

Now let's look at some of the details:

### **Designing an evaluation function**

Heuristic evaluation function is typically a weighted sum of feature measures:

$$w_1f_1 + w_2f_2 + \dots + w_nf_n$$

Steps in designing an evaluation function:

- Pick informative features.
- Find the weights that make the program play well.
- Be sure that the features can be measured/calculated efficiently.

Should agree with the utility function on terminal states.

---

### **Improving Minimax: $\alpha$ - $\beta$ pruning**

Minimax search is essentially depth-first search with the additional overhead of applying the evaluation function and keeping track of state values.

While the space complexity of depth-first search is good, the time complexity is not.

Is there a way to improve upon that? Yes. There might be opportunities to not expand nodes at the “right” of the search tree, based upon what you’ve learned at the left side. Techniques that do this type of pruning of the search space are called branch and bound techniques.

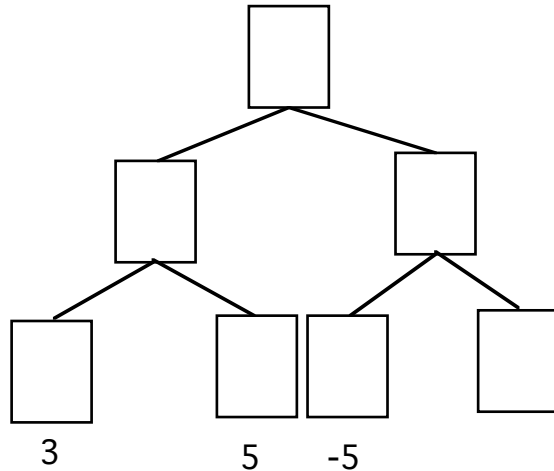
Alpha-beta ( $\alpha$ - $\beta$ ) pruning is a branch-and-bound algorithm for two bounds.

---

### **The $\alpha$ - $\beta$ principle**

If you have an idea that is surely bad, do not take time to see how truly awful it is.

Here’s a very small example. Consider the following search tree:



The first player to move is the maximizer. The maximizer has explored the left side of the tree, determining that the expected value of the move to the “left” is 3. The maximizer is now considering the right branch. The maximizer has evaluated the leaf with a value of  $-5$ . At this point, the maximizer realizes that the minimizer will be the one choosing between the move that yields  $-5$  and its sibling. The minimizer will surely choose either  $-5$  or worse! Since the right branch at this point looks very bad for the maximizer, there is no point in actually evaluating the one node that remains.

---

### $\alpha$ - $\beta$ Search

Let

$c$  = search cutoff (i.e., the depth limit)

alpha ( $\alpha$ ) = lower bound on Max’s outcome; initially set to  $-\infty$ .

i.e., the current best score for Max.

beta ( $\beta$ ) = upper bound on Min’s outcome; initially set to  $+\infty$ .

i.e., the current best score for Min.

Idea

On a minimizing level, if you find a value  $< \alpha$ , cut the search.

On a maximizing level, if you find a value  $> \beta$ , cut the search.

---

### $\alpha$ - $\beta$ Search Algorithm

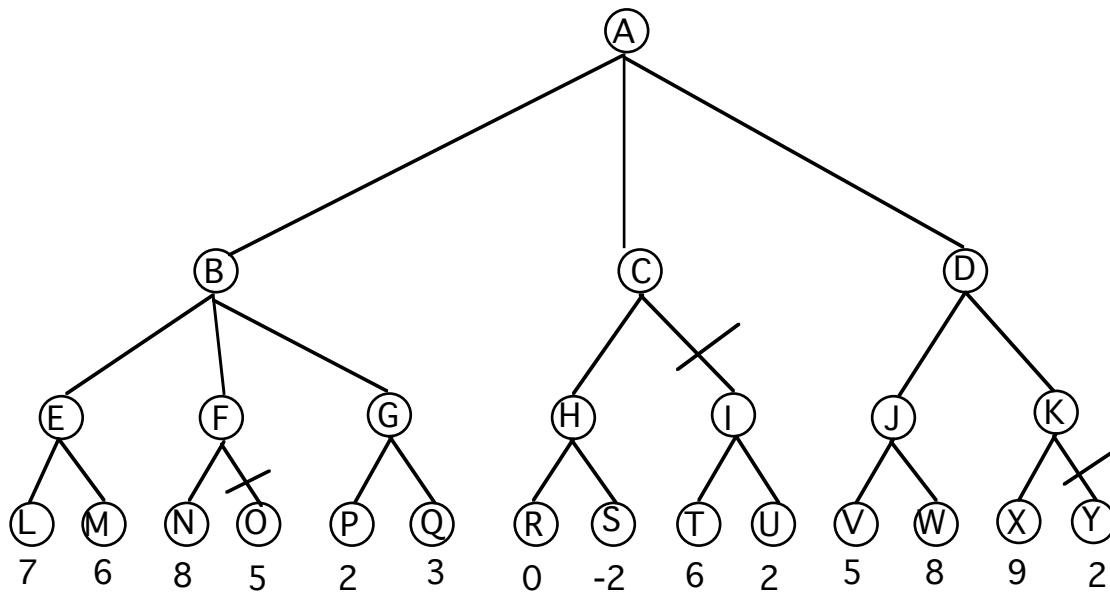
If the limit of search has been reached, compute  $e(n)$  (i.e., evaluation of  $n$ ) and report the result.

Else, if the level is a minimizing level,

While there are children of this node  
 Call  $\alpha$ - $\beta$  search on child with current values of  $\alpha$  and  $\beta$ ; note the value  $v$  that is returned.  
 If  $v < \beta$ , reset  $\beta$  to  $v$ .  
 If  $\alpha \geq \beta$ , return  $\alpha$ .  
 Return  $\beta$ .

Else the level is a maximizing level,  
 While there are children of this node  
 Call  $\alpha$ - $\beta$  search on child with current values of  $\alpha$  and  $\beta$ ; note the value  $v$  that is returned.  
 If  $v > \alpha$ , reset  $\alpha$  to  $v$ .  
 If  $\alpha \geq \beta$ , return  $\beta$ .  
 Return  $\alpha$ .

**Example 4.**  $\alpha$ - $\beta$  returns the same value as minimax.



### Search Space Size Reductions

$\alpha$ - $\beta$ 's success depends upon the ordering of the states.

**Worst Case:** In an ordering where all "worst" options are evaluated first, all nodes must be examined.

**Best Case:** What if “best” nodes are evaluated first?