Announcements (1)

- **Cancelled:**
  - Homework #2 problem 4.d, and Mid-term problems 9.d & 9.e & 9.h.
  - Everybody gets them right, regardless of your actual answers.
- **Homework #2 problem 4.d and Mid-term problem 9.d:**
  - Uniform-cost search (sort queue by g(n)) is both complete and optimal when the path cost never decreases and at most a finite number of paths have a cost below the optimal path cost.
  - Step costs $\geq \varepsilon > 0$ imply this condition.
  - A* also requires this condition for completeness.
- **Mid-term problem 9.e & 9.h:**
  - Greedy best-first search is both complete and optimal when the heuristic is optimal.
    - There is no such thing as an "optimal" heuristic.
  - If the search space contains only a single local maximum (i.e., the global maximum = the only local maximum), then hill-climbing is guaranteed to climb that single hill and will find the global maximum.
    - Your book shows several problems that confound hill-climbing.
  - However, I can see where the phrasing could be confusing.

Announcements (2)

- **The Mid-term exam is now a pedagogical device.**

- **You can recover 50% of your missed points by showing that you have debugged and repaired your knowledge base.**

- **For each item where points were deducted:**
  - Write 2-4 sentences, and perhaps an equation or two.
  - Describe:
    - What was the bug in the knowledge base leading to the error?
    - How has the knowledge base been repaired so that the error will not happen again?
  - Turn in, with your exam, on Tuesday, May 18 (in place of HW #5).
  - 50% of your missed points will be forgiven for each correct repair.

- **Homework #5 is cancelled to give you time to do this.**

Game-Playing & Adversarial Search

Reading: R&N, "Adversarial Search"
Ch. 5 (3rd ed.); Ch. 6 (2nd ed.)

For Thursday: R&N, "Constraint Satisfaction Problems"
Ch. 6 (3rd ed.); Ch 5 (2nd ed.)

Overview

- **Minimax Search with Perfect Decisions**
  - Impractical in most cases, but theoretical basis for analysis
- **Minimax Search with Cut-off**
  - Replace terminal leaf utility by heuristic evaluation function
- **Alpha-Beta Pruning**
  - The fact of the adversary leads to an advantage in search!
- **Practical Considerations**
  - Redundant path elimination, look-up tables, etc.
- **Game Search with Chance**
  - Expectiminimax search

Types of Games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	**battleship Kriegspiel**	bridge, poker, scrabble nuclear war

Not Considered: Physical games like tennis, croquet, ice hockey, etc.
(but see "robot soccer" http://www.robocup.org/)

Typical assumptions

- **Two agents whose actions alternate**
- **Utility values for each agent are the opposite of the other**
  - This creates the adversarial situation
- **Fully observable environments**
- **In game theory terms:**
  - “Deterministic, turn-taking, zero-sum games of perfect information”
- **Generalizes to stochastic games, multiple players, non zero-sum, etc.**

Grundy's game - special case of nim

Given a set of coins, a player takes a set and divides it into two unequal sets.
The player who cannot make a play, looses.

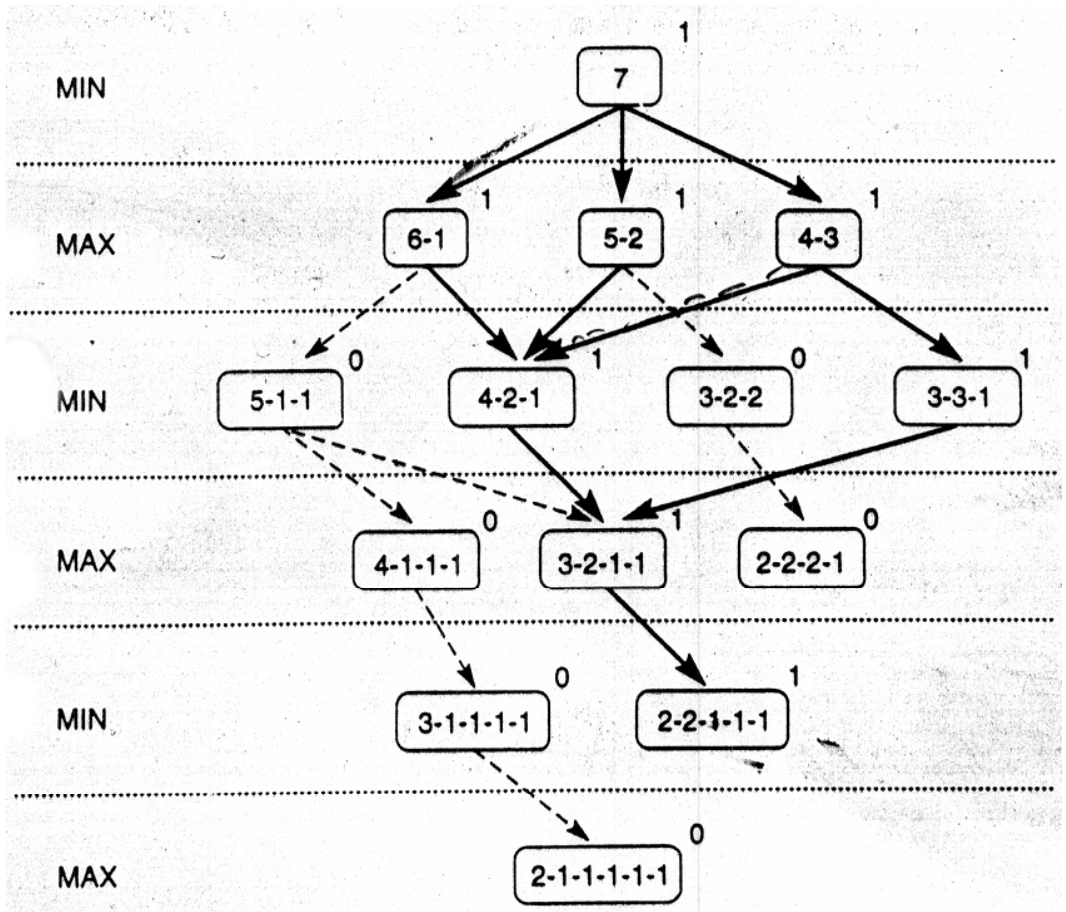
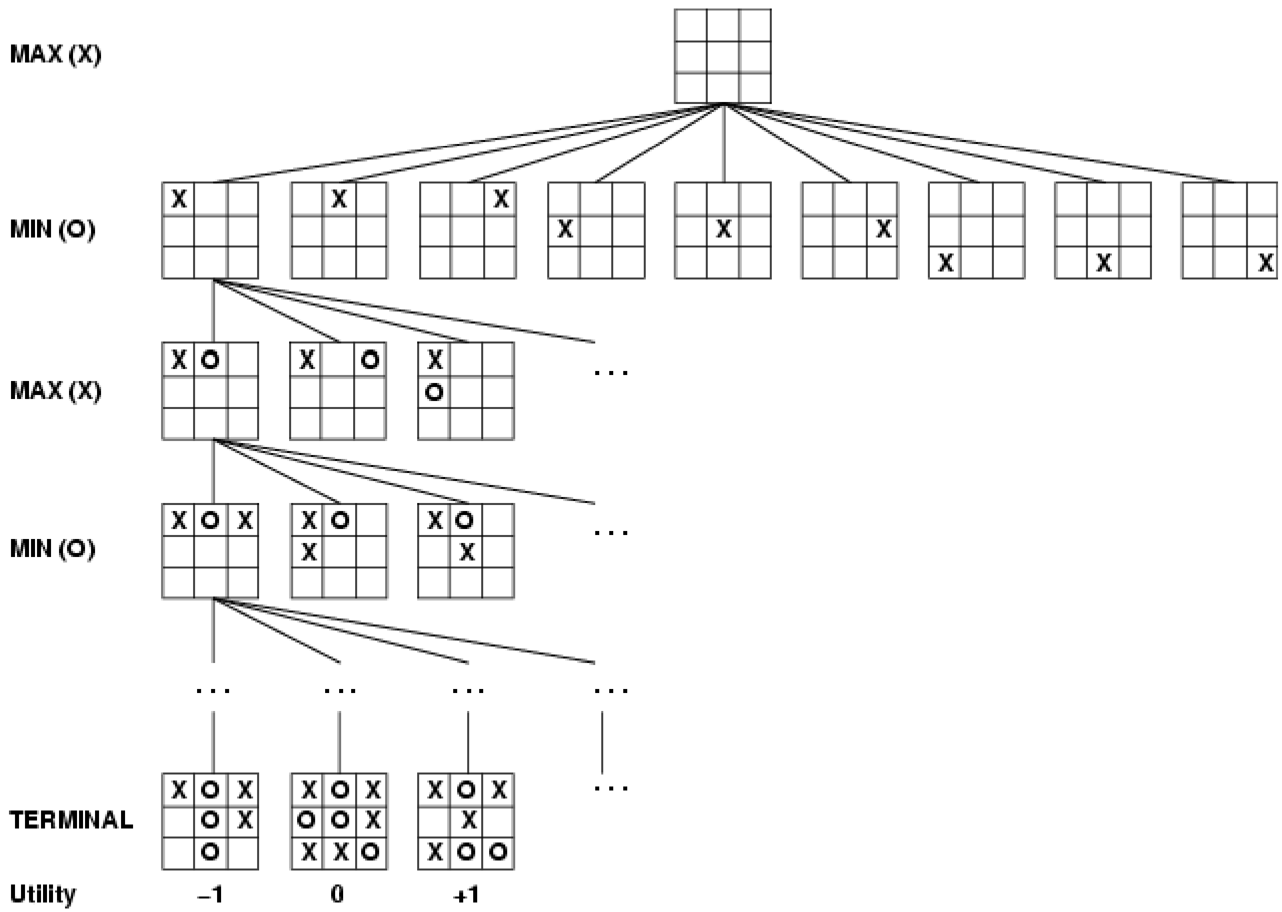


Figure 4.14 Exhaustive minimax for the game of nim. Bold lines indicate forced win for MAX. Each node is marked with its derived value (0 or 1) under minimax.

How do we search this tree to find the optimal move?

Game tree (2-player, deterministic, turns)



How do we search this tree to find the optimal move?

Search versus Games

- **Search – no adversary**
  - Solution is (heuristic) method for finding goal
  - Heuristics and CSP techniques can find *optimal* solution
  - Evaluation function: estimate of cost from start to goal through given node
  - Examples: path planning, scheduling activities

- **Games – adversary**
  - Solution is strategy
    - strategy specifies move for every possible opponent reply.
  - Time limits force an *approximate* solution
  - Evaluation function: evaluate “goodness” of game position
  - Examples: chess, checkers, Othello, backgammon

Games as Search

- **Two players: MAX and MIN**

- **MAX moves first and they take turns until the game is over**
  - Winner gets reward, loser gets penalty.
  - "Zero sum" means the sum of the reward and the penalty is a constant.

- **Formal definition as a search problem:**
  - **Initial state:** Set-up specified by the rules, e.g., initial board configuration of chess.
  - **Player(s):** Defines which player has the move in a state.
  - **Actions(s):** Returns the set of legal moves in a state.
  - **Result(s,a):** Transition model defines the result of a move.
  - (**2nd ed.: Successor function:** list of (move,state) pairs specifying legal moves.)
  - **Terminal-Test(s):** Is the game finished? True if finished, false otherwise.
  - **Utility function(s,p):** Gives numerical value of terminal state s for player p.
    - E.g., win (+1), lose (-1), and draw (0) in tic-tac-toe.
    - E.g., win (+1), lose (0), and draw (1/2) in chess.

- **MAX uses search tree to determine next move.**

An optimal procedure: The Min-Max method

Designed to find the optimal strategy for Max and find best move:

- **1. Generate the whole game tree, down to the leaves.**
- **2. Apply utility (payoff) function to each leaf.**
- **3. Back-up values from leaves through branch nodes:**
  - a Max node computes the Max of its child values
  - a Min node computes the Min of its child values
- **4. At root: choose the move leading to the child of highest value.**

Game Trees

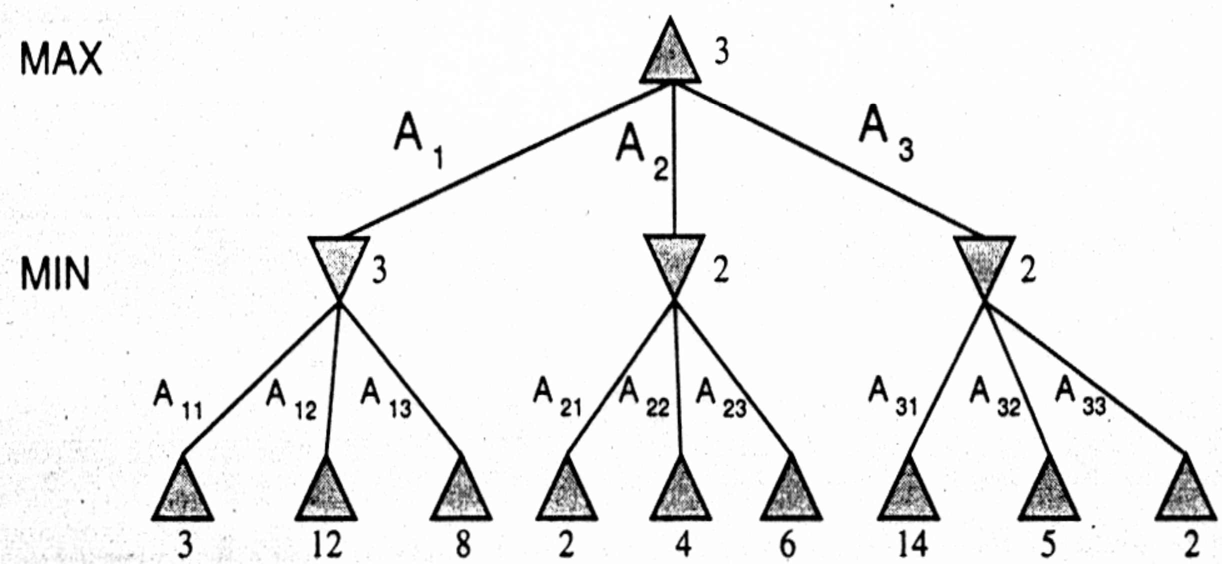
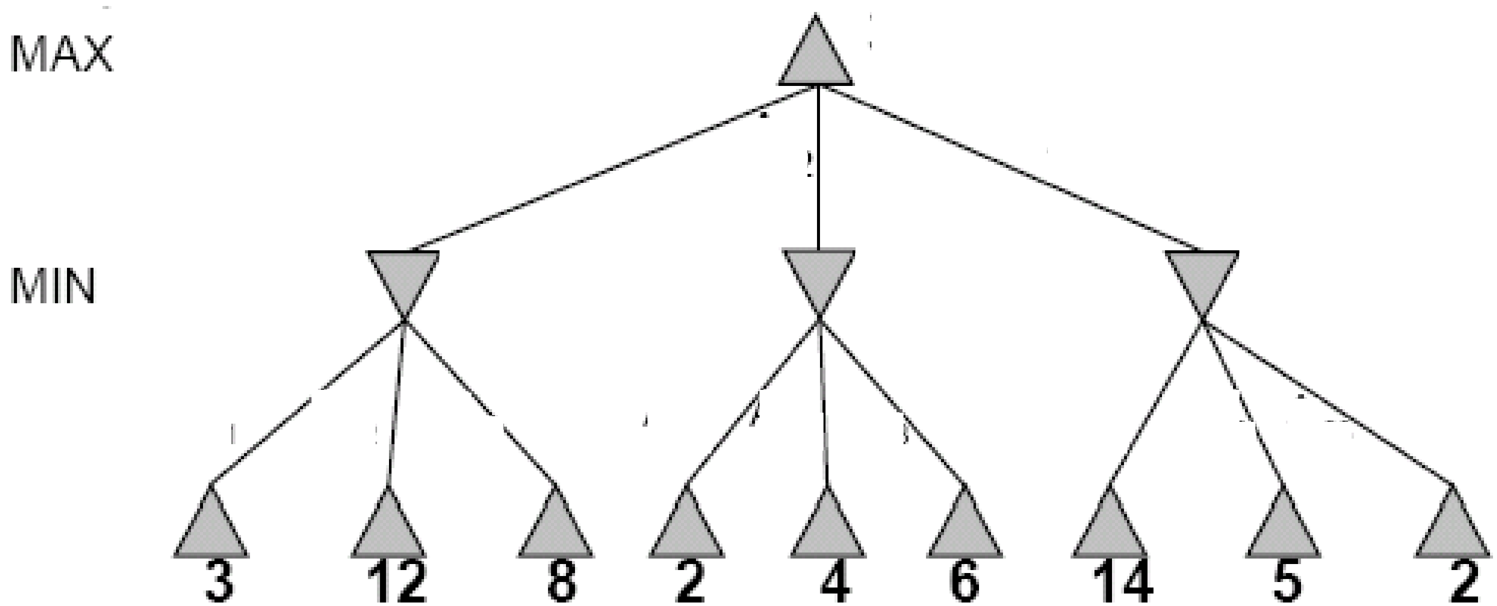
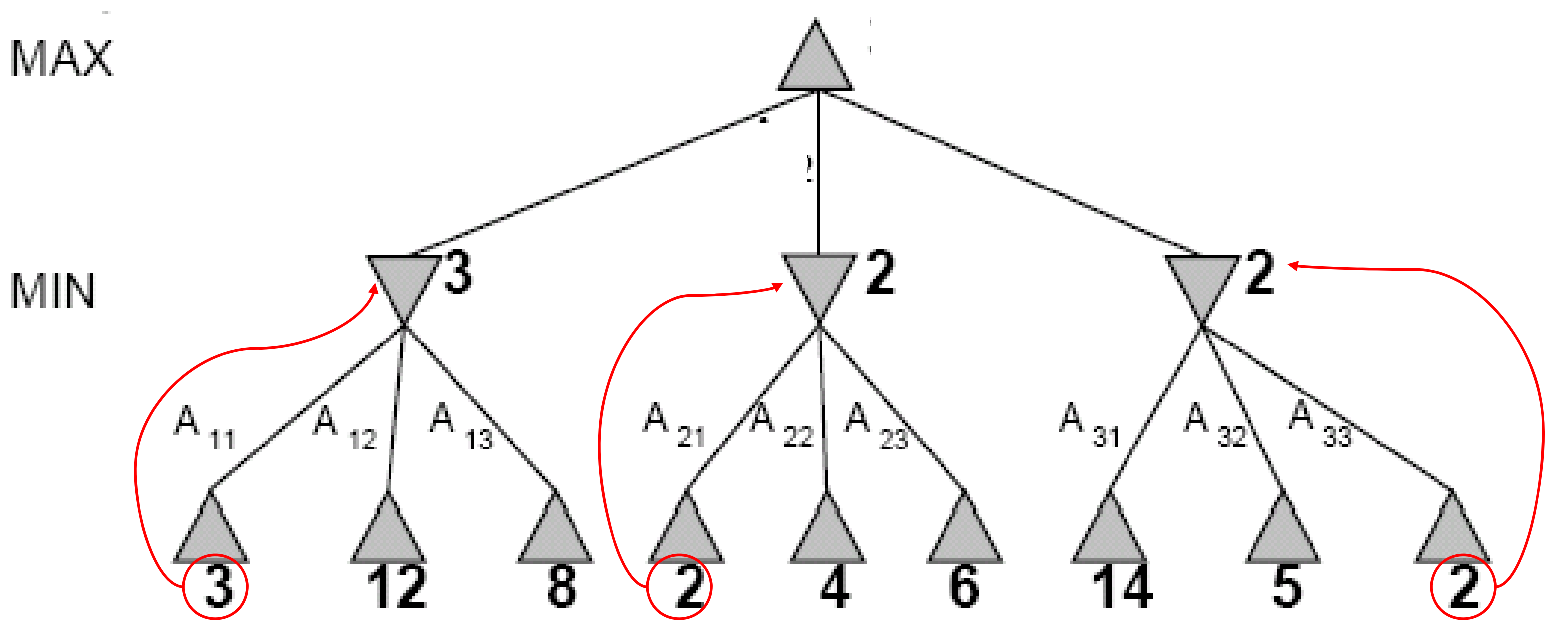


Figure 5.2 A two-ply game tree as generated by the minimax algorithm. The △ nodes are moves by MAX and the ▽ nodes are moves by MIN. The terminal nodes show the utility value for MAX computed by the utility function (i.e., by the rules of the game), whereas the utilities of the other nodes are computed by the minimax algorithm from the utilities of their successors. MAX's best move is A_1, and MIN's best reply is A_{11}.

Two-Ply Game Tree

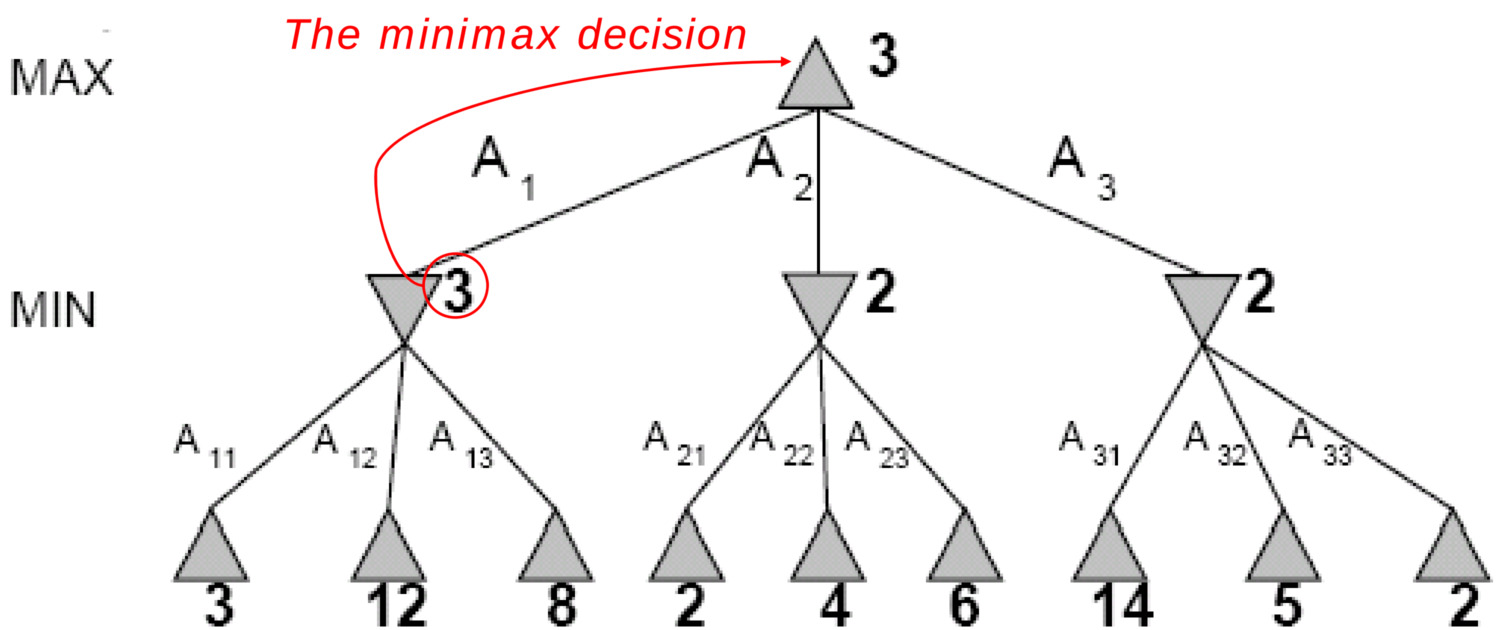


Two-Ply Game Tree



Two-Ply Game Tree

Minimax maximizes the utility for the worst-case outcome for max



Pseudocode for Minimax Algorithm

```
function MINIMAX-DECISION(state) returns an action
  inputs: state, current state in game
return arg max_{a∈ACTIONS(state)} MIN-VALUE(Result(state,a))
```

```
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a in ACTIONS(state) do
    v ← MAX(v,MIN-VALUE(Result(state,a)))
  return v
```

```
function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for a in ACTIONS(state) do
    v ← MIN(v,MAX-VALUE(Result(state,a)))
  return v
```

Properties of minimax

- **Complete?**
  - Yes (if tree is finite).

- **Optimal?**
  - Yes (against an optimal opponent).
  - Can it be beaten by an opponent playing sub-optimally?
    - No. (Why not?)

- **Time complexity?**
  - $O(b^m)$

- **Space complexity?**
  - O(bm) (depth-first search, generate all actions at once)
  - O(m) (depth-first search, generate actions one at a time)

Game Tree Size

- **Tic-Tac-Toe**
  - b ≈ 5 legal actions per state on average, total of 9 plies in game.
    - "ply" = one action by one player, "move" = two plies.
  - 5^9 = 1,953,125
  - 9! = 362,880 (Computer goes first)
  - 8! = 40,320 (Computer goes second)
  - → **exact solution quite reasonable**

- **Chess**
  - b ≈ 35 (approximate average branching factor)
  - d ≈ 100 (depth of game tree for "typical" game)
  - $b^d \approx 35^{100} \approx 10^{154}$ nodes!!
  - → **exact solution completely infeasible**

- **It is usually impossible to develop the whole search tree.**

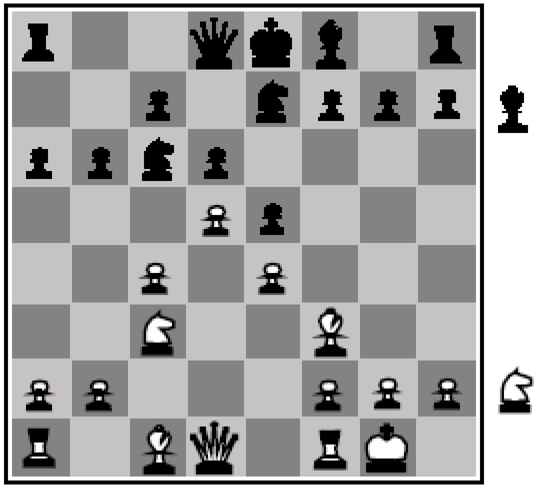
Static (Heuristic) Evaluation Functions

- **An Evaluation Function:**
  - Estimates how good the current board configuration is for a player.
  - Typically, evaluate how good it is for the player, how good it is for the opponent, then subtract the opponent's score from the player's.
  - Othello: Number of white pieces - Number of black pieces
  - Chess: Value of all white pieces - Value of all black pieces

- **Typical values from -infinity (loss) to +infinity (win) or [-1, +1].**

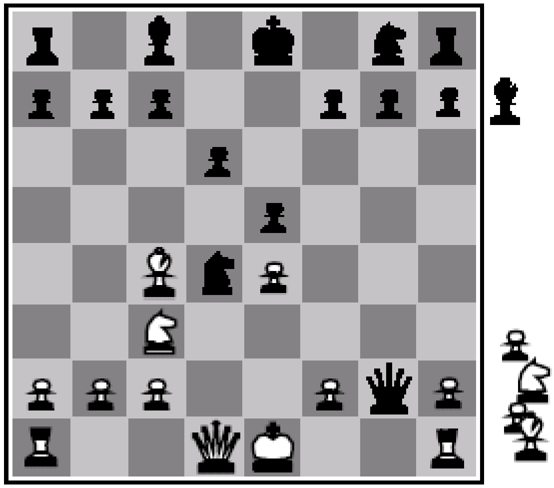
- **If the board evaluation is X for a player, it's -X for the opponent**
  - "Zero-sum game"

Evaluation functions



Black to move

White slightly better



White to move

Black winning

For chess, typically *linear* weighted sum of features

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

e.g., $w_1 = 9$ with
$f_1(s) =$ (number of white queens) – (number of black queens), etc.

Cutting off search

MinimaxCutoff is identical to MinimaxValue except

1. Terminal? is replaced by Cutoff?
2. Utility is replaced by Eval

Does it work in practice?

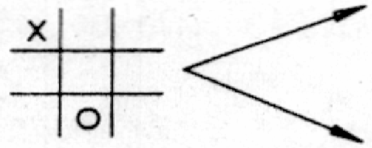
$$b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4$$

4-ply lookahead is a hopeless chess player!

4-ply $\approx$ human novice
8-ply $\approx$ typical PC, human master
12-ply $\approx$ Deep Blue, Kasparov

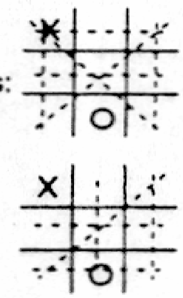
Applying MiniMax to tic-tac-toe

- **The static evaluation function heuristic**

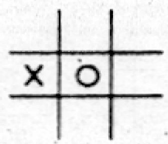


X has 6 possible win paths:

O has 5 possible wins:

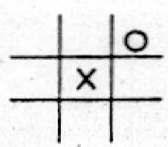


E(n) = 6 – 5 = 1



X has 4 possible win paths;
O has 6 possible wins

E(n) = 4 – 6 = –2



X has 5 possible win paths;
O has 4 possible wins

E(n) = 5 – 4 = 1

Heuristic is E(n) = M(n) – O(n)
where M(n) is the total of My possible winning lines
O(n) is total of Opponent's possible winning lines
E(n) is the total Evaluation for state n

Figure 4.16 Heuristic measuring conflict applied to states of tic-tac-toe.

Backup Values

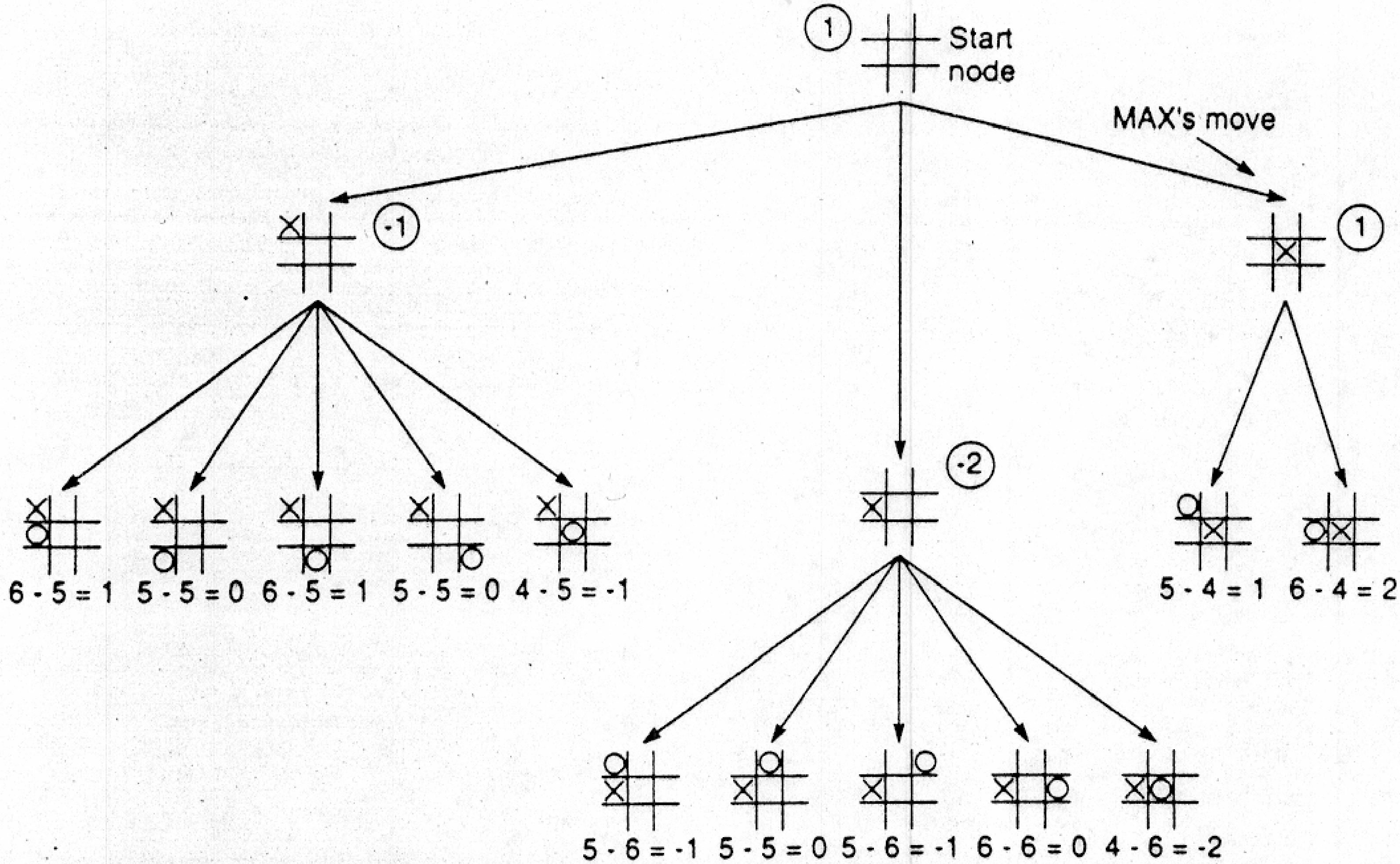


Figure 4.17 Two-ply minimax applied to the opening move of tic-tac-toe.

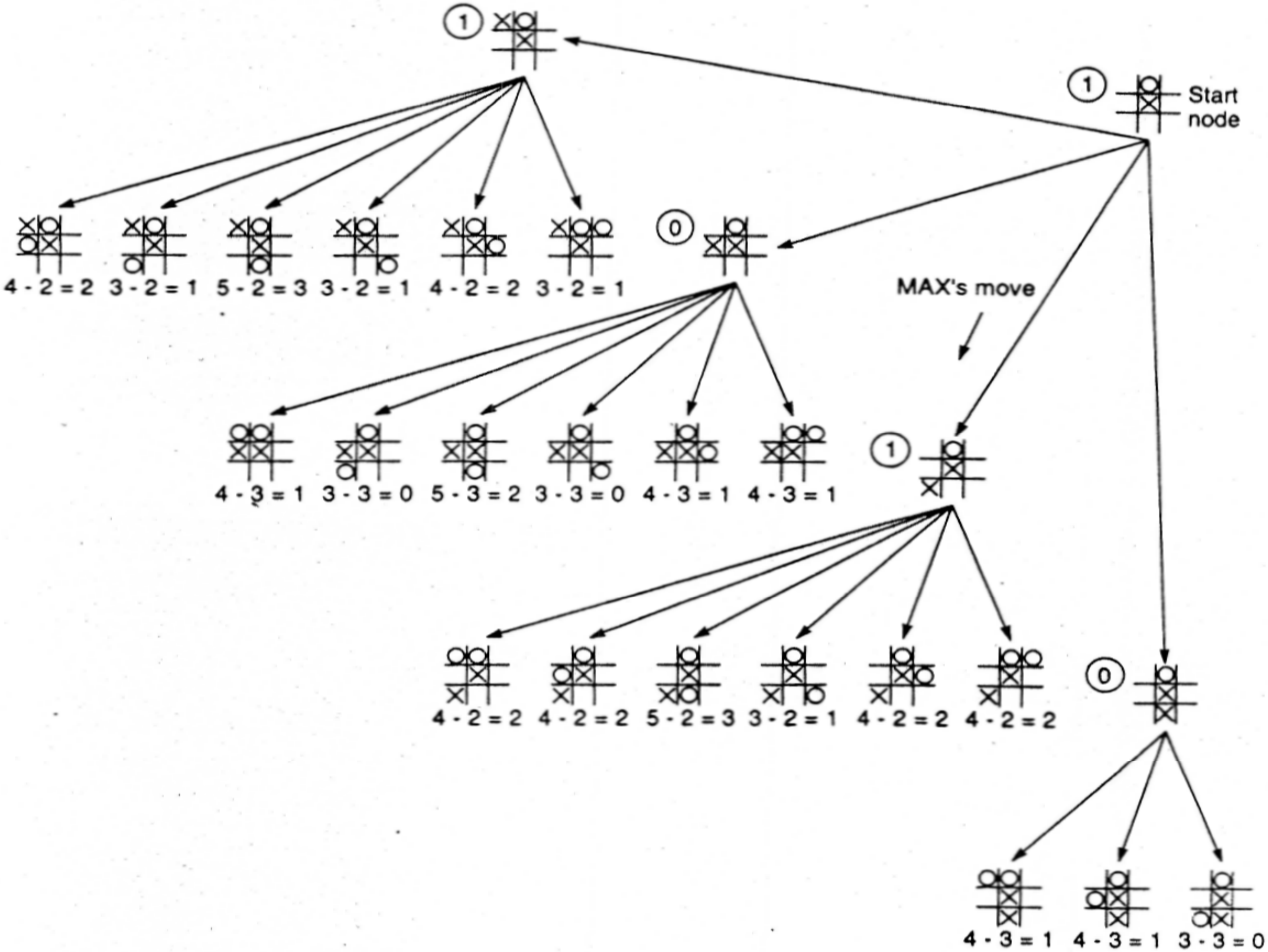


Figure 4.18 Two-ply minimax applied to **X**'s second move of tic-tac-toe.

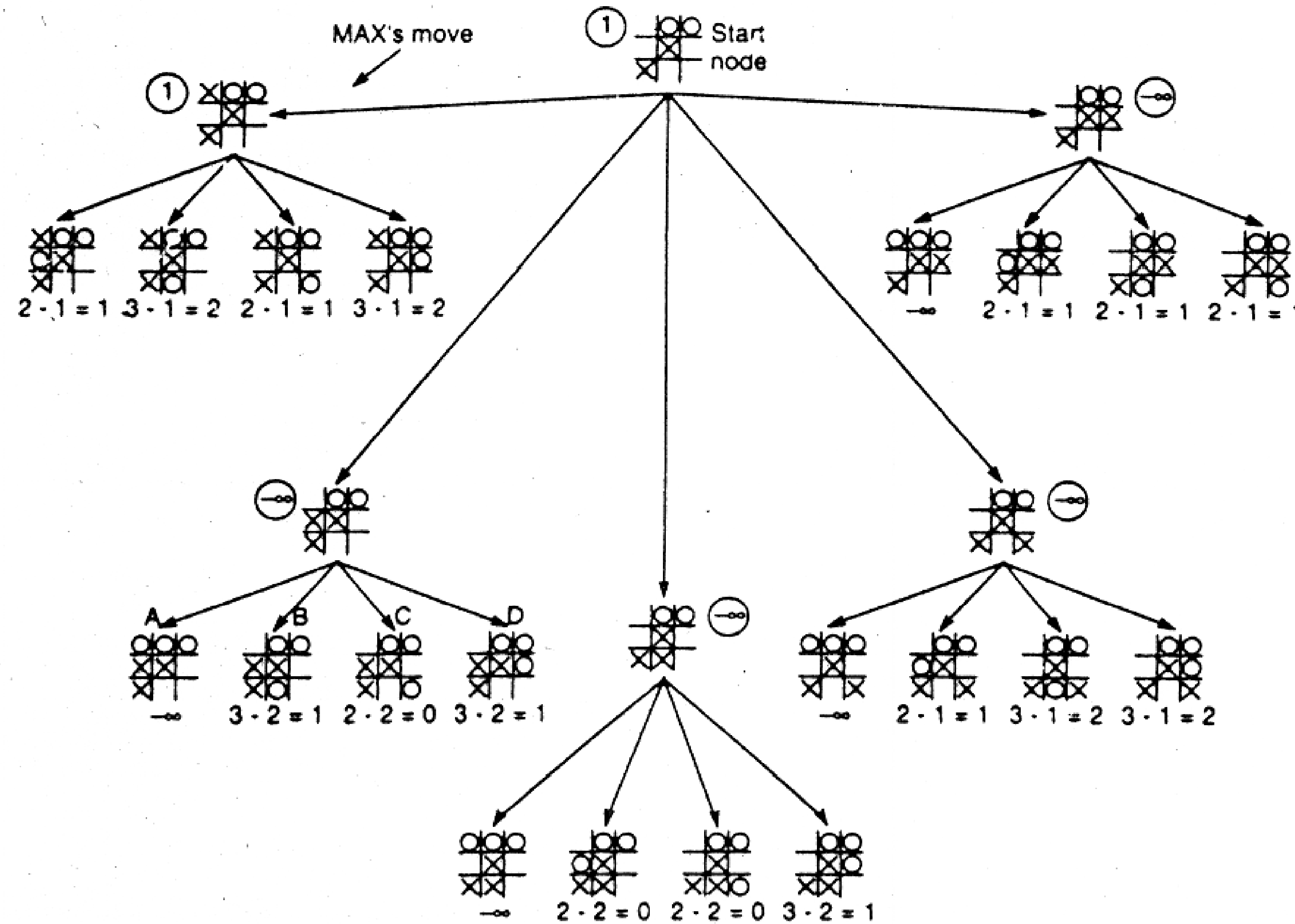
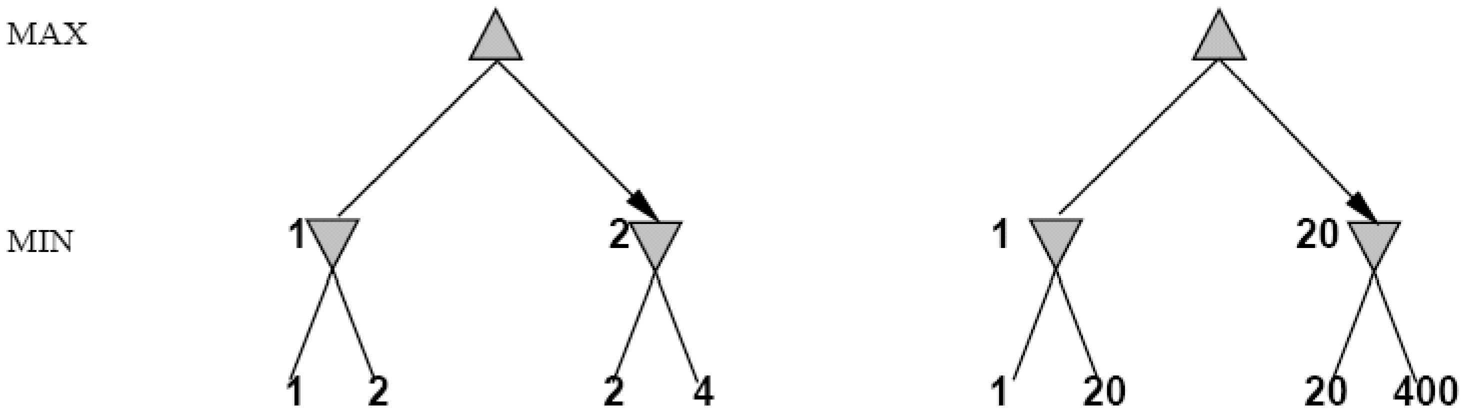


Figure 4.19 Two-ply minimax applied to **X**'s move near end game.

Digression: Exact values don't matter



Behaviour is preserved under any *monotonic* transformation of EVAL

Only the order matters:
payoff in deterministic games acts as an *ordinal utility* function

Alpha-Beta Pruning
Exploiting the Fact of an Adversary

- **If a position is provably bad:**
  - It is NO USE expending search time to find out exactly how bad
- **If the adversary can force a bad position:**
  - It is NO USE expending search time to find out the good positions that the adversary won't let you achieve anyway
- **Bad = not better than we already know we can achieve elsewhere.**
- **Contrast normal search:**
  - ANY node might be a winner.
  - ALL nodes must be considered.
  - (A* avoids this through knowledge, i.e., heuristics)

Tic-Tac-Toe Example with Alpha-Beta Pruning

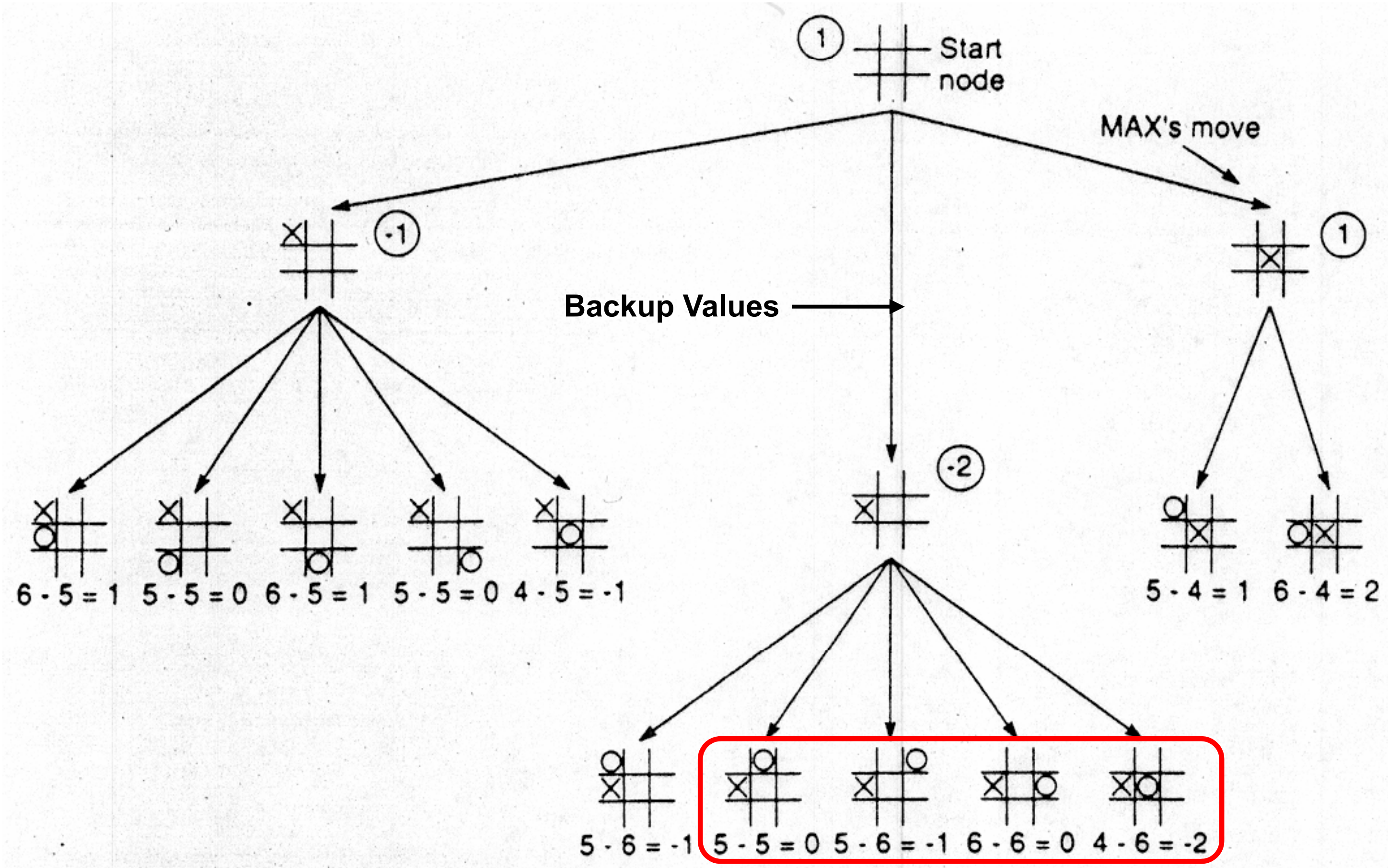
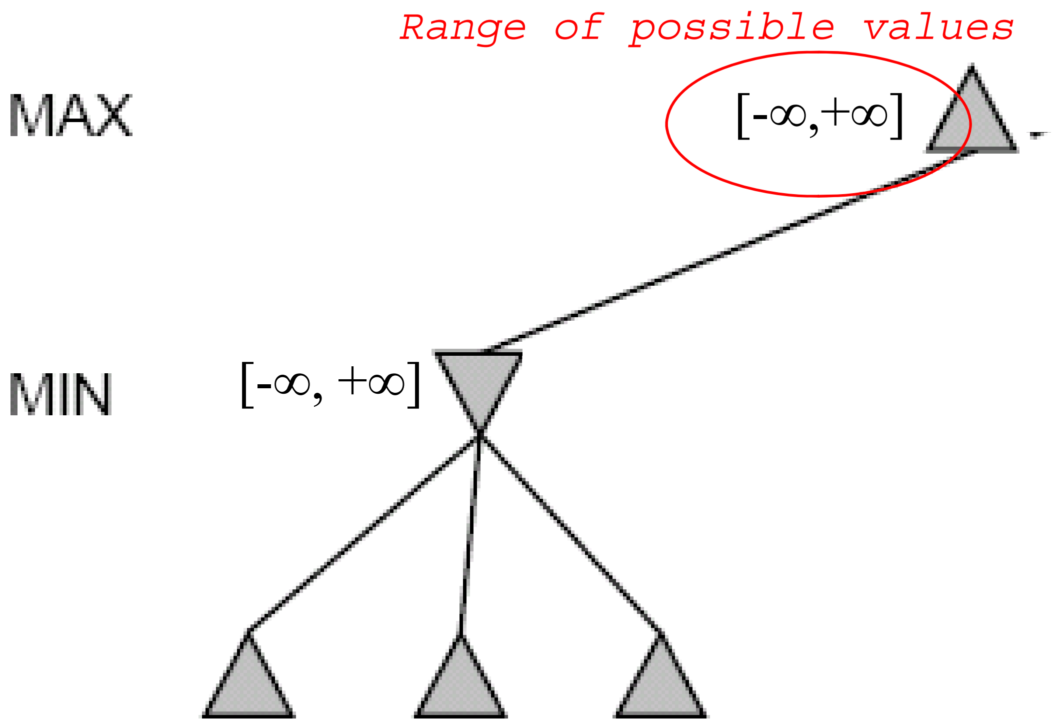


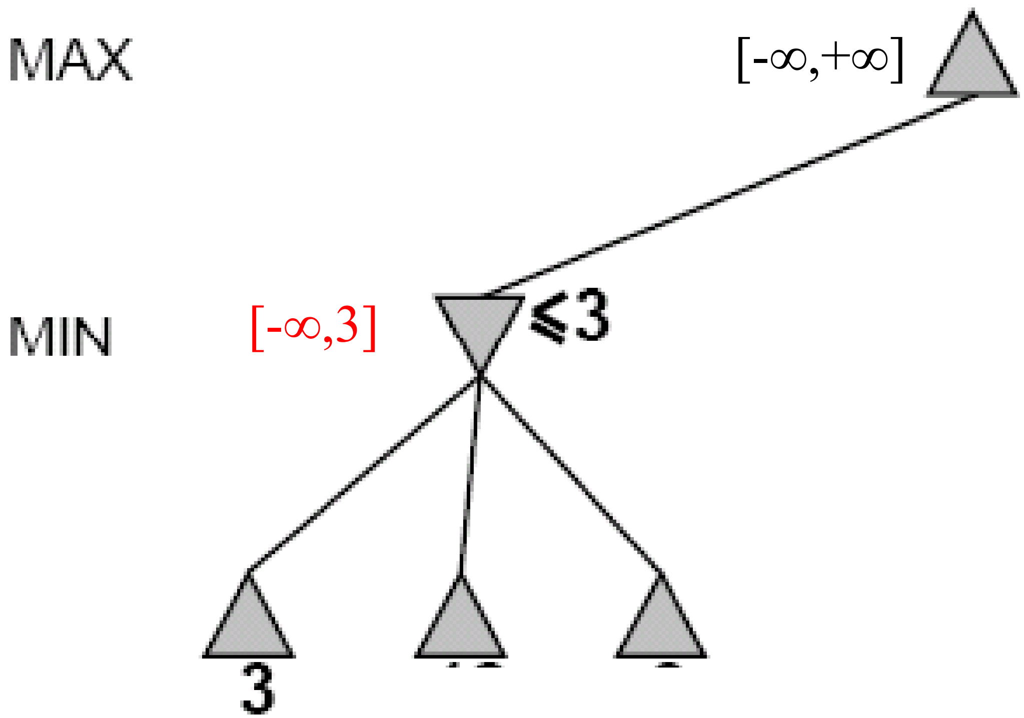
Figure 4.17 Two-ply minimax applied to the opening move of tic-tac-toe.

Another Alpha-Beta Example

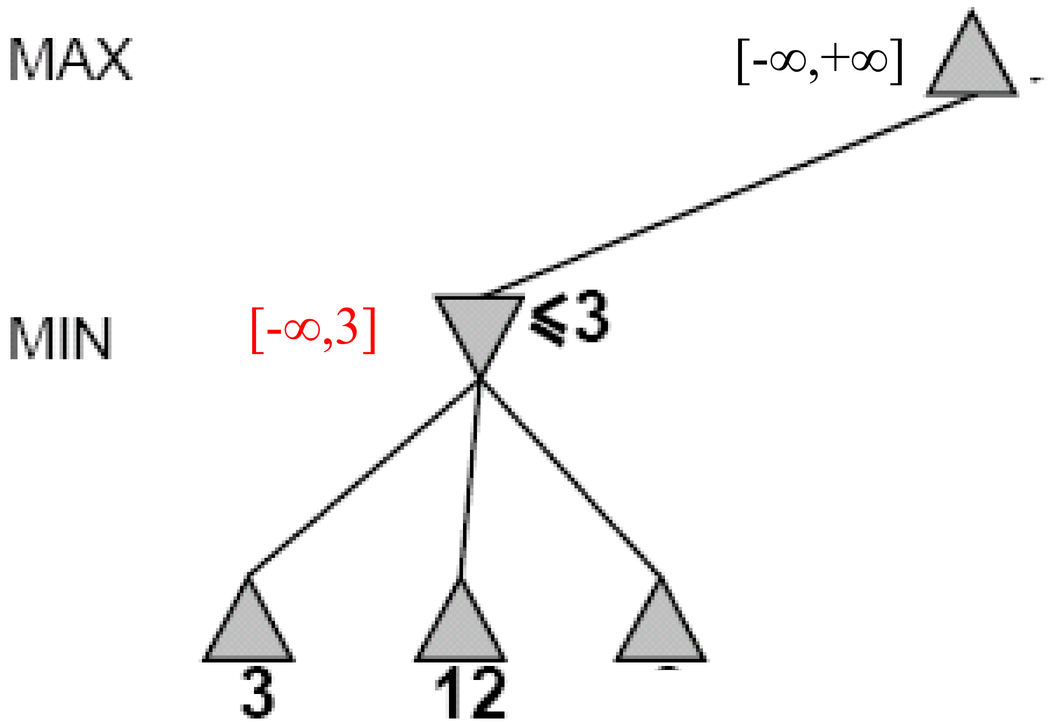
Do DF-search until first leaf



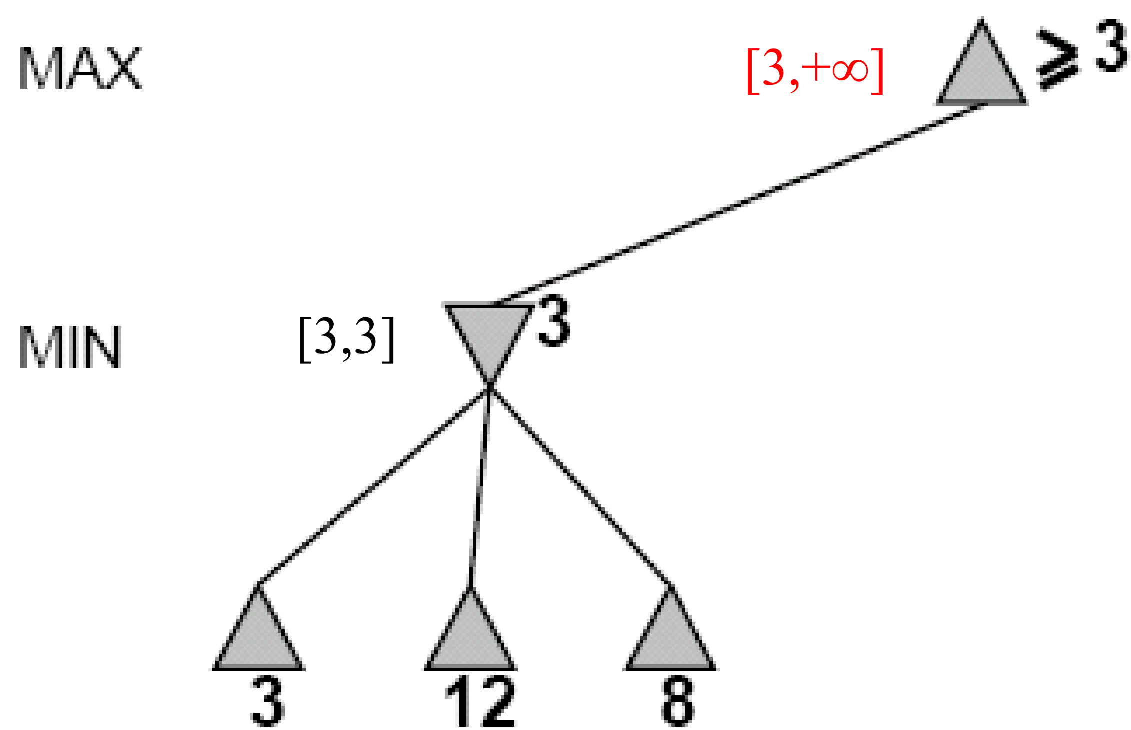
Alpha-Beta Example (continued)



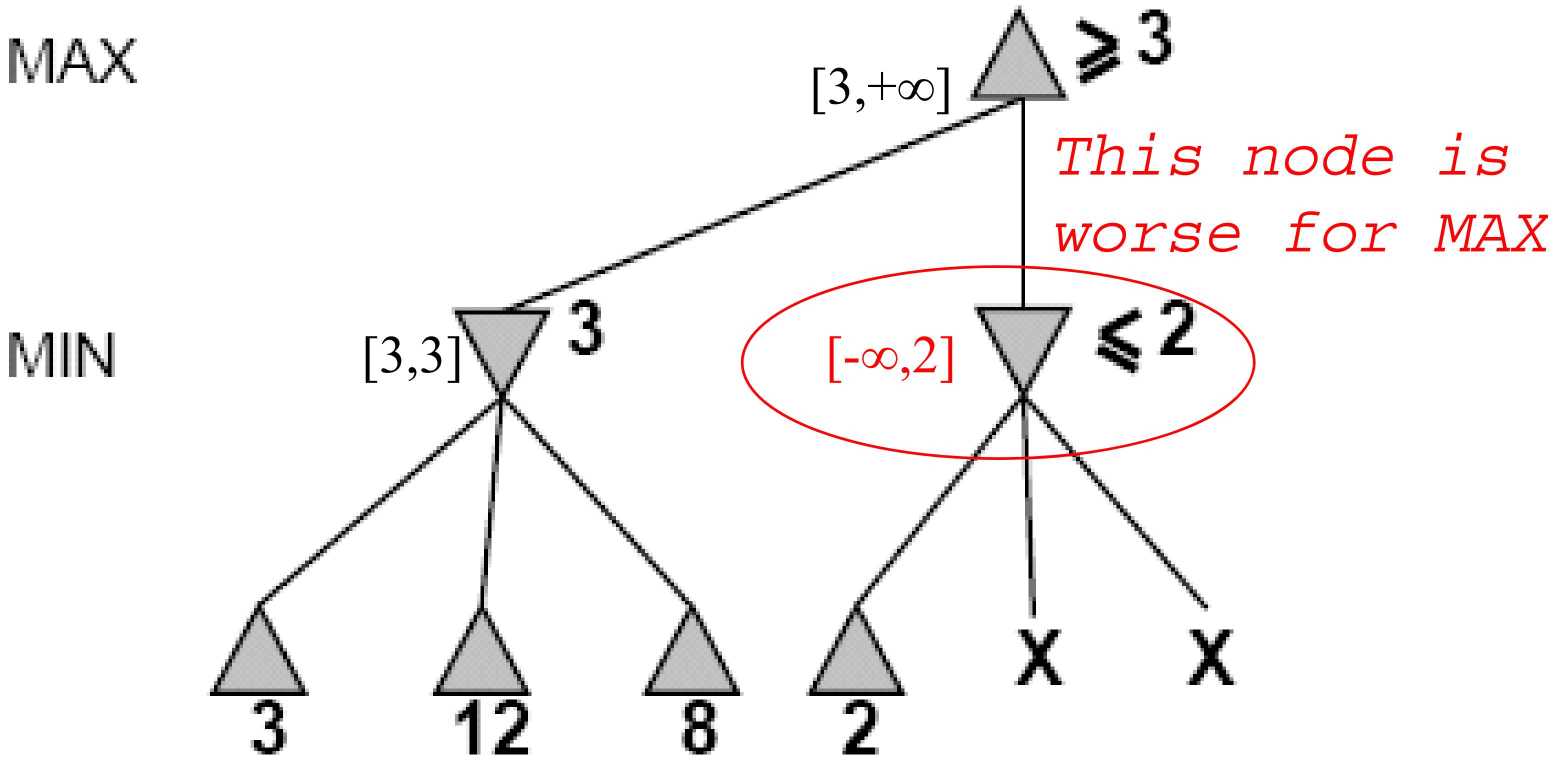
Alpha-Beta Example (continued)



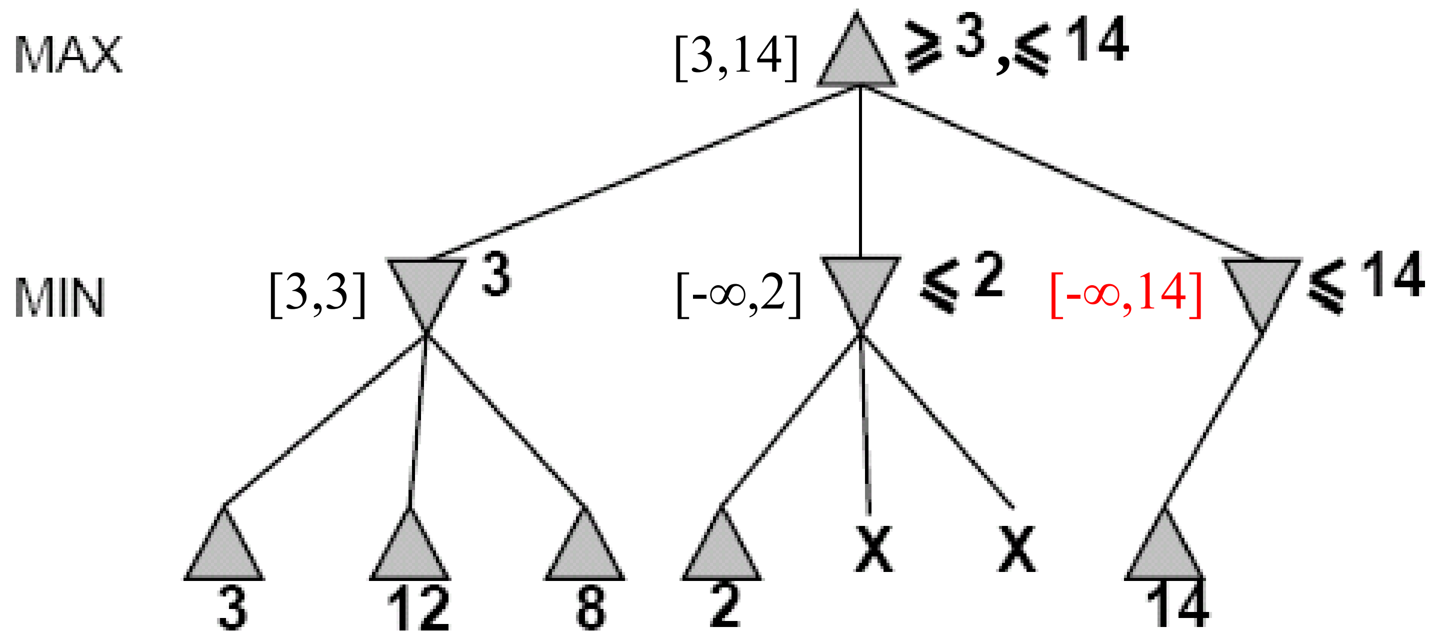
Alpha-Beta Example (continued)



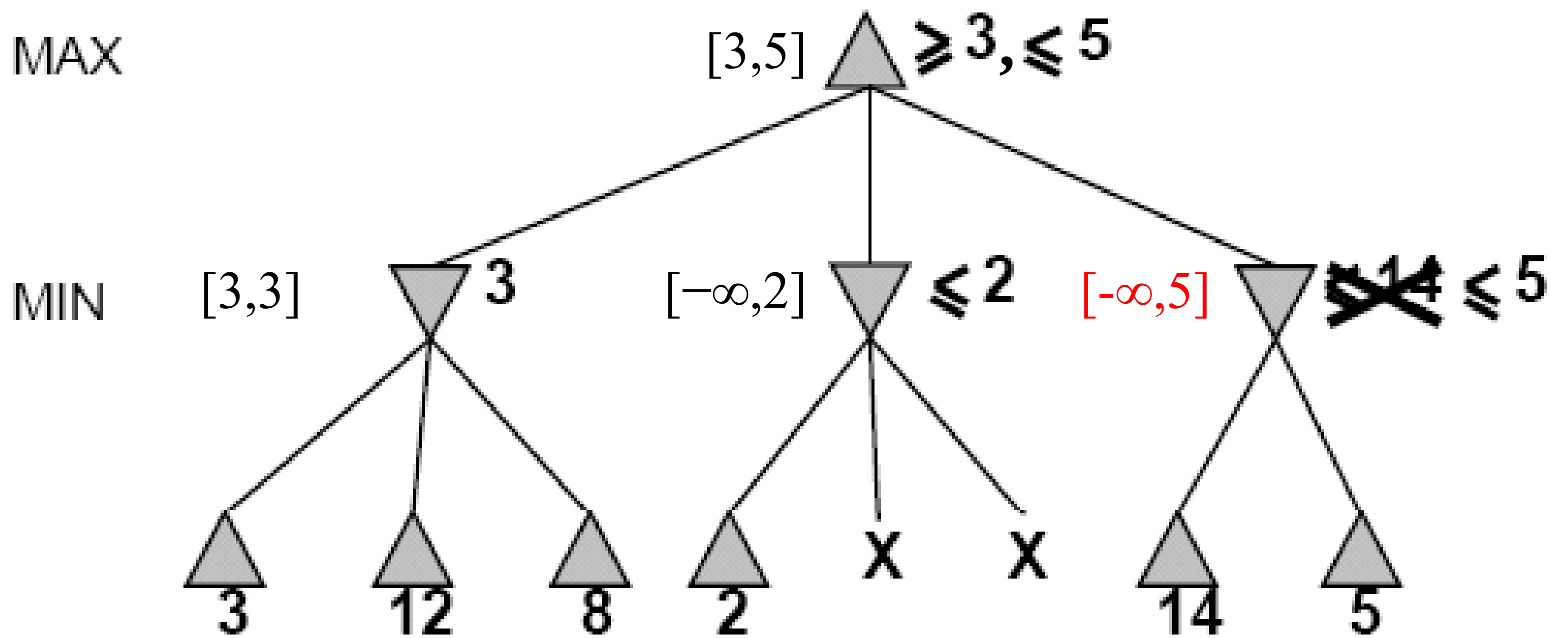
Alpha-Beta Example (continued)



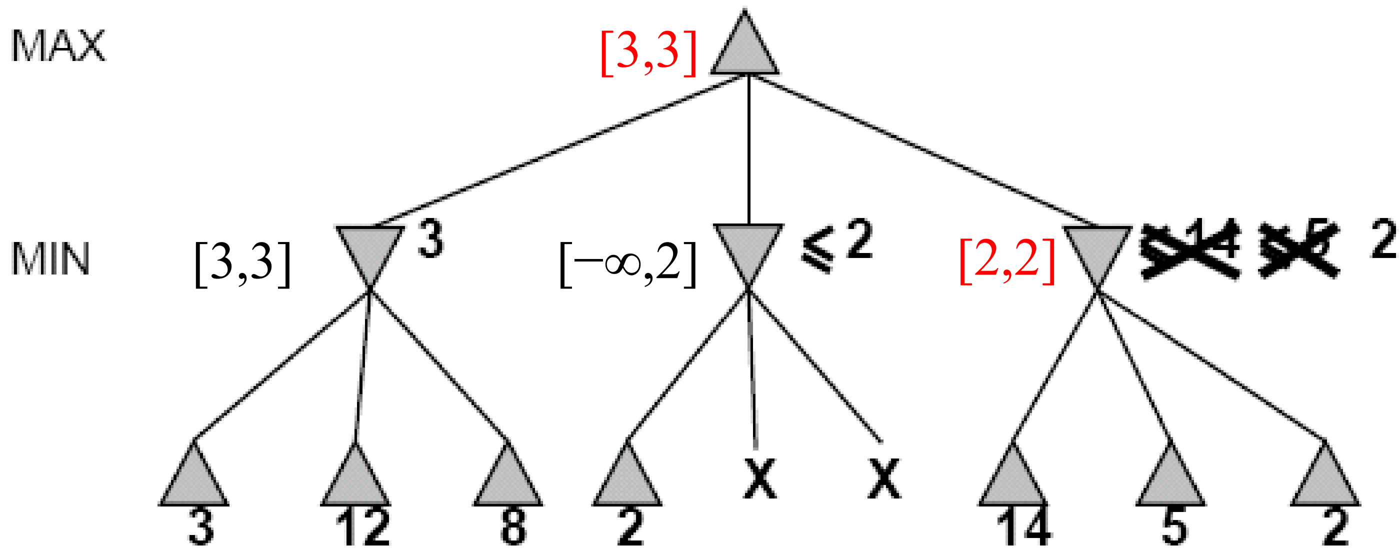
Alpha-Beta Example (continued)



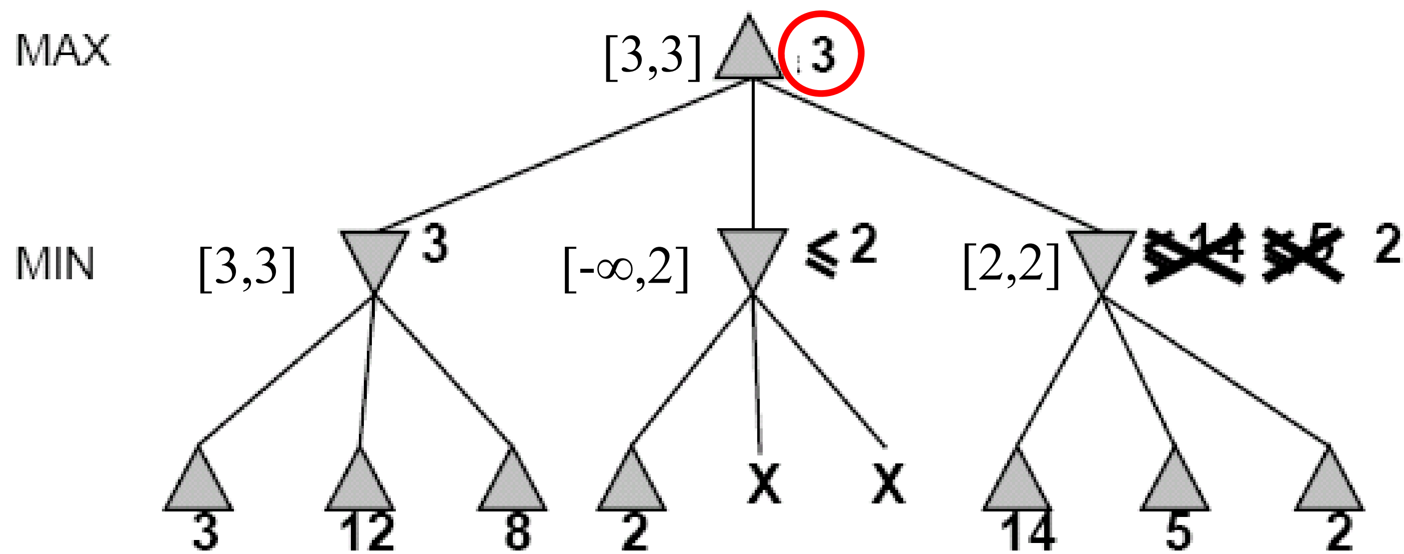
Alpha-Beta Example (continued)



Alpha-Beta Example (continued)

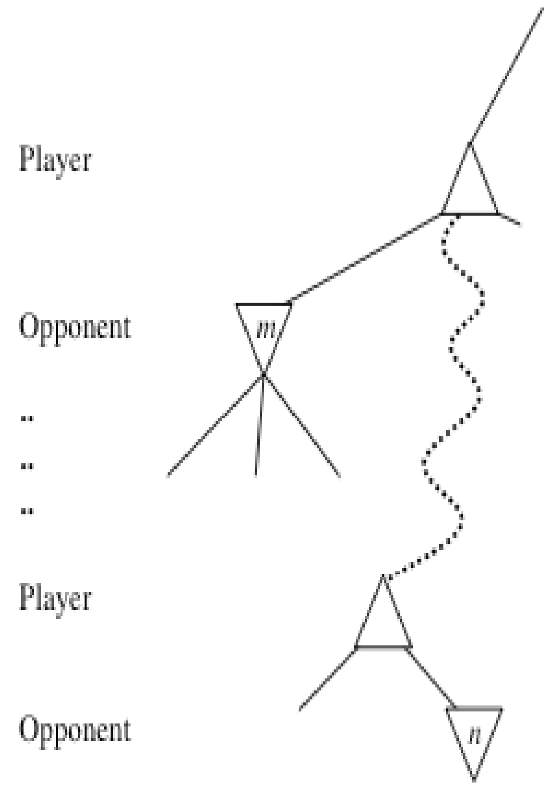


Alpha-Beta Example (continued)



General alpha-beta pruning

- **Consider a node *n* in the tree ---**
- **If player has a better choice at:**
  - Parent node of n
  - Or any choice point further up
- **Then *n* will never be reached in play**
- **Hence, when that much is known about *n*, it can be pruned.**



Alpha-beta Algorithm

- **Depth first search**
  - only considers nodes along a single path from root at any time

α = highest-value choice found at any choice point of path for MAX

(initially, α = −infinity)

β = lowest-value choice found at any choice point of path for MIN

(initially, β = +infinity)

- **Pass current values of α and β down to child nodes during search.**
- **Update values of α and β during search:**
  - MAX updates α at MAX nodes
  - MIN updates β at MIN nodes
- **Prune remaining branches at a node when $\alpha \geq \beta$**

When to Prune

- **Prune whenever $\alpha \geq \beta$.**
  - Prune below a Max node whose alpha value becomes greater than or equal to the beta value of its ancestors.
    - **Max nodes update alpha** based on children's returned values.
  - Prune below a Min node whose beta value becomes less than or equal to the alpha value of its ancestors.
    - **Min nodes update beta** based on children's returned values.

Pseudocode for Alpha-Beta Algorithm

function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game
  $\boxed{v \leftarrow \text{MAX-VALUE}(\mathit{state}, -\infty, +\infty)}$
  return the *action* in SUCCESSORS(*state*) with value *v*

Pseudocode for Alpha-Beta Algorithm

function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty$, $+\infty$)
  return the *action* in ACTIONS(*state*) with value v

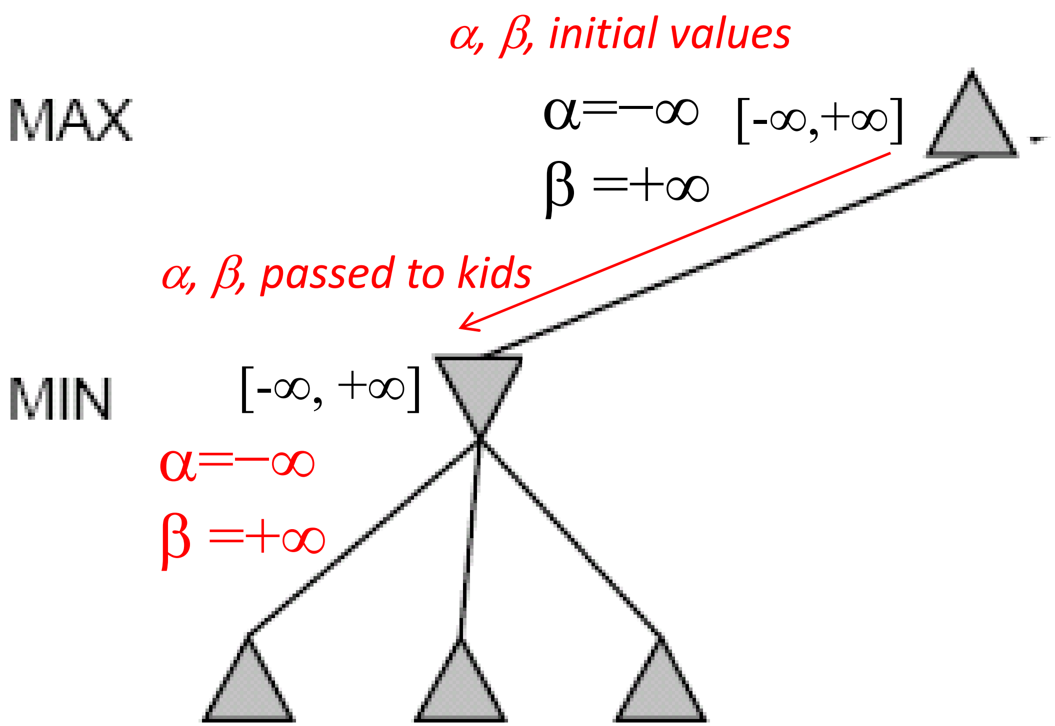
---

function MAX-VALUE(*state*, α , β) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for *a* in ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(Result(*s*,a), α , β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α , v)
  return v

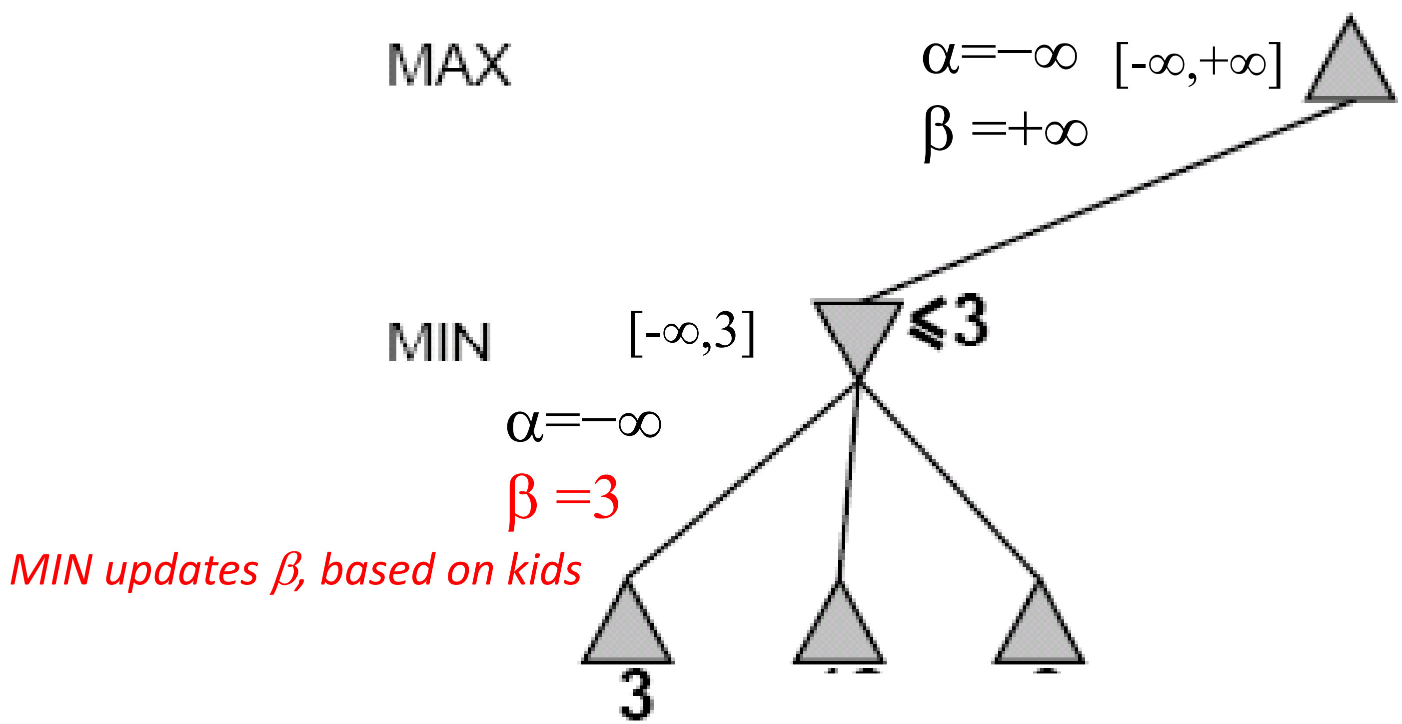
(MIN-VALUE is defined analogously)

Alpha-Beta Example Revisited

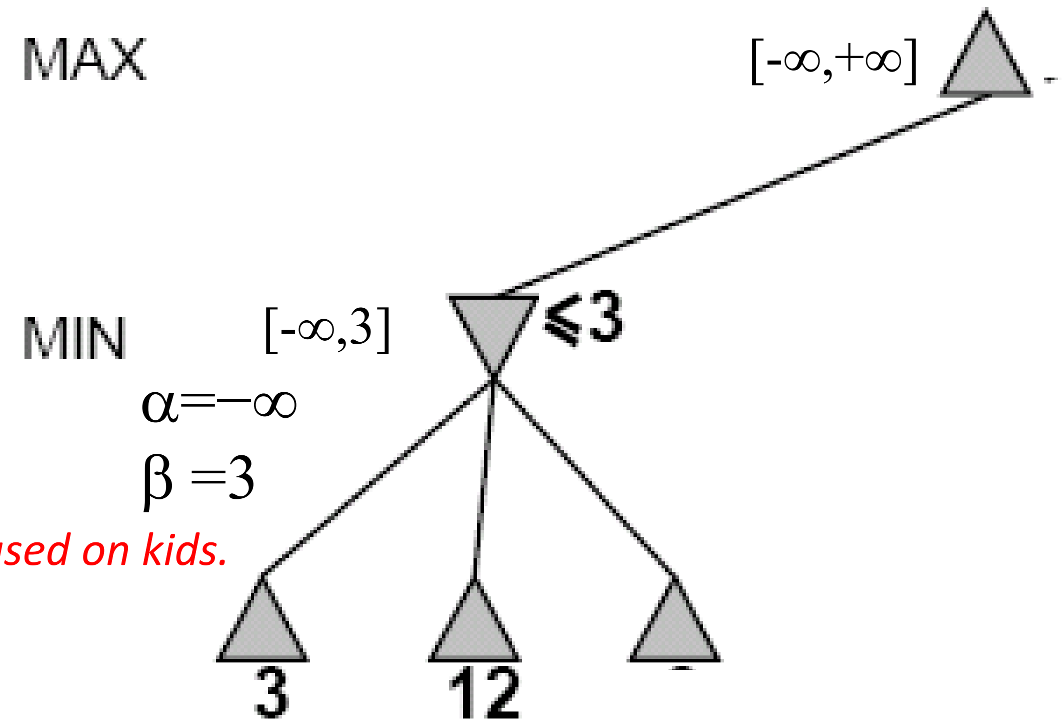
Do DF-search until first leaf



Alpha-Beta Example (continued)

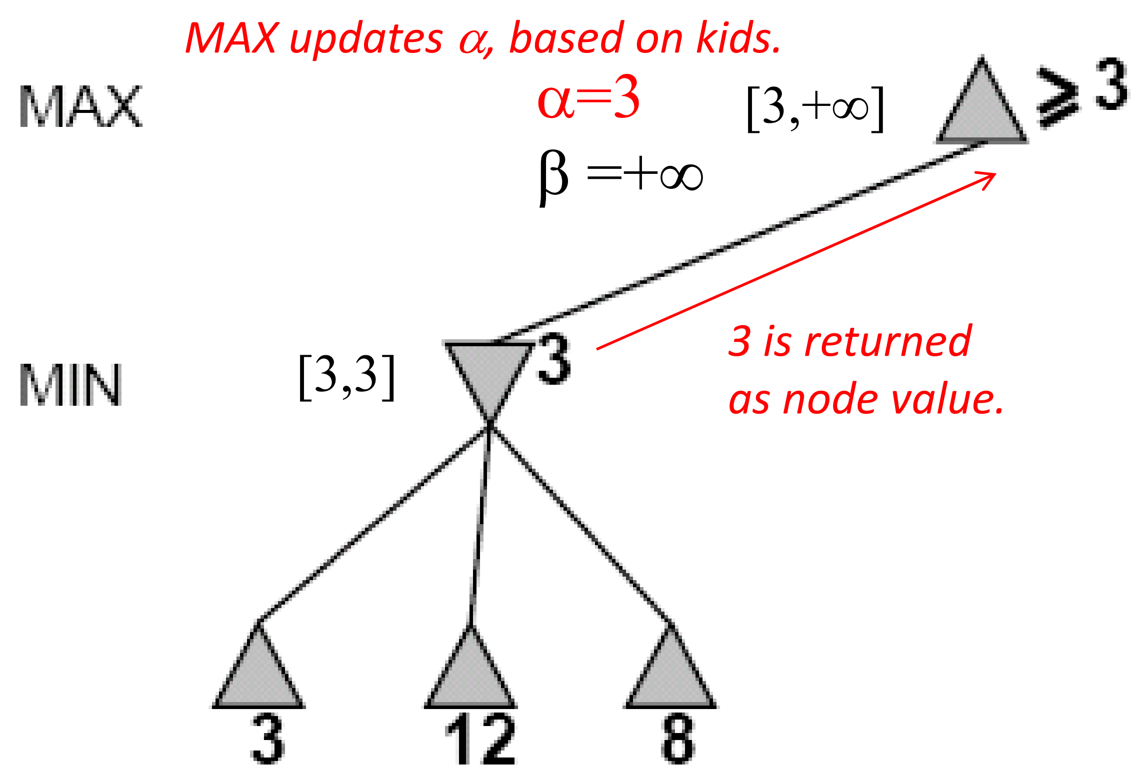


Alpha-Beta Example (continued)

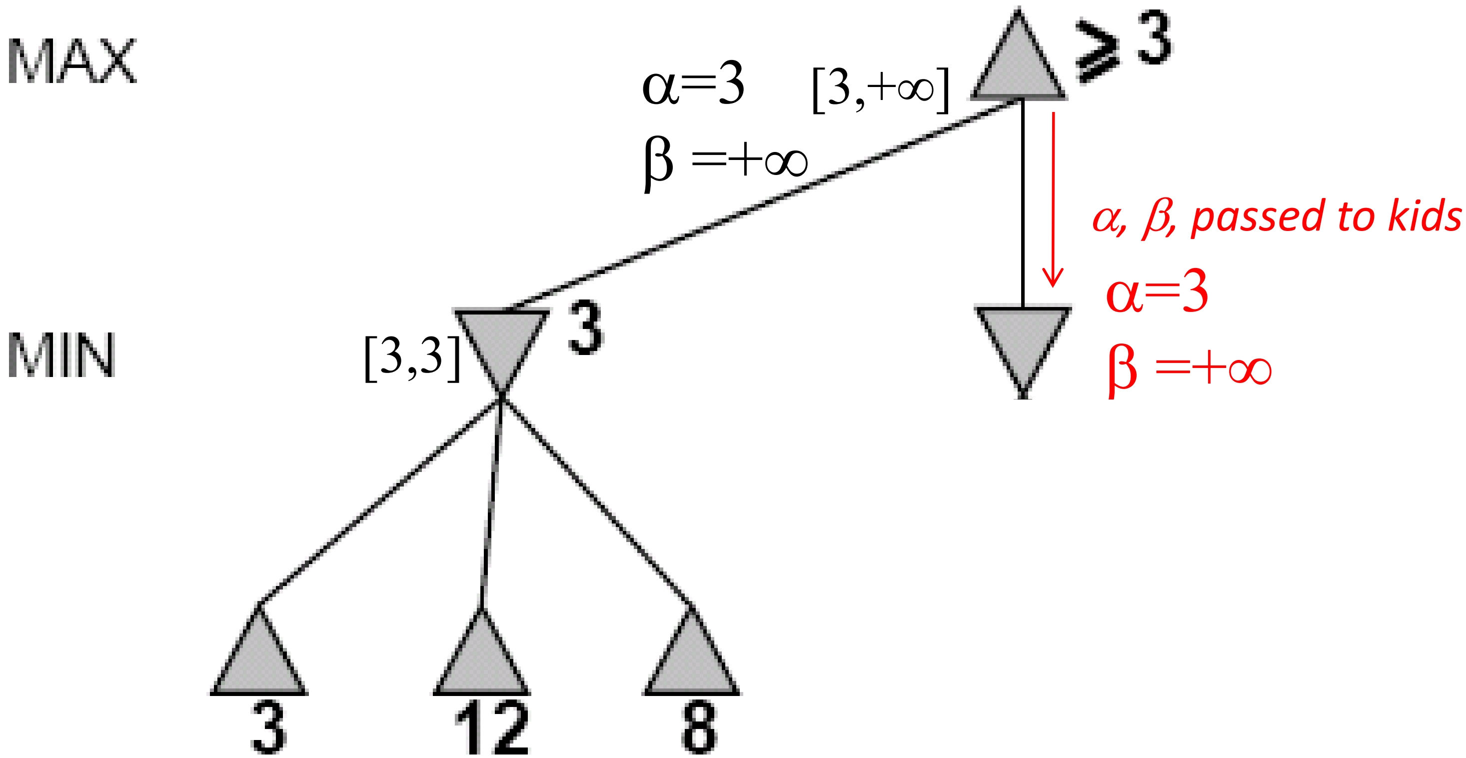


MIN updates β, based on kids.
No change.

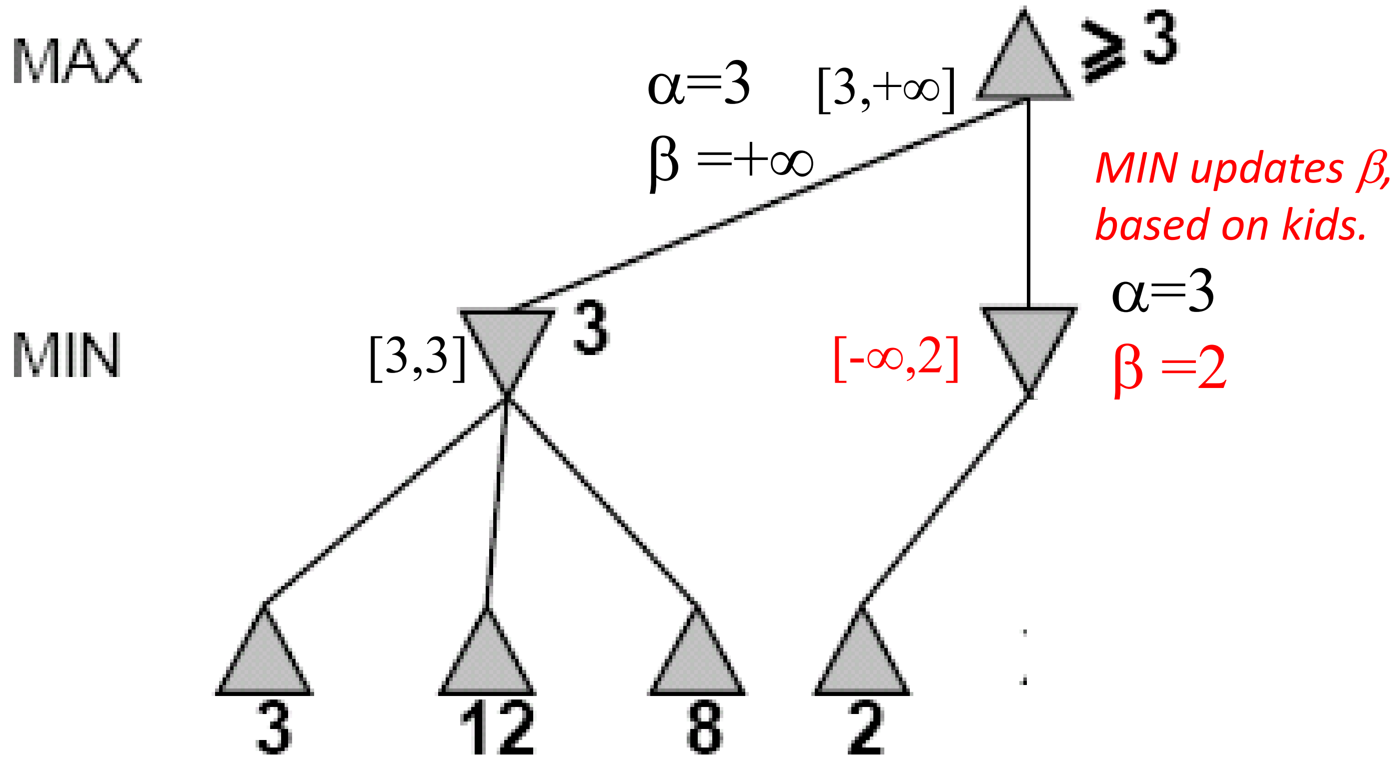
Alpha-Beta Example (continued)



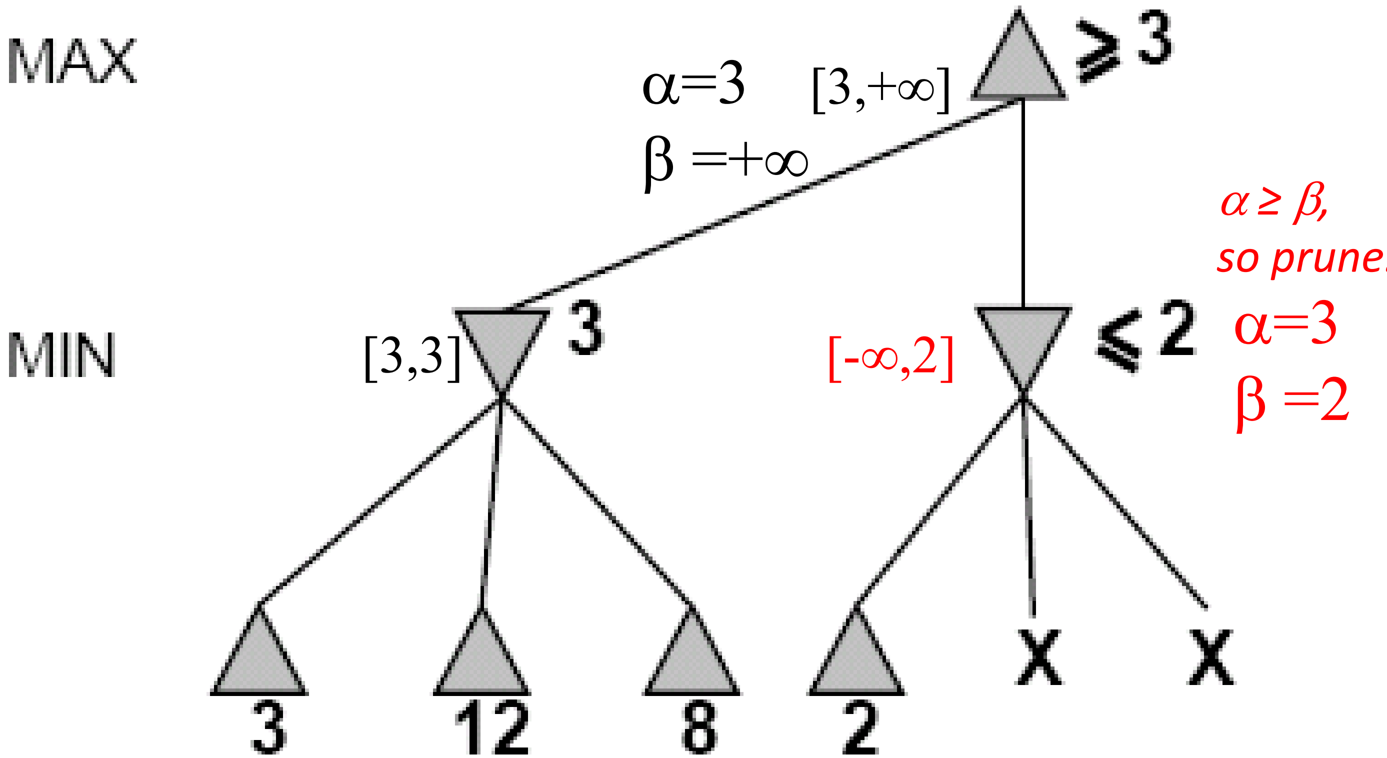
Alpha-Beta Example (continued)



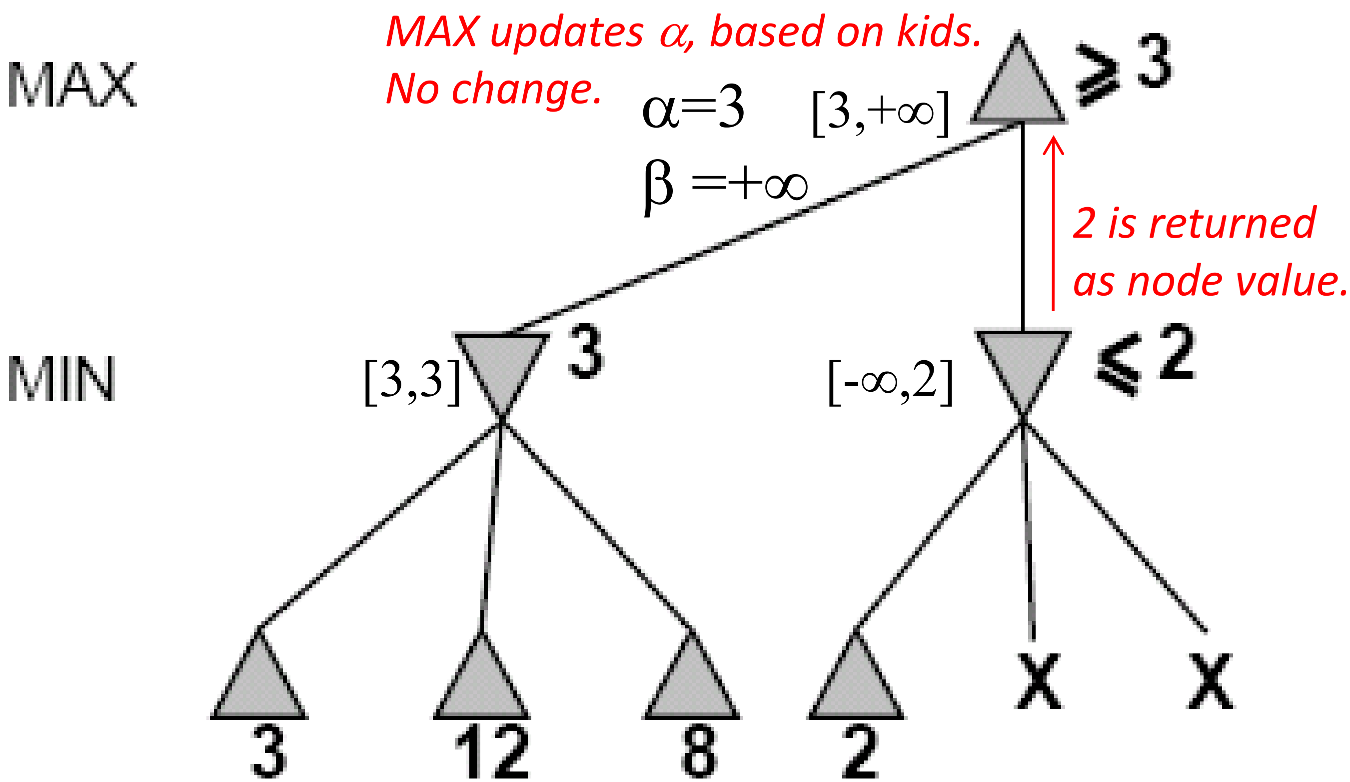
Alpha-Beta Example (continued)



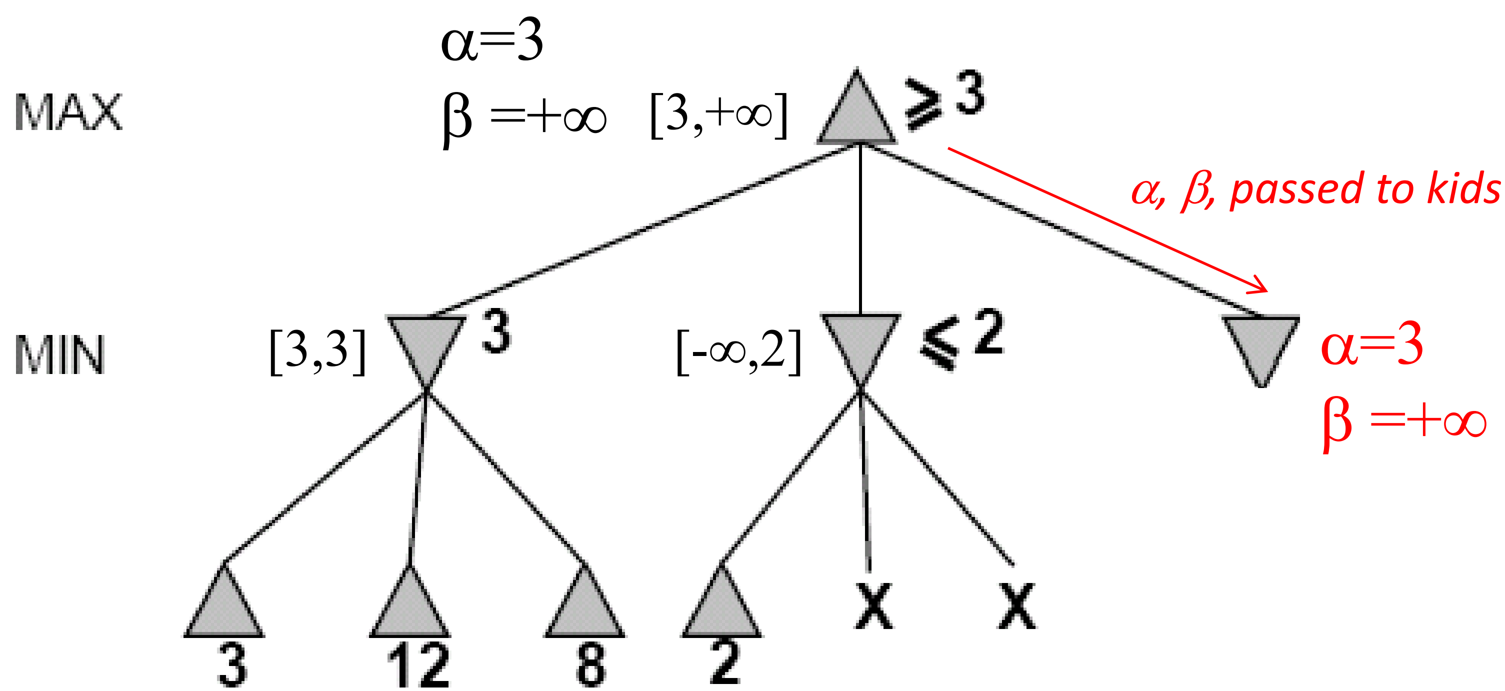
Alpha-Beta Example (continued)



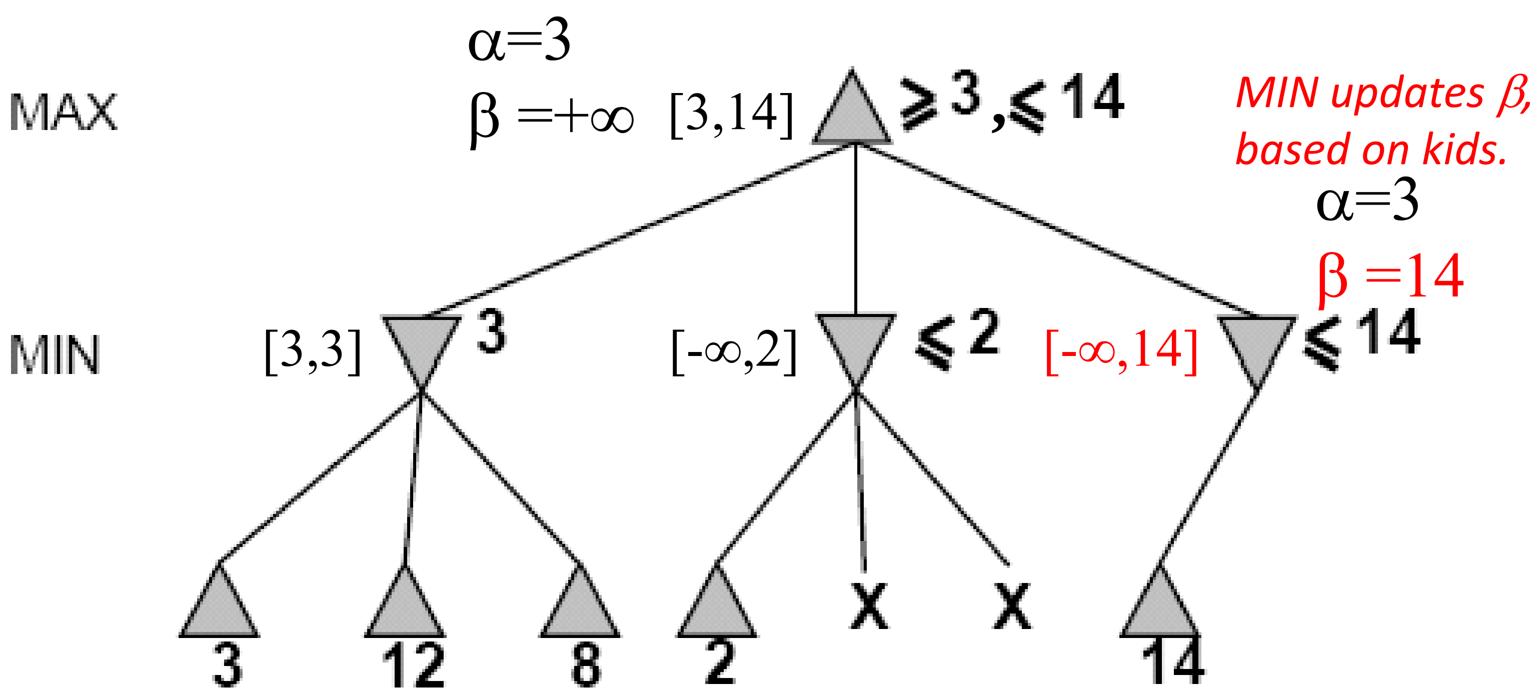
Alpha-Beta Example (continued)



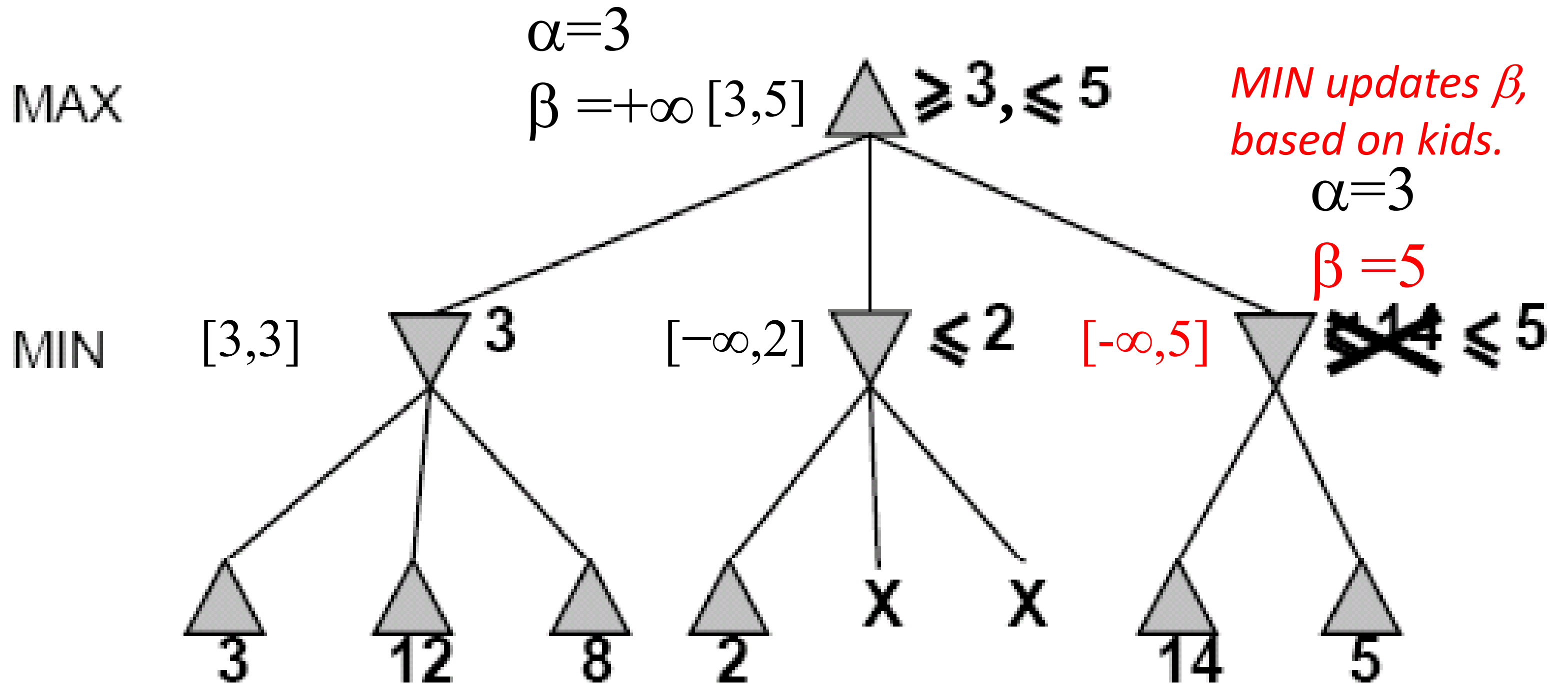
Alpha-Beta Example (continued)



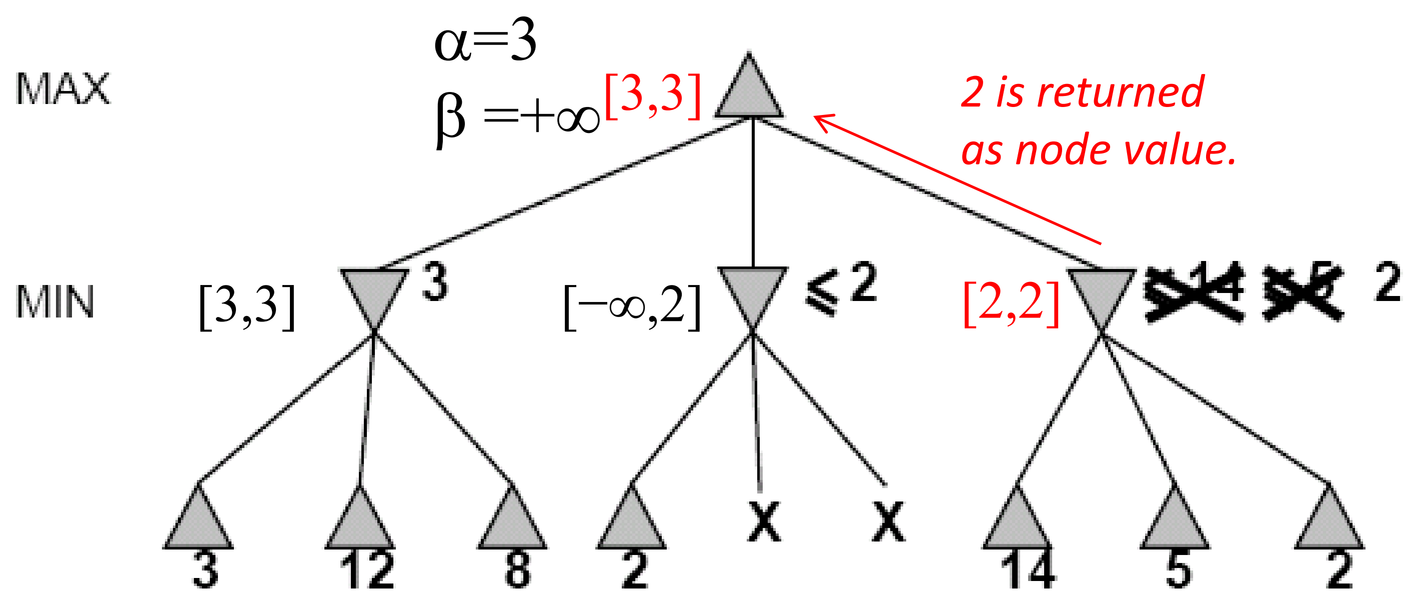
Alpha-Beta Example (continued)



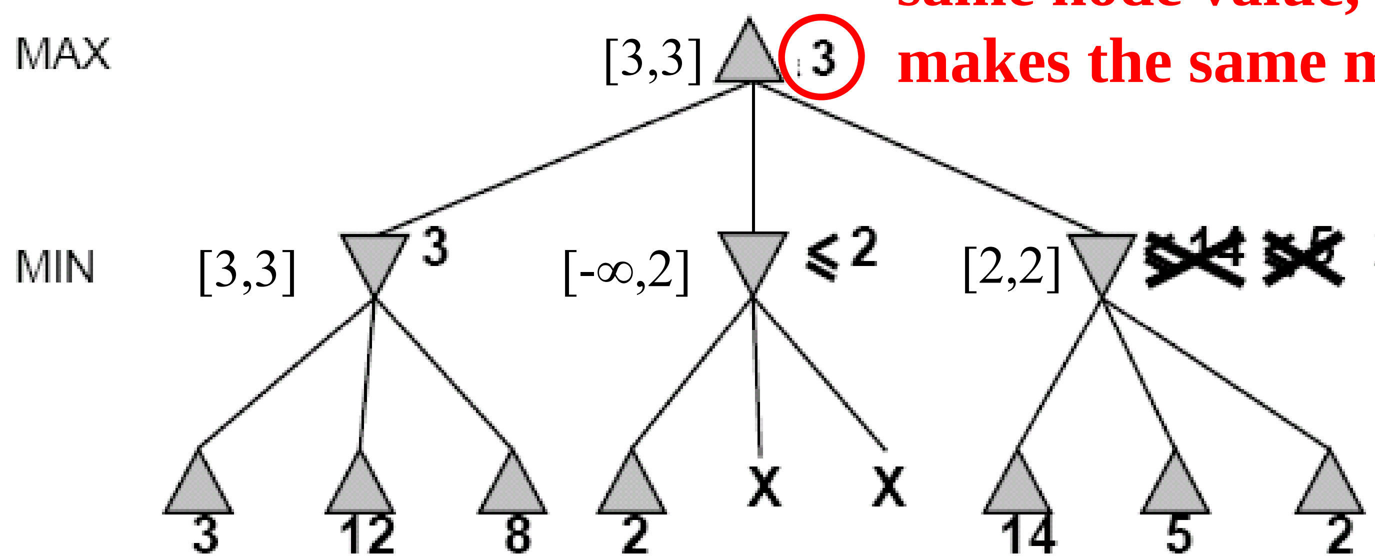
Alpha-Beta Example (continued)



Alpha-Beta Example (continued)



Alpha-Beta Example (continued)



Max calculates the same node value, and makes the same move!

Effectiveness of Alpha-Beta Search

- **Worst-Case**
  - branches are ordered so that no pruning takes place. In this case alpha-beta gives no improvement over exhaustive search

- **Best-Case**
  - each player's best move is the left-most child (i.e., evaluated first)
  - in practice, performance is closer to best rather than worst-case
  - E.g., sort moves by the remembered move values found last time.
  - E.g., expand captures first, then threats, then forward moves, etc.
  - E.g., run Iterative Deepening search, sort by value last iteration.

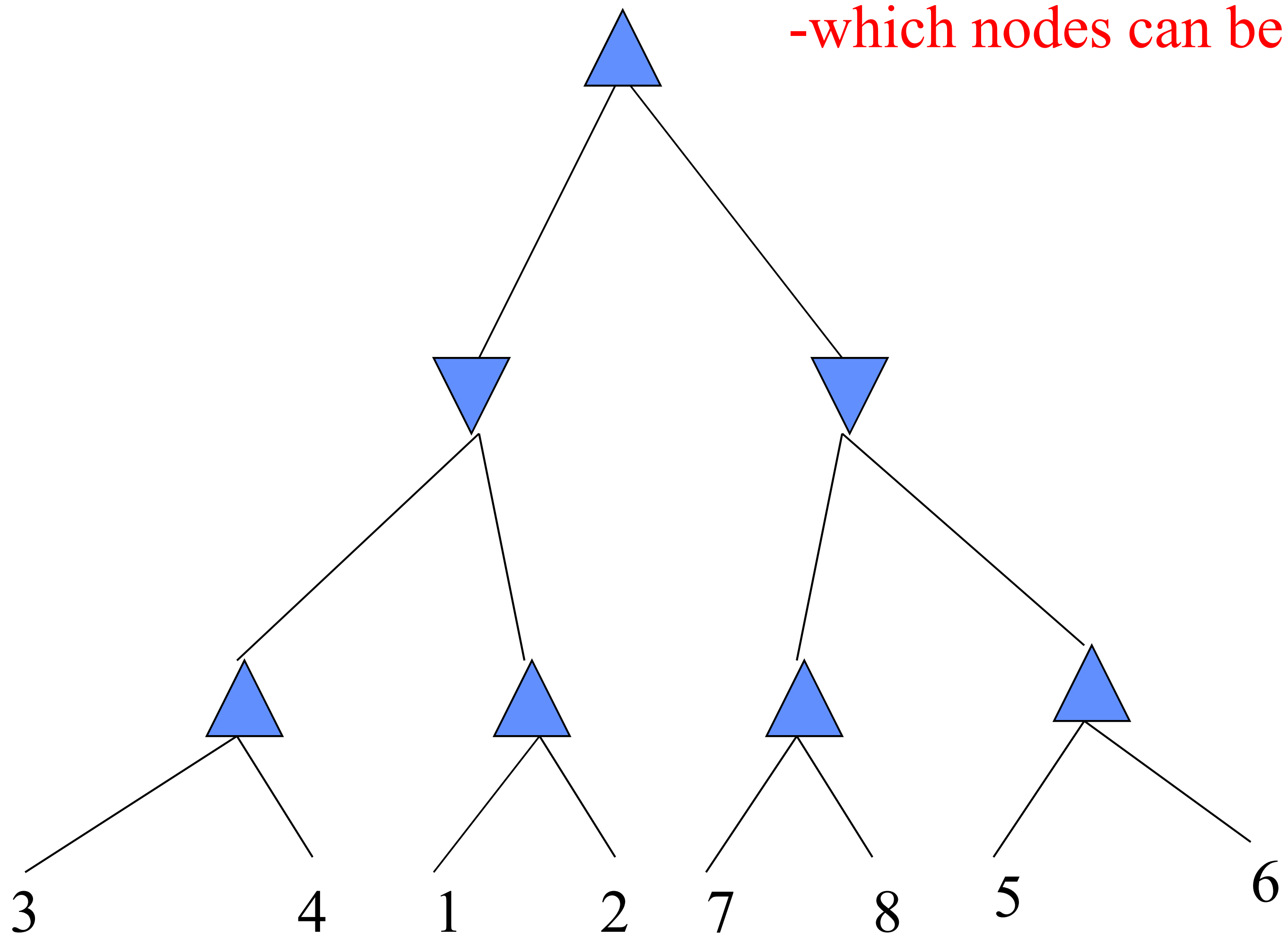
- **In practice often get $O(b^{(d/2)})$ rather than $O(b^d)$**
  - this is the same as having a branching factor of sqrt(b),
    - $(\mathrm{sqrt}(b))^d = b^{(d/2)}$,i.e., we effectively go from b to square root of b
  - e.g., in chess go from b ~ 35 to b ~ 6
    - this permits much deeper search in the same amount of time

Final Comments about Alpha-Beta Pruning

- **Pruning does not affect final results**
- **Entire subtrees can be pruned.**
- **Good move *ordering* improves effectiveness of pruning**
- **Repeated states are again possible.**
  - Store them in memory = transposition table

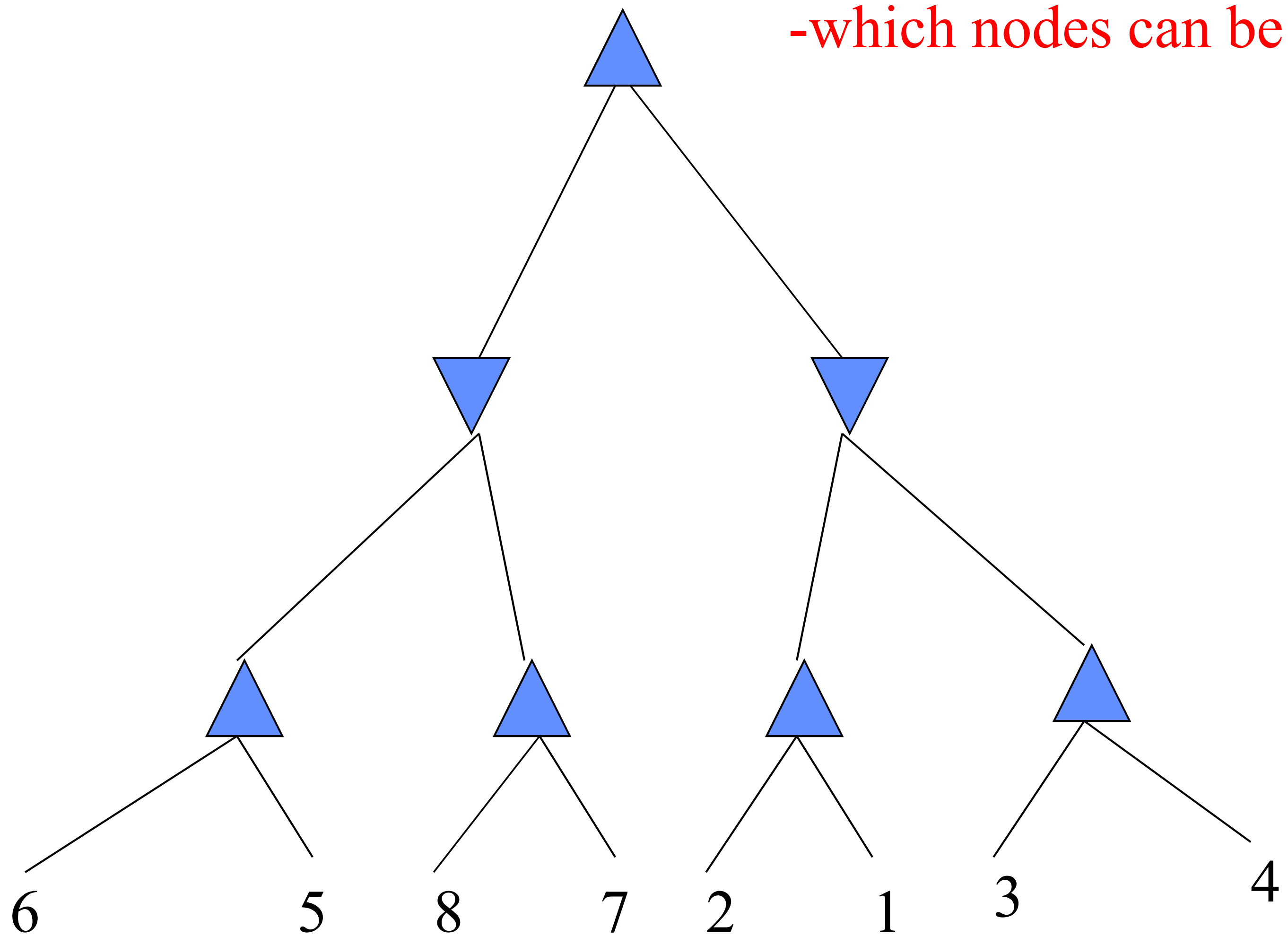
Example

-which nodes can be pruned?



Second Example

-which nodes can be pruned?



Deterministic games in practice

Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.

Chess: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

Othello: human champions refuse to compete against computers, who are too good.

Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

Iterative (Progressive) Deepening

- **In real games, there is usually a time limit T on making a move**

- **How do we take this into account?**
- **using alpha-beta we cannot use "partial" results with any confidence unless the full breadth of the tree has been searched**
  - So, we could be conservative and set a conservative depth-limit which guarantees that we will find a move in time < T
    - disadvantage is that we may finish early, could do more search

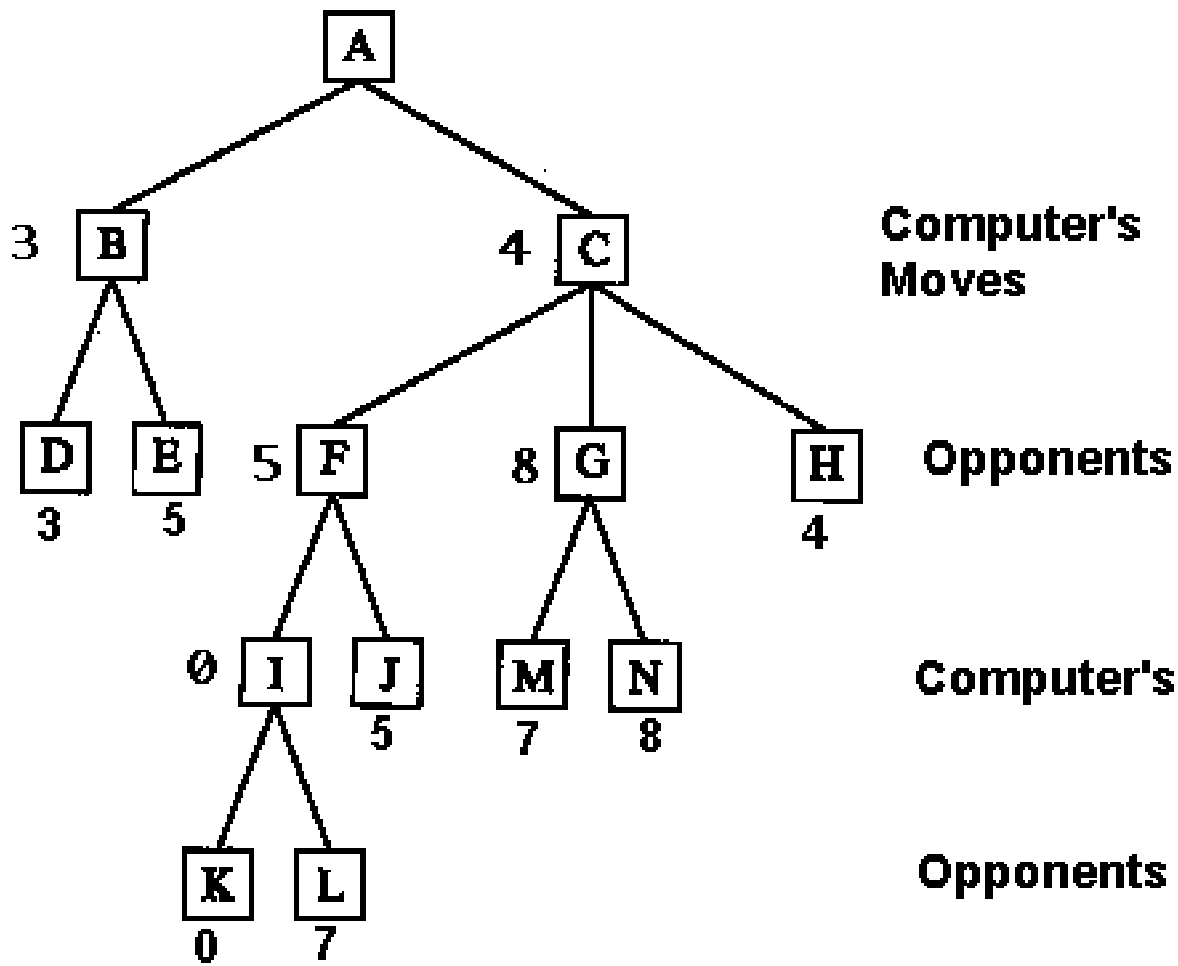
- **In practice, iterative deepening search (IDS) is used**
  - IDS runs depth-first search with an increasing depth-limit
  - when the clock runs out we use the solution found at the previous depth limit

Heuristics and Game Tree Search: limited horizon

- **The Horizon Effect**
  - sometimes there's a major "effect" (such as a piece being captured) which is just "below" the depth to which the tree has been expanded.
  - the computer cannot see that this major event could happen because it has a "limited horizon".
  - there are heuristics to try to follow certain branches more deeply to detect such important events
  - this helps to avoid catastrophic losses due to "short-sightedness"

- **Heuristics for Tree Exploration**
  - it may be better to explore some branches more deeply in the allotted time
  - various heuristics exist to identify "promising" branches

Deeper Game Trees



Eliminate Redundant Nodes

- **On average, each board position appears in the search tree approximately ~10^{150} / ~10^{40} ≈ 10^{100} times.**

  => Vastly redundant search effort.

- **Can't remember all nodes (too many).**

  => Can't eliminate all redundant nodes.

- **However, some short move sequences provably lead to a redundant position.**
  - These can be deleted dynamically with no memory cost

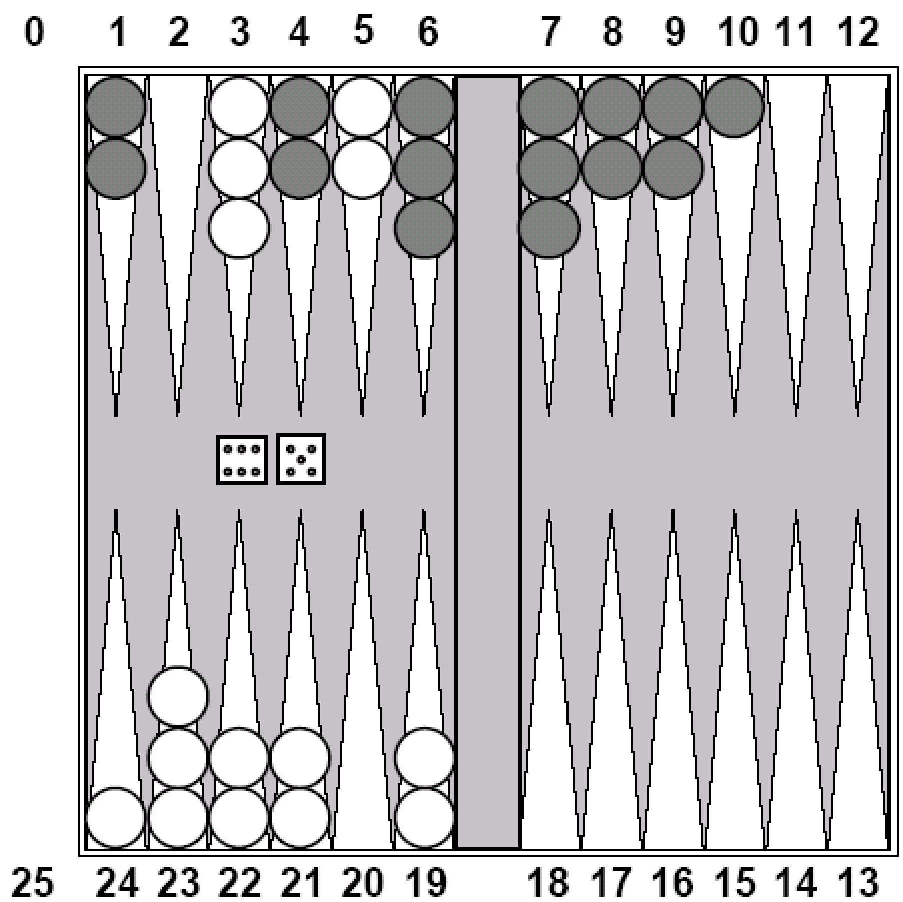
- **Example:**

  1. P-QR4 P-QR4; 2. P-KR4 P-KR4

  leads to the same position as

  1. P-QR4 P-KR4; 2. P-KR4 P-QR4

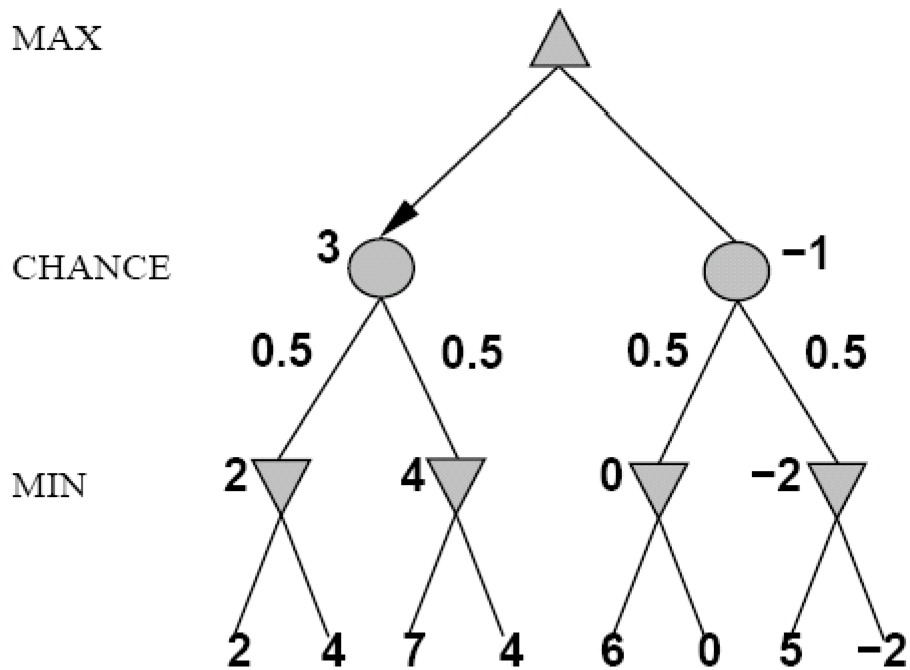
Nondeterministic games: backgammon



Nondeterministic games in general

In nondeterministic games, chance introduced by dice, card-shuffling

Simplified example with coin-flipping:



Algorithm for nondeterministic games

Expectiminimax gives perfect play

Just like Minimax, except we must also handle chance nodes:

```
. . .
if state is a Max node then
    return the highest ExpectiMinimax-Value of Successors(state)
if state is a Min node then
    return the lowest ExpectiMinimax-Value of Successors(state)
if state is a chance node then
    return average of ExpectiMinimax-Value of Successors(state)
. . .
```

The State of Play

- **Checkers:**
  - Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994.

- **Chess:**
  - Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997.

- **Othello:**
  - human champions refuse to compete against computers: they are too good.

- **Go:**
  - human champions refuse to compete against computers: they are too bad
  - $b > 300$ (!)

- **See (e.g.) http://www.cs.ualberta.ca/~games/ for more information**

The University of Alberta GAMES Group

Game-playing,
Analytical methods,
Minimax search and
Empirical
Studies

Announcements

- Weekly **GAMES group meetings** are from 4-5pm on Thursdays at CSC333. You can check the schedule here.
- The University of Alberta GAMES Group has an **opening for a postdoctoral fellow** in the area of Artificial Intelligence in Commercial (Video) Games. Check here for details.

Projects

Checkers *Chinook* is the official world checkers champion. Play	**Poker** *Poki* is the strongest poker AI in the world. Play	**Lines of Action** *YL* & *Mona* are two of the best LoA programs in the world. Play	**Hex** *Queenbee* is one of the best Hex programs in the world. Play
Go The Computer Go group has developed two programs, Explorer and NeuroGo.	**Real-Time Stategy** We are trying to apply AI to real-time stategy games.	**Othello** *Logistello* defeated the human world Othello champion, 6-0, in 1997. *Keyano* is another strong program.	**Sokoban** *Rolling Stone* pushes the boundaries of single agent search.
RoShamBo Home of the *International RoShamBo Programming Competition*	**Amazons** Three programs, and several theoretical contributions.	**Spades & Hearts** Spades & Hearts are the test beds for the research on multi-player games.	**Shogi** *ISshogi* has won the world computer Shogi championship many times (currently inactive).

Previous Projects: Chess | Awari (Play) | Chinese Chess (Play) | Post's Correspondence Problem | Domineering

Applied Research

Done

Deep Blue

- **1957: Herbert Simon**
  - “within 10 years a computer will beat the world chess champion”

- **1997: Deep Blue beats Kasparov**

- **Parallel machine with 30 processors for “software” and 480 VLSI processors for “hardware search”**

- **Searched 126 million nodes per second on average**
  - Generated up to 30 billion positions per move
  - Reached depth 14 routinely

- **Uses iterative-deepening alpha-beta search with transpositioning**
  - Can explore beyond depth-limit for interesting moves

Summary

- Game playing is best modeled as a search problem
- Game trees represent alternate computer/opponent moves
- Evaluation functions estimate the quality of a given board configuration for the Max player.
- Minimax is a procedure which chooses moves by assuming that the opponent will always choose the move which is best for them
- Alpha-Beta is a procedure which can prune large parts of the search tree and allow search to go deeper
- For many well-known games, computer algorithms based on heuristic search match or out-perform human world experts.
- Reading:R&N Chapter 6 (3^{rd} ed.), Chapter 5 (2^{nd} ed.).
  - For Thursday: R&N, “Constraint Satisfaction Problems”
    - Ch. 6 (3rd ed.); Ch 5 (2nd ed.)