DRONACHARYA
College of Engineering

INTELLIGENT SYSTEMS (CSE-303-F)

Section A

Alpha Beta Search

Artificial Intelligence

Alpha Beta Search

Part I : The idea of Alpha Beta Search

Part II: The details of Alpha Beta Search

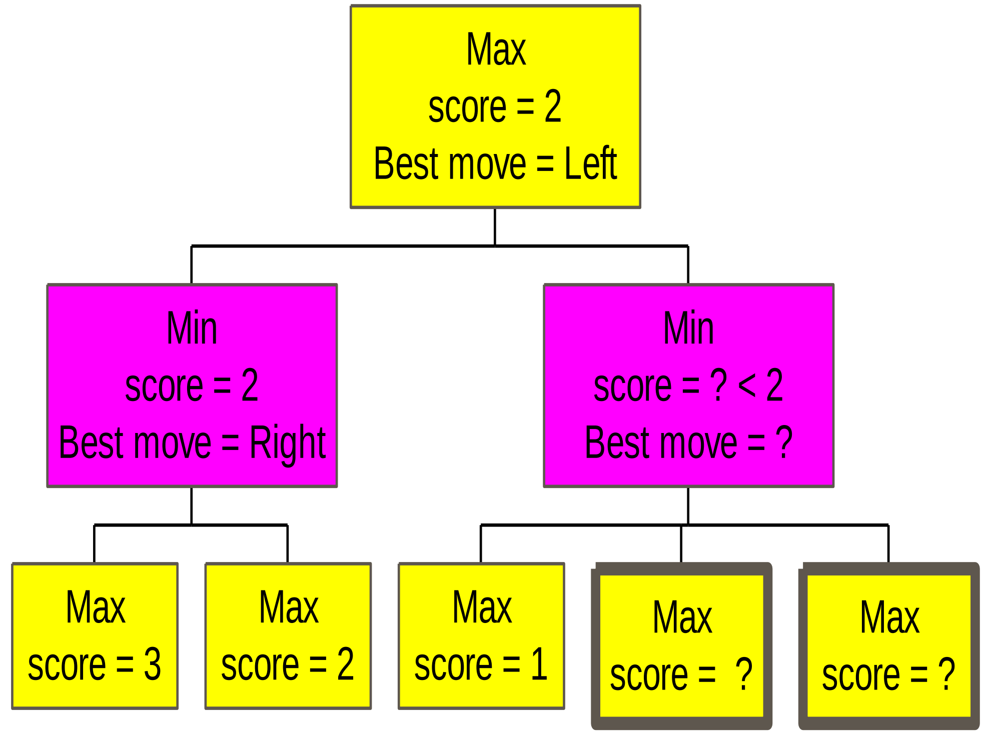
Part III: Results of using Alpha Beta

Reminder

- We consider 2 player perfect information games
- Two players, Min and Mαx
- Leaf nodes given definite score
- backing up by MiniMax defines score for all nodes
- Usually can’t search whole tree
  - Use static evaluation function instead
- MiniMax hopelessly inefficient

What's wrong with MiniMax

- Minimax is horrendously inefficient
- If we go to depth d, branching rate b,
  - we must explore b^d nodes
- but many nodes are wasted
- We needlessly calculate the exact score at every node
- but at many nodes we don't need to know exact score
- e.g. outlined nodes are irrelevant



The Solution

- Start propagating costs as soon as leaf nodes are generated
- Don’t explore nodes which cannot affect the choice of move
  - I.e. don’t explore those that we can prove are no better than the best found so far
- This is the idea behind alpha-beta search

Alpha-Beta search

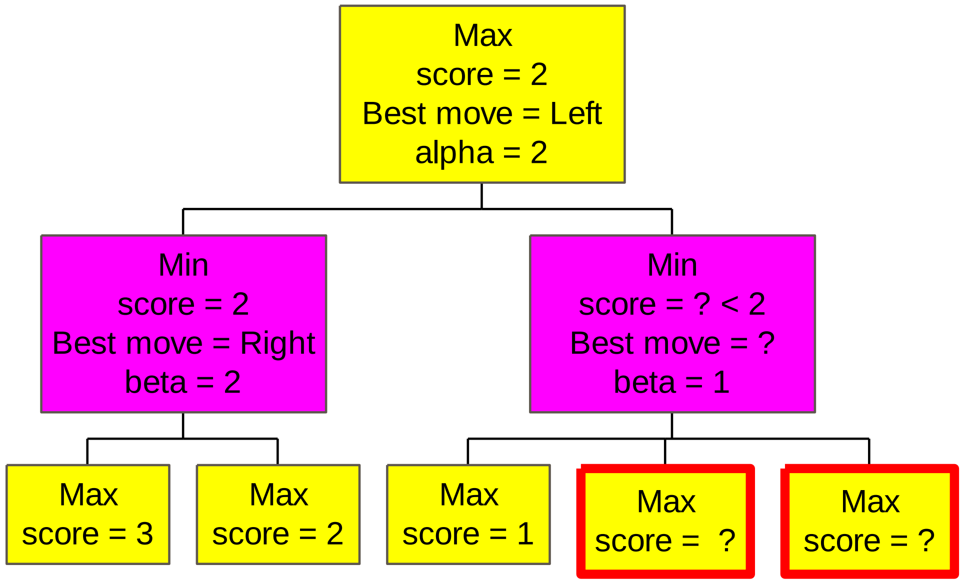
- Alpha-Beta = $\alpha-\beta$
- Uses same insight as branch and bound
- When we cannot do better than the best so far
  - we can cut off search in this part of the tree
- More complicated because of opposite score functions
- To implement this we will manipulate alpha and beta values, and store them on internal nodes in the search tree

Alpha and Beta values

- At a Mαx node we will store an alpha value
  - the alpha value is *lower bound* on the exact minimax score
  - the true value might be $\geq \alpha$
  - if we know Min can choose moves with score $< \alpha$
    - then Min will never choose to let Max go to a node where the score will be α or more
- At a Min node, β value is similar but opposite
- Alpha-Beta search uses these values to cut search

Alpha Beta in Action

- Why can we cut off search?
- Beta = 1 < alpha = 2 where the alpha value is at an ancestor node
- At the ancestor node, Max had a choice to get a score of at least 2 (maybe more)
- Max is not going to move right to let Min guarantee a score of 1 (maybe less)



Alpha and Beta values

- Max node has α value
  - the alpha value is *lower bound* on the exact minimax score
  - with best play M αx can guarantee scoring at least α
- Min node has β value
  - the beta value is *upper bound* on the exact minimax score
  - with best play Min can guarantee scoring no more than β
- At Max node, if an ancestor Min node has $\beta < \alpha$
  - Min's best play must never let Max move to this node
    - therefore this node is irrelevant
  - if $\beta = \alpha$, Min can do as well without letting Max get here
    - so again we need not continue

Alpha-Beta Pruning Rule

- Two key points:
  - alpha values can *never decrease*
  - beta values can *never increase*
- Search can be discontinued at a node if:
  - It is a Max node and
    - the alpha value is ≥ the **beta** of any Min ancestor
    - this is *beta cutoff*
  - Or it is a Min node and
    - the beta value is ≤ the **alpha** of any Max ancestor
    - this is *alpha cutoff*

Calculating Alpha-Beta values

- Alpha-Beta calculations are similar to Minimax
  - but the pruning rule cuts down search
- Use concept of ‘final backed up value’ of node
  - this might be the minimax value
  - or it might be an approximation where search cut off
    - *less* than the true minimax value at a Max node
    - *more* than the true minimax value at a Min node
    - in either case, we don’t need to know the true value

Final backed up value

- Like MiniMax
- At a Max node:
  - the final backed up value is equal to the:
    - largest final backed up value of its successors
    - this can be all successors (if no beta cutoff)
    - or all successors used until beta cutoff occurs
- At a Min node
  - the smallest final backed up value is equal to the
    - smallest final backed up value of its successors
    - min of all successors until alpha cutoff occurs

Calculating alpha values

- At a Mαx node
  - after we obtain the final backed up value of the first child
    - we can set α of the node to this value
  - when we get the final backed up value of the second child
    - we can increase α if the new value is larger
  - when we have the final child, or if beta cutoff occurs
    - the stored α becomes the final backed up value
    - only then can we set the β of the parent Min node
    - only then can we guarantee that β will not increase
- Note the difference
  - setting alpha value of current node as we go along
  - *vs.* propagating value up only when it is finalised

Calculating beta values

- At a Min node
  - after we obtain the final backed up value of the first child
    - we can set β of the node to this value
  - when we get the final backed up value of the second child
    - we can decrease β if the new value is smaller
  - when we have the final child, or if alpha cutoff occurs
    - the stored β becomes the final backed up value
    - only then can we set the α of the parent Max node
    - only then can we guarantee that α will not decrease
- Note the difference
  - setting beta value of current node as we go along
  - *vs.* propagating value up only when it is finalised

Move ordering Heuristics

- Variable ordering heuristics irrelevant
- value ordering heuristics = move ordering heuristic
- The optimal move ordering heuristic for alpha-beta ..
  - … is to consider the best move first
  - I.e. test the move which will turn out to have best final backed up value
  - of course this is impossible in practice
- The pessimal move ordering heuristic …
  - … is to consider the worst move first
  - I.e. test move which will have worst final backed up value

Move ordering Heuristics

- In practice we need quick and dirty heuristics
- will neither be optimal nor pessimal
- E.g. order moves by static evaluation function
  - if it's reasonable, most promising likely to give good score
  - should be nearer optimal than random
- If static evaluation function is expensive
  - need even quicker heuristics
- In practice move ordering heuristics vital

Theoretical Results

- With pessimal move ordering,
  - alpha beta makes no reduction in search cost
- With optimal move ordering
  - alpha beta cuts the amount of search to the square root
  - I.e. From b^d to $\sqrt{b^d} = b^{d/2}$
  - Equivalently, we can search to twice the depth
    - at the same cost
- With heuristics, performance is in between
- alpha beta search vital to successful computer play in 2 player perfect information games

Summary and Next Lecture

- Game trees are similar to search trees
  - but have opposing players
- Minimax characterises the value of nodes in the tree
  - but is horribly inefficient
- Use static evaluation when tree too big
- Alpha-beta can cut off nodes that need not be searched
  - can allow search up to twice as deep as minimax
- Next Time:
  - Chinook, world champion Checkers player