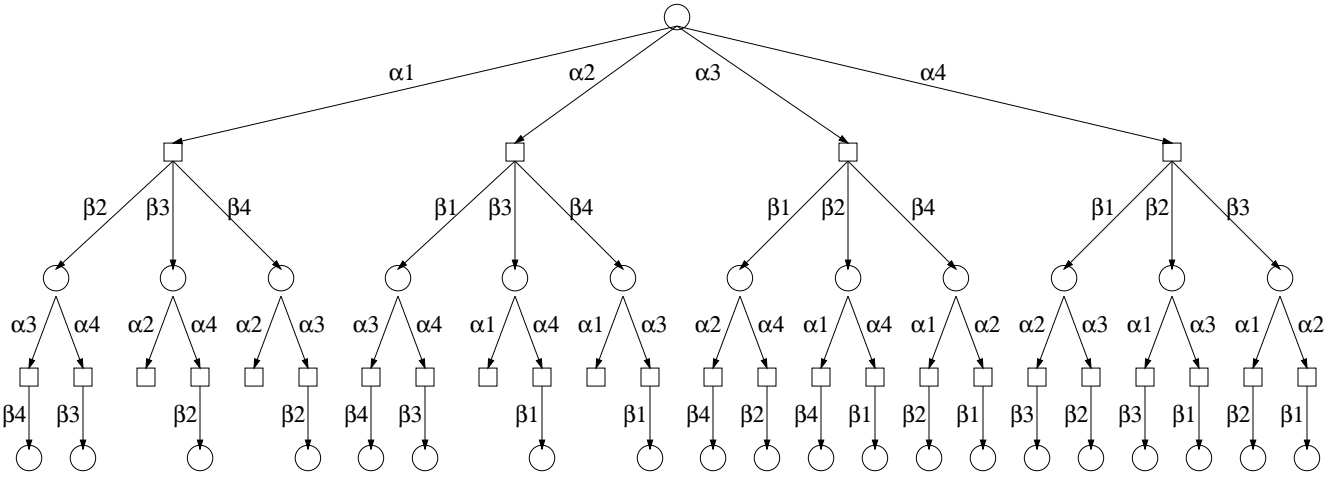
CS/ENGRI 172, Fall 2003: Computation, Information, and Intelligence
9/15/03: Game Playing

Game Trees

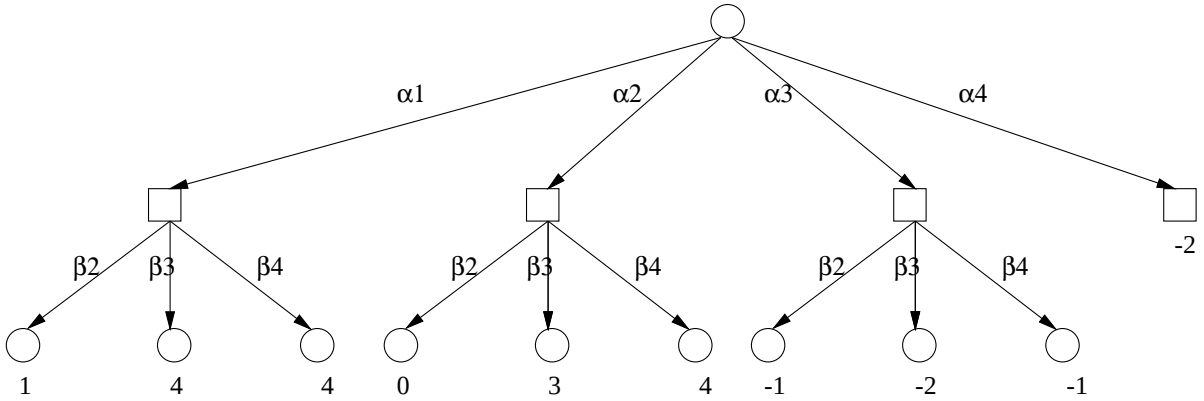
The "3-game" is two players taking turns selecting unique numbers from 1 to 4; a player wins if they get two numbers summing to 3. We'll let ◯ be nodes where the corresponding problem space state has player 1 move next, and □ be nodes where the corresponding problem space state has player 2 move next. The operator α_i means that player 1 choses the number i; the operator β_j means that player 2 choses the number j. The following is the *complete* game tree for the 3-game:



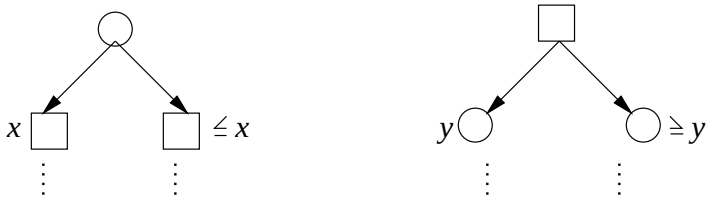
The *minimax score* of a node (in a zero-sum game, with respect to a given assignment of values to the game tree's leaf nodes) is the eventual benefit to player 1, assuming optimal play by all parties, of being in the state corresponding to that node. We can compute the minimax score of any internal node if we know the minimax score of all its children.

Evaluation function example

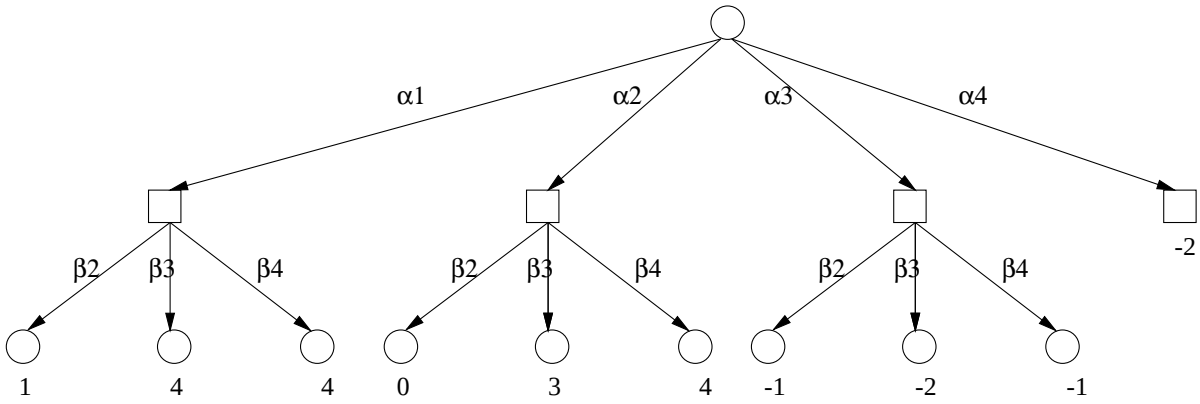
Here is an example of using an evaluation function on a search-limited (i.e., pruned) version of our 3-game game tree. This particular evaluation function happens to be pretty good - usually relatively high scores are given to nodes corresponding to states from which player 1 could potentially win.



Alpha-beta pruning (This name is unrelated to the operator labels we've been using.)
The fundamental principle is that nodes whose (corresponding states') values can't affect decisions "higher up" need not be examined. We will use DFS as an organizing principle for examining nodes and *propagating constraints*. Two canonical situations in which nodes can be pruned are as follows; the shape of the node indicates whose turn it is to play in the corresponding state, and next to each node our current knowledge about its minimax value is indicated:



In either situation, it is not necessary to consult or compute the minimax values of any heretofore unseen nodes in the right nodes' subtrees, because the leftmost operator will be chosen regardless of their values.



Pruning self-checks
As a self-check, see that you can verify the following results for yourself. "Pseudo-minimax" refers to values calculated in a minimax fashion from an evaluation function's approximation of the leaves' true minimax values.

1. Applying dfs-style alpha-beta pruning to the full 3-game game tree on the previous handout (assigning values of 1 to leaves where p1 wins, -1 where p1 loses, and 0 where there is a tie) should result in having to look at *only* the values of which leaves? (See footnote[1].)

2. The alpha-beta pruning constraints for a given node are always consistent with the node's actual (psuedo-)minimax value. (This gives you a way to check your work, by the way.)

3. Assume a full game tree is accessible, and that player 1 adopts the "minimax strategy", always choosing the operator that yields maximum minimax value. Player 2 *cannot*, via suboptimal play, cause player 1 to get benefit less than the minimax value of the root. However, if player 2 plays suboptimally, player 1 may get lower benefit via the minimax strategy than would have been possible if they had known player 2's strategy ahead of time and incorporated it into their choice of moves.

[1]The only leaves touched are 1.1.1.1, 1.1.2.1, 1.2.1, 1.3.1, 2.1.1.1, 2.1.2.1, 3.1.1.1, 3.1.2.1, 4.1.1.1, and 4.1.2.1.