Adversarial Search

Sven Koenig, USC

Russell and Norvig, 3rd Edition, Sections 5.1-5.3

These slides are new and can contain mistakes and typos.
Please report them to Sven (skoenig@usc.edu).

Game Playing

- IBM's Deep Blue beat Garry Kasparov 3.5 – 2.5 in May 1997

1997

Der Spiegel

Wikipedia

Game Playing

2007

SCIENTIFIC AMERICAN™

Computers Solve Checkers—It's a Draw

King me! Top computer scientist proves perfect play leads to draw, recounts battle for world championship, gets kinged

2014

Science AAAS

[Heads-Up Limit] Texas Hold 'em poker solved by computer

Game Playing

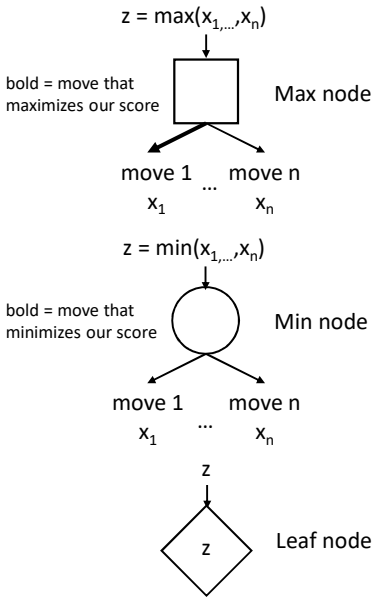
- Classifying games
  - Chess
  - Checkers
  - Poker
  - Bridge
  - Backgammon
  - Scrabble
  - Go
  - …

Game Playing

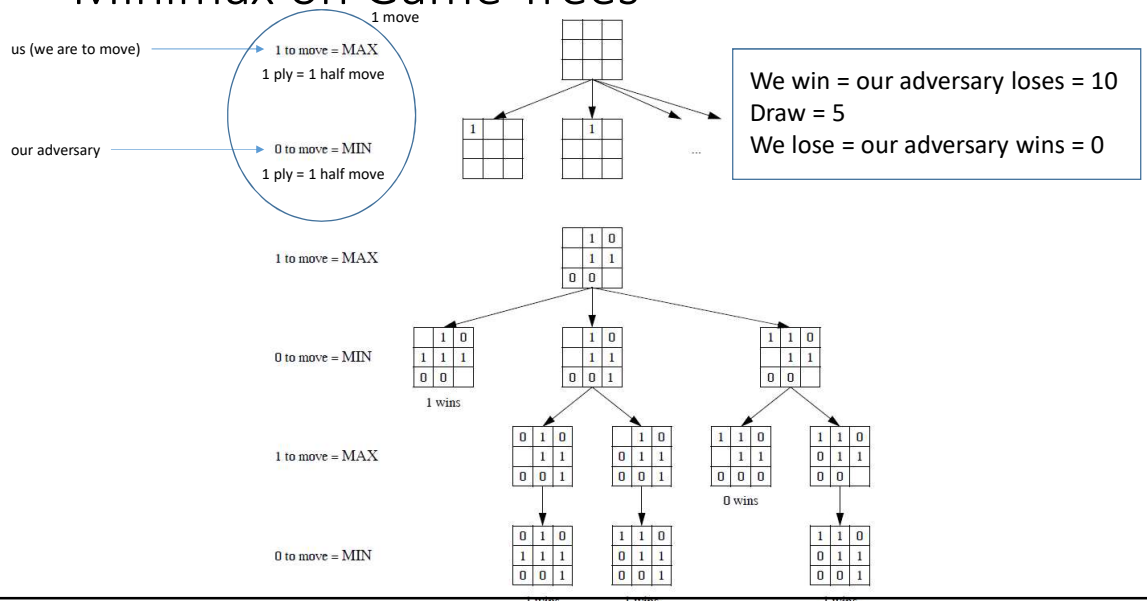
- Classifying games
  - How many players are there? Here: 2
  - Are the players competing or cooperating? Here: competing.
  - Is the state completely known? Here: yes
  - Is there a probabilistic element? Here: no
- We study deterministic, perfect information, 2-player, zero-sum games, like chess or tic-tac-toe.

Game Trees

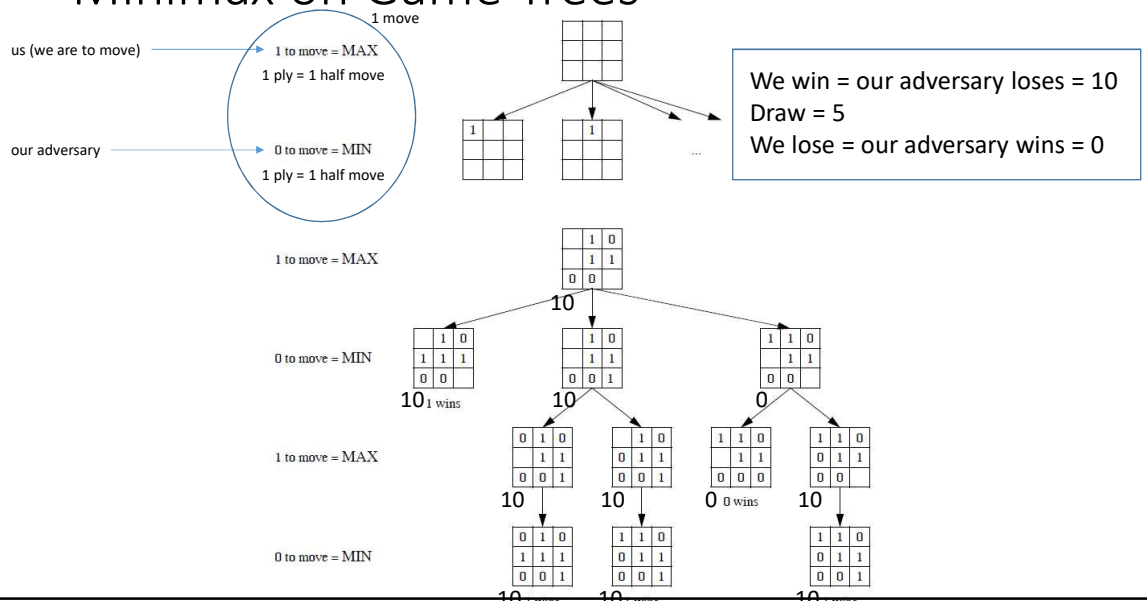
- We are playing a game against an adversary.
- Max nodes:
  We pick the move that maximizes our score.
- Min nodes:
  Our adversary picks the move that minimizes our score (i.e. maximizes their score).
- Leaf nodes (terminal game positions):
  We receive the given score.



Minimax on Game Trees



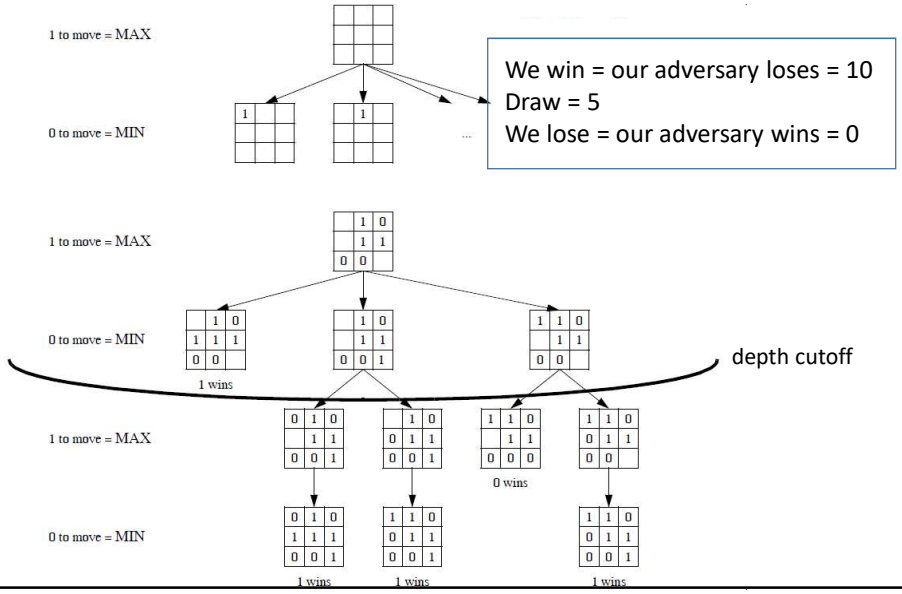
Minimax on Game Trees



Minimax on Game Trees

- Game trees can be huge and then take too long to search.
- Tic-Tac-Toe has at most 3^9 different legal positions.
- But chess, for example, has about
  - 10^{40} different legal positions and
  - 35^{100} nodes in an average game tree.

Minimax on Game Trees



Minimax on Game Trees

- Evaluation function
  - Returns actual value for a terminal node
    (e.g. value of “we win” for a terminal node where we win)
  - Returns a value between “we win” and “we lose” for a non-terminal node,
    - which is roughly proportional to the likelihood of us winning,
    - which can be calculated quickly, and
    - which is often a weighted average of values of hand-selected features with learned weights.
- Features for Tic-Tac-Toe
  - control of the center
  - number of our “open files” minus number of adversary’s “open files”
  - …

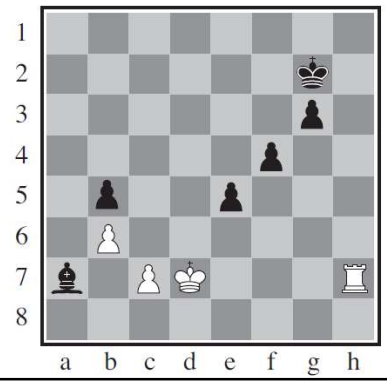
Minimax on Game Trees

- Evaluation functions are often too inexact for the initial positions and endgame positions.
- In this case, one uses move libraries that simply store the best moves for these positions.

Minimax on Game Trees

- One wants to search beyond the depth cutoff until quiescence (i.e. until the evaluations of a node and its ancestor(s) are similar) to avoid the horizon effect

black to move



Minimax on Game Trees

Implement this as a depth-first search, including its memory-saving techniques

- call MAX-VALUE(node = current game position);
- MAX-VALUE(node)
  if node is a terminal node (or to be treated like one) then
    return the value of the evaluation function for that node;
  else
    alpha := value of "we lose";
    for each successor n of node do
      alpha := MAX(alpha, MIN-VALUE(n));
    return alpha;
- MIN-VALUE(node)
  if node is a terminal node (or to be treated like one) then
    return the value of the evaluation function for that node;
  else
    beta := value of "we win";
    for each successor n of node do
      beta := MIN(beta, MAX-VALUE(n));
    return beta;

Alpha-Beta on Game Trees

- There are nodes in game trees whose evaluations do not matter for determining the value of the game, i.e. the value of the root node of the game tree.
- One does not need to determine the values of such nodes but can "prune" them by backtracking from them immediately.
- This can save a lot of effort.
- In fact, Alpha-Beta determines the same action as Minimax and the same value of the game but can often search a game tree twice as deep as Minimax in the same amount of time.

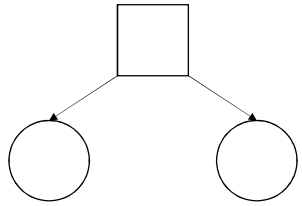
Alpha-Beta on Game Trees

MAX

Alpha-Beta on Game Trees

MAX

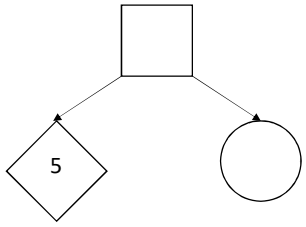
MIN



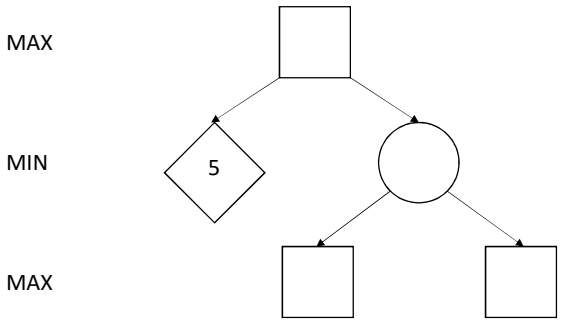
Alpha-Beta on Game Trees

MAX

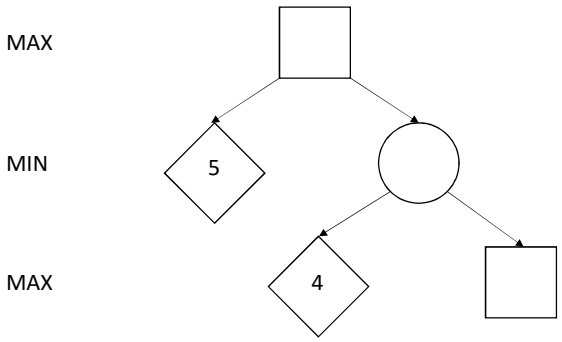
MIN



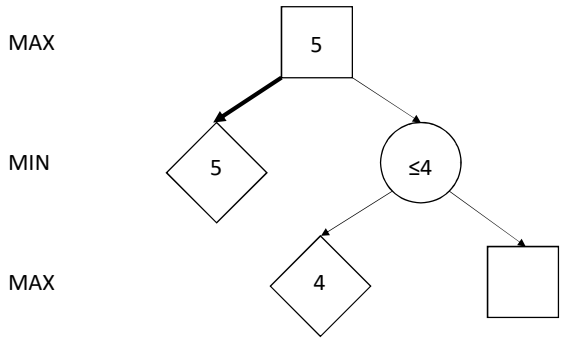
Alpha-Beta on Game Trees



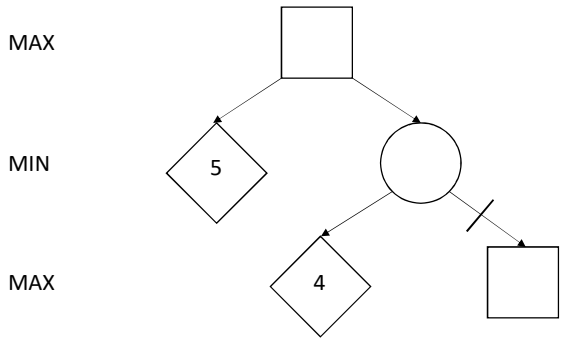
Alpha-Beta on Game Trees



Alpha-Beta on Game Trees

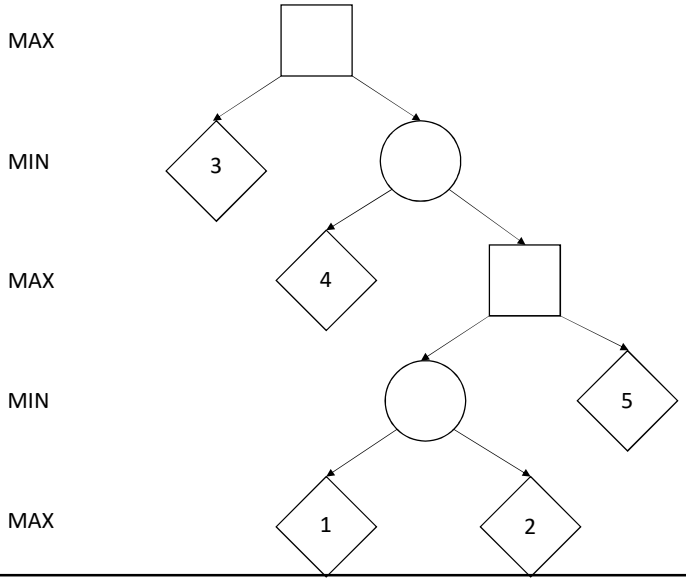


Alpha-Beta on Game Trees



There might be a large subtree here that does not need to be searched.

Alpha-Beta on Game Trees

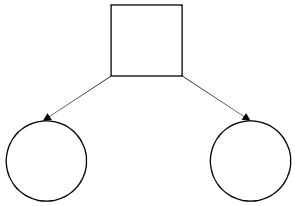


Alpha-Beta on Game Trees

Alpha-Beta on Game Trees

MAX

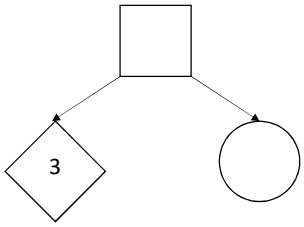
MIN



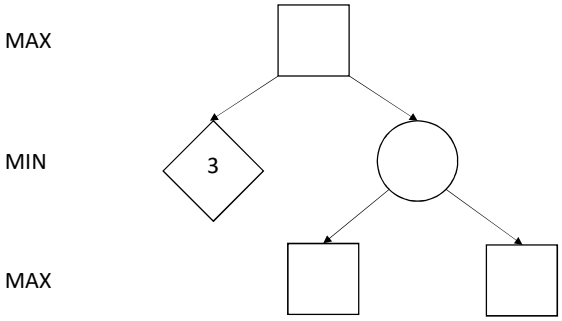
Alpha-Beta on Game Trees

MAX

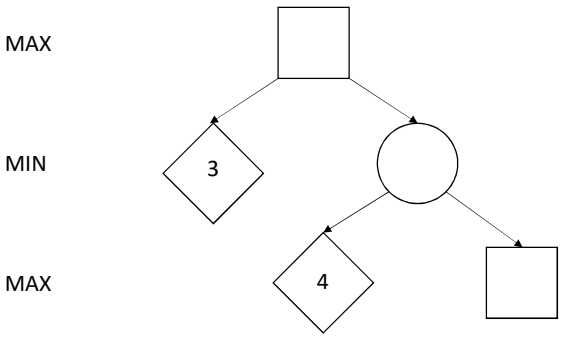
MIN



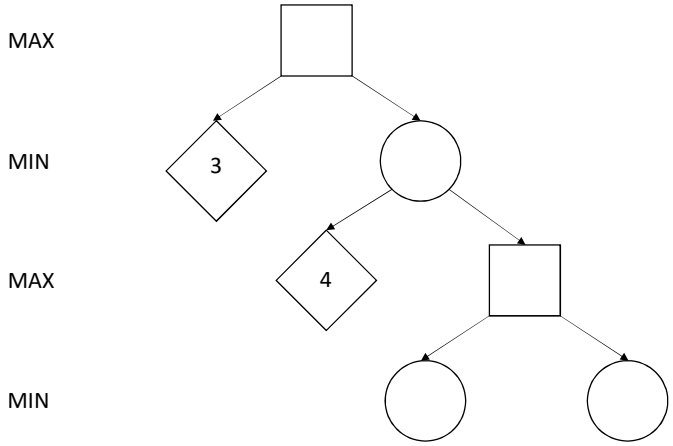
Alpha-Beta on Game Trees



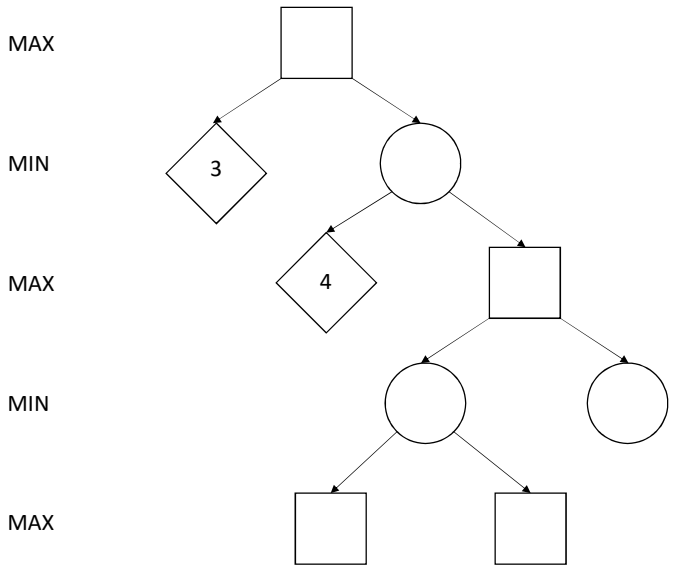
Alpha-Beta on Game Trees



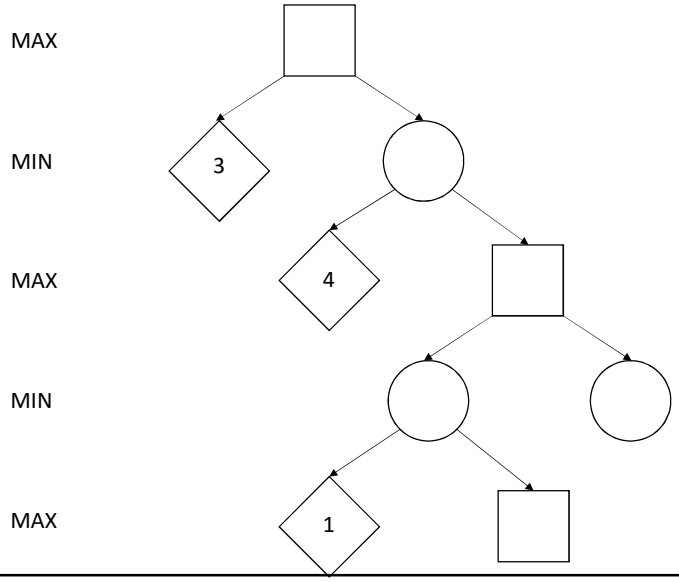
Alpha-Beta on Game Trees



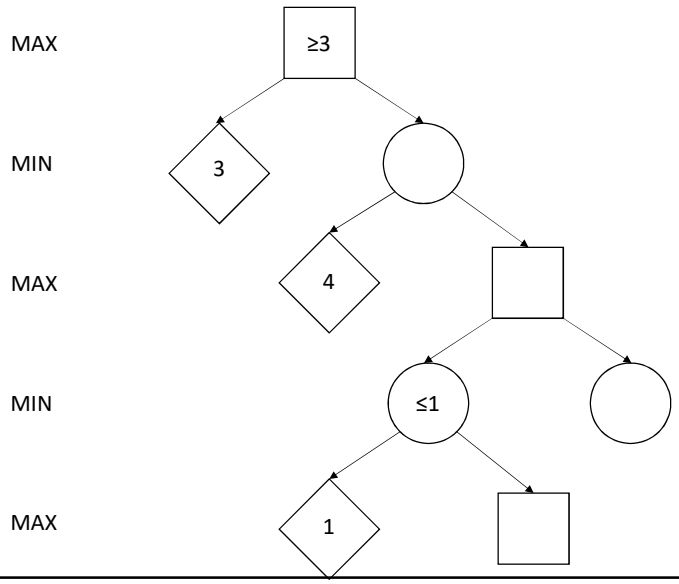
Alpha-Beta on Game Trees



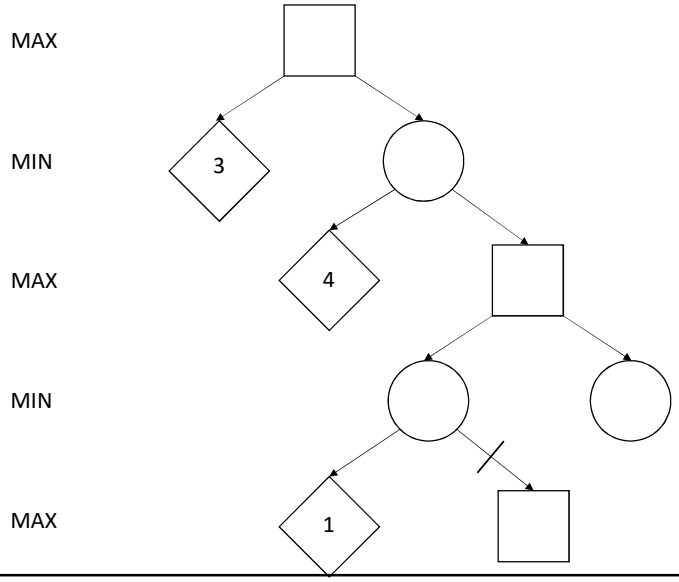
Alpha-Beta on Game Trees



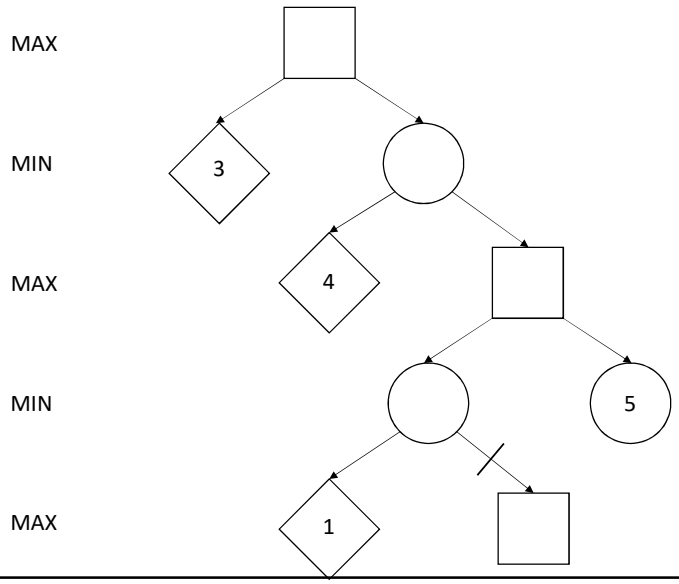
Alpha-Beta on Game Trees



Alpha-Beta on Game Trees



Alpha-Beta on Game Trees



Alpha-Beta on Game Trees

Implement this as a depth-first search, including its memory-saving techniques

- call MAX-VALUE(node = current game position, alpha=value of "we lose", beta="value of "we win");
- MAX-VALUE(node, alpha, beta)
  if node is a terminal node (or to be treated like one) then
    return the value of the evaluation function for that node;
  else
    for each successor n of node do
      alpha := MAX(alpha, MIN-VALUE(n, alpha, beta));
      if alpha ≥ beta then return alpha;
    return alpha;
- MIN-VALUE(node, alpha, beta)
  if node is a terminal node (or to be treated like one) then
    return the value of the evaluation function for that node;
  else
    for each successor n of node do
      beta := MIN(beta, MAX-VALUE(n, alpha, beta));
      if alpha ≥ beta then return beta;
    return beta;

alpha = largest minimax value MAX is guaranteed to achieve if node "node" is reached;
beta = smallest minimax value MIN is guaranteed to achieve if node "node" is reached;

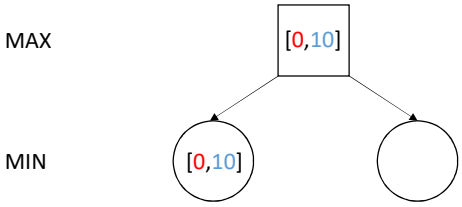
Alpha-Beta on Game Trees

Initialize alpha-beta interval.

MAX ["we lose","we win"] = [0,10]

alpha = largest minimax value MAX is guaranteed to achieve if the node is reached;
beta = smallest minimax value MIN is guaranteed to achieve if the node is reached;

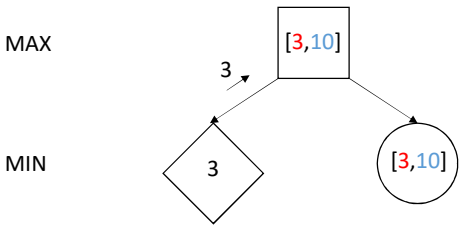
Alpha-Beta on Game Trees



Propagate alpha-beta interval down.

alpha = largest minimax value MAX is guaranteed to achieve if the node is reached;
beta = smallest minimax value MIN is guaranteed to achieve if the node is reached;

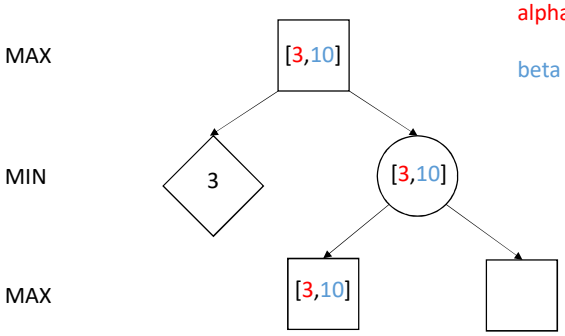
Alpha-Beta on Game Trees



Evaluate node, propagate node value up, increase alpha value of MAX node if possible, propagate alpha-beta interval down.

alpha = largest minimax value MAX is guaranteed to achieve if the node is reached;
beta = smallest minimax value MIN is guaranteed to achieve if the node is reached;

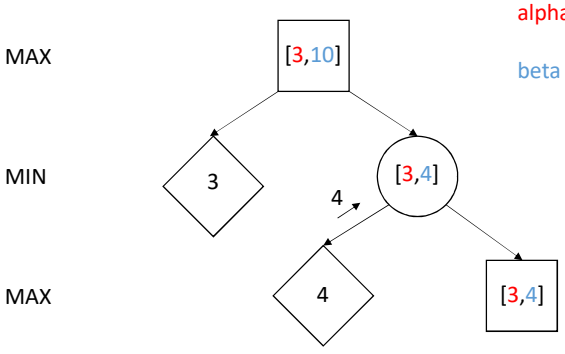
Alpha-Beta on Game Trees



Propagate alpha-beta interval down.

alpha = largest minimax value MAX is guaranteed to achieve if the node is reached;
beta = smallest minimax value MIN is guaranteed to achieve if the node is reached;

Alpha-Beta on Game Trees



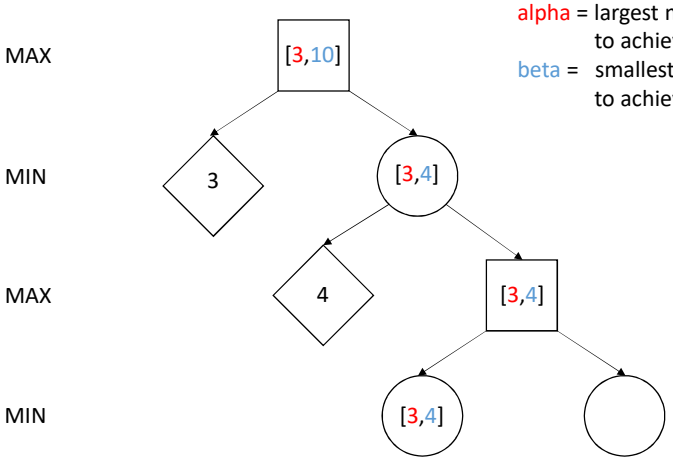
Evaluate node, propagate node value up, decrease beta value of MIN node if possible, propagate alpha-beta interval down.

alpha = largest minimax value MAX is guaranteed to achieve if the node is reached;
beta = smallest minimax value MIN is guaranteed to achieve if the node is reached;

Alpha-Beta on Game Trees

Propagate alpha-beta interval down.

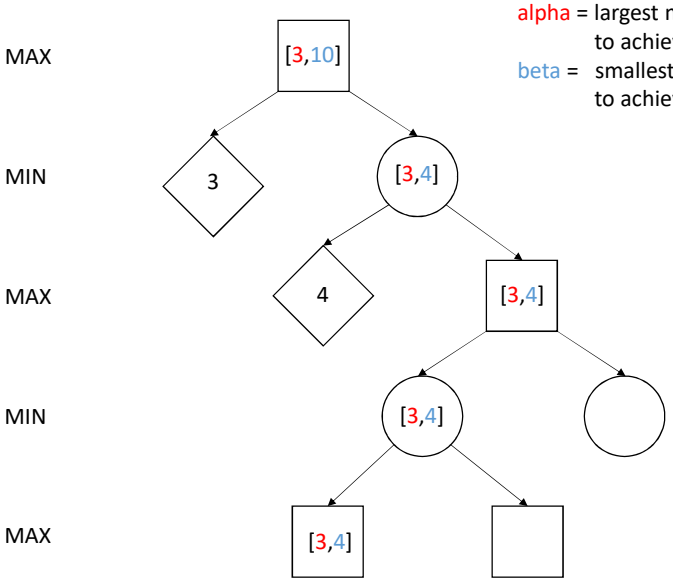
alpha = largest minimax value MAX is guaranteed to achieve if the node is reached;
beta = smallest minimax value MIN is guaranteed to achieve if the node is reached;



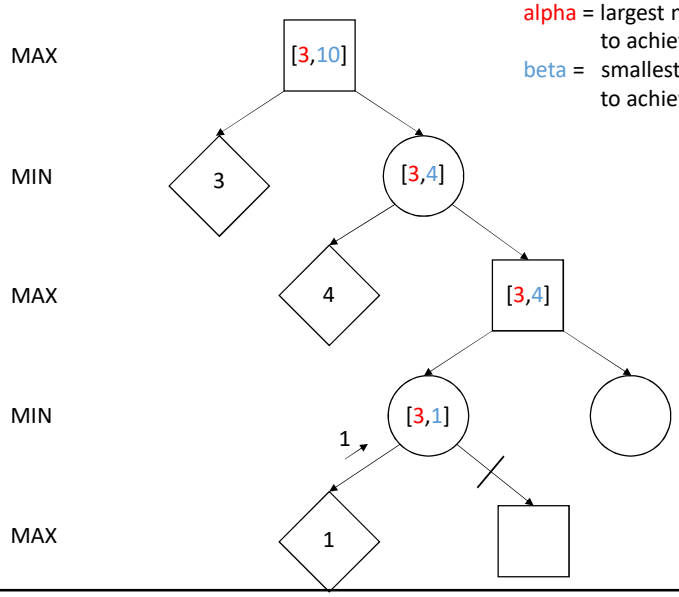
Alpha-Beta on Game Trees

Propagate alpha-beta interval down.

alpha = largest minimax value MAX is guaranteed to achieve if the node is reached;
beta = smallest minimax value MIN is guaranteed to achieve if the node is reached;



Alpha-Beta on Game Trees

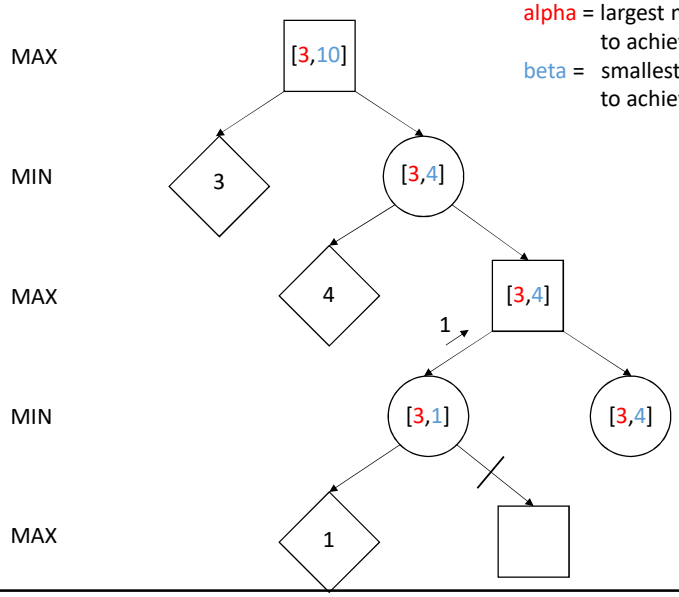


Evaluate node, propagate node value up, decrease beta value of MIN node if possible, backtrack since the interval is empty or a point.

alpha = largest minimax value MAX is guaranteed to achieve if the node is reached;

beta = smallest minimax value MIN is guaranteed to achieve if the node is reached;

Alpha-Beta on Game Trees



Propagate beta value of MIN node up, increase alpha value of MAX node if possible, propagate alpha-beta interval down.

alpha = largest minimax value MAX is guaranteed to achieve if the node is reached;

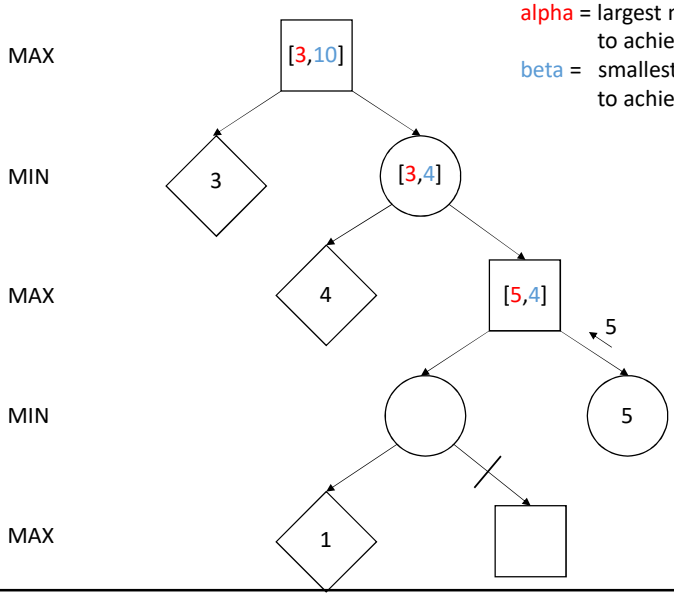
beta = smallest minimax value MIN is guaranteed to achieve if the node is reached;

Alpha-Beta on Game Trees

Evaluate node, propagate node value up, increase alpha value of MAX node if possible, backtrack since the interval is empty or a point.

alpha = largest minimax value MAX is guaranteed to achieve if the node is reached;

beta = smallest minimax value MIN is guaranteed to achieve if the node is reached;

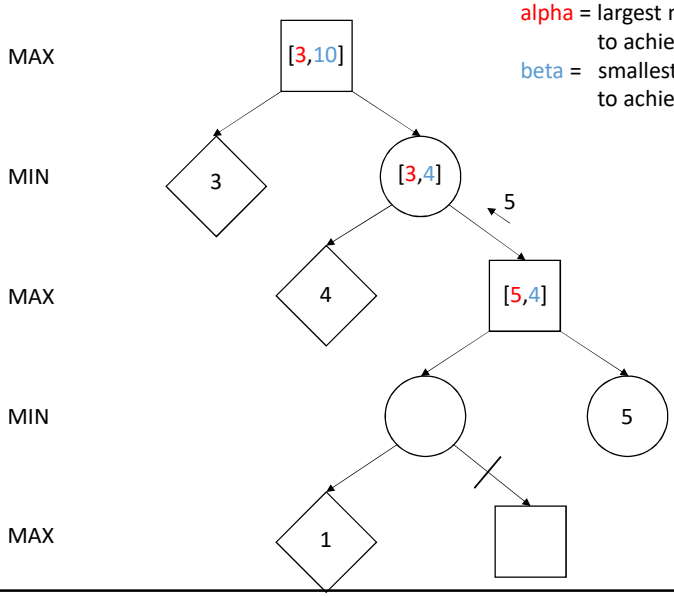


Alpha-Beta on Game Trees

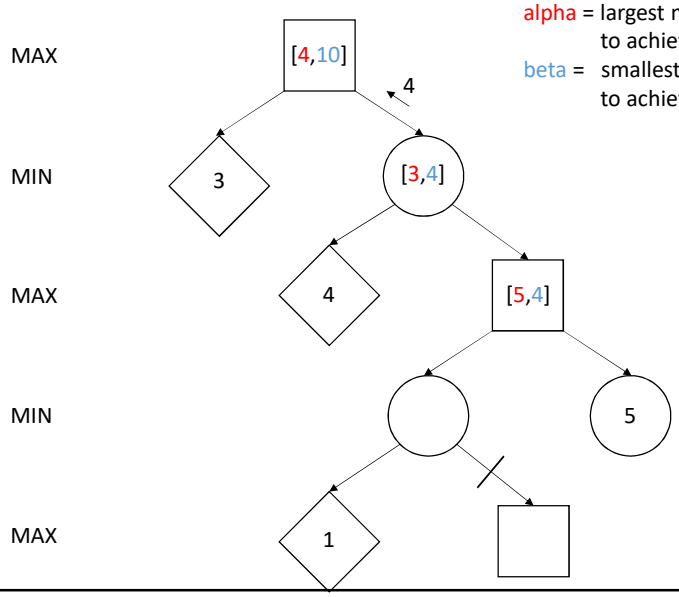
Propagate alpha value of MAX node up, decrease beta value of MIN node if possible.

alpha = largest minimax value MAX is guaranteed to achieve if the node is reached;

beta = smallest minimax value MIN is guaranteed to achieve if the node is reached;



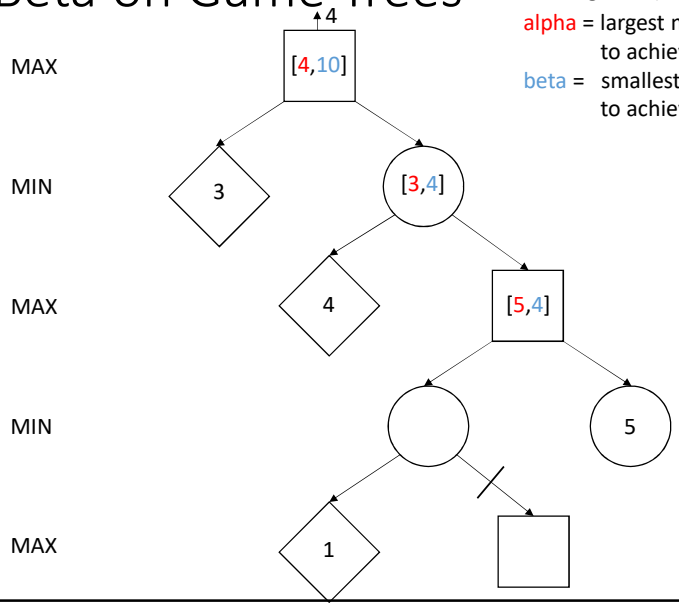
Alpha-Beta on Game Trees



Propagate beta value of MIN node up, increase alpha value of MAX node if possible.

alpha = largest minimax value MAX is guaranteed to achieve if the node is reached;
beta = smallest minimax value MIN is guaranteed to achieve if the node is reached;

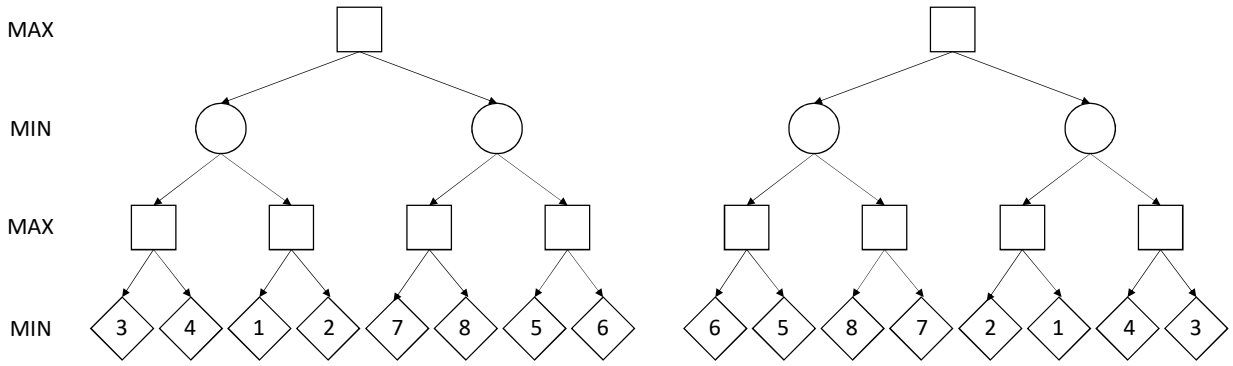
Alpha-Beta on Game Trees



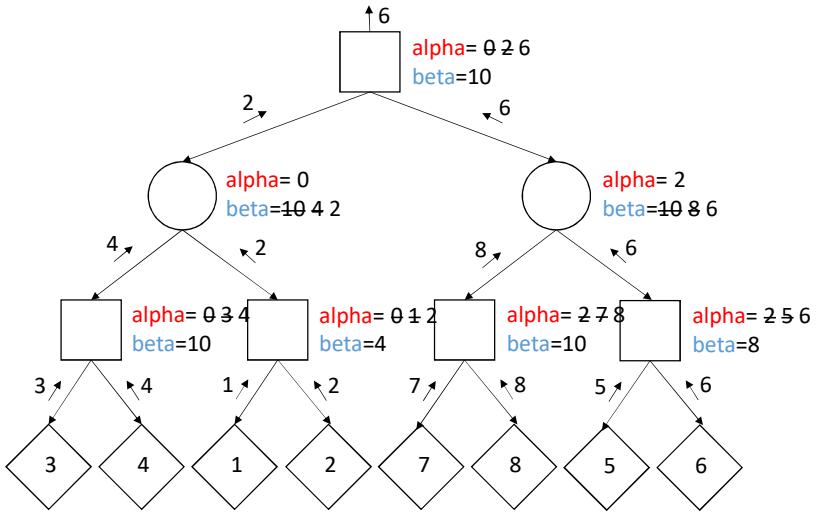
The alpha value of the MAX node (which would need to be propagated up) is the minimax value of the game. (Use this as a sanity check.)

alpha = largest minimax value MAX is guaranteed to achieve if the node is reached;
beta = smallest minimax value MIN is guaranteed to achieve if the node is reached;

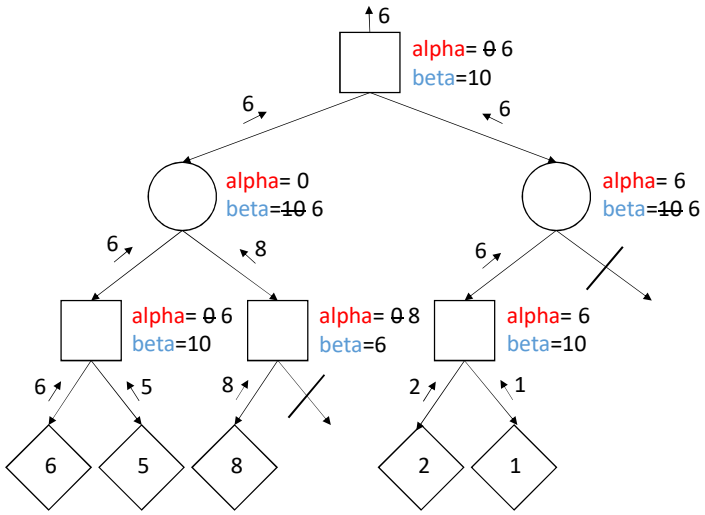
Alpha-Beta on Game Trees



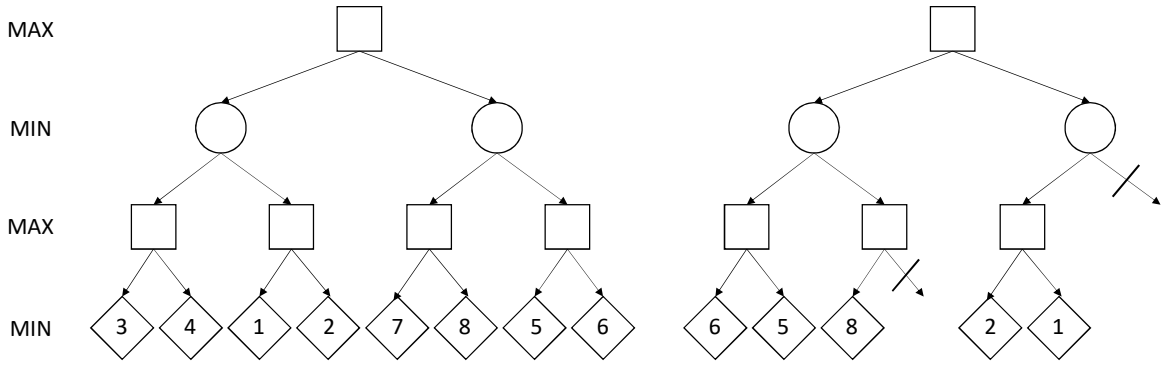
Alpha-Beta on Game Trees



Alpha-Beta on Game Trees



Alpha-Beta on Game Trees



These two game trees are identical. Just the way the moves at the MAX and MIN nodes are ordered are different!

Alpha-Beta on Game Trees

- For alpha-beta to prune lots of nodes, one needs to try the (likely) strong moves ("killer moves") first.
- At MAX nodes, try the best moves for MAX first (i.e. those that lead to positions with large values).
- At MIN nodes, try the best moves for MIN first (i.e. those that lead to positions with small values).
- In chess, for example, try moves at MAX and MIN nodes first that result in the capture of pieces since these are typically strong moves for the player who is to move at the node