, the deeper we go into the game tree and the better the decision about which of the choices for the player is best. Therefore, applications of progressive deepening lead to what are called *anytime algorithms*.

We can use progressive deepening along with Alpha-Beta by using the solutions from applying Alpha-Beta up to a certain depth to reorder the nodes in the game tree so as to hopefully make later runs of Alpha-Beta that go deeper into the tree more efficient. Another heuristic that can be used during progressive deepening is to use the $\alpha - \beta$ bounds that Alpha-Beta obtained at each node in previous steps of progressive deepening to initialize the bounds for further applications of progressive deepening (instead of just using $[\alpha = -\infty, \beta = \infty]$ every time we restart at the root of the current game subtree).

Computational Considerations

In the discussion that follows on the running time and space complexity of the search in games algorithms described in the previous sections of this note, assume that the *branching factor* b and the *cutoff depth* d of the game subtree under consideration are constant. For simplicity of presentation, the expressions involving b and d should be interpreted as "on the order of." (For those familiar with the theory of algorithms, this means that the O-notation is omitted from the expressions.)

Space Complexity

All the methods require *bd space*.

Running Times

Both Minimax and Alpha-Beta exhibit *exponential explosion* in the worst case: their worst-case running time is b^d. For Minimax this is in fact *always* the case because, like in British-Museum search, it always evaluates *every* possible sequence of play before making a decision. In the *best case*, Alpha-Beta, i.e., minimax with alpha-beta pruning, runs in time $b^{d/2}$, thus allowing us to evaluate a game tree *twice* as deep as we could with plain Minimax. More specifically, the best-case running time of Alpha-Beta is $2b^{d/2} - 1$ if d is even and $b^{(d+1)/2} + b^{(d-1)/2} - 1$ if d is odd; on the *average-case*, it is $b^{3d/4}$, making it quite effective in practice.