Alpha-Beta Pruning

Removing Irrelevant Decisions in Two Person Games

Background

• What is Alpha-Beta Pruning?
  – It’s an algorithm that is more efficient that a minimax search but returns the same result
• Why use Alpha-Beta Pruning?
  – Increase efficiency by avoiding decisions that have no impact on the outcome
• Where to use Alpha-Beta Pruning?
  – When computing a minimax search on a search tree, and:
    • opponent always makes perfect decision
    • all nodes have same branching factor
    • all paths reach a fixed depth
The A/B Pruning Principle

• If there is some node N in the tree such that the player has a choice of moving to that node. If the player has a better choice M either at the parent node of N or at any choice point further up, then N will never be reached in actual play

• In other words, a node’s value isn’t important if an opponent will never go to it

A/B Pruning Principle (cont’d)
What are Alpha and Beta

- Alpha is the best choice so far for Max at a node
  - The highest value of all directly subordinate nodes
- Beta is the best choice so far for Min at a node
  - The lowest value of all directly subordinate nodes

Chess Example

- Assuming we can search about 1000 positions per second, and are given 150 seconds per move, we can look at 150,000 positions
- Using a minimax search we can only look ahead about 3 or 4 moves, since chess has a branching factor of about 35
- An average human player can look ahead 6 to 8 moves
Alpha-Beta Pruning Example

MAX

MIN

MAX

<= 3
Max function

function Max-Value(state, game, A, B) returns the minimax value of state
inputs:
    state, current state in the game
    game, game description,
    A the best score for MAX along the path to state
    B, the best score for MIN along the path to state

if CUTOFF-TEST(state) then return EVAL(state)
for each s in SUCCESSORS(state) do
    A ← MAX(A, MIN-VALUE(s, game, A, B)) if A ≥ B then return B
end
return A
Min Function

function Min-Value(state, game, A, B) returns the minimax value of state

if CUTOFF-TEST(state) then return EVAL(state)
for each s in SUCCESSORS(state) do
    β ← MAX(β, MAX-VALUE(s, game, A, B)) if B ≤ A then return A
end
return B

Chess with A/B Pruning

• Instead of having a O(b^d) like with a standard minimax search, we achieve O(b^(d/2)) where b is the branching factor and d is the search depth
  – For example, if d was 4 and b was 35
    • With minimax we would get 35^4=1,500,625
    • With A/B pruning we get 35^2=1225
    • This mean we have replaced b with the square root of b since 35^2=35^(1/2)^4
  – This means we can look ahead twice as far in the same amount of time
Summary

• We can increase the look number of moves that we look ahead by eliminating moves that a sensible player would never make
• There are many restrictions on the application of this algorithm that make it far from ideal

References

Artificial Intelligence: A Modern Approach
Stuart Russell and Peter Norvig

sem.ucalgary.ca/courses/CPSC/533/W99/presentations/L2_5B_Lima_Neitz/abpruning.ppt

http://www.cs.uwyo.edu/~dspears/courses/4550/lecture6.1.ppt