Set 4: Game-Playing

ICS 271 Fall 2012

Overview

- **Computer programs which play 2-player games**
  - game-playing as search
  - with the complication of an opponent

- **General principles of game-playing and search**
  - evaluation functions
  - minimax principle
  - alpha-beta-pruning
  - heuristic techniques

- **Status of Game-Playing Systems**
  - in chess, checkers, backgammon, Othello, etc, computers routinely defeat leading world players

- **Motivation: multiagent competitive environments**
  - think of “nature” as an opponent
  - economics, war-gaming, medical drug treatment

Solving 2-players Games

- **Two players, perfect information**
- **Examples: e.g., chess, checkers, tic-tac-toe**
- **Configuration of the board = unique arrangement of “pieces”**
- **Statement of Game as a Search Problem:**
  - **States** = board configurations
  - **Operators** = legal moves. The transition model
  - **Initial State** = current configuration
  - **Goal** = winning configuration
  - **payoff function (utility)**= gives numerical value of outcome of the game
- **A working example: Grundy's game**
  - Given a set of coins, a player takes a set and divides it into two unequal sets. The player who plays last, looses.
  - What is a state? Moves? Goal?

Grundy's game - special case of nim

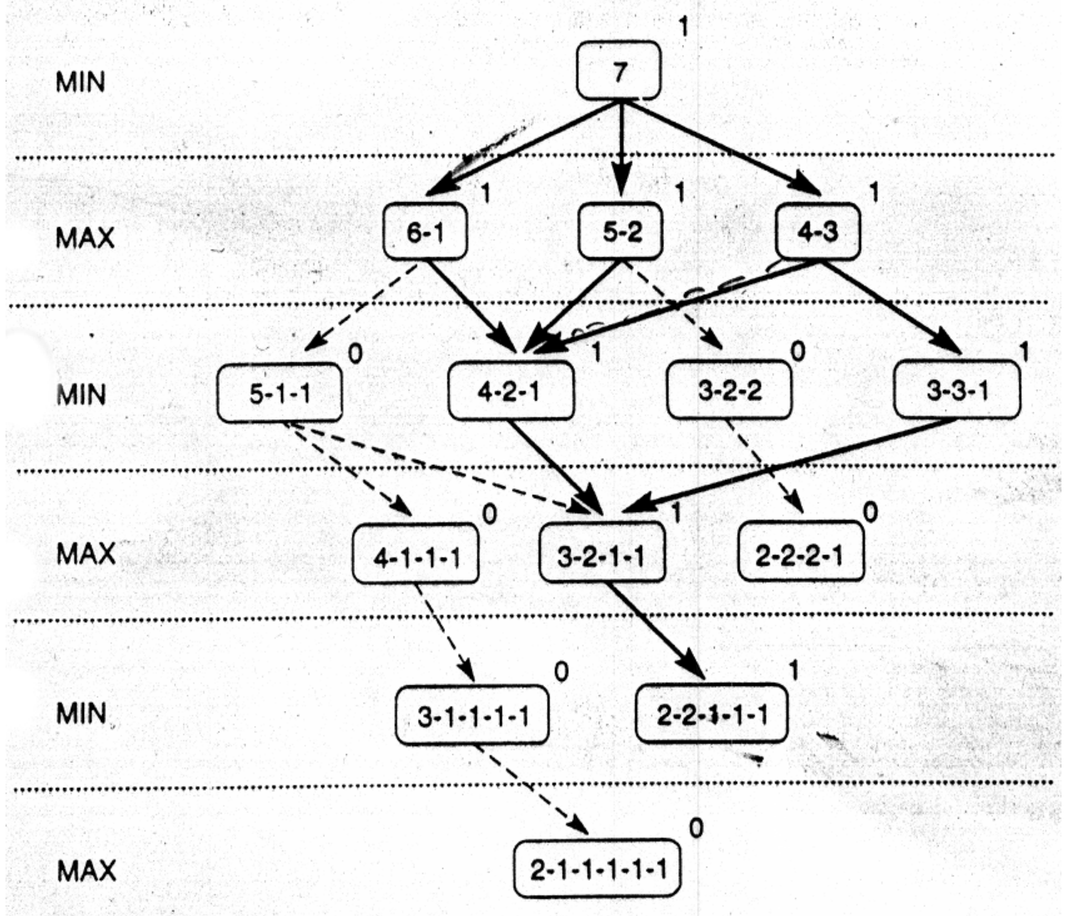


Figure 4.14 Exhaustive minimax for the game of nim. Bold lines indicate forced win for MAX. Each node is marked with its derived value (0 or 1) under minimax.

Types of games

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

Game Trees: Tic-tac-toe

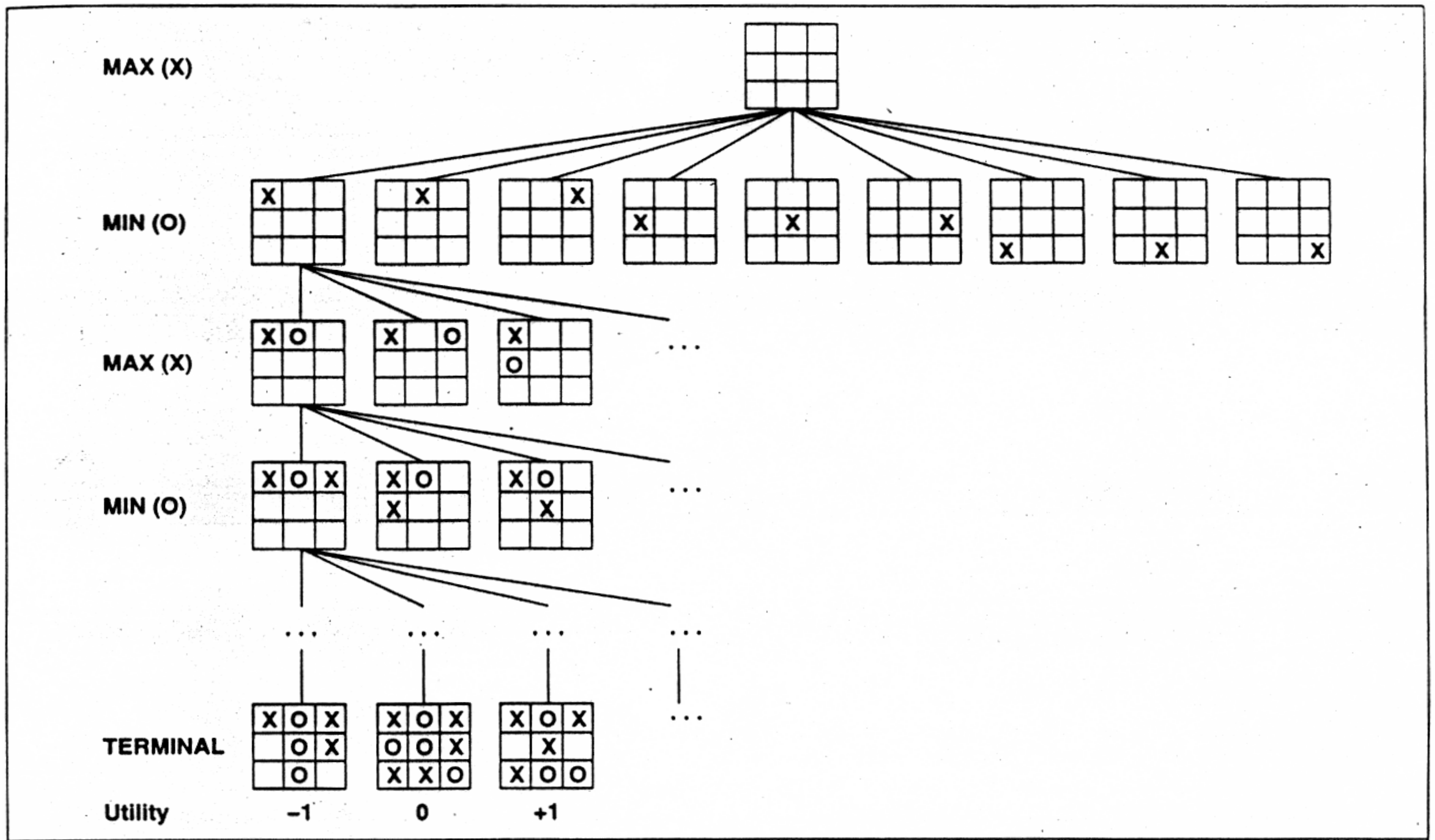


Figure 5.1 A (partial) search tree for the game of Tic-Tac-Toe. The top node is the initial state, and MAX moves first, placing an X in some square. We show part of the search tree, giving alternating moves by MIN (O) and MAX, until we eventually reach terminal states, which can be assigned utilities according to the rules of the game.

Game Trees

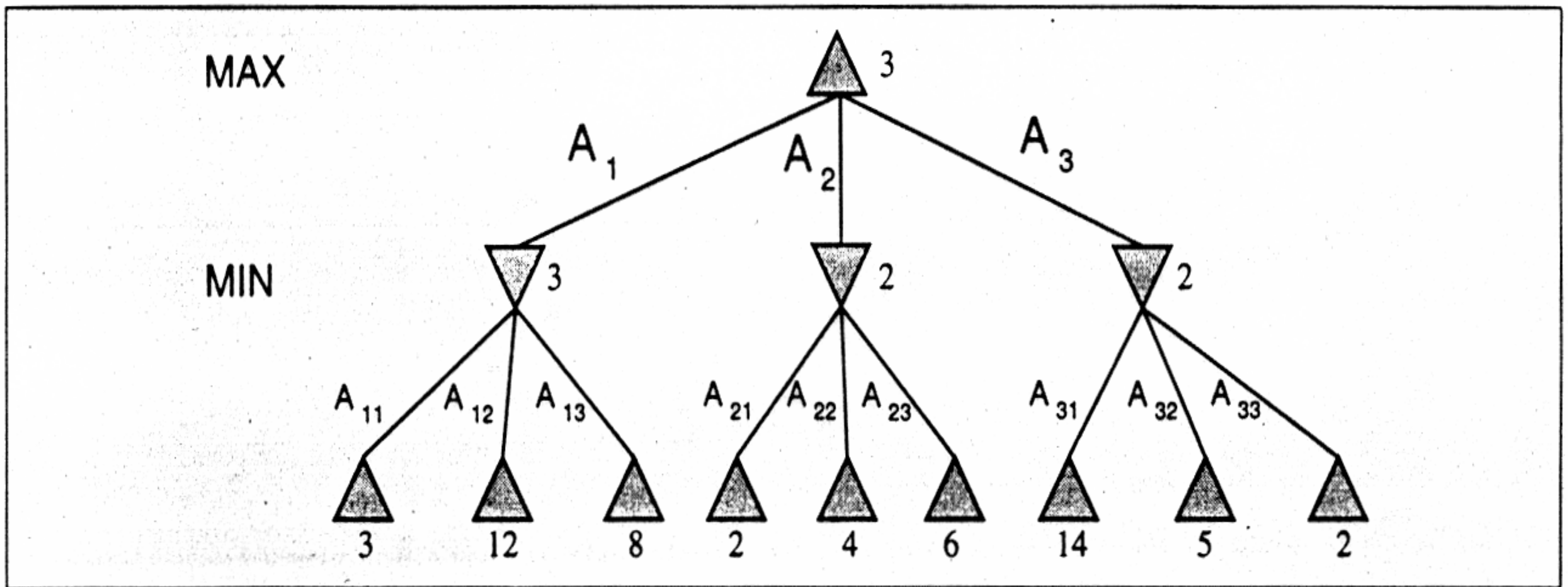


Figure 5.2 A two-ply game tree as generated by the minimax algorithm. The △ nodes are moves by MAX and the ▽ nodes are moves by MIN. The terminal nodes show the utility value for MAX computed by the utility function (i.e., by the rules of the game), whereas the utilities of the other nodes are computed 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}.

The Minimax Algorithm

- Designed to find the optimal strategy for Max and find best move
- Explores the game (and/or) search tree in a depth-first search manner
- The search space is the game-tree.
- We wish to find an optimal strategy, or just optimal first move.

Brute-force:

- 1. Generate the whole game tree to leaves
- 2. Apply utility (payoff) function to leaves
- 3. Back-up values from leaves toward the root:
  - a Max node computes the max of its child values
  - a Min node computes the Min of its child values
- 4. When value reaches the root: choose max value and the corresponding move.

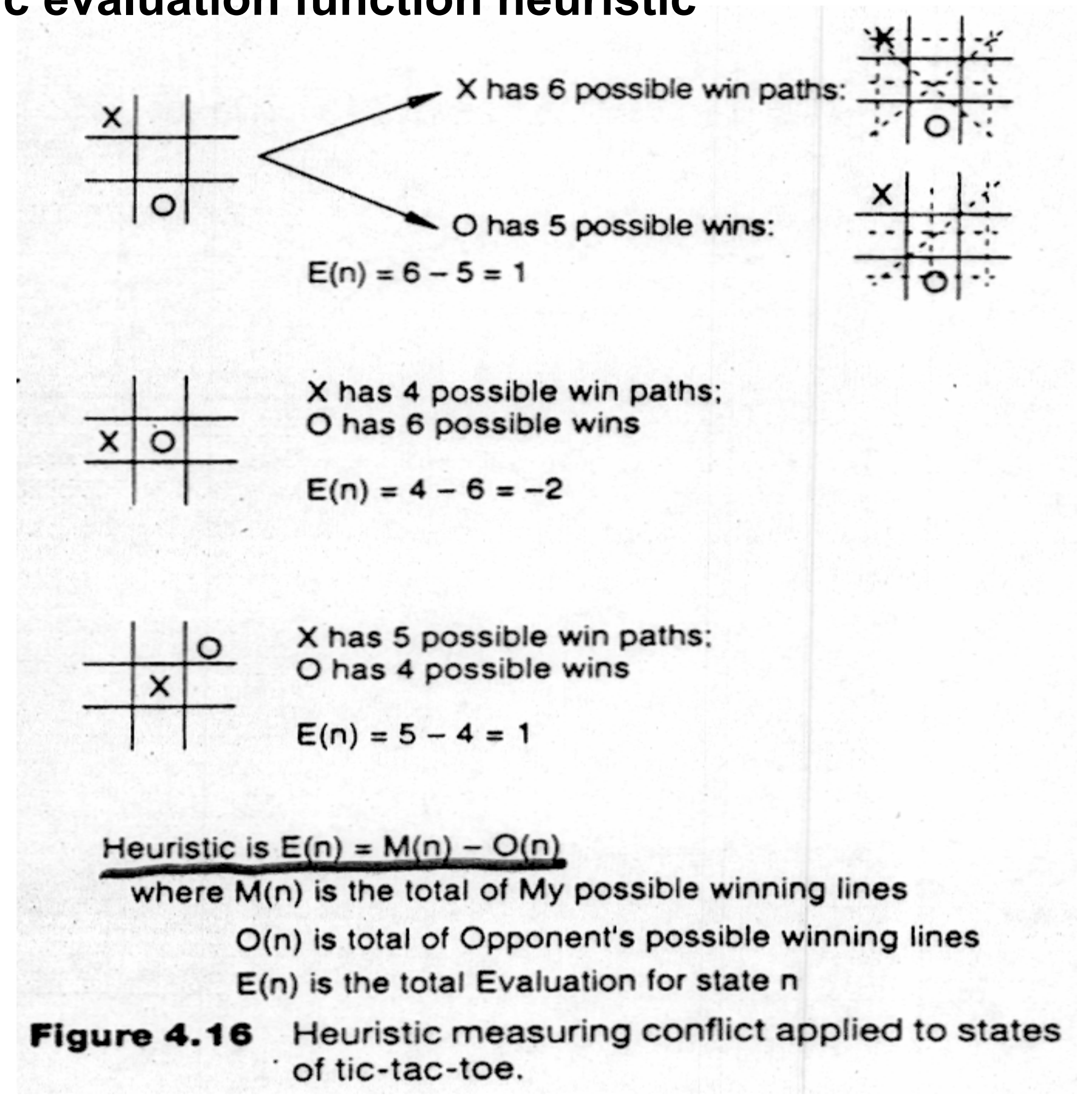
Minimax:

1. Search the game-tree in a DFS manner to find the value of the root.

- **However: It is impossible to develop the whole search tree. Instead develop part of the tree and evaluate promise of leaves using a static evaluation function.**

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

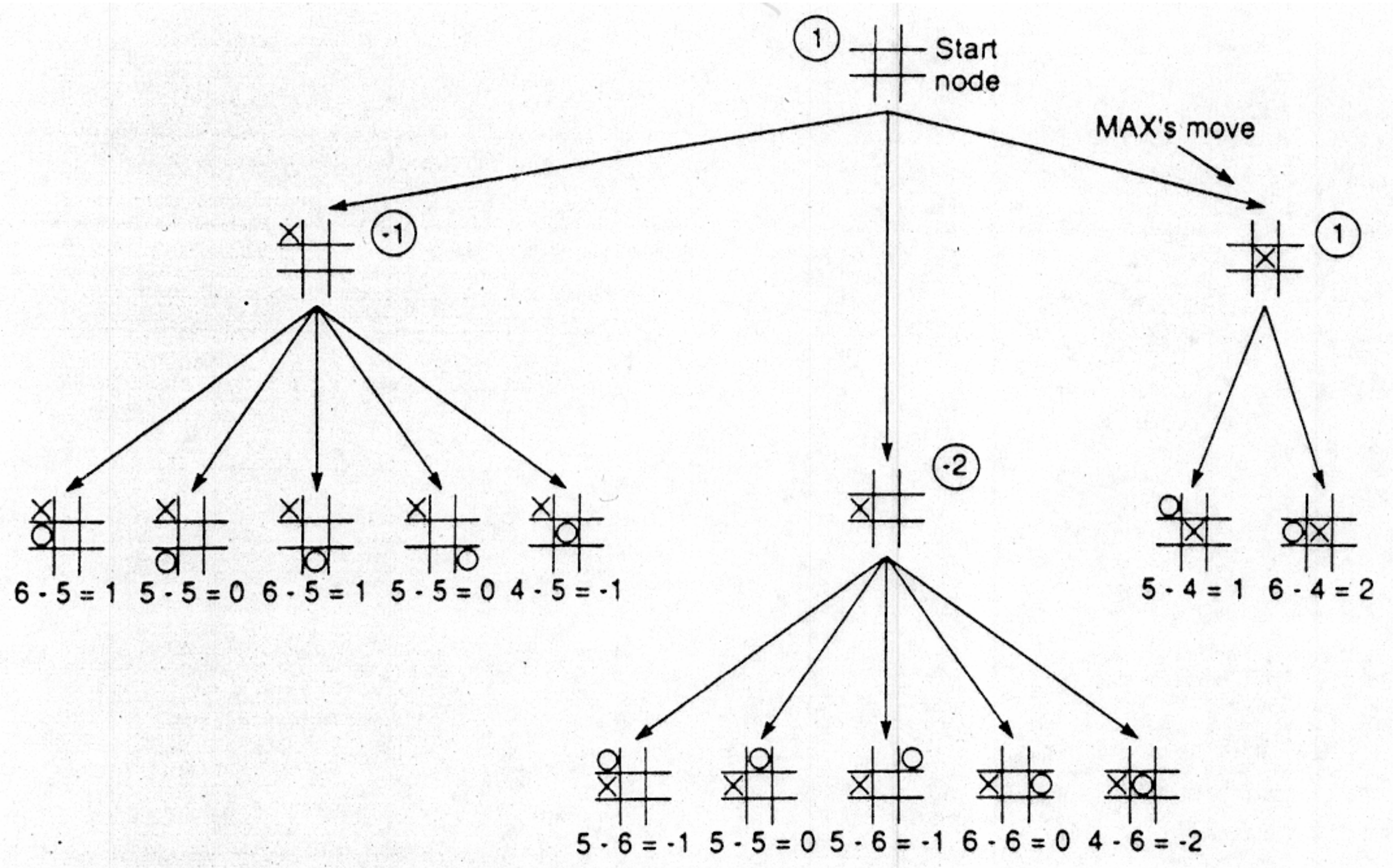


Figure 4.17 Two-ply minimax applied to the opening move of tic-tac-toe.

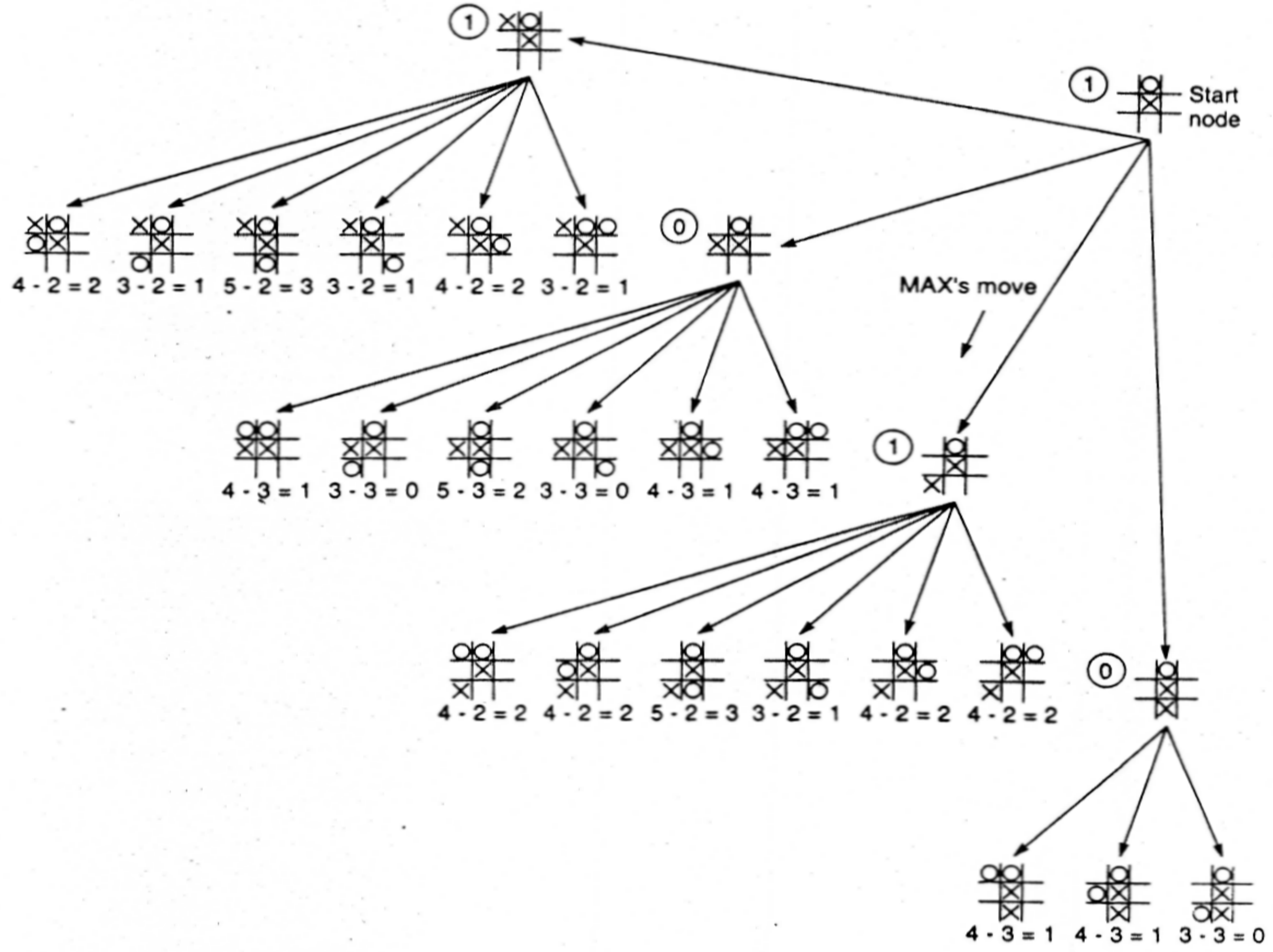


Figure 4.18 Two-ply minimax applied to **X**'s second move of tic-tac-toe.

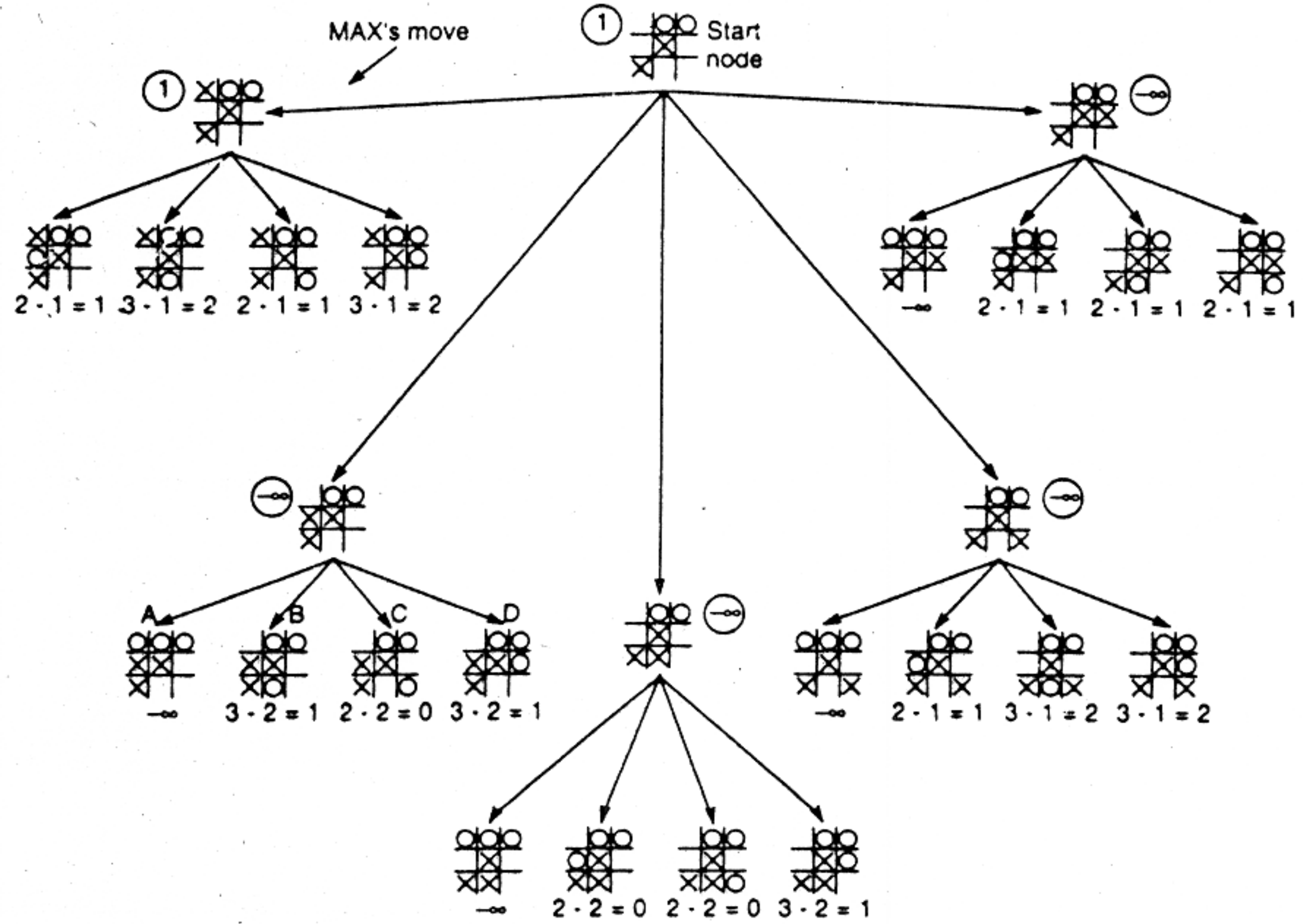


Figure 4.19 Two-ply minimax applied to **X**'s move near end game.

Properties of Minimax

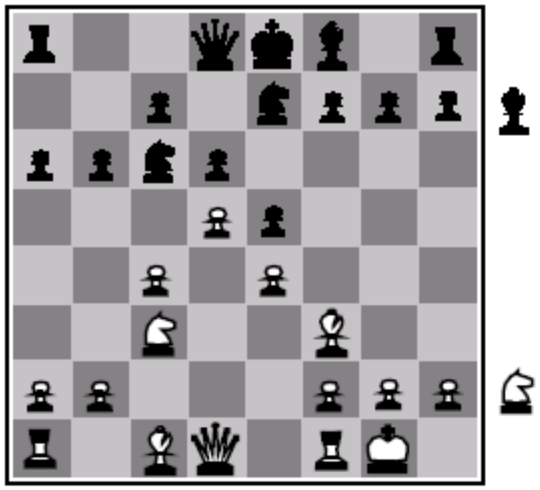
- Complete?
  - Yes (if tree is finite)
- Complete for Optimality?
  - Yes (against an optimal opponent)
- Time complexity? O(b^m), b is the brunching degree, m is depth of tree.
- Space complexity? O(bm) (depth-first exploration)

- For chess, for "reasonable" games exact solution completely infeasible
  - Chess:
    - b ~ 35 (average branching factor)
    - d ~ 100 (depth of game tree for typical game)
    - $b^d \sim 35^{100} \sim 10^{154}$ nodes!!
  - Tic-Tac-Toe
    - ~5 legal moves, total of 9 moves
    - 5^9 = 1,953,125
    - 9! = 362,880 (Computer goes first)
    - 8! = 40,320 (Computer goes second)

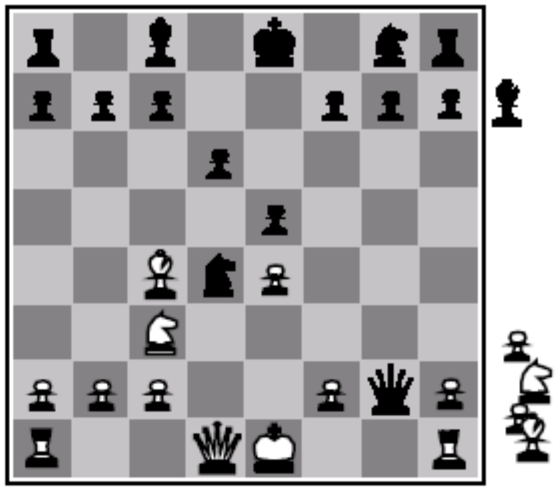
Static (Heuristic) Evaluation Functions

- **An Evaluation Function:**
  - Estimates how good the current board configuration is for a player.
  - Typically, one figures how good it is for the player, and how good it is for the opponent, and subtracts the opponents score from the player.
  - 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**
- **Example:**
  - Evaluating chess boards,
  - Checkers
  - Tic-tac-toe

Evaluation functions


Black to move

White slightly better


White to move

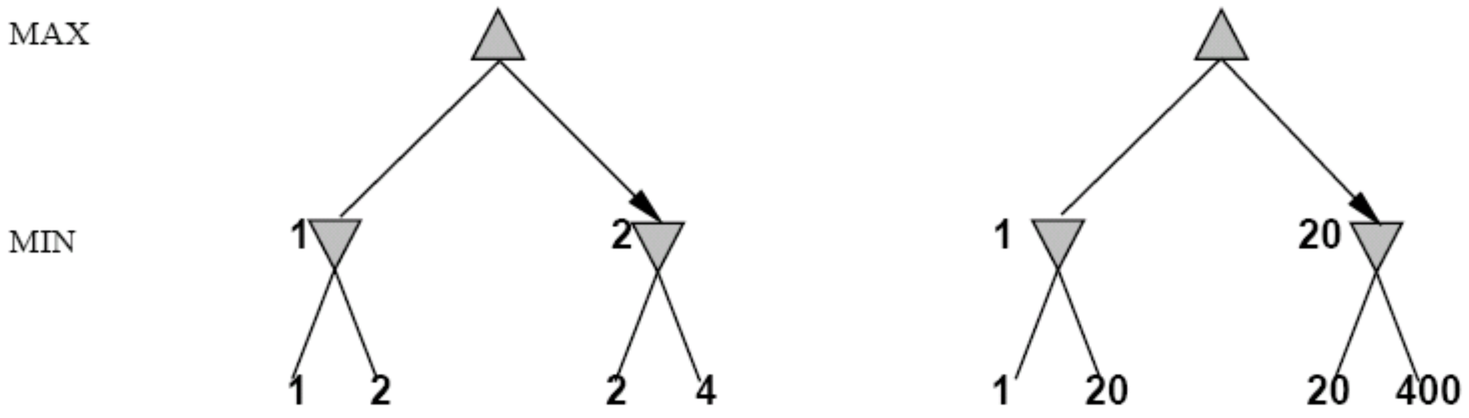
Black winning

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.

Digression: Exact values don't matter

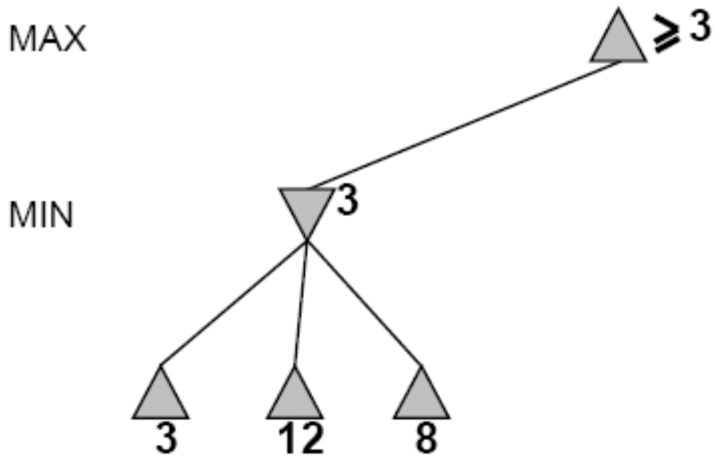


Behaviour is preserved under any *monotonic* transformation of EVAL

Only the order matters:

payoff in deterministic games acts as an *ordinal utility* function

Can we prune depth-first search?



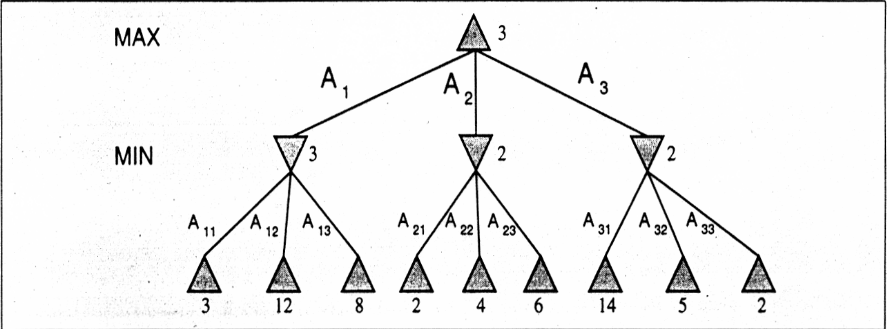
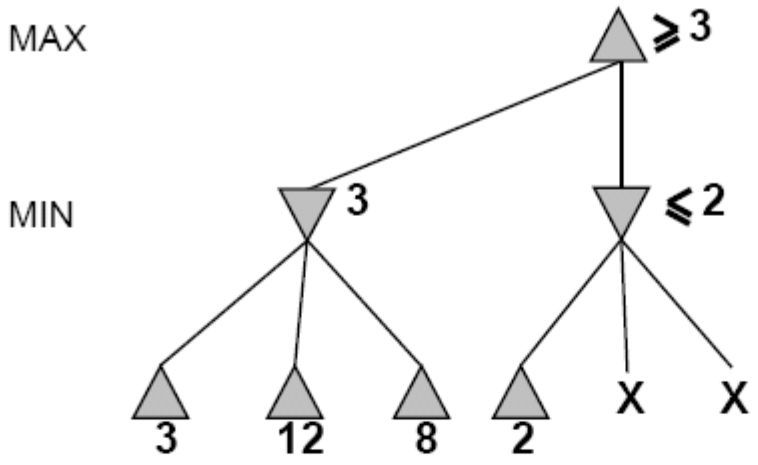


Figure 5.2 A two-ply game tree as generated by the minimax algorithm. The △ nodes are moves by MAX and the ▽ nodes are moves by MIN. The terminal nodes show the utility value for MAX computed by the utility function (i.e., by the rules of the game), whereas the utilities of the other nodes are computed 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}.

α–β pruning example



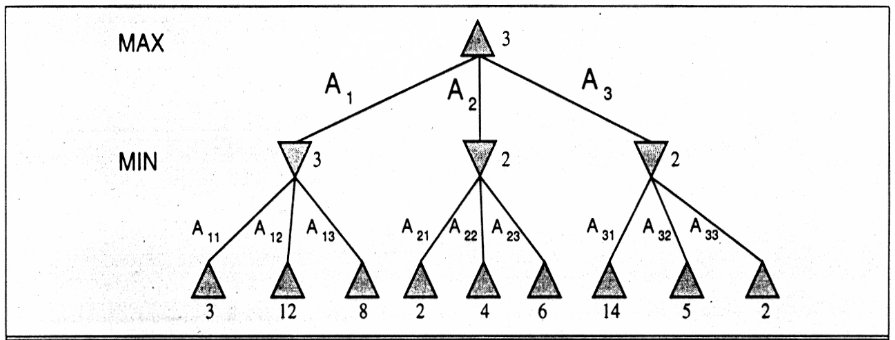


Figure 5.2 A two-ply game tree as generated by the minimax algorithm. The △ nodes are moves by MAX and the ▽ nodes are moves by MIN. The terminal nodes show the utility value for MAX computed by the utility function (i.e., by the rules of the game), whereas the utilities of the other nodes are computed 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}.

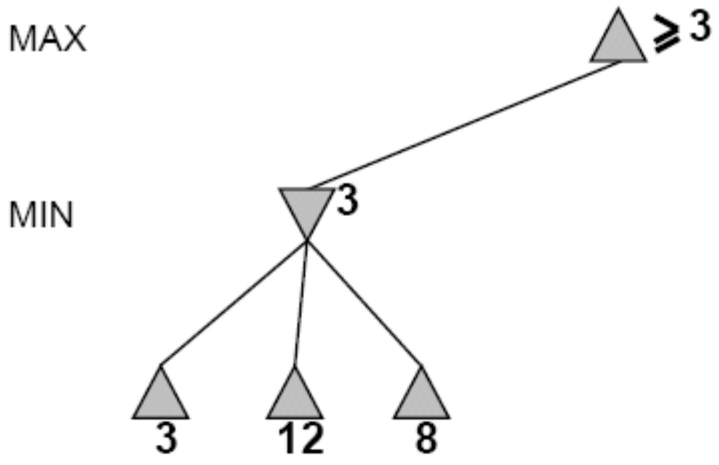
Alpha Beta Procedure

- **Idea:**
  - Do depth first search to generate partial game tree,
  - Give static evaluation function to leaves,
  - Compute bound on internal nodes.
- **Alpha, Beta bounds:**
  - Alpha value for max node means that max real value is at least alpha.
  - Beta for min node means that min can guarantee a value below beta.
- **Computation:**
  - Alpha of a max node is the maximum value of its seen children.
  - Beta of a min node is the minimum value seen of its child node .

When to Prune

- **Pruning**
  - Below a Min node whose beta value is lower than or equal to the alpha value of its ancestors.
  - Below a Max node having an alpha value greater than or equal to the beta value of any of its Min nodes ancestors.

Can we prune depth-first search?



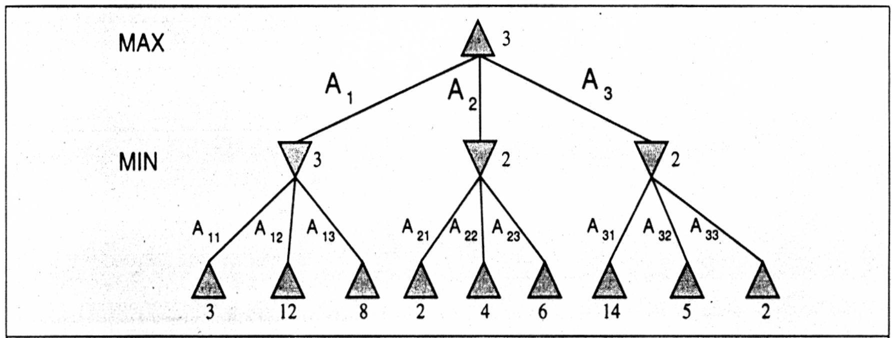
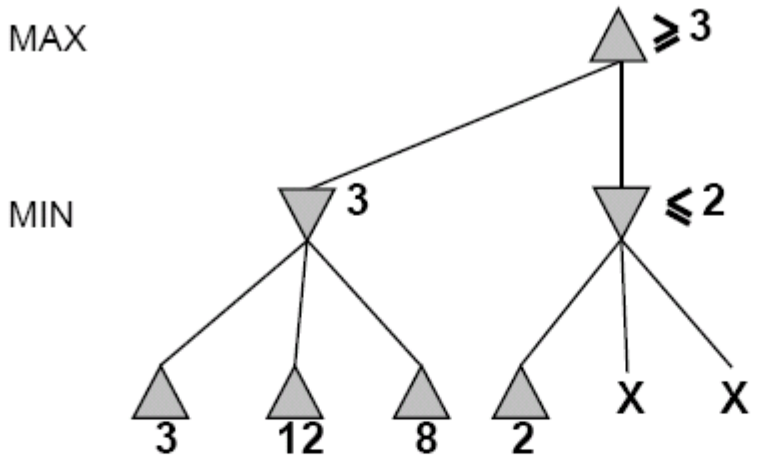


Figure 5.2 A two-ply game tree as generated by the minimax algorithm. The △ nodes are moves by MAX and the ▽ nodes are moves by MIN. The terminal nodes show the utility value for MAX computed by the utility function (i.e., by the rules of the game), whereas the utilities of the other nodes are computed 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}.

α–β pruning example



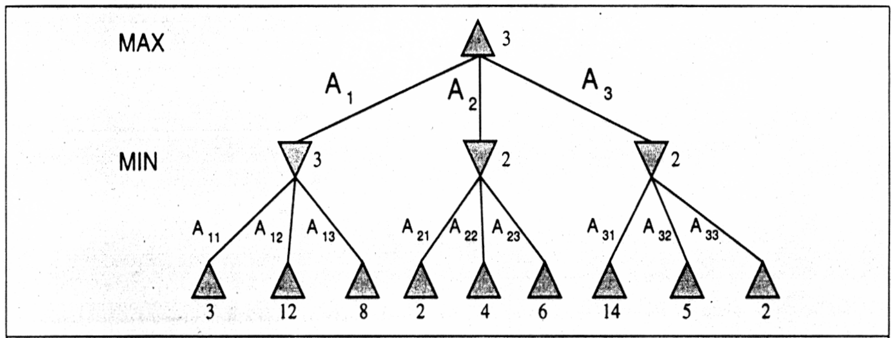
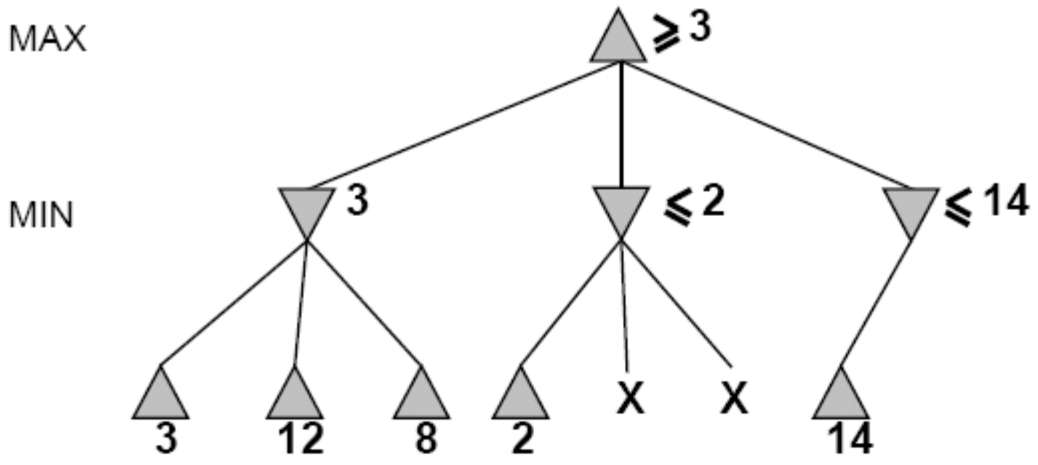


Figure 5.2 A two-ply game tree as generated by the minimax algorithm. The $\triangle$ nodes are moves by MAX and the ∇ nodes are moves by MIN. The terminal nodes show the utility value for MAX computed by the utility function (i.e., by the rules of the game), whereas the utilities of the other nodes 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}.

α–β pruning example



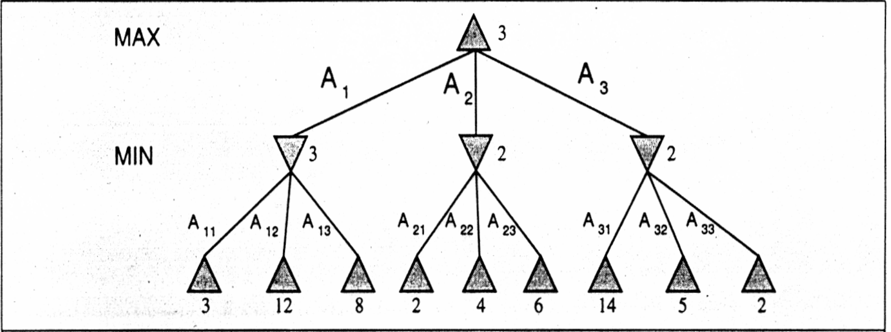
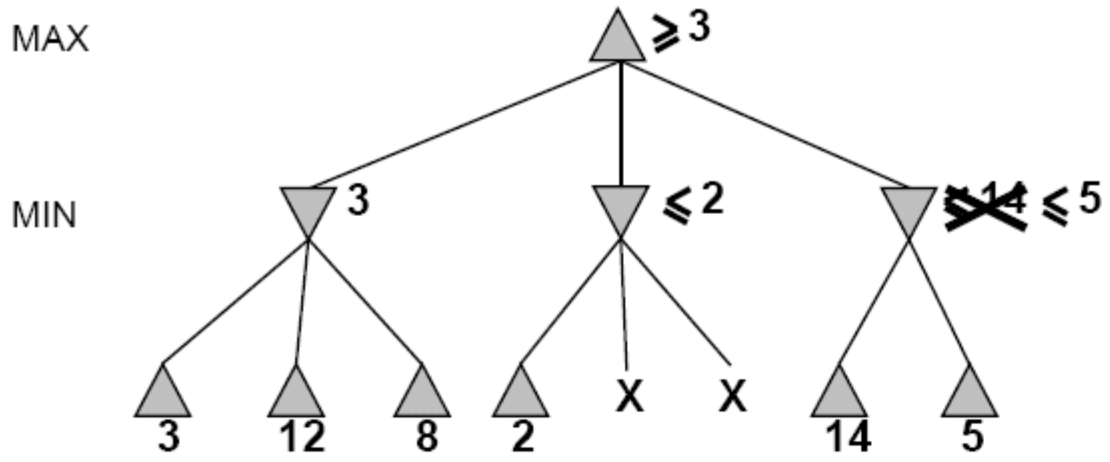


Figure 5.2 A two-ply game tree as generated by the minimax algorithm. The △ nodes are moves by MAX and the ▽ nodes are moves by MIN. The terminal nodes show the utility value for MAX computed by the utility function (i.e., by the rules of the game), whereas the utilities of the other nodes are computed 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}.

α–β pruning example



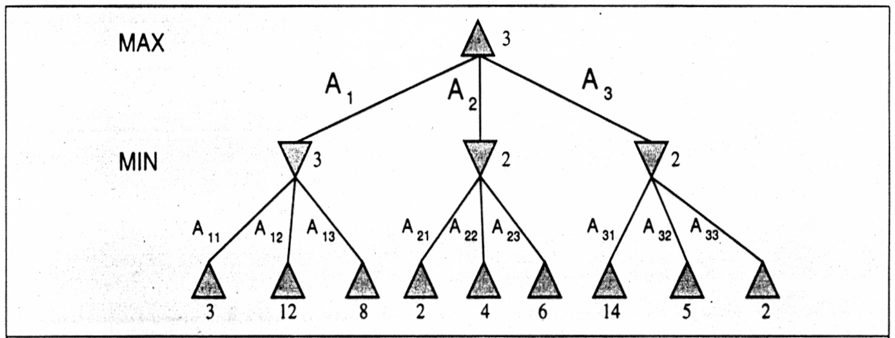
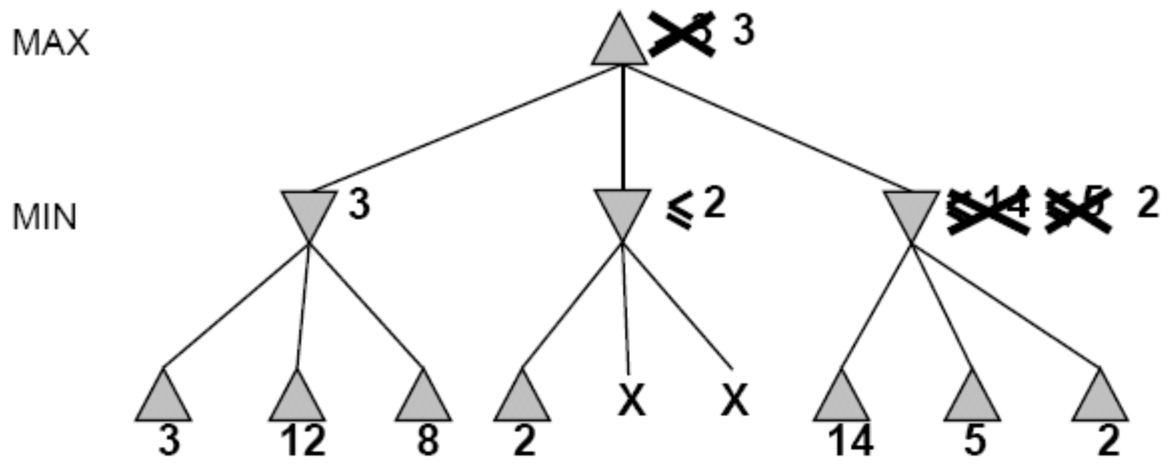


Figure 5.2 A two-ply game tree as generated by the minimax algorithm. The △ nodes are moves by MAX and the ▽ nodes are moves by MIN. The terminal nodes show the utility value for MAX computed by the utility function (i.e., by the rules of the game), whereas the utilities of the other nodes are computed 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}.

α–β pruning example



Alpha Beta Procedure

- **Idea:**
  - Do depth first search to generate partial game tree,
  - Give static evaluation function to leaves,
  - Compute bound on internal nodes.
- **Alpha, Beta bounds:**
  - Alpha value for max node means that max real value is at least alpha.
  - Beta for min node means that min can guarantee a value below beta.
- **Computation:**
  - Alpha of a max node is the maximum value of its seen children.
  - Beta of a min node is the minimum value seen of its child node .
- **Graph-search using Transposition tables:**
  - Even in depth-first search we can store the result of an evaluation in a hash table of previously seen positions. Like the notion of “explored” list in graph-search.

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 alternative (i.e., evaluated first)
  - Assume we have an optimal strategy and we order it with best move for max is first to the left.
  - In practice, performance is closer to best rather than worst-case
- **Alpha/beta best case is $O(b^{(d/2)})$ rather than $O(b^d)$**
  - This is the same as having a branching factor of sqrt(b),
    - since $(\text{sqrt}(b))^d = b^{(d/2)}$ (i.e., we have effectively gone from b to square root of b)
  - In chess go from b ~ 35 to b ~ 6
    - permiting much deeper search in the same amount of time
  - IN practice it is often $b^{(d3/2)}$

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

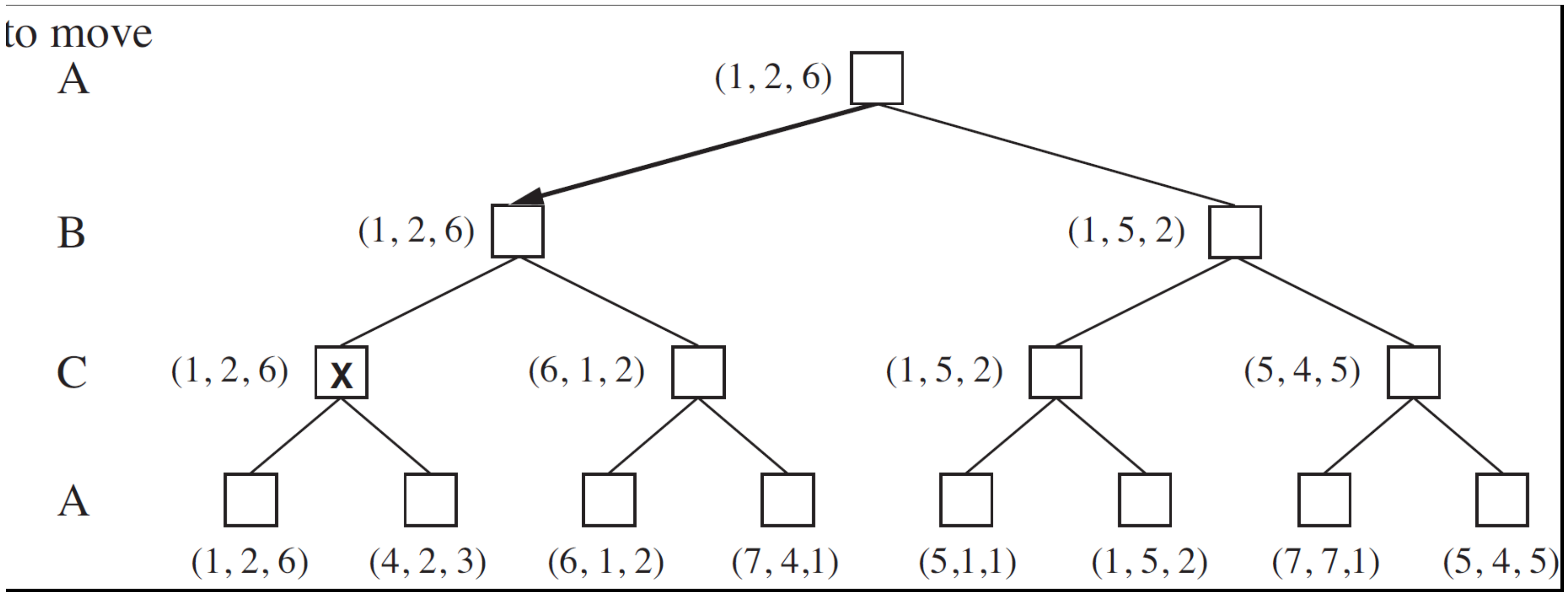
- **In practice, iterative deepening search (IDS) is used**
  - IDS runs depth-first search with an increasing depth-limit
  - when the clock runs out we use the solution found at the previous depth limit

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
- **Search versus lookup tables**
  - (e.g., chess endgames)

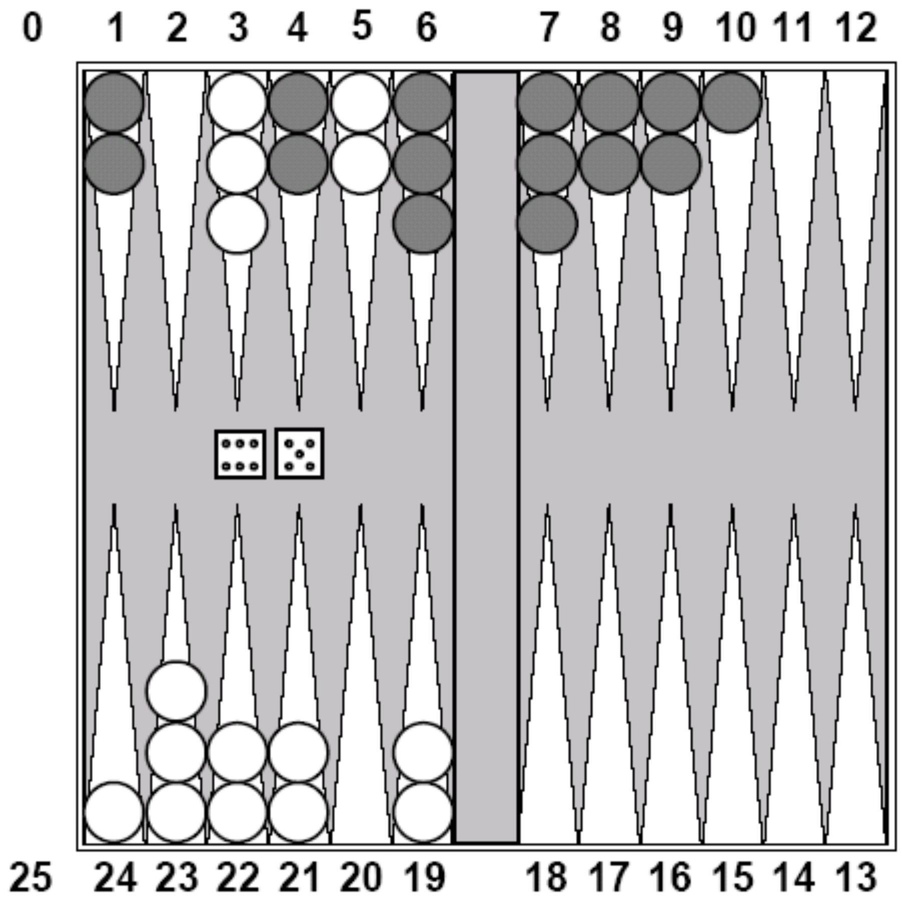
Multiplayer Games



- Multiplayer games often involve alliances: If A and B are in a weak position they can collaborateAnd act against C

- If games are not zero-sum, collaboration can also occur in two-game plays: if (1000,1000_ Is a best payoff for both, then they will cooperate towards getting there and not towards minimax value.

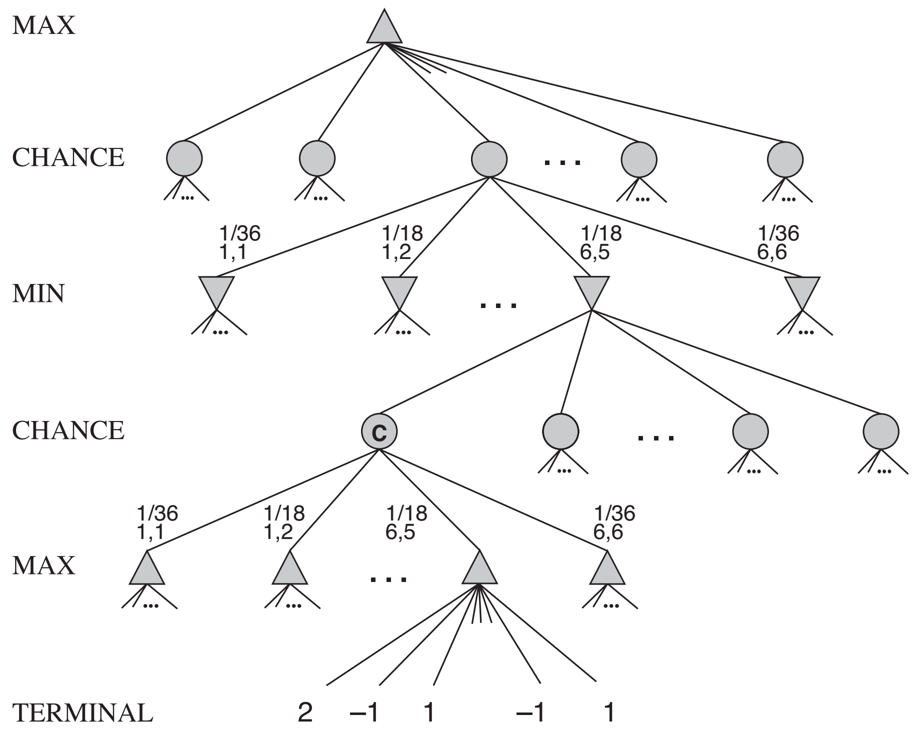
Nondeterministic games: backgammon



In real life there are many unpredictable external events

A game tree in Backgammon must include chance nodes

Schematic Game Tree for Backgammon Position



- How do we evaluate good move?
- By expected utility leading to expected minimax
- Utility for max is highest expected value of child nodes
- Utility of min-nodes is the lowest expected value of child nodes
- Chance node take the expected value of their child nodes.

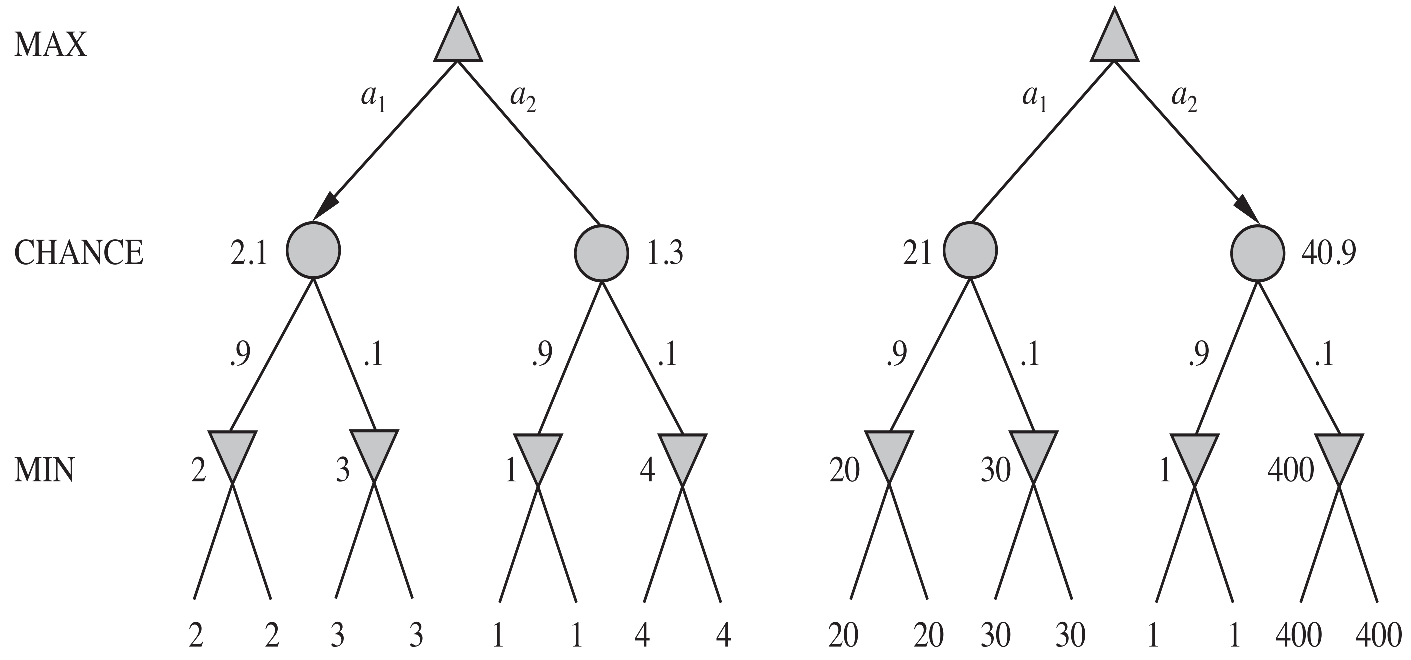
Algorithm for nondeterministic games

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)
...
```

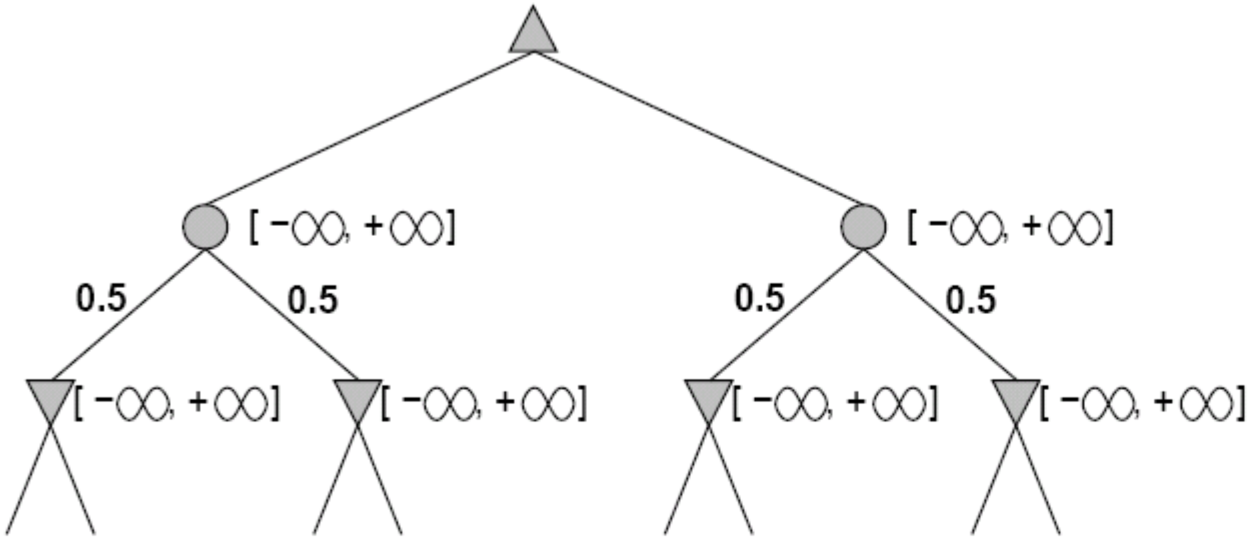
Evaluation functions for stochastic games



- Sensitivity to the absolute values
- The evaluation function should related to the probability of winning from a position, or to the expected utility from the position
- Complexity: O((bn)^m) where m is the depth and n is branching of chance nodes

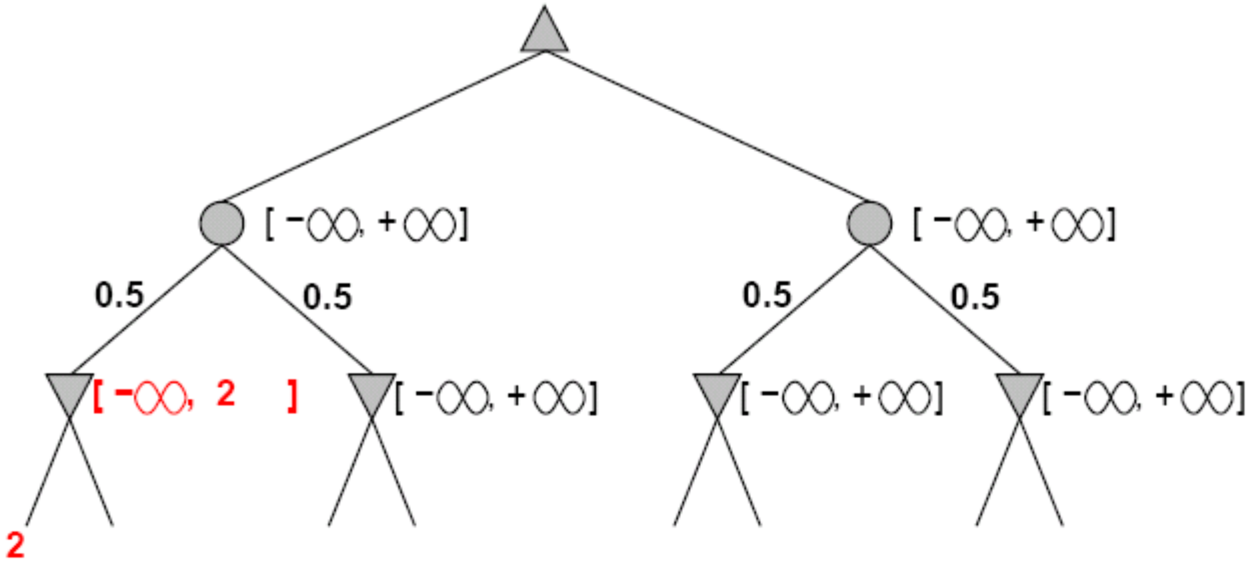
Pruning in nondeterministic game trees

A version of α-β pruning is possible:



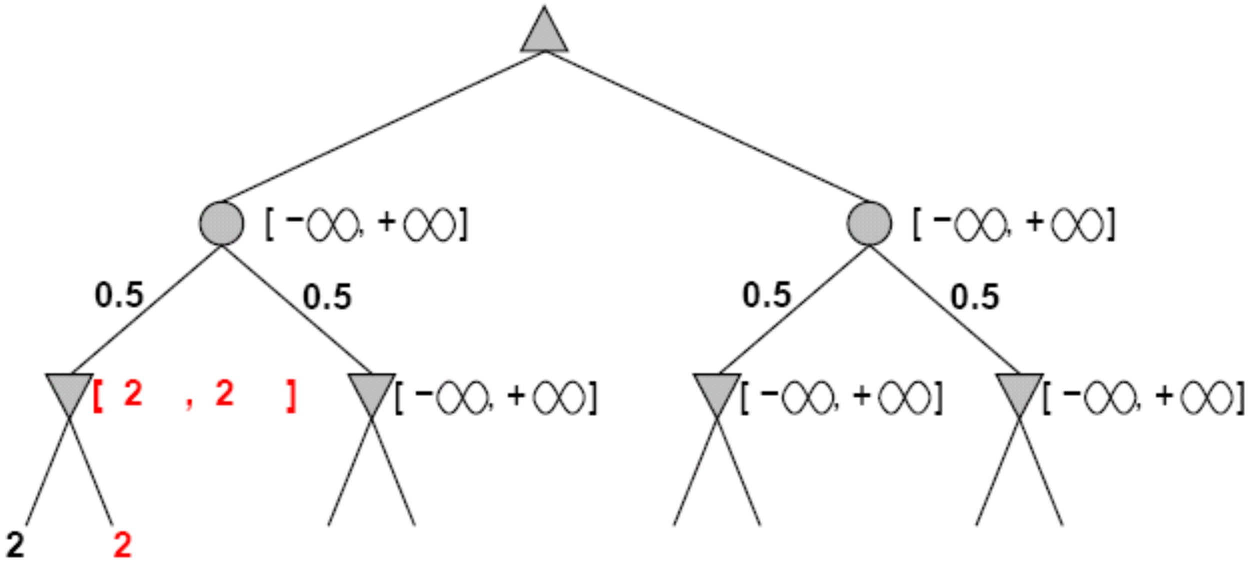
Pruning in nondeterministic game trees

A version of α-β pruning is possible:



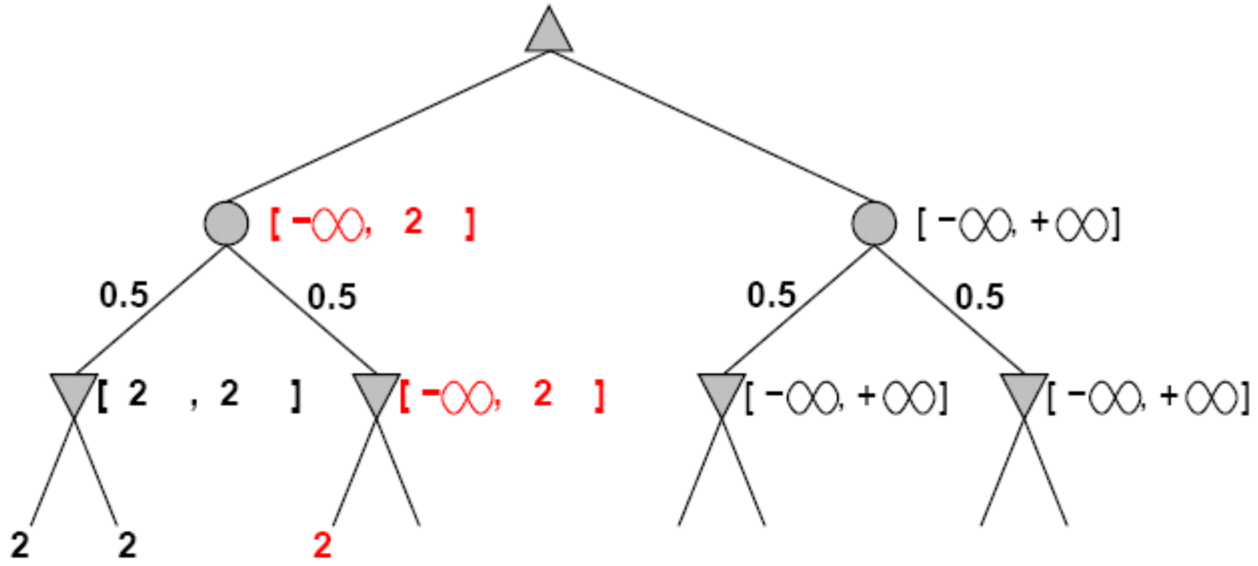
Pruning in nondeterministic game trees

A version of α-β pruning is possible:



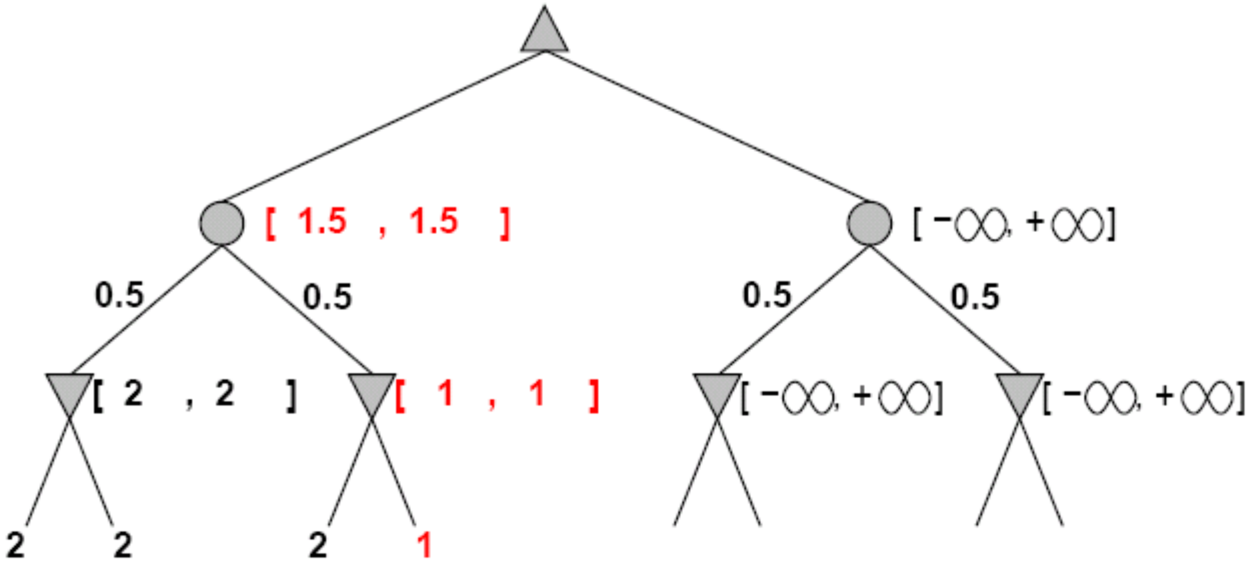
Pruning in nondeterministic game trees

A version of α-β pruning is possible:



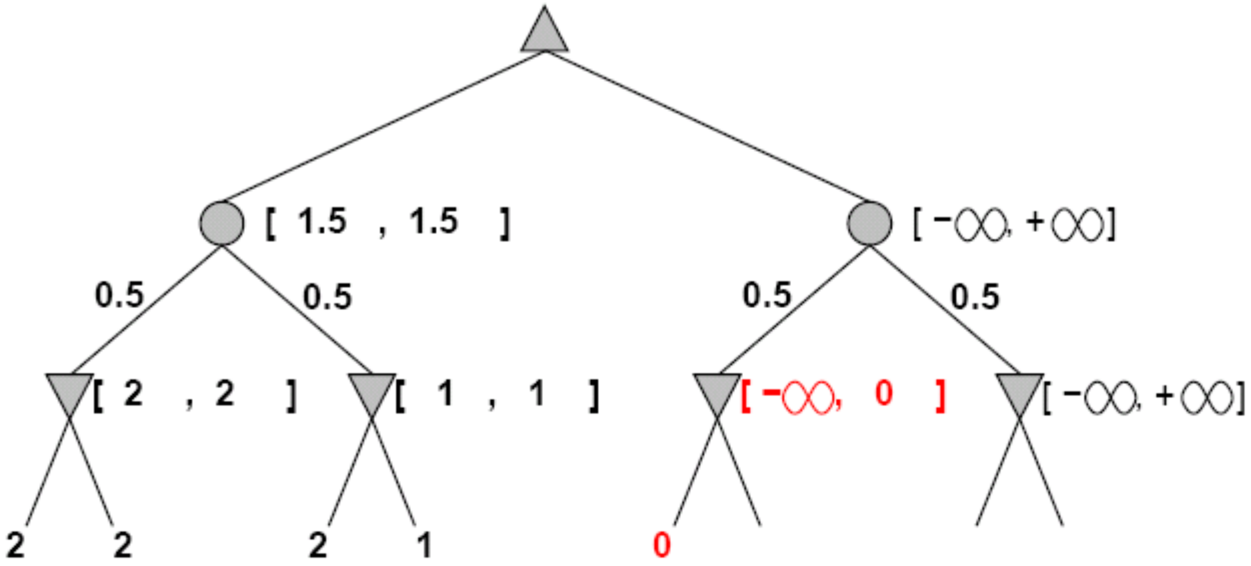
Pruning in nondeterministic game trees

A version of α-β pruning is possible:



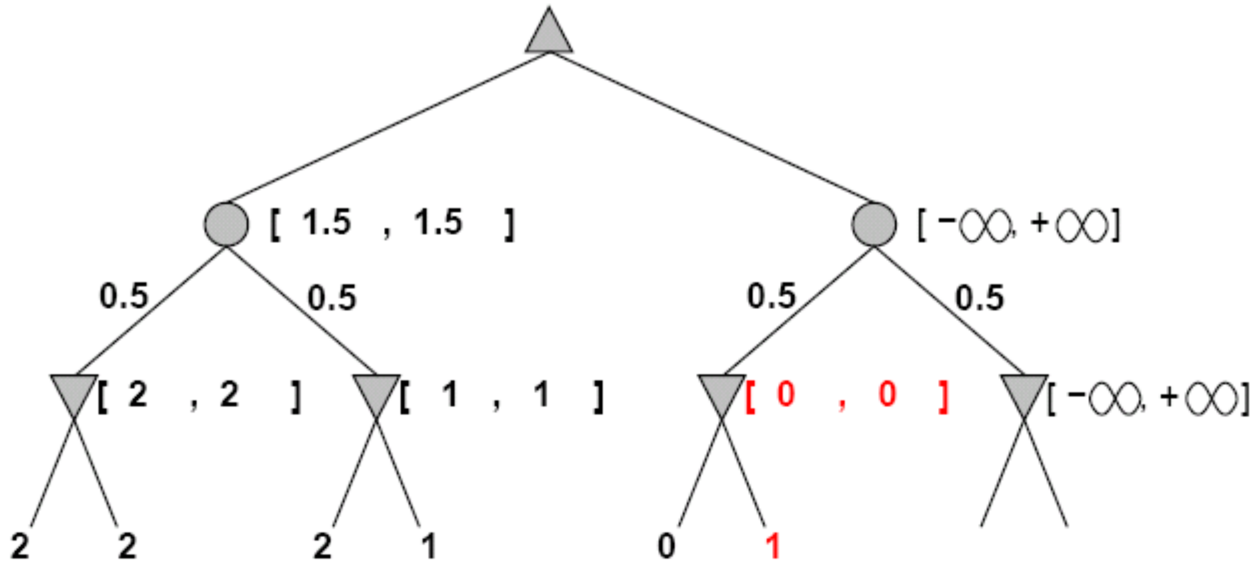
Pruning in nondeterministic game trees

A version of α-β pruning is possible:



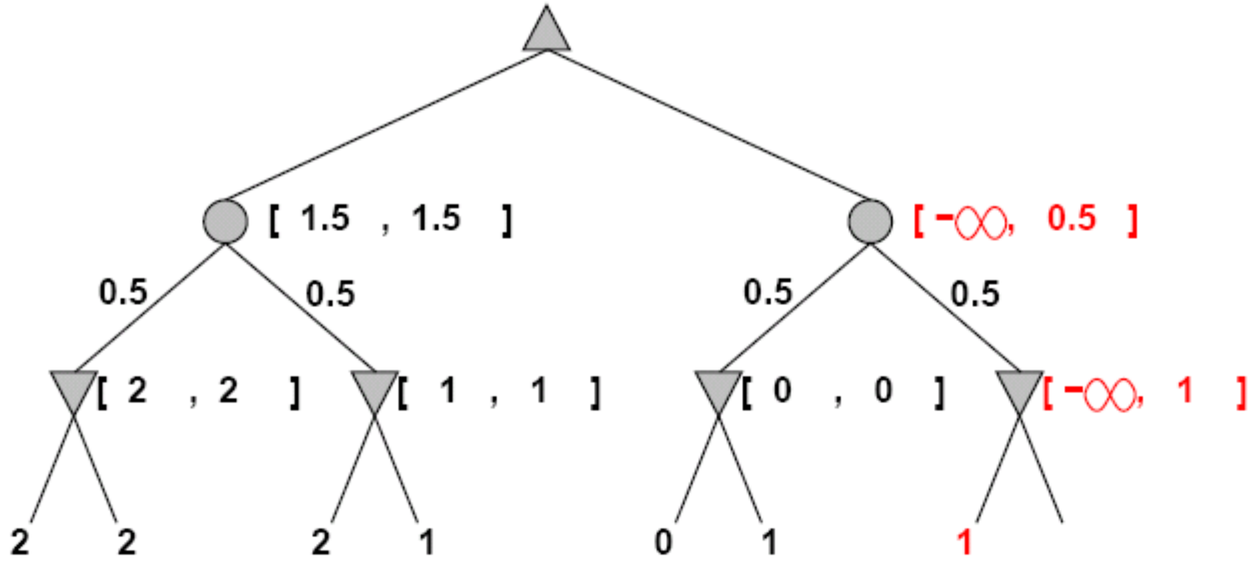
Pruning in nondeterministic game trees

A version of α-β pruning is possible:



Pruning in nondeterministic game trees

A version of α-β pruning is possible:



An alternative: Monte Carlo simulations:
Play thousands of games of the program against itself
Using random dice rolls. Record the percentage of win
From a position.

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.
- Stochastic games
- Partially observable games
- Reading:R&N Chapter 5.

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.