Lecture

Refinements to Adversary Search: Alpha-Beta Pruning;

[Same Reading]
Min Max Trees: From Last Time......

The Algorithm:

(1) Generate the tree (more on this later!);

(2) Label the nodes by traversing the tree post-order and:

   (A) At leaf nodes, use the eval() function;

   (B) At Max nodes, back up the maximum value of any child; and

   (C) At Min nodes, back up the minimum value of any child.

(3) Choose the move that corresponds to the largest child of the root (which gave the root its value).
Min-Max Trees: From Last Time.....

Depth-bounded post-order traversal of a min-max tree down to some fixed depth D:

```java
Move chooseMove(Node t) {
    int max = -Inf;   Move best;
    for (each move m to a child c of t) {
        int val = minMax( c, 1 );
        if (val > max) {
            best = m; max = val;
        }
    }
    return best;
}
```

```java
int minMax(Node t, int depth) {
    if (t is a leaf node || depth == D)
        return eval(t);
    else if (t is max node) {
        int val = -Inf;
        for (each child c of t)
            val = max(val, minMax( c, depth+1 ));
        return val;
    } else {                       // is a min node
        int val = Inf;
        for (each child c of t)
            val = min(val, minMax( c, depth+1 ));
        return val;
    }
}
```
What are my next two questions?

One: What’s wrong with this (if anything)?

Two: How can we make it more efficient?

How can we “prune” the search space so that we find the best paths faster and eliminate useless nodes?
Refinements to Min-Max Search

**Main Problem:**
You have only a limited amount of time to search, and the combinatorial explosion of the search space makes it imperative that you use your time effectively to find the best move.

How can we find good nodes faster?
How can we avoid useless nodes?
Refinements to Min-Max Search

**How to Find Good Nodes Faster**

Basic Idea: We can apply best-first search to the Min-Max tree:

Instead of searching children in some random order, apply `eval()` to each child, and search them in descending order of value.

So: generate all children, sort (descending order at Max nodes, ascending order at Min nodes), then search in order. You could cut off the search after some number of children (e.g., just search half, or use a threshold value).

Note: Since we applying `eval()` to all nodes, perhaps we want to create a simpler, more efficient version for this part of the search.
How can we avoid useless nodes?

Some of the nodes you examine might be completely useless – they will NEVER be on the intended path which determines the choice of the next move.

The intended path is easy to see: it is the path the root’s value took from the leaf (where it was created by eval() ) to the top of the tree.
How can we avoid useless nodes?

Some of the nodes you examine might be completely useless – they will NEVER be on the intended path which determines the choice of the best next move. Why?

a) Clearly bad for you (e.g., you sacrifice your Queen to capture a Pawn!) or a WIN for you
How can we avoid useless nodes?

Some of the nodes you examine might be completely useless – they will NEVER be on the intended path which determines the choice of the best next move.

a) Clearly bad for you (e.g., you lose your Queen) or a WIN for you.

Solution: Apply Eval() during search and set thresholds at which you stop searching below that node.

Need separate thresholds for Min and Max, or just reverse them by multiplying by -1.

```java
final int MAX_THRESHOLD = Infinity;
final int MIN_THRESHOLD = -300;

int minMax(Node t, int depth) {
    int e = eval(t);
    if( t is a leaf node (no moves) || depth == D)
        return e;
    else if( t is max node ) {
        if( e >= MAX_THRESHOLD || e <= MIN_THRESHOLD)
            return e;
    ...
    }
    else if( t is min node ) {
        if( e <= -MAX_THRESHOLD || e >= -MIN_THRESHOLD)
            return e;
    ...
    }
```
How can we avoid useless nodes?

Some of the nodes you examine might be completely useless – they will NEVER be on the intended path which determines the choice of the best next move.

a) Clearly bad for you (e.g., you lose your Queen) or a WIN for you. Solution: Use eval to filter out “useless nodes.”

b) Some nodes are useless because we have already found “good enough” nodes in other parts of the tree. This is called α-β-Pruning
α-β-Pruning: Examples

We have seen that we can stop searching below a node when we have found a “good enough” node such as a Win:

Why search these nodes?
Refinements to Min-Max Search: Pruning the search space

**α-β-Pruning:** Examples

But then why search any of the siblings? You’ve already found a “good enough” node to pass to the parent!

The siblings are useless as well!

Why search any of these nodes?

So, if you find a Win when searching the children left to right, stop searching and return Infinity!
Refinements to Min-Max Search: Pruning the search space

**α-β-Pruning: Examples**

But then why search any of the siblings? You’ve already found a “good enough” node to pass to the parent!

The siblings are useless as well!

A similar argument works at Minimizing nodes: if you find a Losing node, stop (that’s the one your opponent will choose!)
α-β-Pruning: Examples

The general idea is: If you can not possible improve on the best value you’ve found so far going post-order through the children, STOP!
Refinements to Min-Max Search: Pruning the search space

**α-β-Pruning: Examples**

**The general idea is:** If you can not possible improve on the best value you’ve found so far going post-order through the children, STOP!

The best value found so far!
Refinements to Min-Max Search: Pruning the search space

α-β-Pruning: Examples

The general idea is: If you can not possible improve on the best value you’ve found so far going post-order through the children, STOP!

How could you find a better value here??

The best value found so far!
Refinements to Min-Max Search: Pruning the search space

α-β-Pruning: Examples

The general idea is: If you can not possible improve on the best value you’ve found so far going post-order through the children, STOP!
Refinements to Min-Max Search: Pruning the search space

**α-β-Pruning: Examples**

But sometimes this kind of argument works among different levels of the tree!

Suppose we start searching post-order in this tree...

The root “knows” he can get at least a 5 from the left-most child, but what if a value > 5 can be found in the rest of the children? Keep searching!

The best value found so far! Final value will be ≥ 5.
Refinements to Min-Max Search: Pruning the search space

**α-β-Pruning: Examples**

But sometimes this kind of argument works among different levels of the tree!

YES! Found a 13. Can a value > 13 be found in the remaining child? Keep searching......

The best value found so far! Final value will be $\geq 13$. 

The image shows a decision tree with nodes labeled with values and arrows indicating the direction of the search. The tree branches out with max and min nodes, illustrating the pruning process where nodes are pruned based on the values they contain.
α-β-Pruning: Examples

But sometimes this kind of argument works among different levels of the tree!

At the next level down, a 2 has been backed up to a “grandparent”. This means the right-most (minimizing) child knows he can get ≤ 2.
α-β-Pruning: Examples

But sometimes this kind of argument works among different levels of the tree!

At the next level down, a 2 has been backed up to a “grandchild”. This means the right most (minimizing) child “knows” he can get 2 or less.

But then there is NO WAY any value from the right-most child can “beat” the 13. STOP!
α-β-Pruning: Basic Principles

Among adjacent levels, then, we could generalize this as follows for post-order traversal below a Max node:

Keep track of the max value $\alpha$ among the children of all Max nodes, and the min value $\beta$ among the children of all Min nodes. These are the “best-so-far” values.

If the best-so-far value of a grandchild is less than a Max nodes best-so-far value, i.e.,

$$\text{if}(\ \beta < \alpha\ ) \ \text{STOP!}$$
**α-β-Pruning: Basic Principles**

Way Complicated, right?

Not really..... let’s look at it this way.....

If we could prove that a value K in a node can NEVER make its way from the leaf (where it got generated by Eval()) to the root, then we don’t have to evaluate that node or any nodes below it.

How does a value K survive the trip root-ward?

It has to be the largest child at Max nodes and the smallest child at Min nodes.
Refinements to Min-Max Search: Pruning the search space

α-β-Pruning: Basic Principles

Way Complicated, right?

Not really..... let’s look at it this way.....

If we could prove that a value K in a node can NEVER make its way from the leaf (where it got generated by Eval()) to the root, then we don’t have to evaluate that node or any nodes below it.

How does a value K survive the trip root-ward?

It has to be the largest child at Max nodes and the smallest child at Min nodes.

Suppose the blue values are best-so-far found during post-order traversal. We can update the best-so-far based on finding the 15 at a leaf node.....
α-β-Pruning: Basic Principles

Way Complicated, right?

Not really..... let’s look at it this way.....

If we could prove that a value K in a node can NEVER make its way from the leaf (where it got generated by Eval()) to the root, then we don’t have to evaluate that node or any nodes below it.

How does a value K survive the trip root-ward?

It has to be the largest child at Max nodes and the smallest child at Min nodes.
**α-β-Pruning: Basic Principles**

Way Complicated, right?

Not really..... let’s look at it this way.....

If we could prove that a value K in a node can NEVER make its way from the leaf (where it got generated by Eval()) to the root, then we don’t have to evaluate that node or any nodes below it.

How does a value K survive the trip root-ward?

It has to be the largest child at Max nodes and the smallest child at Min nodes.
α-β-Pruning: Basic Principles

Way Complicated, right?

Not really..... let’s look at it this way.....

If we could prove that a value K in a node can NEVER make its way from the leaf (where it got generated by Eval()) to the root, then we don’t have to evaluate that node or any nodes below it.

How does a value K survive the trip root-ward?

It has to be the largest child at Max nodes and the smallest child at Min nodes.
α-β-Pruning: Basic Principles

Way Complicated, right?

Not really..... let’s look at it this way.....

If we could prove that a value K in a node can NEVER make its way from the leaf (where it got generated by Eval()) to the root, then we don’t have to evaluate that node or any nodes below it.

How does a value K survive the trip root-ward?

It has to be the largest child at Max nodes and the smallest child at Min nodes.
Refinements to Min-Max Search: Pruning the search space

**α-β-Pruning: Basic Principles**

Way Complicated, right?

Not really..... let’s look at it this way.....

If we could prove that a value K in a node can NEVER make its way from the leaf (where it got generated by `Eval()`) to the root, then we don’t have to evaluate that node or any nodes below it.

How does a value K survive the trip root-ward?

It has to be the largest child at Max nodes and the smallest child at Min nodes.
Refinements to Min-Max Search: Pruning the search space

α-β-Pruning: Basic Principles

Way Complicated, right?

Not really..... let's look at it this way.....

If we could prove that a value K in a node can NEVER make its way from the leaf (where it got generated by Eval() ) to the root, then we don’t have to evaluate that node or any nodes below it.

How does a value K survive the trip root-ward?

It has to be the largest child at Max nodes and the smallest child at Min nodes.
Refinements to Min-Max Search: Pruning the search space

α-β-Pruning: Basic Principles

So, here is the simpler way to look at it:

15 > every value at Max nodes
   is same as
15 > max( all values at Max nodes)

and

15 < every value at Min nodes
   is same as
15 < min( all values at Min nodes)
α-β-Pruning: Basic Principles

So, here is the simpler way to look at it:

Suppose

- N is a node we are visiting, with a value calculated as $V_N$;
- $\alpha_0 ... \alpha_k$ is the set of values calculated at Max nodes on the path from N to root;
- $\beta_0 ... \beta_j$ is the set of values calculated at Min nodes on the path from N to root;
Refinements to Min-Max Search: Pruning the search space

α-β-Pruning: Basic Principles

So, here the simpler way to look at it:

Suppose

- N is a node we are visiting, with a value calculated as $V_N$;
- $\alpha_0 \ldots \alpha_k$ is the set of values calculated at Max nodes on the path from N to root;
- $\beta_0 \ldots \beta_j$ is the set of values calculated at Min nodes on the path from N to root;

Then N is only **useful** if

$$\max(\alpha_0 \ldots \alpha_k) \leq V_N \leq \min(\beta_0 \ldots \beta_j)$$
Refinements to Min-Max Search: Pruning the search space

**α-β-Pruning: Basic Principles**

So, here the simpler way to look at it:

**Suppose**

- N is a node we are visiting, with a value calculated as $V_N$;
- $\alpha_0, \ldots, \alpha_k$ is the set of values calculated at Max nodes on the path from N to root;
- $\beta_0, \ldots, \beta_j$ is the set of values calculated at Min nodes on the path from N to root;

Then N is only **useful** if

$$\max(\alpha_0, \ldots, \alpha_k) \leq V_N \leq \min(\beta_0, \ldots, \beta_j)$$

**Punchline:** If $\max(\alpha_0, \ldots, \alpha_k) > \min(\beta_0, \ldots, \beta_j)$ then $V_N$ can NEVER be useful. STOP!

$$\alpha_3 \ldots \alpha_0 \ldots \alpha_k \ldots \alpha_7$$
$$\beta_4 \ldots \beta_2 \ldots \beta_0 \ldots \beta_j$$

No possible $V_N$ values!
Refinements to Min-Max Search: Pruning the search space

α-β-Pruning: Basic Principles

How can we make this into an algorithm?

- Keep track of the max value α found so far at Max nodes;
- Keep track of the min value β found so far at Min nodes;
- If ever β < α, STOP!

```c
int final Inf = 1000000

Move chooseMove(Node t) {
    int max = -Inf; Move best;
    for(each move m to a child c of t) {
        int val = minMax( c, 1, -Inf, Inf );
        if(val > max) { best = m; max = val ;}
    }
    return best; }

int minMax(Node t, int depth, int alpha, int beta ) {
    if( t is a leaf node (no moves) || depth == D)
        return eval(t); // stop searching and return eval
    else if( t is max node ) {
        int val = -Inf;
        for(each child c of t) {
            alpha = max(alpha, val); // update alpha with max so far
            if(beta < alpha) break; // terminate loop
            val = max(val, minMax( c, depth+1, alpha, beta ));
        }
        return val;
    } else { // is a min node
        int val = Inf;
        for(each child c of t) {
            beta = min(beta, val); // update beta with min so far
            if(beta < alpha) break; // terminate loop
            val = min(val, minMax( c, depth+1, alpha, beta ));
        }
        return val;
    }
}
```
α-β-Pruning: Basic Principles

How can we make this into an algorithm?

- Keep track of the max value $\alpha$ found so far at Max nodes;
- Keep track of the min value $\beta$ found so far at Min nodes;
- If ever $\beta < \alpha$, STOP!

LET'S TRY IT.....
Refinements to Min-Max Search: Pruning the search space

α-β-Pruning: Basic Principles

How can we make this into an algorithm?
- Keep track of the max value $\alpha$ found so far at Max nodes;
- Keep track of the min value $\beta$ found so far at Min nodes;
- If ever $\beta < \alpha$, STOP!

LET'S TRY IT.....
**α-β-Pruning:**

**Basic Principles**

How can we make this into an algorithm?
- Keep track of the max value \( \alpha \) found so far at Max nodes;
- Keep track of the min value \( \beta \) found so far at Min nodes;
- If ever \( \beta < \alpha \), STOP!

**LET’S TRY IT…..**
Refinements to Min-Max Search: Pruning the search space

α-β-Pruning: Basic Principles

How can we make this into an algorithm?
- Keep track of the max value \( \alpha \) found so far at Max nodes;
- Keep track of the min value \( \beta \) found so far at Min nodes;
- If ever \( \beta < \alpha \), STOP!

LET’S TRY IT.....
Refinements to Min-Max Search: Pruning the search space

**α-β-Pruning:**
Basic Principles

How can we make this into an algorithm?
- Keep track of the max value $\alpha$ found so far at Max nodes;
- Keep track of the min value $\beta$ found so far at Min nodes;
- If ever $\beta < \alpha$, STOP!

**LET’S TRY IT…..**
Refinements to Min-Max Search: Pruning the search space

α-β-Pruning:
Basic Principles

How can we make this into an algorithm?

- Keep track of the max value α found so far at Max nodes above or at the current node;
- Keep track of the min value β found so far at Min nodes above or at the current node;
- If ever β < α, STOP!

LET’S TRY IT.....
α-β-Pruning: Basic Principles

How can we make this into an algorithm?

- Keep track of the max value \( \alpha \) found so far at Max nodes;
- Keep track of the min value \( \beta \) found so far at Min nodes;
- If ever \( \beta < \alpha \), STOP!

LET'S TRY IT.....
**α-β-Pruning:**

**Basic Principles**

How can we make this into an algorithm?

- Keep track of the max value $\alpha$ found so far at Max nodes;
- Keep track of the min value $\beta$ found so far at Min nodes;
- If ever $\beta < \alpha$, STOP!

**LET’S TRY IT…..**
**α-β-Pruning: Basic Principles**

How can we make this into an algorithm?

- Keep track of the max value $\alpha$ found so far at Max nodes;
- Keep track of the min value $\beta$ found so far at Min nodes;
- If ever $\beta < \alpha$, STOP!

**LET'S TRY IT.....**
Refinements to Min-Max Search: Pruning the search space

α-β-Pruning: Basic Principles

How can we make this into an algorithm?
- Keep track of the max value α found so far at Max nodes;
- Keep track of the min value β found so far at Min nodes;
- If ever β < α, STOP!

LET’S TRY IT.....
Refinements to Min-Max Search: Pruning the search space

α-β-Pruning:
Basic Principles

How can we make this into an algorithm?
- Keep track of the max value \( \alpha \) found so far at Max nodes;
- Keep track of the min value \( \beta \) found so far at Min nodes;
- If ever \( \beta < \alpha \), STOP!

LET'S TRY IT.....
α-β-Pruning: Basic Principles

How can we make this into an algorithm?

- Keep track of the max value $\alpha$ found so far at Max nodes;
- Keep track of the min value $\beta$ found so far at Min nodes;
- If ever $\beta < \alpha$, STOP!

LET’S TRY IT…..
Refinements to Min-Max Search: Pruning the search space

\textbf{α-β-Pruning: Basic Principles}

How can we make this into an algorithm?

- Keep track of the max value $\alpha$ found so far at Max nodes;
- Keep track of the min value $\beta$ found so far at Min nodes;
- If ever $\beta < \alpha$, STOP!

\textbf{LET'S TRY IT.....}
Refinements to Min-Max Search: Pruning the search space

α-β-Pruning: Basic Principles

How can we make this into an algorithm?
- Keep track of the max value α found so far at Max nodes;
- Keep track of the min value β found so far at Min nodes;
- If ever β < α, STOP!

LET’S TRY IT…..
### α-β-Pruning: Basic Principles

How can we make this into an algorithm?

- Keep track of the max value $\alpha$ found so far at Max nodes;
- Keep track of the min value $\beta$ found so far at Min nodes;
- If ever $\beta < \alpha$, STOP!

**LET'S TRY IT…..**
**α-β-Pruning:**

**Basic Principles**

How can we make this into an algorithm?

- Keep track of the max value \( \alpha \) found so far at Max nodes;
- Keep track of the min value \( \beta \) found so far at Min nodes;
- If ever \( \beta < \alpha \), STOP!

**LET'S TRY IT....**
Refinements to Min-Max Search: Pruning the search space

**α-β-Pruning: Basic Principles**

How can we make this into an algorithm?
- Keep track of the max value \( \alpha \) found so far at Max nodes;
- Keep track of the min value \( \beta \) found so far at Min nodes;
- If ever \( \beta < \alpha \), STOP!

**LET'S TRY IT.....**
α-β-Pruning: Basic Principles

How can we make this into an algorithm?
- Keep track of the max value α found so far at Max nodes;
- Keep track of the min value β found so far at Min nodes;
- If ever β < α, STOP!

LET’S TRY IT.....
α-β-Pruning: Basic Principles

How can we make this into an algorithm?

- Keep track of the max value $\alpha$ found so far at Max nodes;
- Keep track of the min value $\beta$ found so far at Min nodes;
- If ever $\beta < \alpha$, STOP!

LET’S TRY IT.....
α-β-Pruning: Basic Principles

How can we make this into an algorithm?

- Keep track of the max value α found so far at Max nodes;
- Keep track of the min value β found so far at Min nodes;
- If ever β < α, STOP!

LET’S TRY IT…. 
α-β-Pruning: Basic Principles

How can we make this into an algorithm?
- Keep track of the max value $\alpha$ found so far at Max nodes;
- Keep track of the min value $\beta$ found so far at Min nodes;
- If ever $\beta < \alpha$, STOP!

LET'S TRY IT.....
Refinements to Min-Max Search: Pruning the search space

**α-β-Pruning:** Basic Principles

How can we make this into an algorithm?

- Keep track of the max value \( \alpha \) found so far at Max nodes;
- Keep track of the min value \( \beta \) found so far at Min nodes;
- If ever \( \beta < \alpha \), STOP!

LET'S TRY IT.....
α-β-Pruning: Basic Principles

How can we make this into an algorithm?

- Keep track of the max value $\alpha$ found so far at Max nodes;
- Keep track of the min value $\beta$ found so far at Min nodes;
- If ever $\beta < \alpha$, STOP!

LET'S TRY IT.....
Refinements to Min-Max Search: Pruning the search space

**α-β-Pruning:**

Basic Principles

How can we make this into an algorithm?

- Keep track of the max value \( \alpha \) found so far at Max nodes;
- Keep track of the min value \( \beta \) found so far at Min nodes;
- If ever \( \beta < \alpha \), STOP!

**LET’S TRY IT…..**
α-β-Pruning:
Basic Principles

How can we make this into an algorithm?

- Keep track of the max value α found so far at Max nodes;
- Keep track of the min value β found so far at Min nodes;
- If ever β < α, STOP!

LET'S TRY IT.....
**α-β-Pruning:**

**Basic Principles**

How can we make this into an algorithm?

- Keep track of the max value $\alpha$ found so far at Max nodes;
- Keep track of the min value $\beta$ found so far at Min nodes;
- If ever $\beta < \alpha$, STOP!

LET'S TRY IT.....
Improving the Min-Max Algorithm

How effective is Alpha-Beta Pruning?

Using a vanilla implementation of the my Connect4 program, I counted how many total board positions were examined, with and without AB Pruning. Here are the results, expressed as the percentage of boards examined under AB Pruning compared with no pruning:

<table>
<thead>
<tr>
<th>Depth</th>
<th>Without AB</th>
<th>With AB</th>
<th>With/Without</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>8</td>
<td>100.00%</td>
</tr>
<tr>
<td>2</td>
<td>72</td>
<td>72</td>
<td>100.00%</td>
</tr>
<tr>
<td>3</td>
<td>584</td>
<td>386</td>
<td>66.10%</td>
</tr>
<tr>
<td>4</td>
<td>4209</td>
<td>2364</td>
<td>56.16%</td>
</tr>
<tr>
<td>5</td>
<td>33380</td>
<td>12197</td>
<td>36.54%</td>
</tr>
<tr>
<td>6</td>
<td>255247</td>
<td>48912</td>
<td>19.16%</td>
</tr>
<tr>
<td>7</td>
<td>2322941</td>
<td>312565</td>
<td>13.46%</td>
</tr>
</tbody>
</table>

The improvement is about \((2/3)^{(\text{depth} - 2)}\), which is an exponential improvement; roughly, this means that using AB Pruning, you can go two layers deeper than you could otherwise.

The conclusion is obvious: Use AB Pruning!!