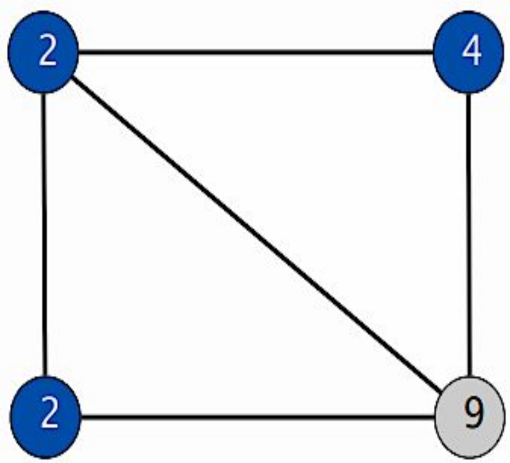
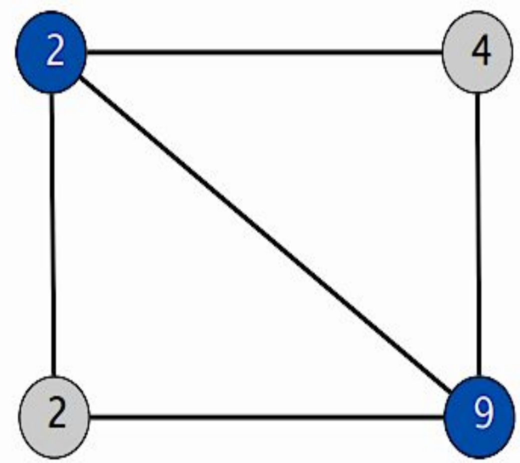
Artificial Intelligence

Constrained minimization example

Weighted vertex cover. Given a graph G with vertex weights, find a vertex cover of minimum weight.



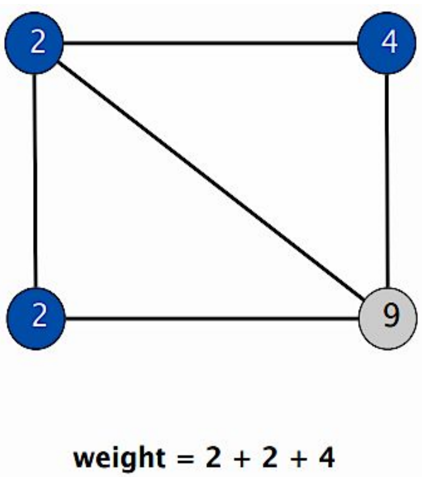
weight = 2 + 2 + 4

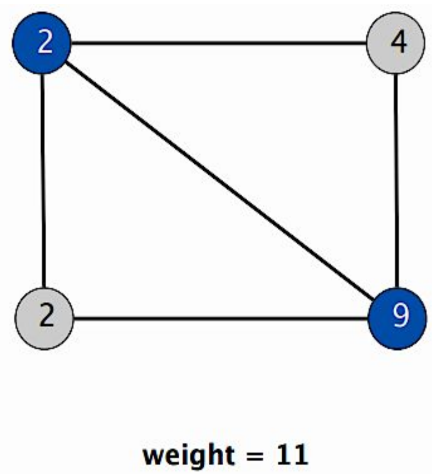


weight = 11

Constrained minimization example

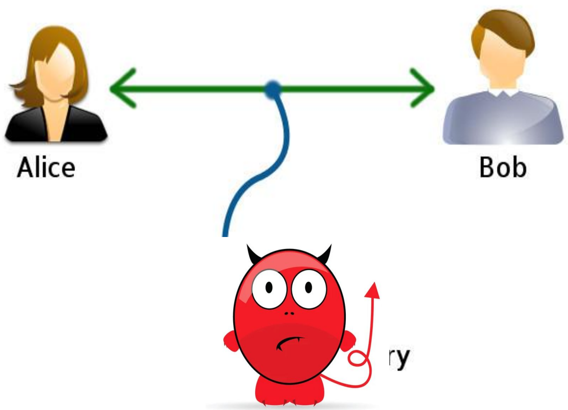
$$
\begin{array}{llcll}
\min & \sum_{i \in V} w_i \, x_i & & & \\
\text{s.t.} & x_i + x_j & \geq & 1 & (i,j) \in E \\
 & x_i & \in & \{0,\ 1\} & i \in V
\end{array}
$$

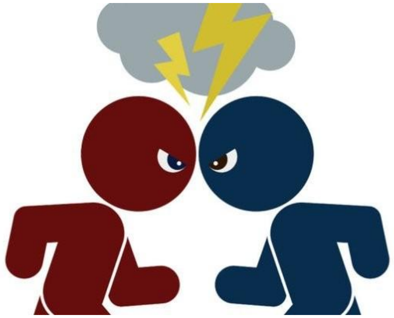


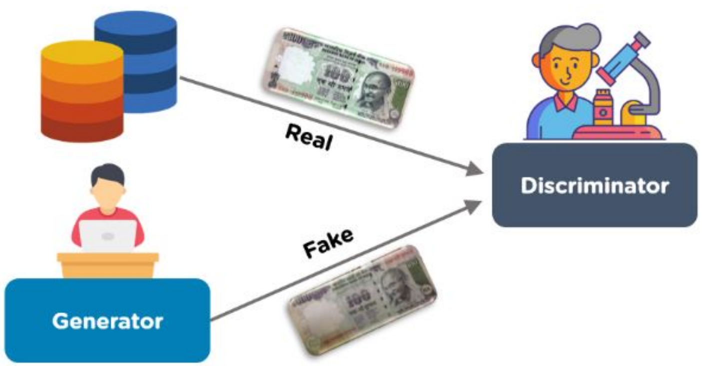


Adversarial Search And Games

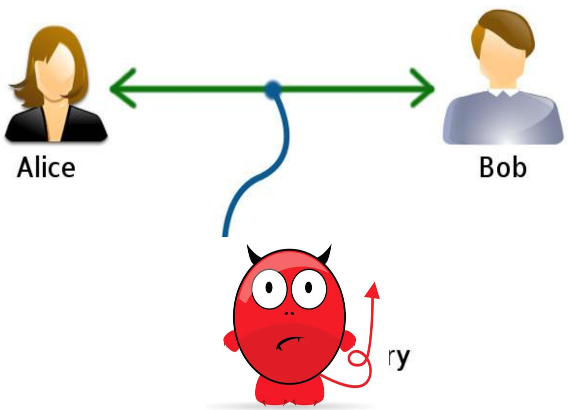
Adversary (an enemy, or an opponent in a competition)

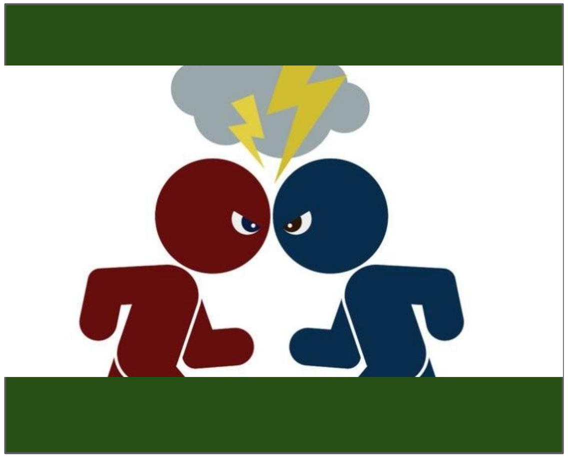


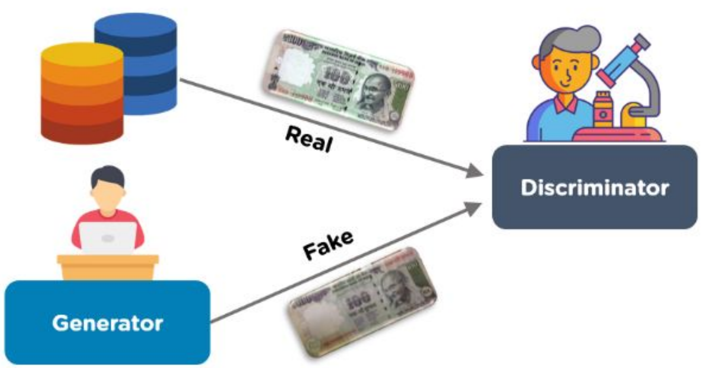




Adversary (an enemy, or an opponent in a competition)







Why GAMES

- **Games** are a form of *multi-agent* environments, in which two or more agents have goals (cooperative vs. competitive)
- Why study games in AI?
  - lots of **fun and fame**; historically entertaining
  - **challenges**: agents restricted to small number of actions with precise rules, computationally hard

Search (with no adversary)

- solution is a method for finding a goal
- heuristics techniques can find optimal solutions
- examples: path planning, scheduling activities, …

Games (with adversary), a.k.a adversarial search

Games (with adversary)

- solution a is strategy (specifies move for every possible opponent reply)
- evaluation function (utility): evaluate “goodness” of game position
- examples: tic-tac-toe, chess, checkers, backgammon, ...

Types of Games

- Here we focus on games, such as tic-tac-toe, chess, and poker.
- Physical games, such as croquet and ice hockey, have more complicated descriptions due to larger range of possible actions.

Types of Games

- Here we focus on games, such as tic-tac-toe, chess, and poker.
- Physical games, such as croquet and ice hockey, have more complicated descriptions due to larger range of possible actions.

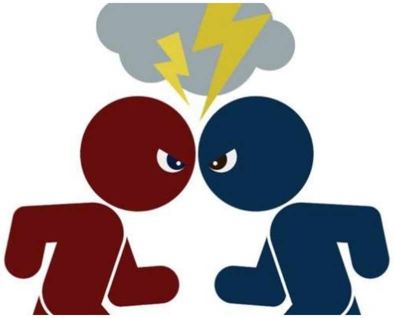
	deterministic	chance
perfect information	chess, checkers,	backgammon
imperfect information	blind tictactoe	bridge, poker

Adversarial game-tree search

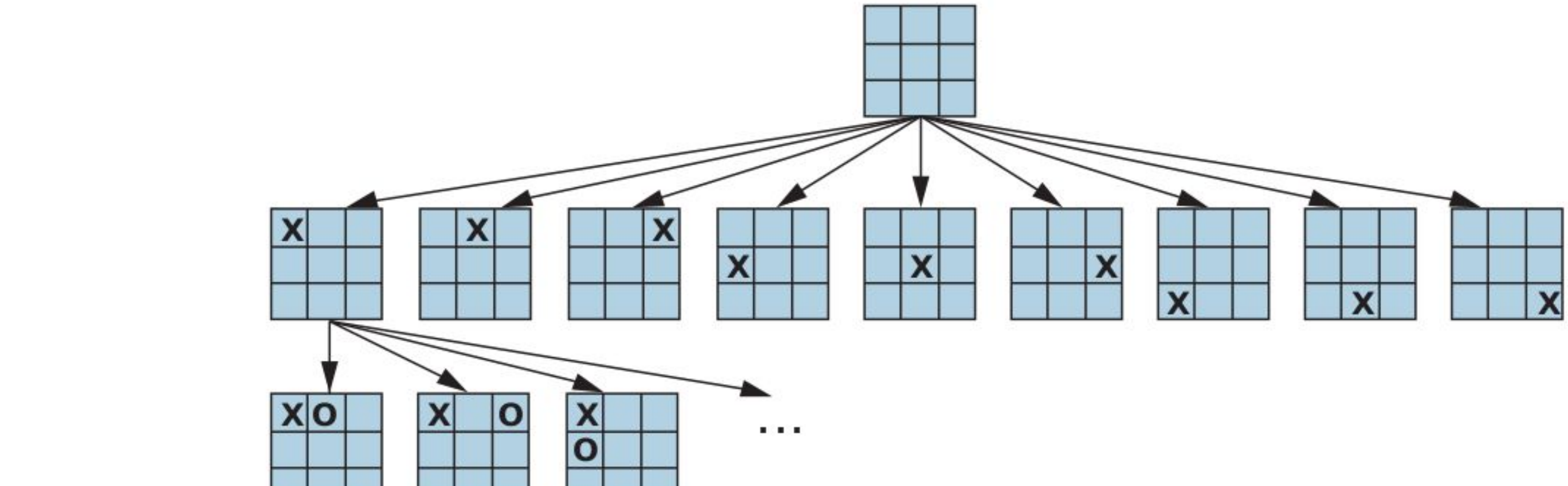
- We can define the complete game tree as a **search tree** that follows every sequence of moves all the way to a terminal state.
- Study restricted class of games and define the **optimal move** and an algorithm (**minimax**) for finding optimal move.
- Aim is to find best strategy (aka policy $\boldsymbol{\pi} : S \longrightarrow A$)

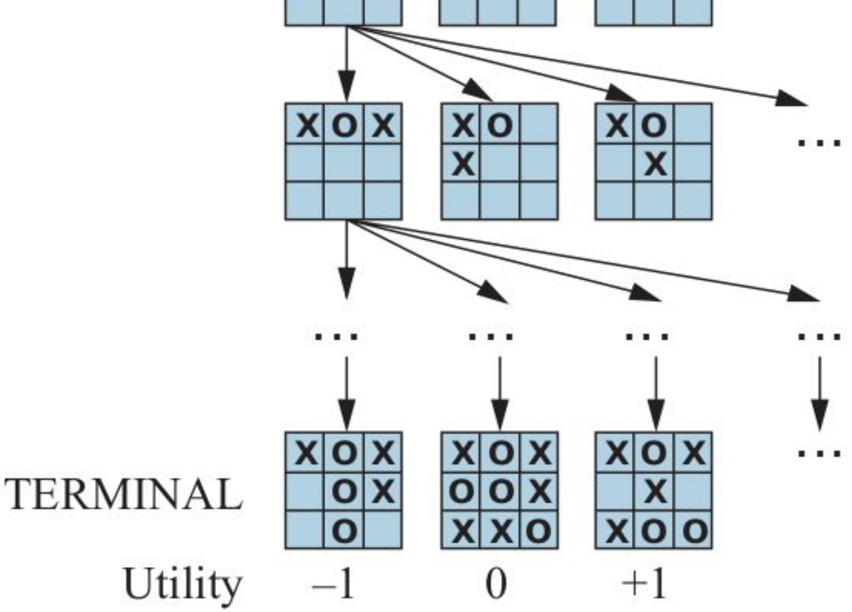
Two-player zero-sum games

- We mostly study a *two-player*, turn-taking, perfect information, *zero-sum* games.
  - “zero-sum” means that what is good for one player is just as bad for the other: there is no “win-win” outcome.
  - “Perfect information” is a synonym for “fully observable”.

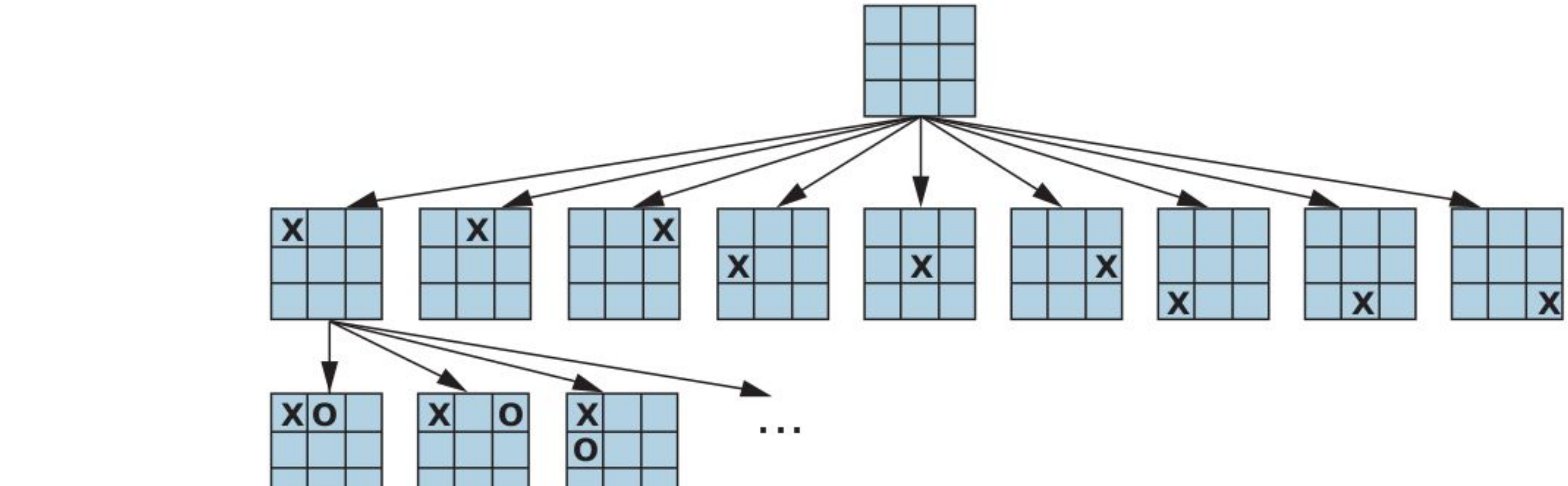


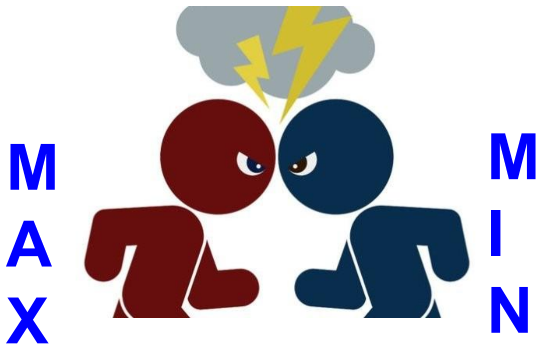
Two-player zero-sum games



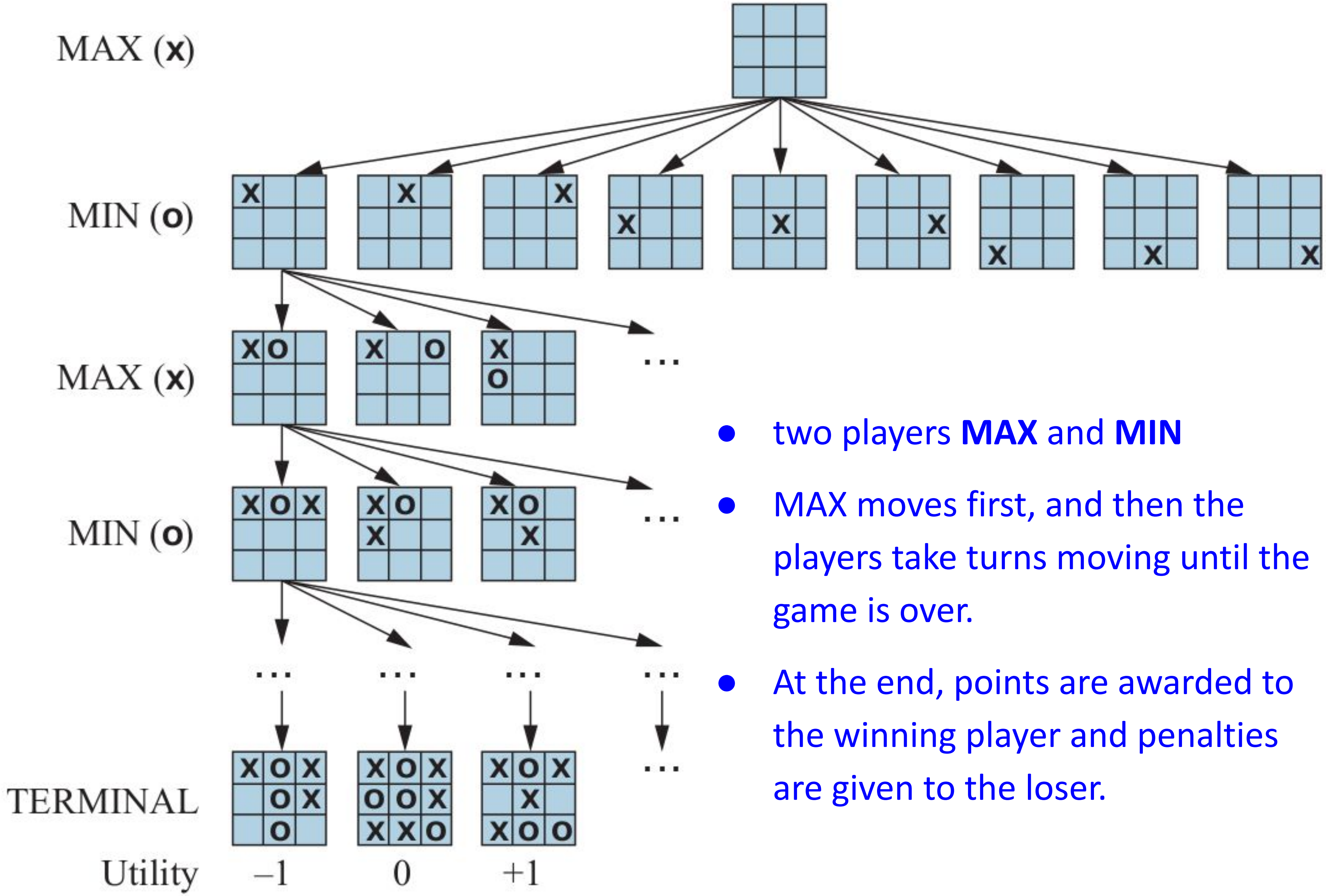


Two-player zero-sum games





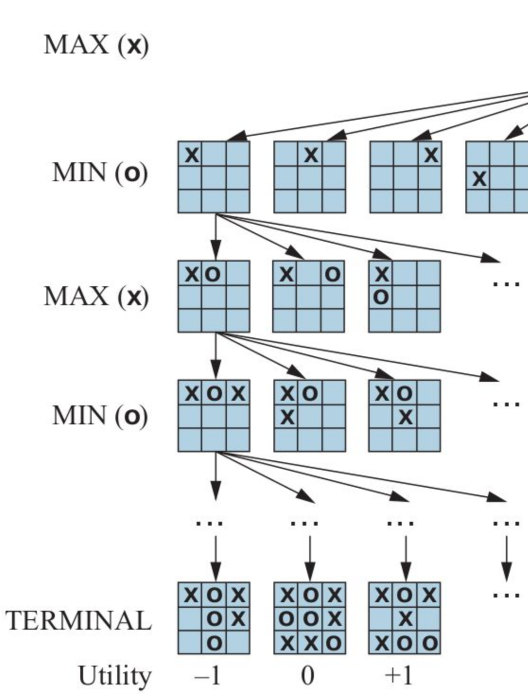
Min-Max



- two players **MAX** and **MIN**
- MAX moves first, and then the players take turns moving until the game is over.
- At the end, points are awarded to the winning player and penalties are given to the loser.

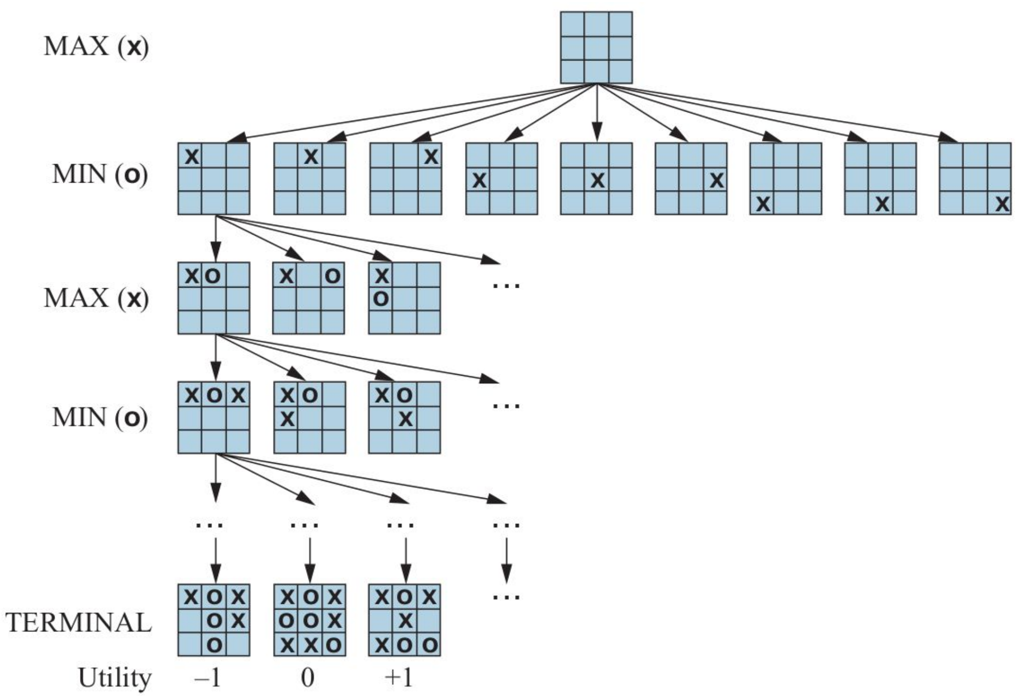
Adversarial Search as Min-Max Search

- Assume MAX and MIN are very smart and always play optimally
- A single-agent move is called half-move or ply
- In a non-terminal state, MAX prefers to move to a state of maximum value and MIN prefers a state of minimum value.



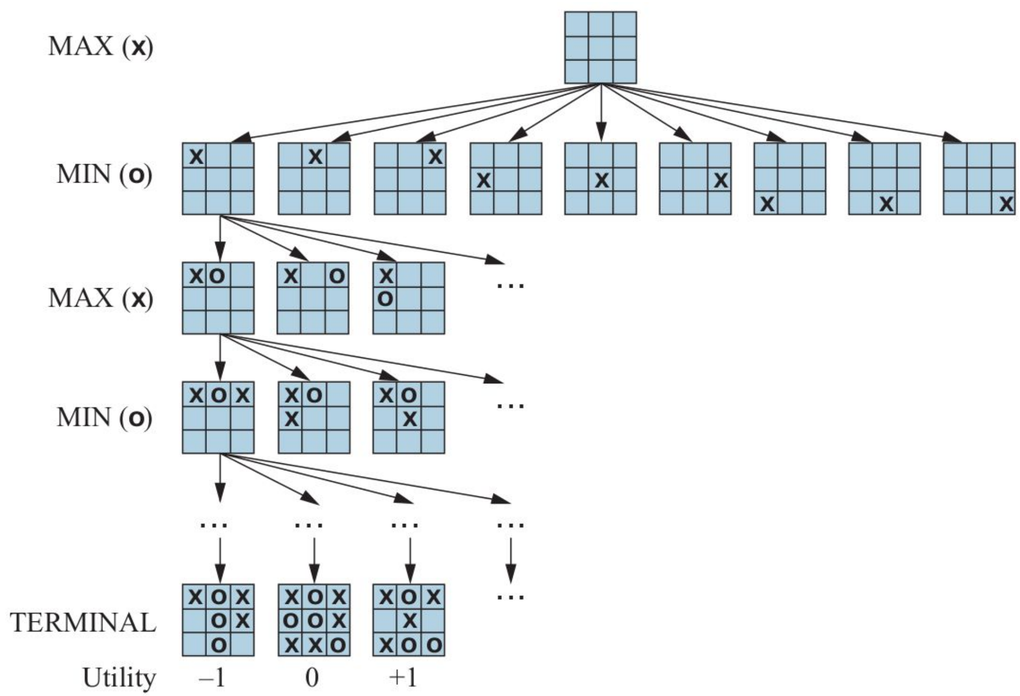
Definitions

- S_0 : The initial state, which specifies how the game is set up at the start.



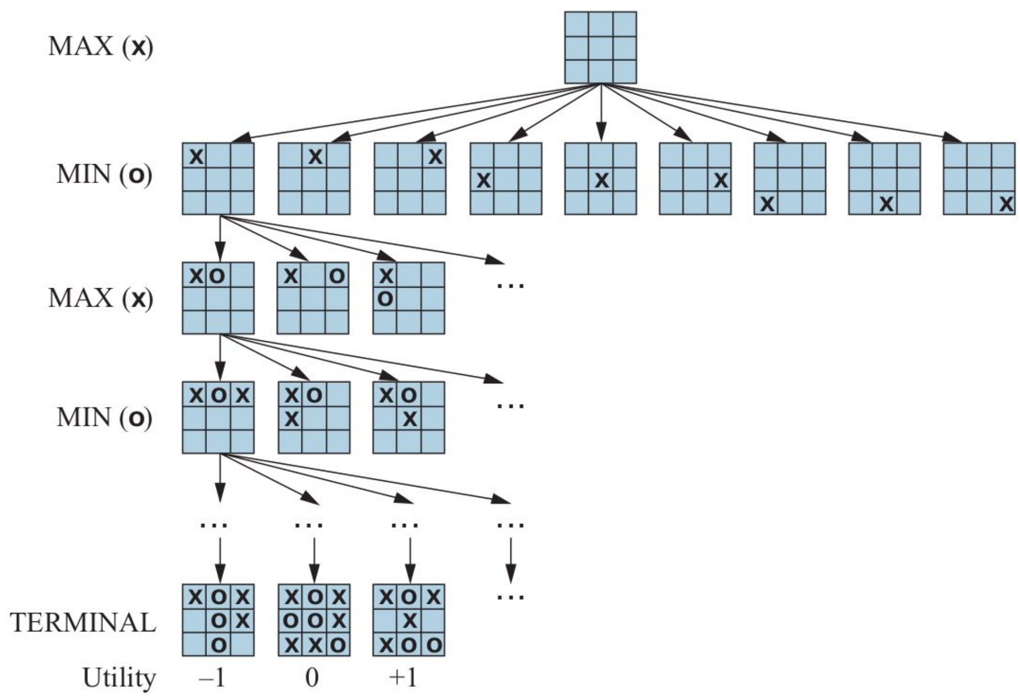
Definitions

- S_0 : The initial state, which specifies how the game is set up at the start.
- To-Move (s) : The player whose turn it is to move in state s.



Definitions

- S_0 : The initial state, which specifies how the game is set up at the start.
- To-Move (s) : The player whose turn it is to move in state s.
- Actions (s) : The set of legal moves in state s.



Definitions

- S_0 : The initial state, which specifies how the game is set up at the start.
- To-Move (s) : The player whose turn it is to move in state s.
- Actions (s) : The set of legal moves in state s.
- Result (s, a) : The transition model, which defines the state resulting from taking action a in state s.

Definitions

- S_0 : The initial state, which specifies how the game is set up at the start.
- To-Move (s) : The player whose turn it is to move in state s.
- Actions (s) : The set of legal moves in state s.
- Result (s, a) : The transition model, which defines the state resulting from taking action a in state s.
- Is-Terminal (s) : A terminal test, which is true when the game is over and false otherwise. States where the game has ended are called terminal states.

Definitions

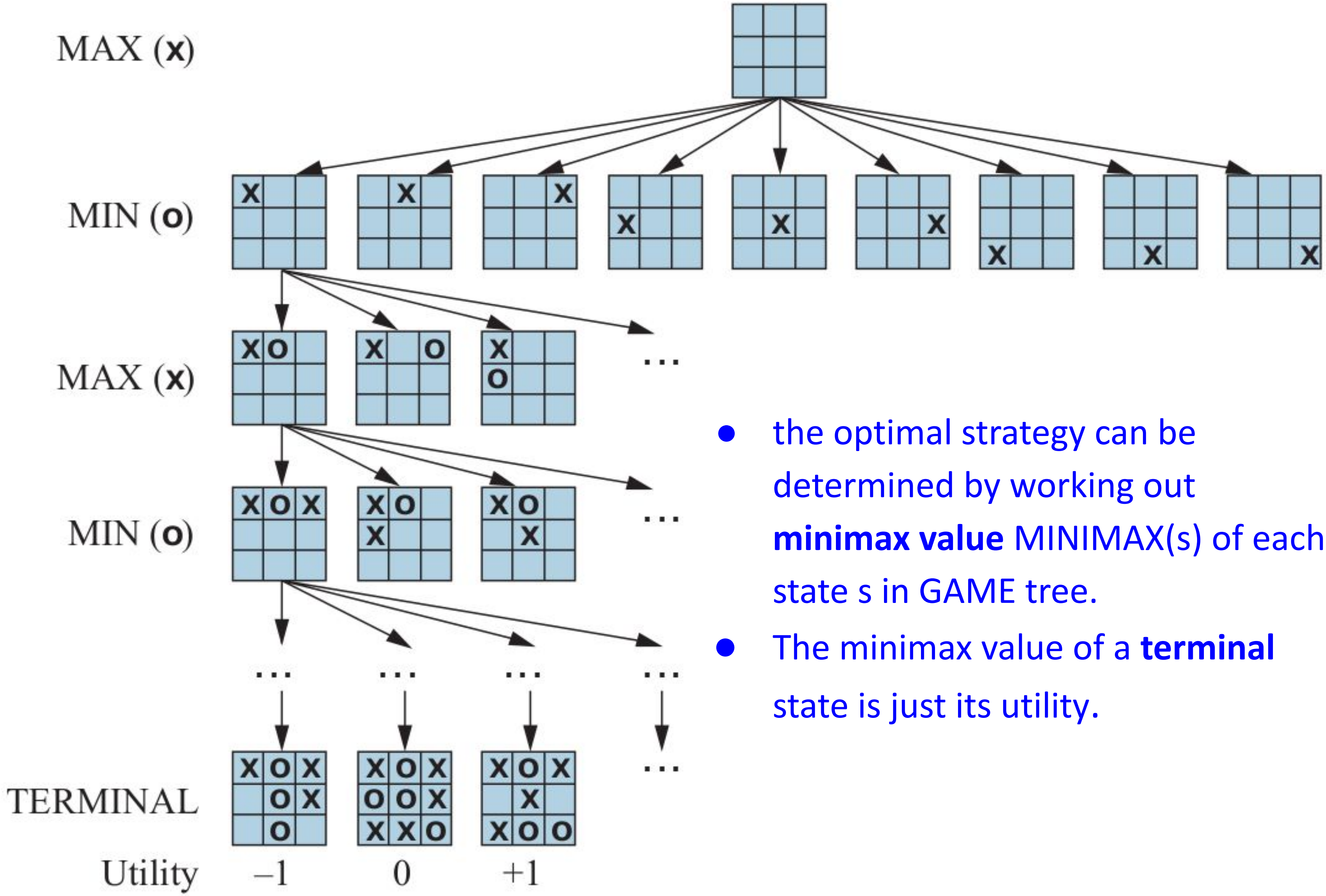
- S_0 : The initial state, which specifies how the game is set up at the start.
- To-Move (s) : The player whose turn it is to move in state s.
- Actions (s) : The set of legal moves in state s.
- Result (s, a) : The transition model, which defines the state resulting from taking action a in state s.
- Is-Terminal (s) : A terminal test, which is true when the game is over and false otherwise. States where the game has ended are called terminal states.
- Utility(s, p) : A utility function (also called an objective function or payoff function), which defines the final numeric value to player p when the game ends in terminal state s.

Adversarial Search as Min-Max Search

- Suppose MINIMAX(**s**) value is utility (for MAX) of being in state **s**.

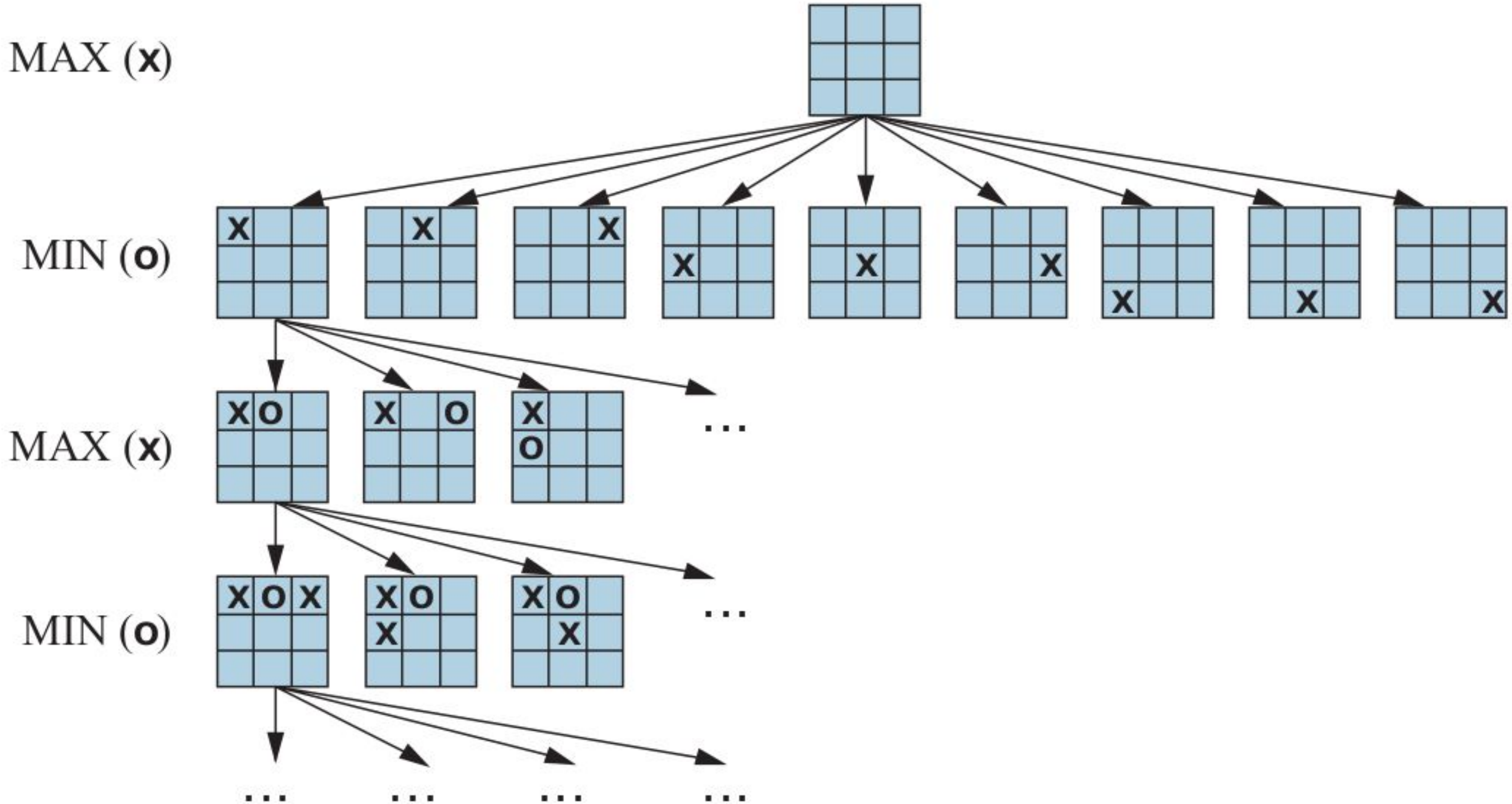
$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s, \textsc{max}) & \text{if } \textsc{Is-Terminal}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s, a)) & \text{if } \textsc{To-Move}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s, a)) & \text{if } \textsc{To-Move}(s) = \textsc{min} \end{cases}$$

Adversarial Search as Min-Max Search



- the optimal strategy can be determined by working out **minimax value** MINIMAX(s) of each state s in GAME tree.
- The minimax value of a **terminal** state is just its utility.

Adversarial Search as Min-Max Search



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.
- $5^9 = 1{,}953{,}125$

Min-Max Search

- Δ nodes are "MAX nodes", ∇ nodes are "MIN nodes",

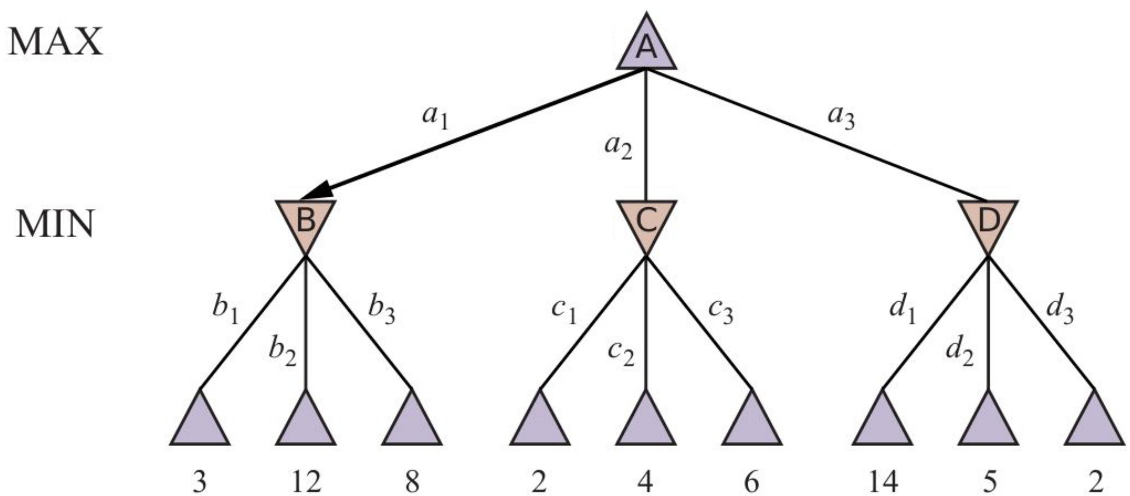


Figure 5.2 A two-ply game tree. The $\triangle$ nodes are "MAX nodes," in which it is MAX's turn to move, and the $\triangledown$ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX's best move at the root is

Min-Max Search

- Δ nodes are "MAX nodes", ∇ nodes are "MIN nodes",
- terminal nodes show the utility values for MAX

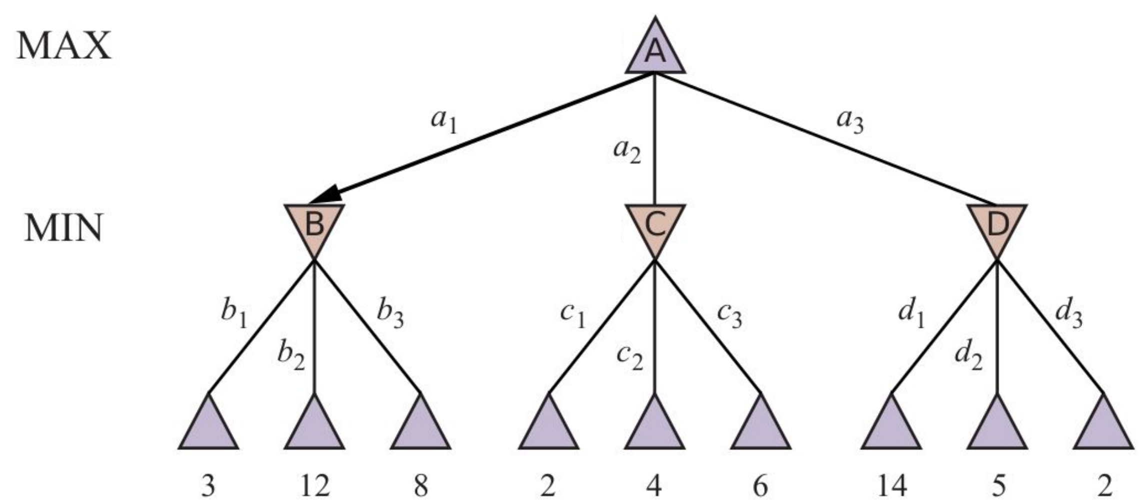


Figure 5.2 A two-ply game tree. The △ nodes are "MAX nodes," in which it is MAX's turn to move, and the ▽ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX's best move at the root is

Min-Max Search

- Δ nodes are "MAX nodes", ∇ nodes are "MIN nodes",
- terminal nodes show the utility values for MAX
- the other nodes are labeled with their minimax value

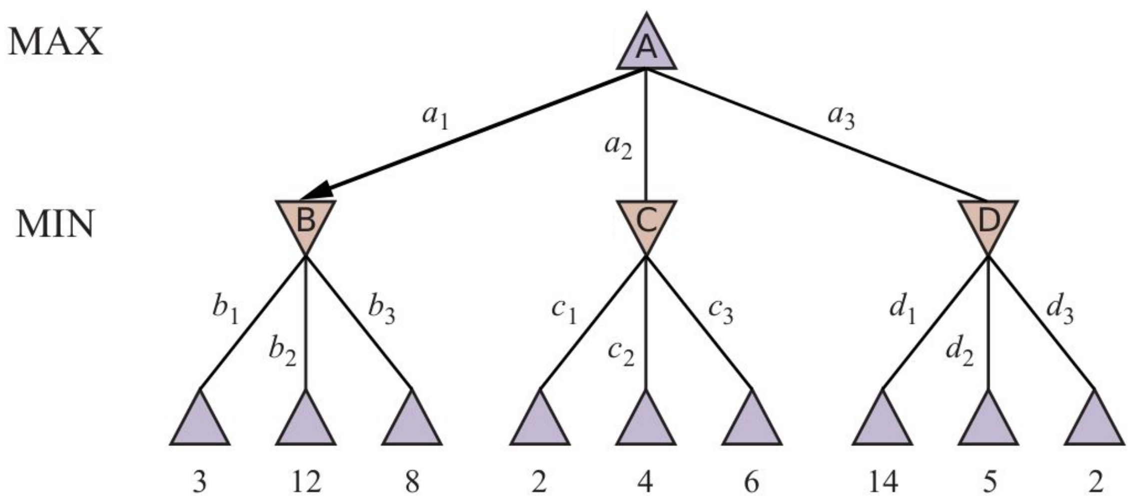


Figure 5.2 A two-ply game tree. The △ nodes are "MAX nodes," in which it is MAX's turn to move, and the ▽ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX's best move at the root is

Min-Max Search

- Δ nodes are "MAX nodes", ∇ nodes are "MIN nodes",
- terminal nodes show the utility values for MAX
- the other nodes are labeled with their minimax value
- Minimax maximizes the worst-case outcome for MAX

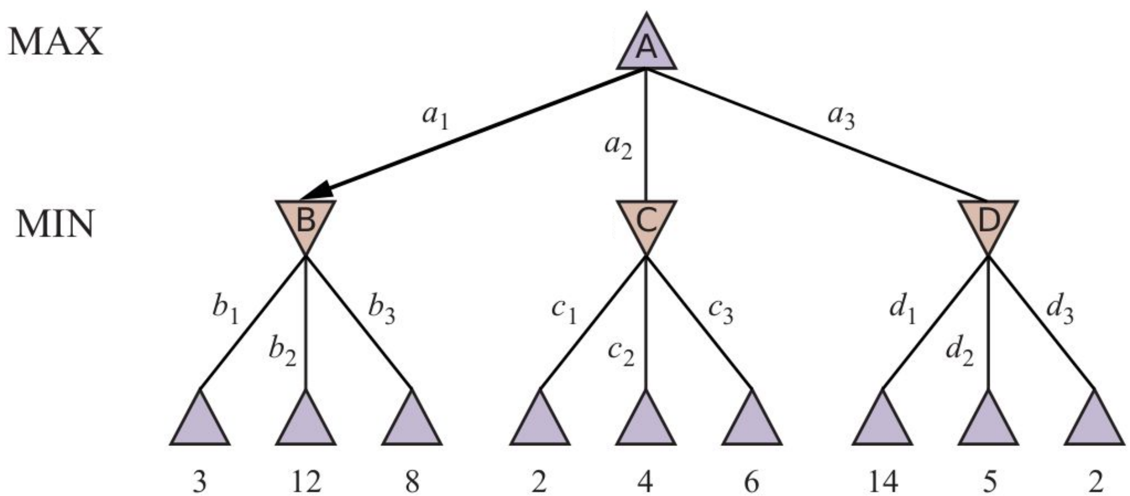


Figure 5.2 A two-ply game tree. The △ nodes are "MAX nodes," in which it is MAX's turn to move, and the ▽ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX's best move at the root is

Min-Max Search

- Δ nodes are "MAX nodes", ∇ nodes are "MIN nodes",
- terminal nodes show the utility values for MAX
- the other nodes are labeled with their minimax value
- Minimax maximizes the worst-case outcome for MAX
- Each node labeled with values from each player's viewpoint

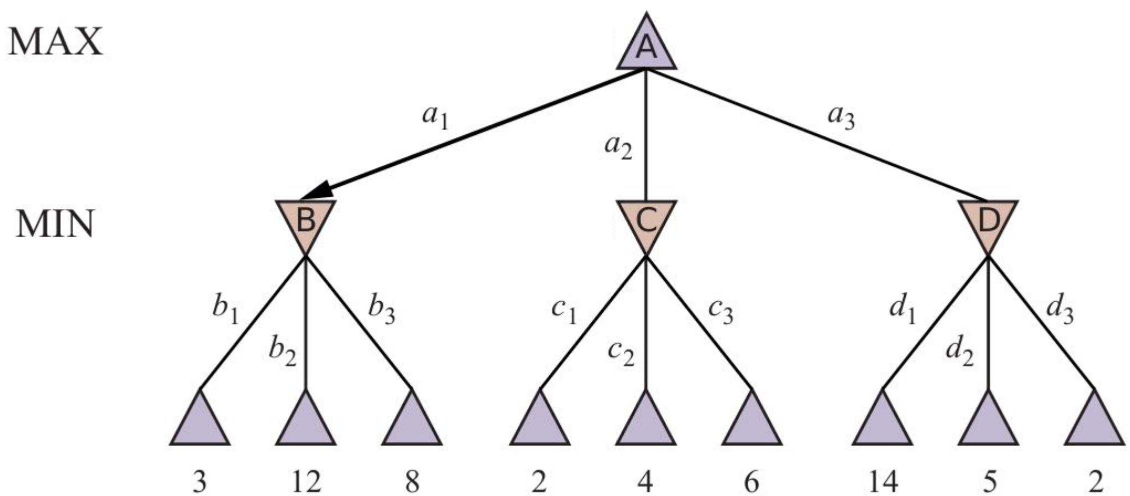


Figure 5.2 A two-ply game tree. The △ nodes are "MAX nodes," in which it is MAX's turn to move, and the ▽ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX's best move at the root is

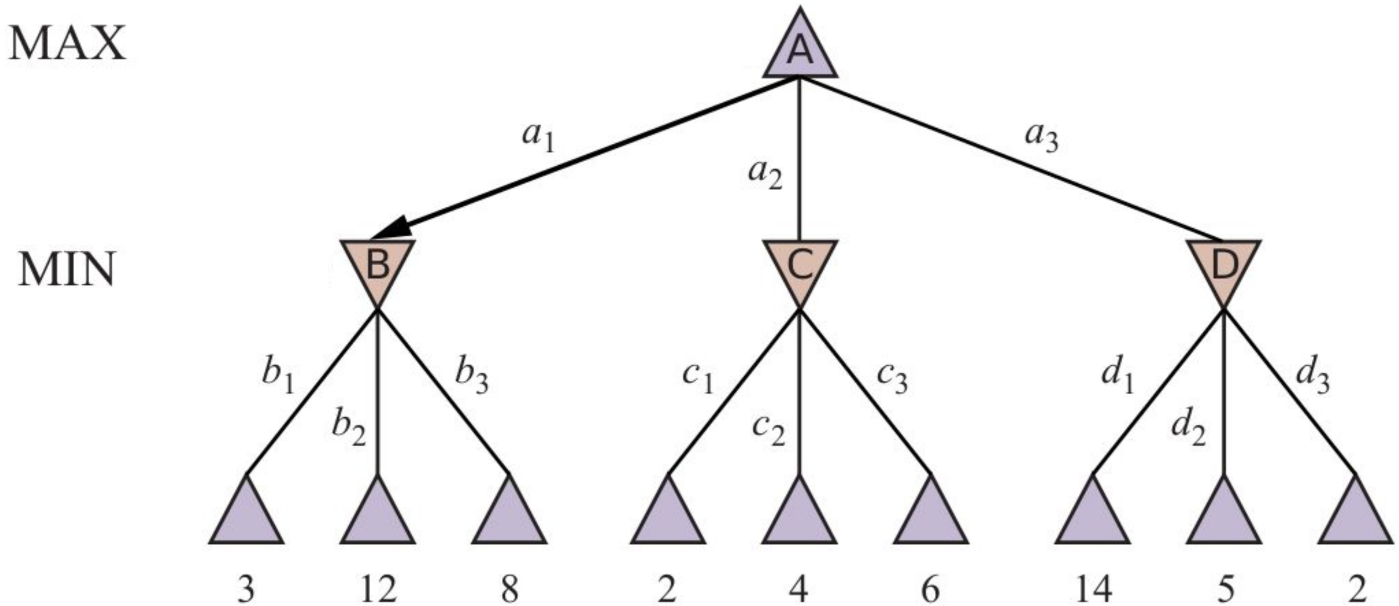


Figure 5.2 A two-ply game tree. The $\triangle$ nodes are "MAX nodes," in which it is MAX's turn to move, and the $\triangledown$ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values.

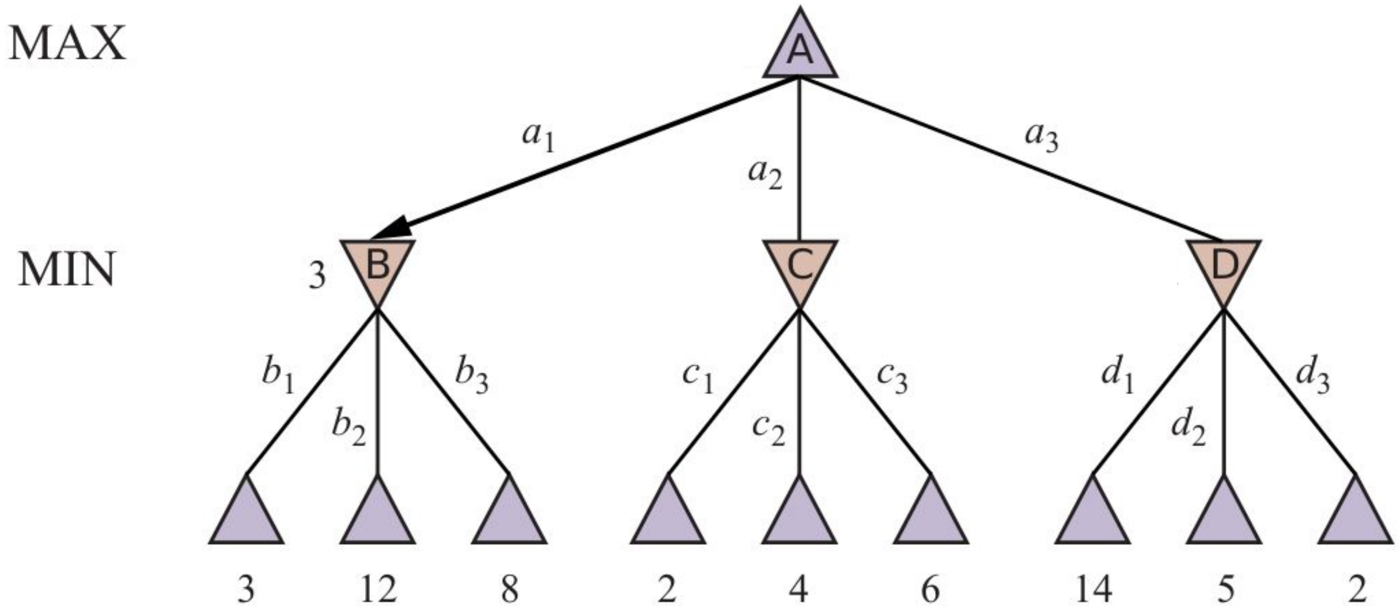


Figure 5.2 A two-ply game tree. The $\triangle$ nodes are "MAX nodes," in which it is MAX's turn to move, and the ∇ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values.

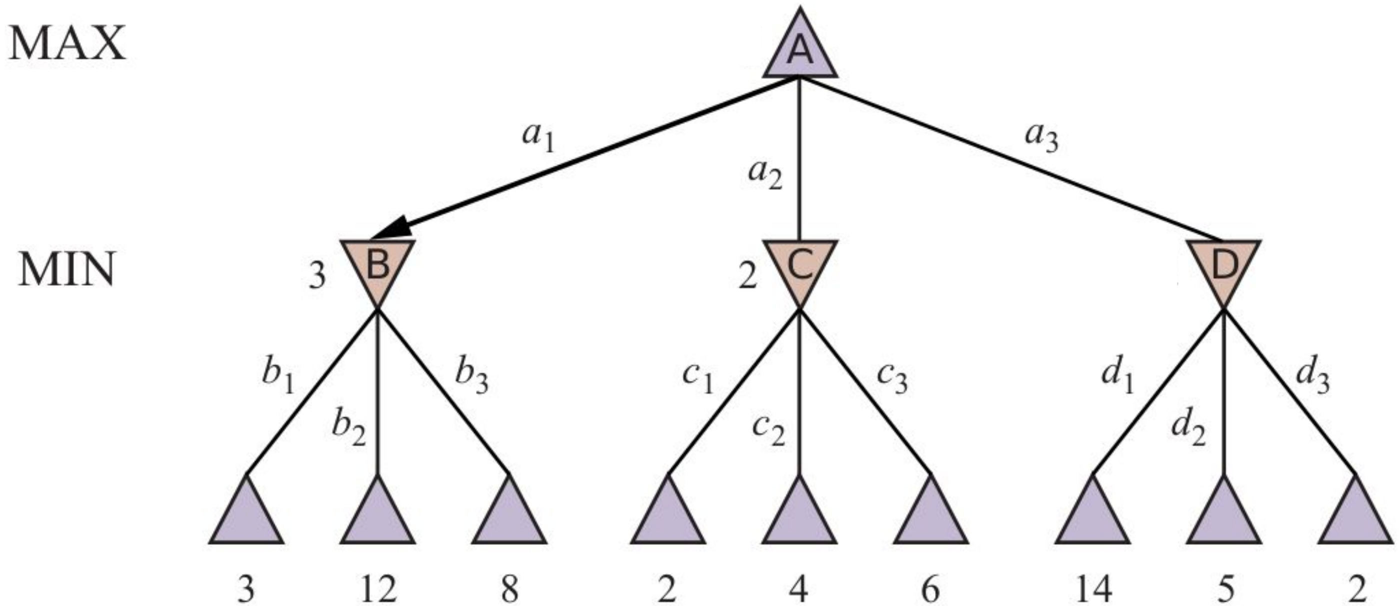


Figure 5.2 A two-ply game tree. The $\triangle$ nodes are "MAX nodes," in which it is MAX's turn to move, and the $\triangledown$ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values.

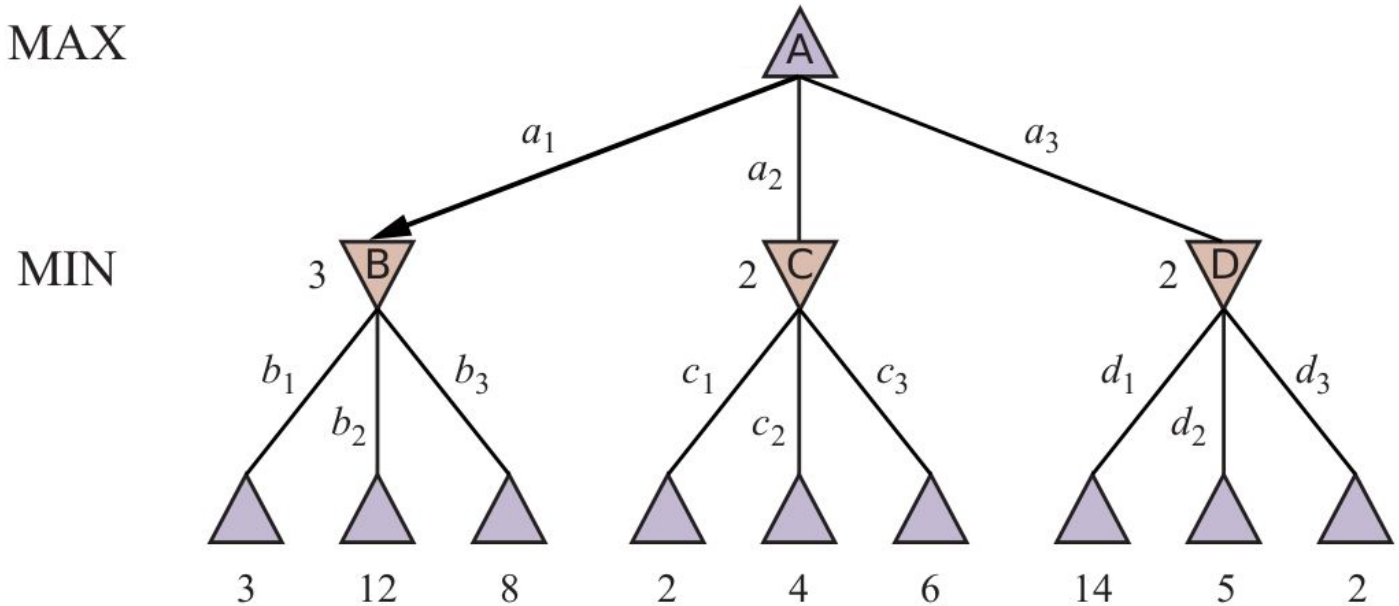


Figure 5.2 A two-ply game tree. The △ nodes are "MAX nodes," in which it is MAX's turn to move, and the ▽ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values.

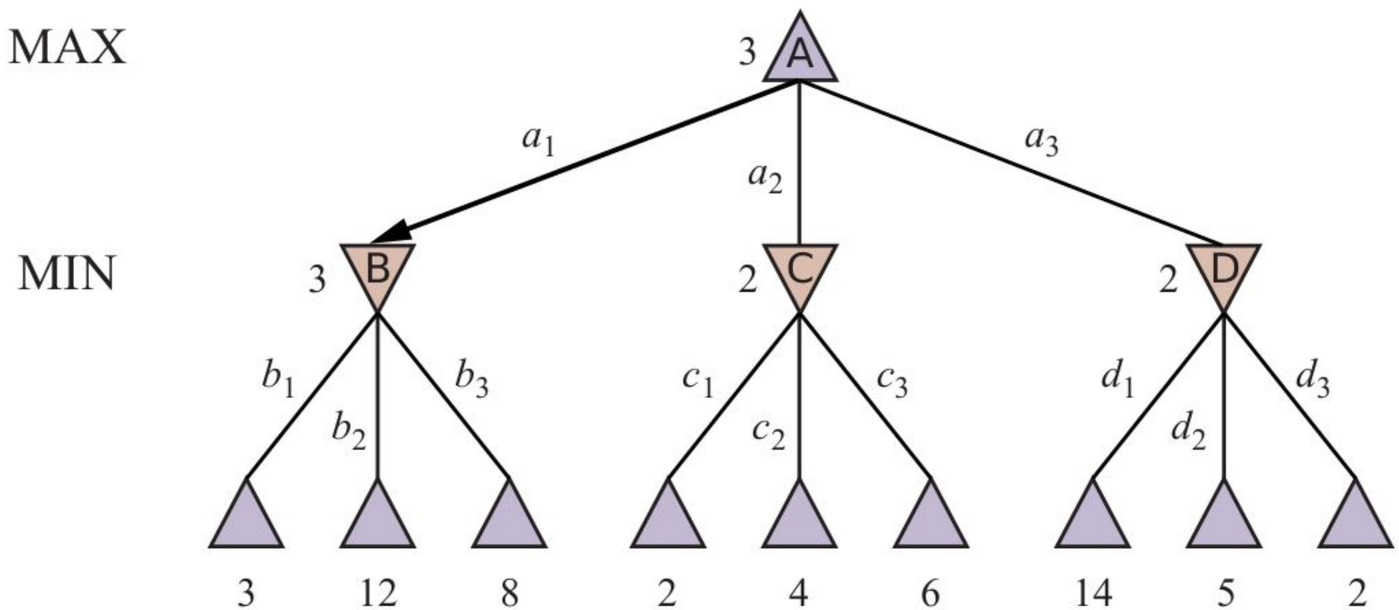


Figure 5.2 A two-ply game tree. The △ nodes are "MAX nodes," in which it is MAX's turn to move, and the ▽ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values.

Properties of minimax

Complete??

Optimal??

Time complexity??

Space complexity??

Properties of minimax

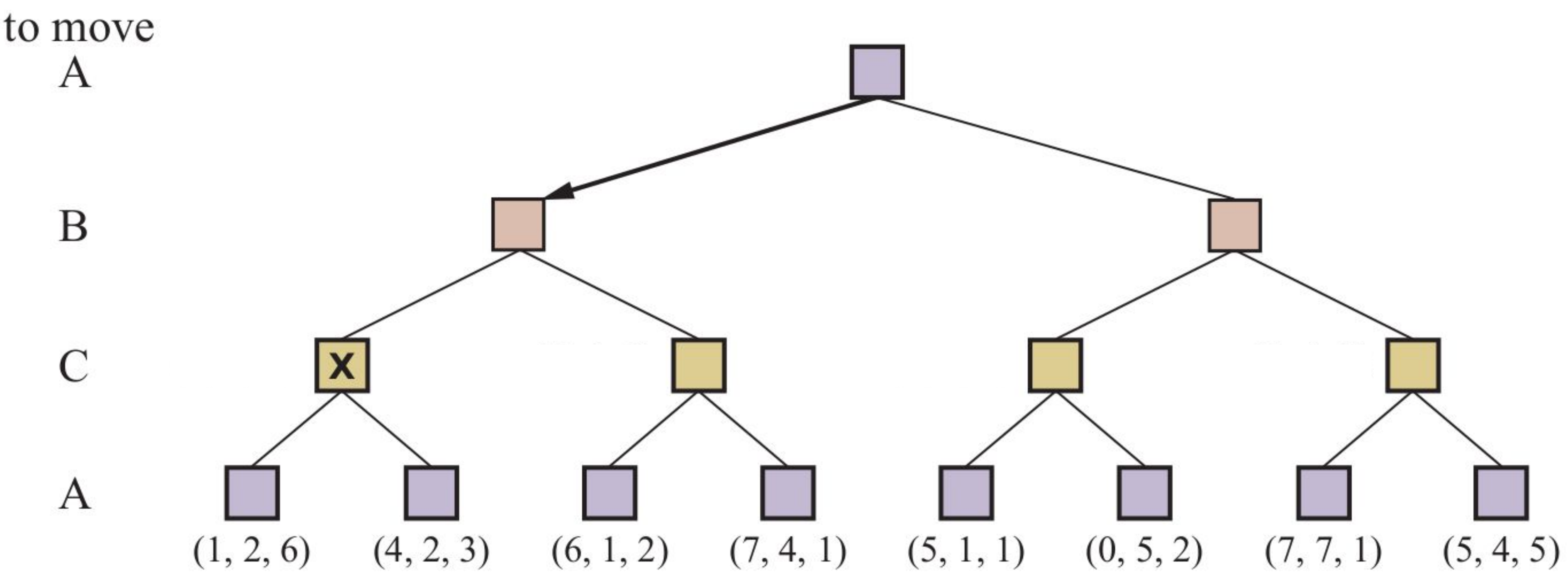
Complete?? Yes, if tree is finite

Optimal?? Yes, against an optimal opponent.

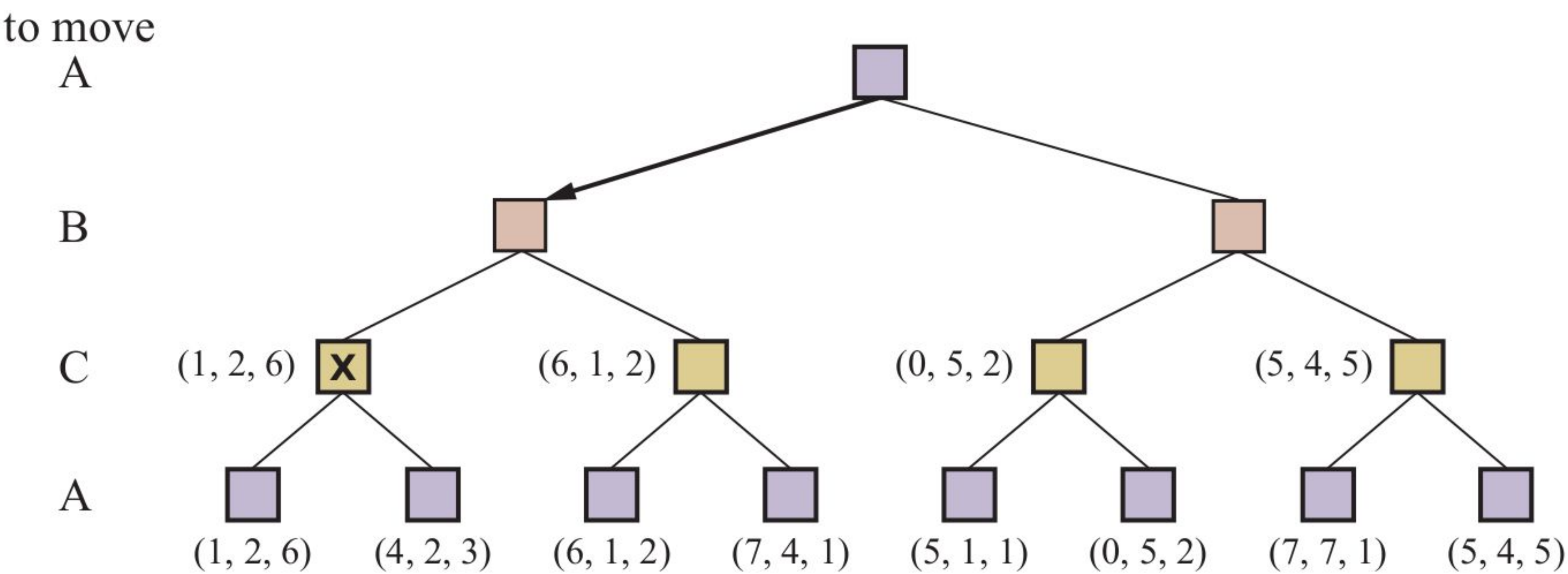
Time complexity?? $O(b^m)$

Space complexity?? $O(bm)$ (depth-first exploration)

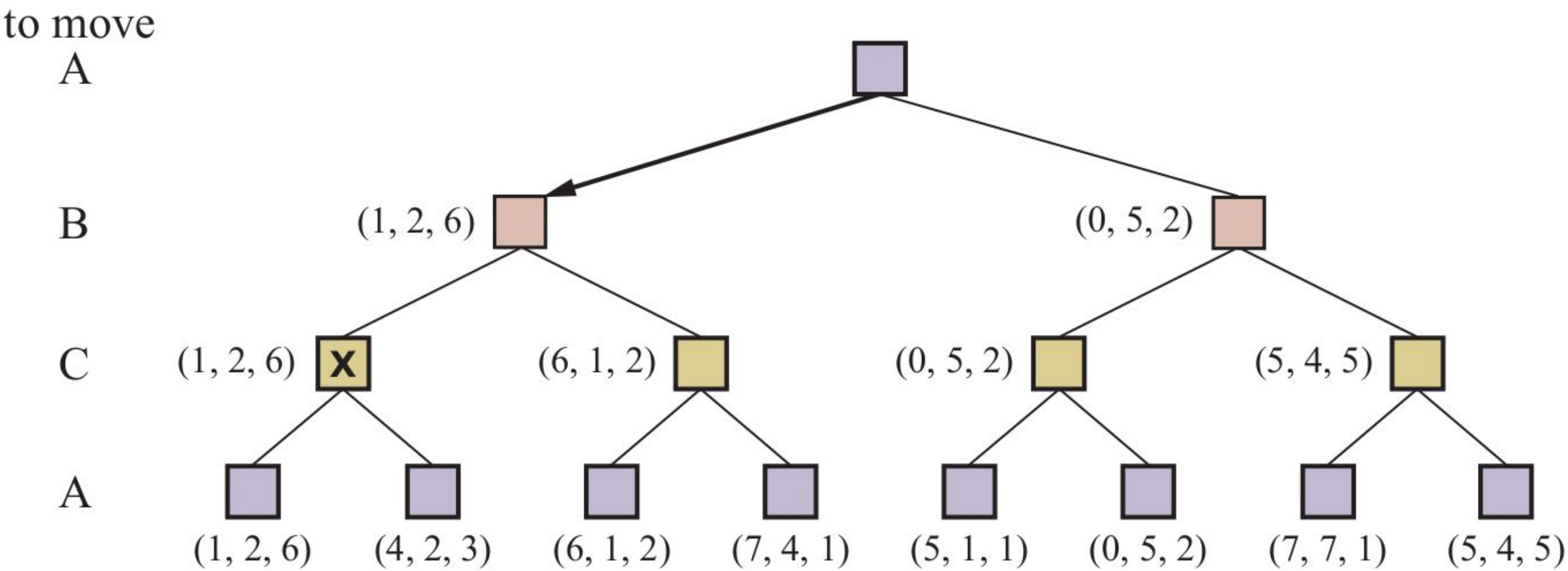
Optimal decisions in multiplayer games



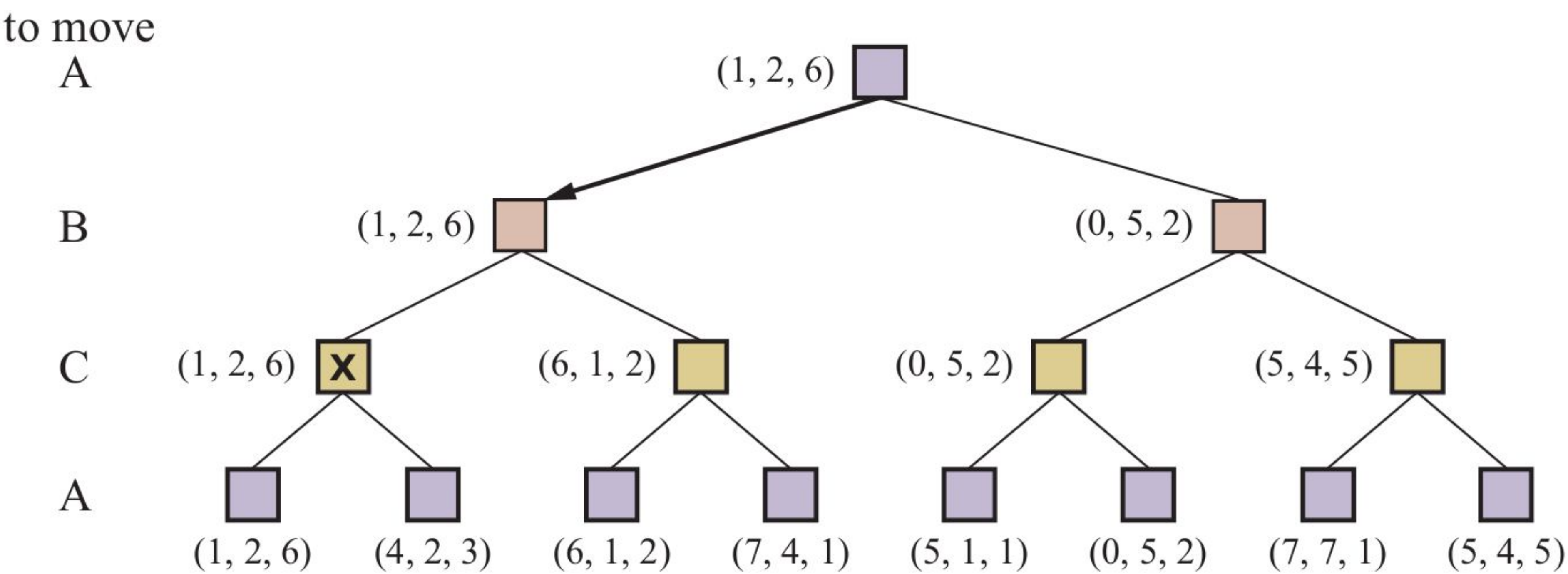
Optimal decisions in multiplayer games



Optimal decisions in multiplayer games



Optimal decisions in multiplayer games



Alpha–Beta Pruning

- The number of game states is exponential in the depth of the tree.
- We can sometimes compute the correct minimax decision without examining every state by pruning large parts of the tree.

Alpha–Beta Pruning

- The number of game states is exponential in the depth of the tree.
- We can sometimes compute the correct minimax decision without examining every state by pruning large parts of the tree.
- Alpha–beta pruning can be applied to trees of any depth, and it is often possible to prune entire subtrees rather than just leaves.
  - **α** = the value of the best choice along the path of MAX.
  - **β** = the value of the best choice along the path of MIN.

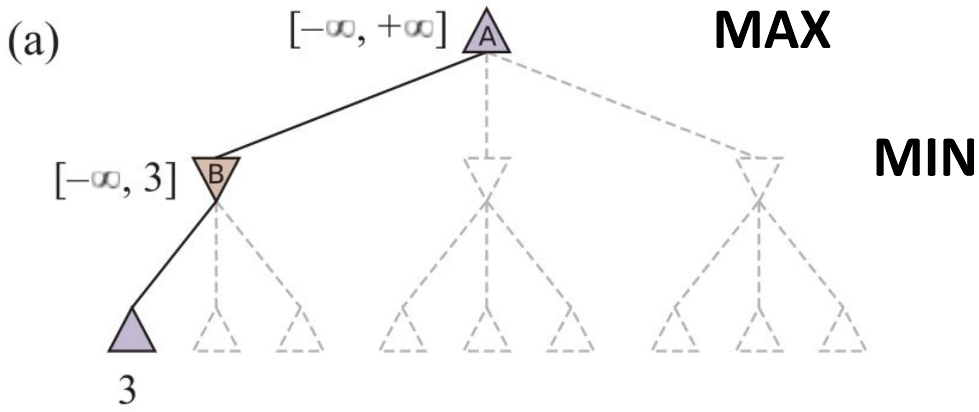
Alpha–Beta Pruning

Key Idea:

- Alpha–beta search updates the values of α and β as the search goes along.
- Prunes the remaining branches at a node as soon as the value of the current node is known to be worse than the current α or β value for MAX or MIN, respectively.
- Note that pruning make no difference to the outcome.

Alpha–Beta Pruning [**α, β**]

Let [min, max] track the currently-known bounds for the search.

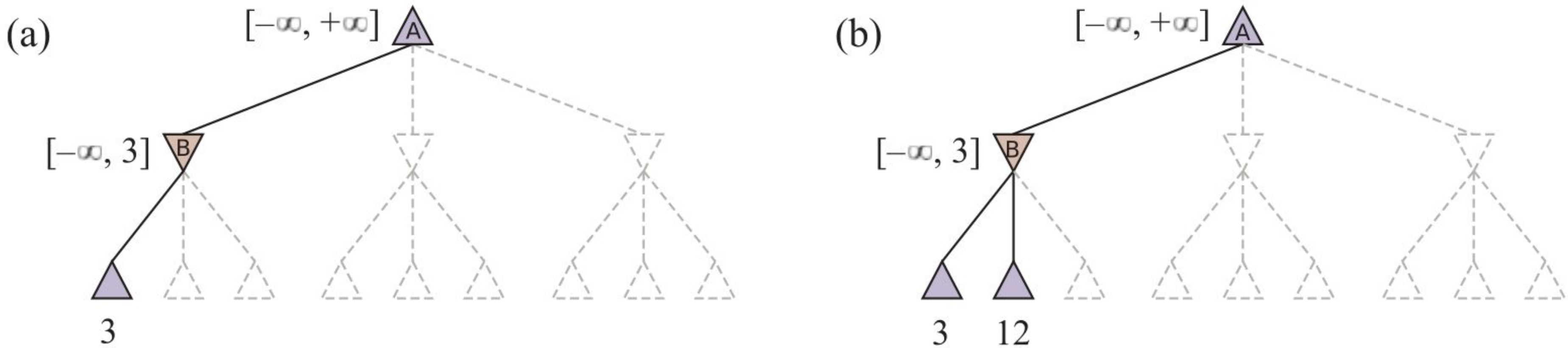


B labeled with [−∞, 3] (MIN will not choose values ≥ 3 for B)

nodes labeled with **[α, β]**

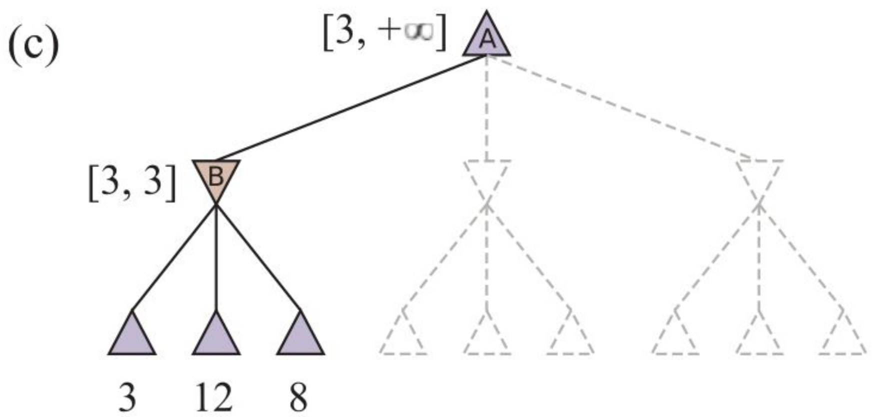
- **α** = best choice along the path of **MAX**.
- **β** = best choice along the path of **MIN**.

Alpha–Beta Pruning



- **α** = best choice along the path of **MAX**.
- **β** = best choice along the path of **MIN**.

Alpha–Beta Pruning

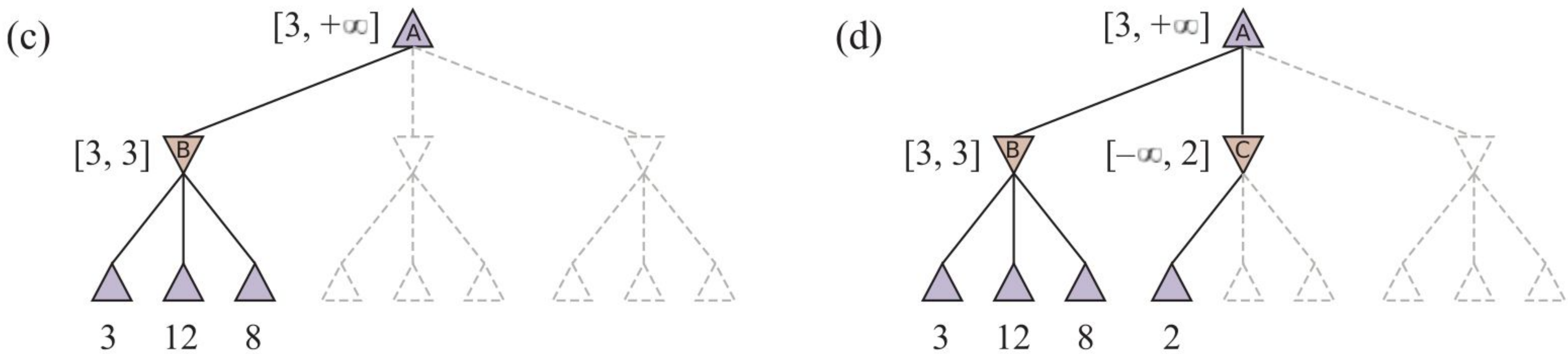


B labeled with [3, 3] (MIN cannot find values ≤ 3 for B)

- **α** = best choice along the path of **MAX**.
- **β** = best choice along the path of **MIN**.

Alpha–Beta Pruning

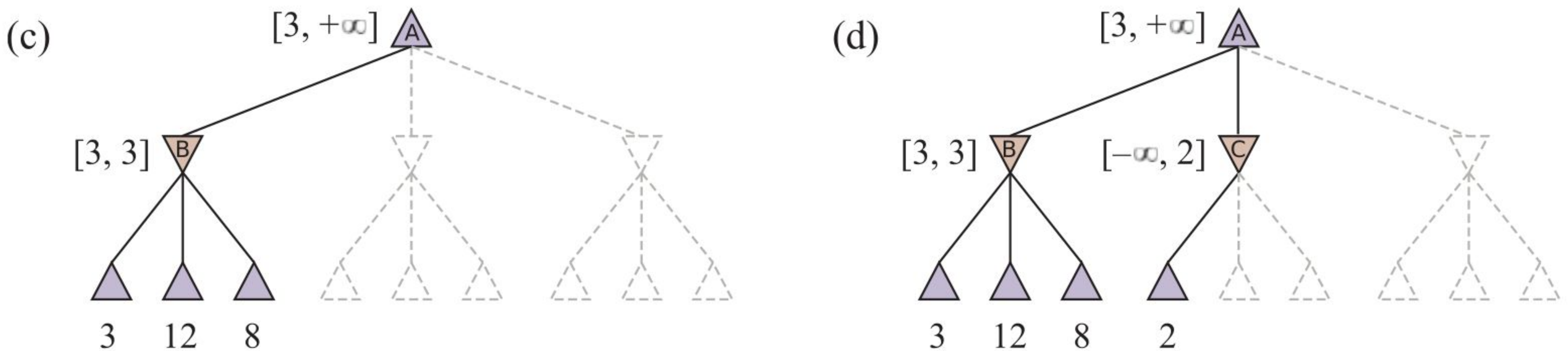
Is it necessary to evaluate the remaining leaves of C?



- **α** = best choice along the path of **MAX**.
- **β** = best choice along the path of **MIN**.

Alpha–Beta Pruning

NO! They cannot produce an upper bound ≥ 2

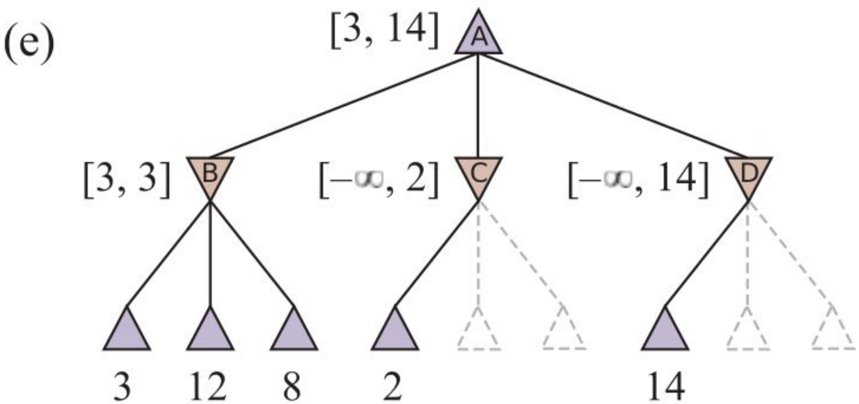


- **α** = best choice along the path of **MAX**.
- **β** = best choice along the path of **MIN**.

Prune: α ⩾ β

Alpha–Beta Pruning

MAX updates the upper bound to
14 (D is last subtree)

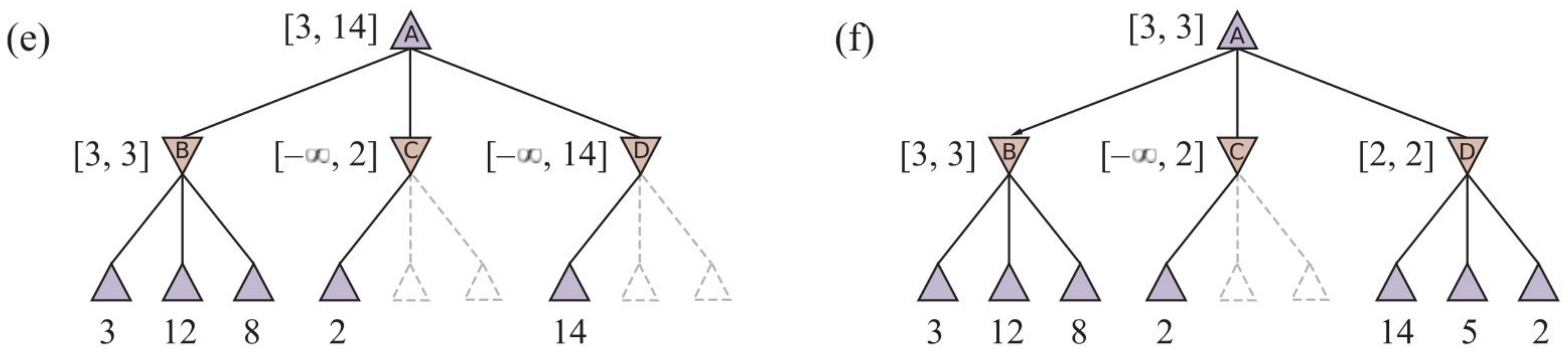


- **α** = best choice along the path of **MAX**.
- **β** = best choice along the path of **MIN**.

Prune: α ⩾ β

Alpha–Beta Pruning

D labeled [2, 2] ⇒ MAX updates the upper bound to 3



- **α** = best choice along the path of **MAX**.
- **β** = best choice along the path of **MIN**.

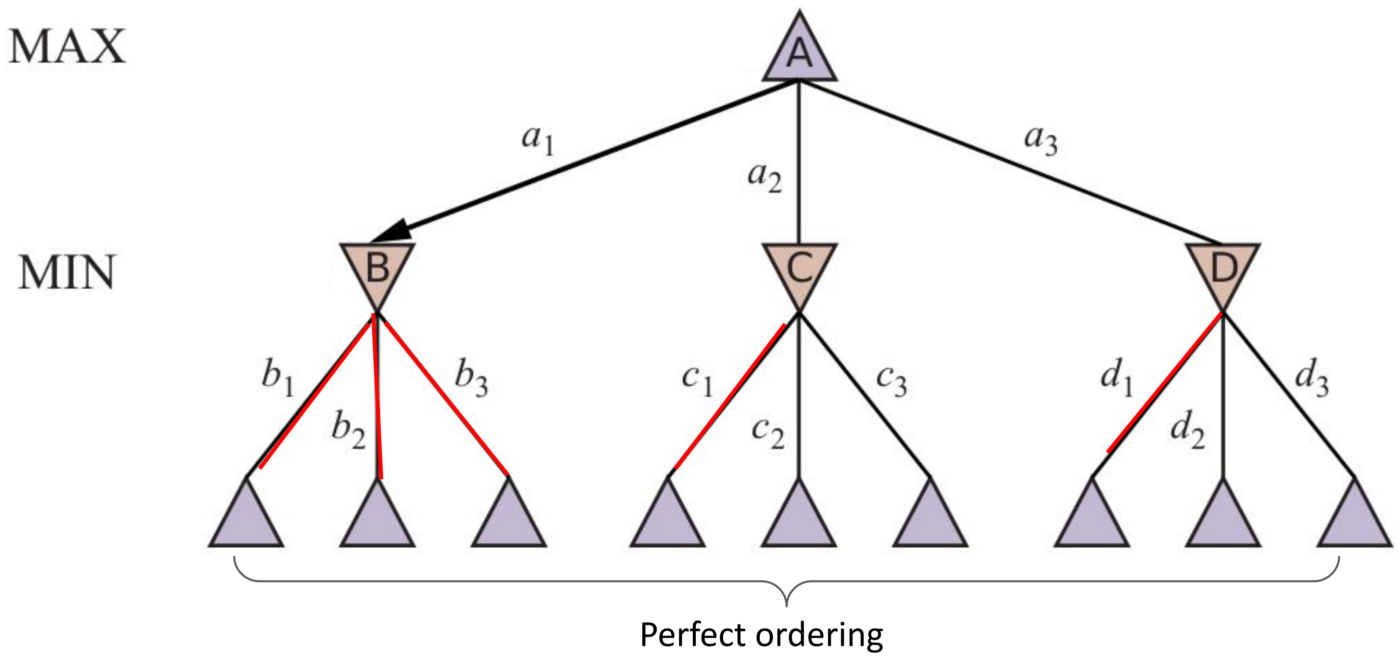
Prune: α ⩾ β

Properties of α–β

Pruning *does not* affect final result

Good move ordering improves effectiveness of pruning

With "perfect ordering," time complexity $= O(b^{m/2})$



Adversarial Search with Resource Limits

- **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**

Evaluation Functions

- A heuristic evaluation function EVAL (s, p) returns an estimate of the expected utility from a given position.
  - Ideal function: returns the actual minimax value of the position
- Should order terminal states the same way as the utility function
  - e.g., wins > draws > losses
- For nonterminal states, should be strongly correlated with the actual chances of winning

Evaluation Functions

- Typically weighted linear sum of features:

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s),$$

- Example chess game:

Symbol					
Piece	pawn	knight	bishop	rook	queen
Value	**1**	**3**	**3**	**5**	**9**

  - $w_{pawns} = 1$, $w_{bishops} = w_{knights} = 3$, $w_{rooks} = 5$, $w_{queens} = 9$

Evaluation Functions

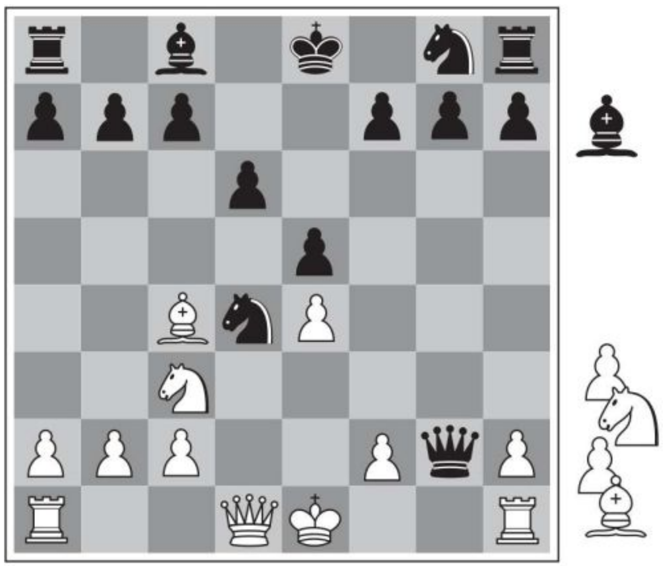
- Typically weighted linear sum of features:

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s) \,,$$

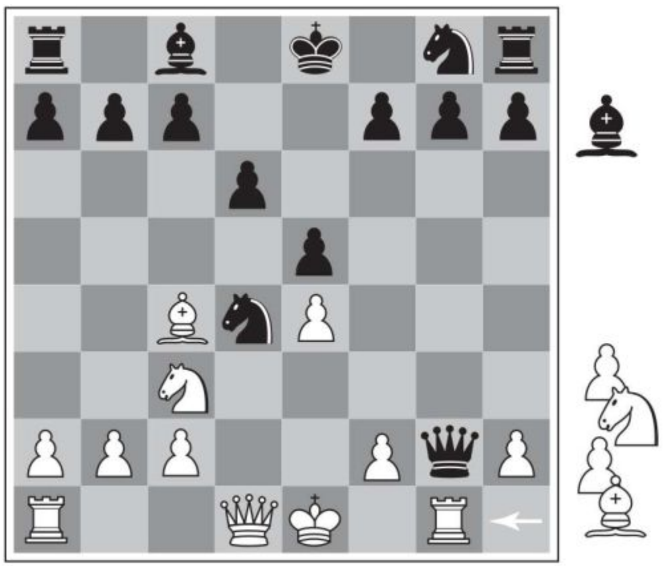
- Example chess game:
  - $f_{queens}(s)$ = #white queens − #black queens,

Evaluation Functions

- Two same-score positions



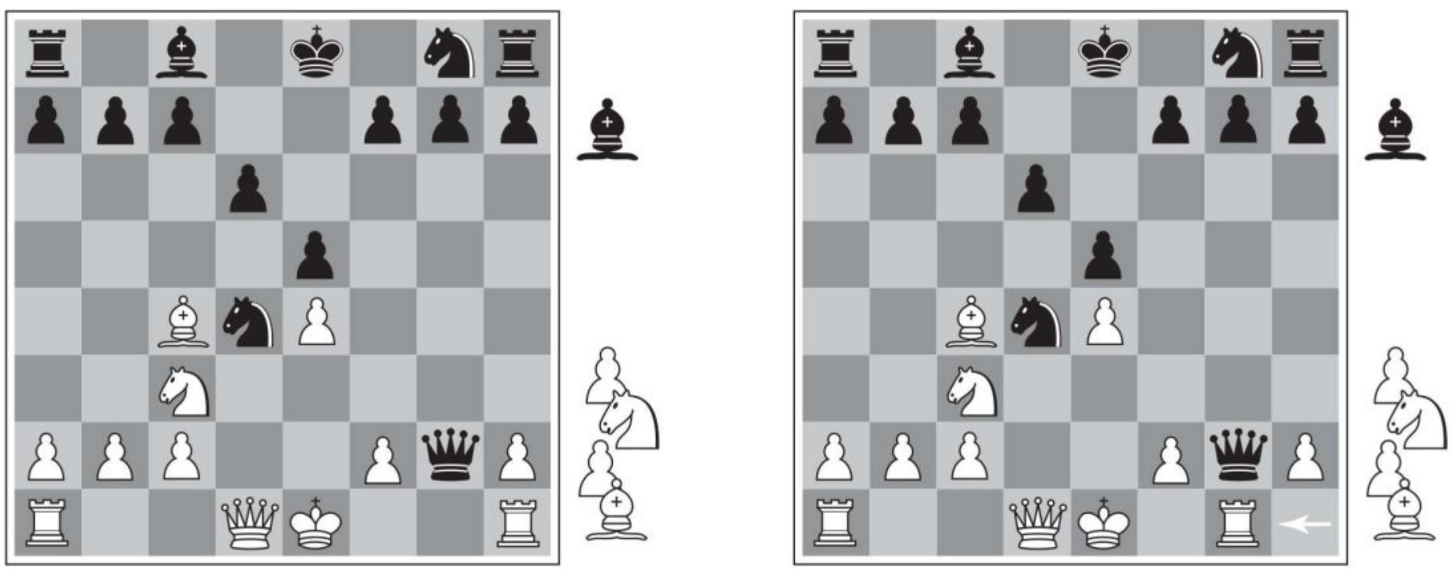
(a) White to move



(b) White to move

Evaluation Functions

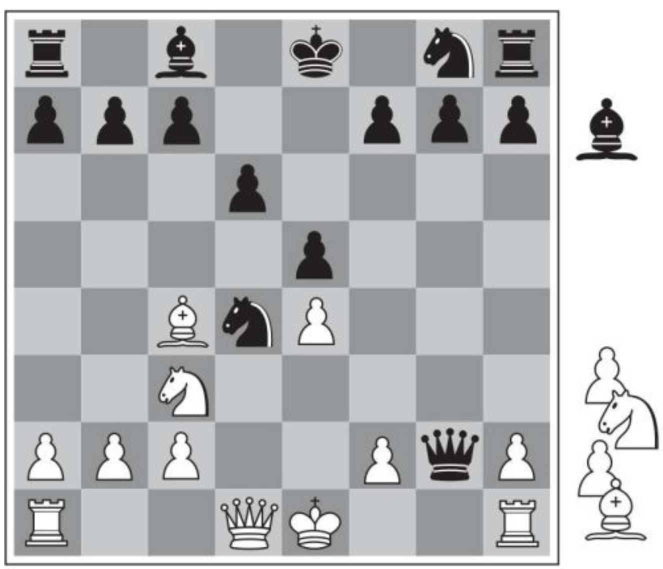
- Two same-score positions (White: -8, Black: -3)



(a) White to move (b) White to move

Evaluation Functions

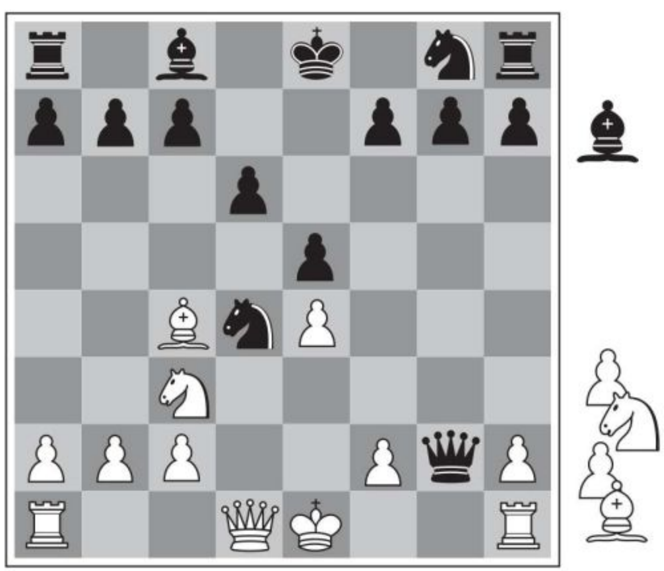
- Two same-score positions (White: -8, Black: -3)
  - (a) Black has an advantage of a knight and two pawns,
  - ⇒ should be enough to win the game



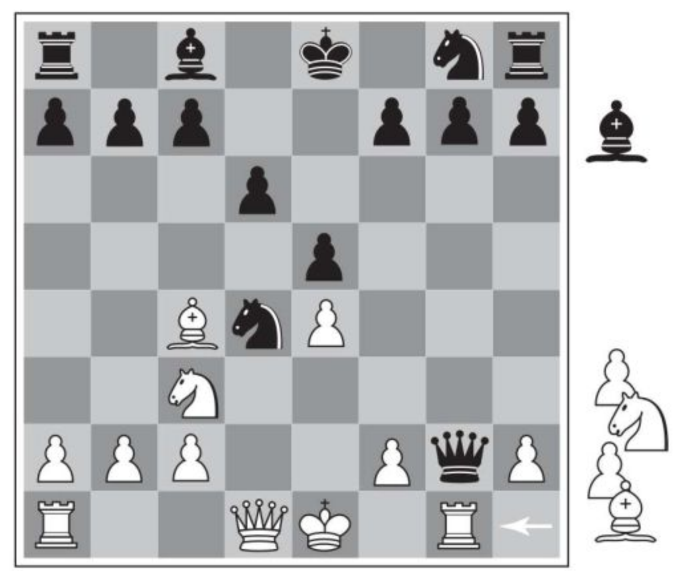
(a) White to move

Evaluation Functions

- Two same-score positions (White: -8, Black: -3)
  - (a) Black has an advantage of a knight and two pawns,
  - ⇒ should be enough to win the game
  - (b) White will capture the queen,
  - ⇒ give it an advantage that should be strong enough to win



(a) White to move

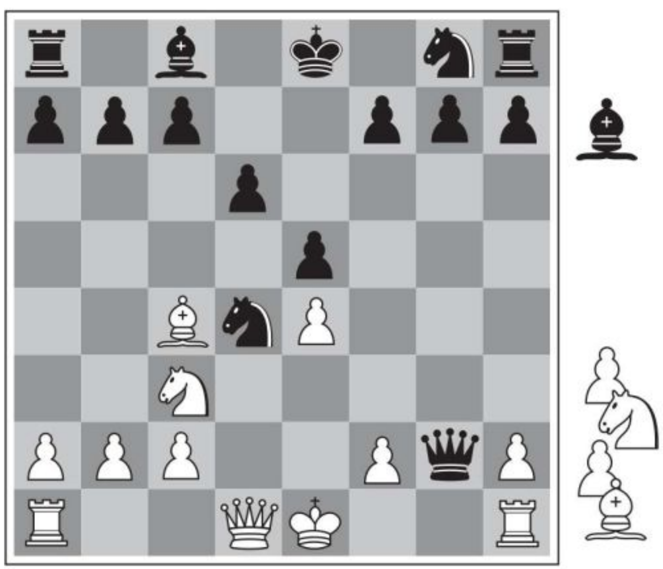


(b) White to move

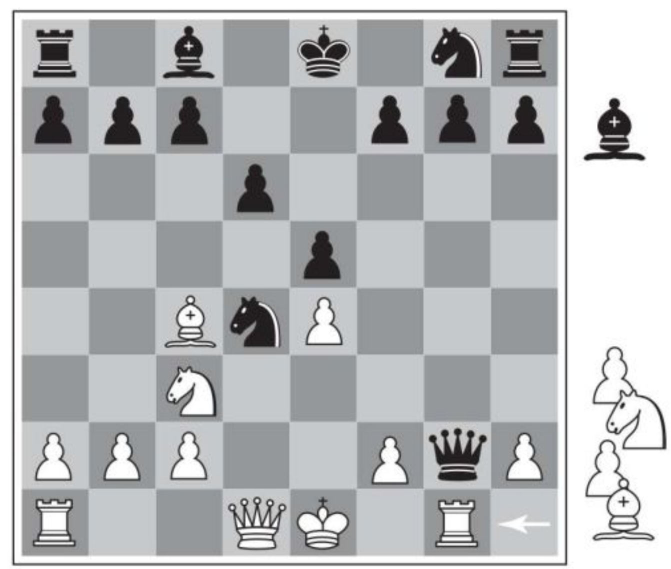
Evaluation Functions

- Two same-score positions (White: -8, Black: -3)
  - (a) Black has an advantage of a knight and two pawns,
  - ⇒ should be enough to win the game
  - (b) White will capture the queen,
  - ⇒ give it an advantage that should be strong enough to win

(bad move)



(a) White to move



(b) White to move

Deterministic Games in Practice

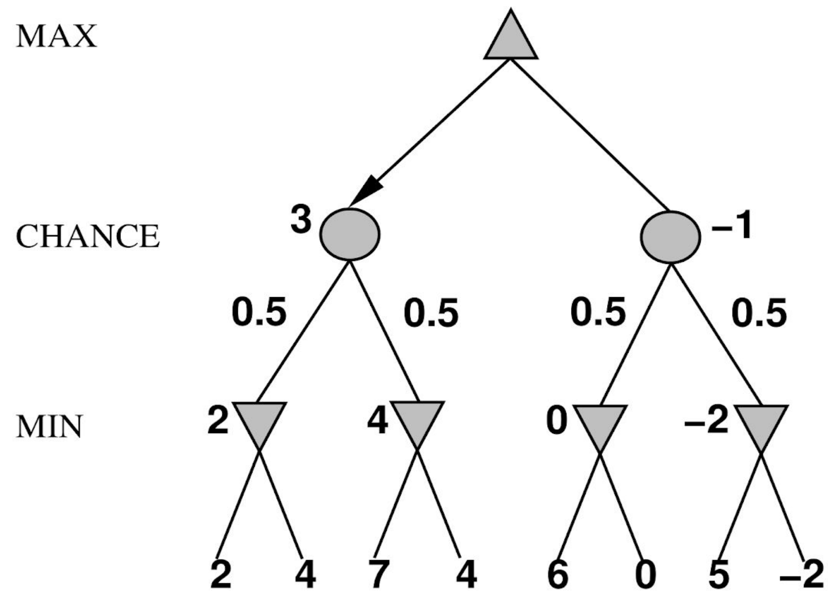
- Checkers: (1994) Chinook ended 40-year-reign of world champion Marion Tinsley
- Chess: (1997) Deep Blue defeated world champion Gary Kasparov in a six-game match
- Go: (2016) AlphaGo beats world champion Lee Sedol

Nondeterministic games

E..g, in backgammon, the dice rolls determine the legal moves

Simplified example with coin-flipping instead of dice-rolling:

minimax value?

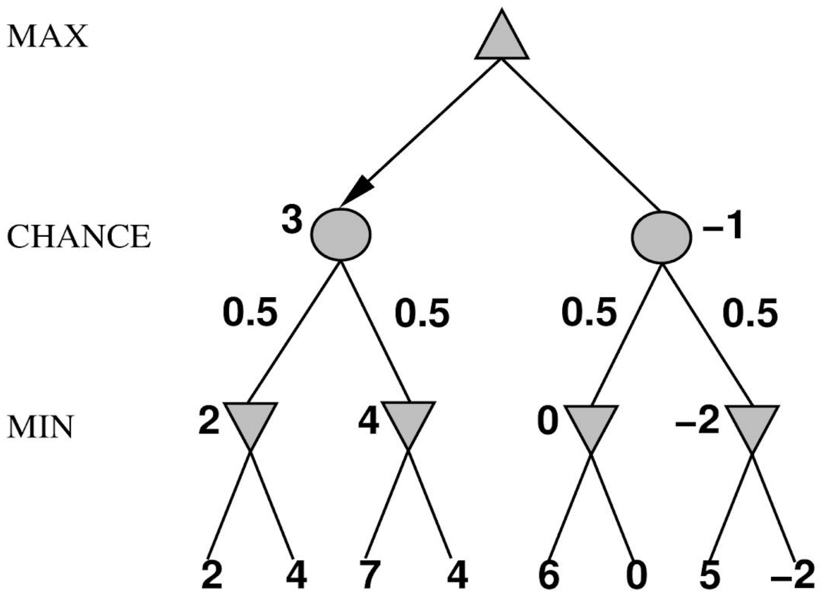


Nondeterministic games

E..g, in backgammon, the dice rolls determine the legal moves

Simplified example with coin-flipping instead of dice-rolling:

Chance nodes take expectations, otherwise like minimax



Cannot calculate definite minimax value, only **expected** values

Expectation

- We can define a function f(X) of a random variable X
- The expected value of a function

$$E\big(f(X)\big) = \sum_x f(X = x)P(X = x)$$

X	P	f
1	1/6	1
2	1/6	2
3	1/6	3
4	1/6	4
5	1/6	5
6	1/6	6

Expectation

- We can define a function f(X) of a random variable X
- The expected value of a function

$$E\big(f(X)\big) = \sum_x f(X = x)P(X = x)$$

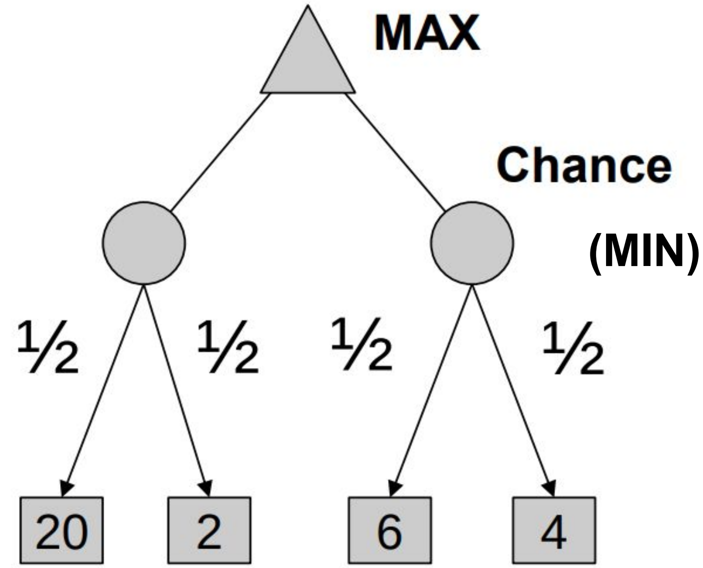
X	P	f
1	1/6	1
2	1/6	2
3	1/6	3
4	1/6	4
5	1/6	5
6	1/6	6

$$1 \times \frac{1}{6} + 2 \times \frac{1}{6} + 3 \times \frac{1}{6} + 4 \times \frac{1}{6} + 5 \times \frac{1}{6} + 6 \times \frac{1}{6}$$

$$= 3.5$$

Game Tree for Stochastic Single-Player Game

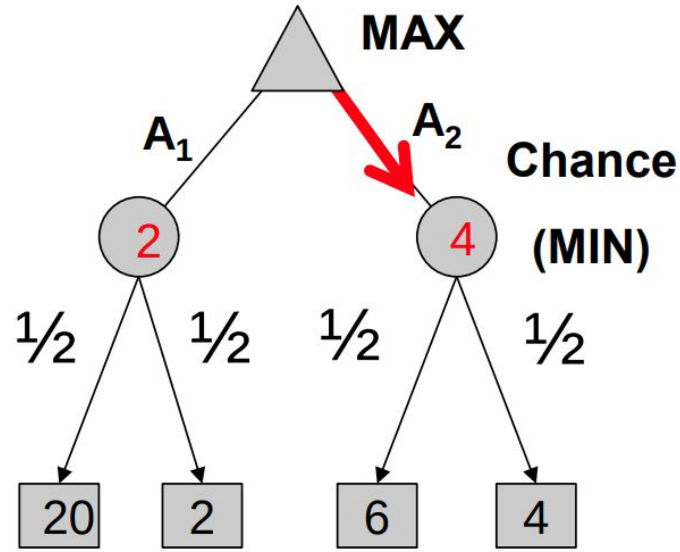
- MAX nodes as before
- Chance nodes: Environment selects an action with some probability
- Minimax strategy: Pick MIN value move at each chance node



Game Tree for Stochastic Single-Player Game

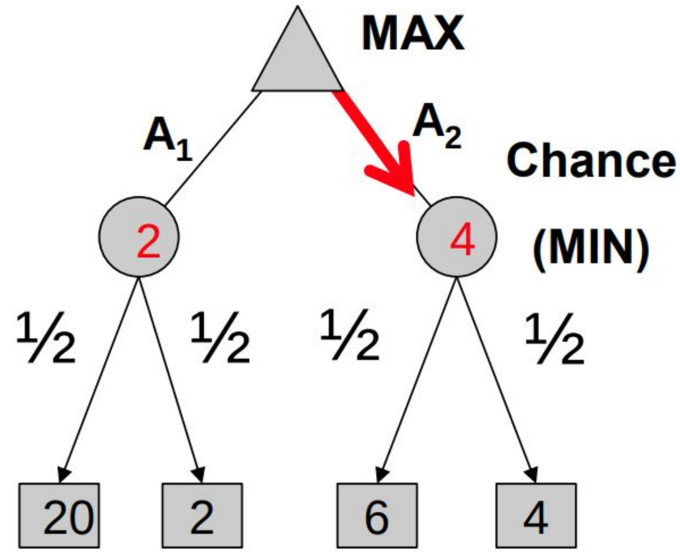
- Suppose MAX always choose A_2
- Average utility = 6/2+4/2 = 5

- If MAX had chosen A1 ?



Game Tree for Stochastic Single-Player Game

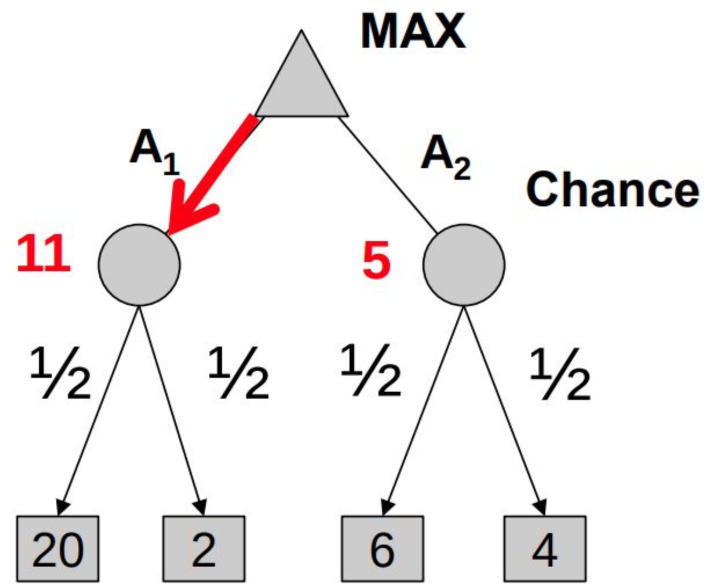
- Suppose MAX always choose A_2
- Average utility = 6/2+4/2 = 5



- If MAX had chosen A1 ?
- Average utility = 11

Expectimax Search

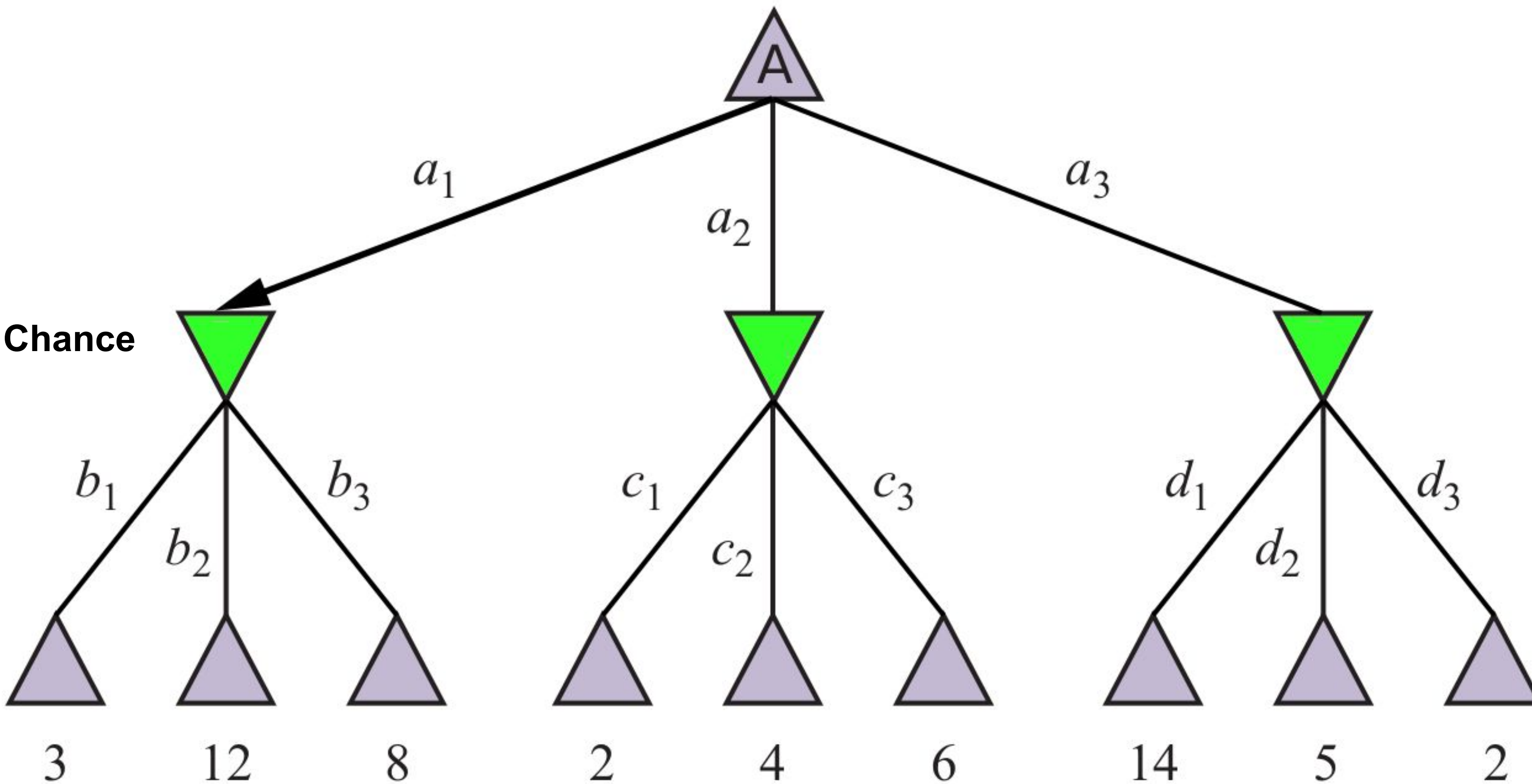
- Expectimax search: Chance nodes take average (expectation) of value of children
- **MAX picks move with maximum expected value**



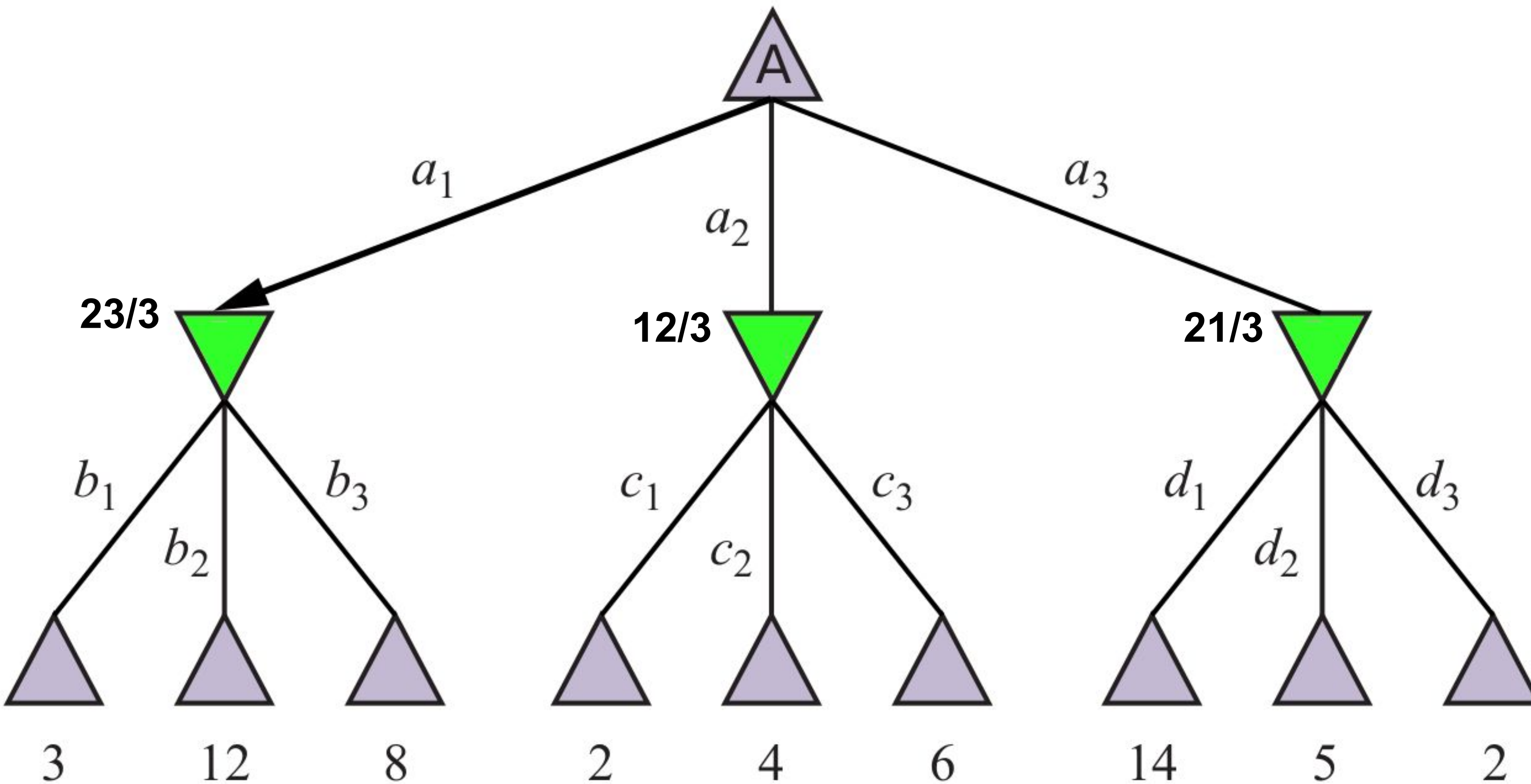
Principle of maximum expected utility: An agent should chose the action which maximizes its expected utility, given its knowledge

Expectimax

Expectimax algorithm is a variation of Minimax, and it is used to maximize expected utility.



Expectimax



Expectimax

Advantages:

- Expectimax is able to model the **non-optimal opponents**.
- Unlike Minimax, Expectimax can take risks and end up in a state with a higher utility as the opponents are random (not optimal).

Expectimax

Disadvantages:

- Expectimax is not optimal, It may lead to agents losing (ending up in a state with lesser utility).
- Expectimax requires the full search tree to be explored. There is no type of pruning that can be done because of the unexplored nodes can influence expectimax values. Thus, it is slow.

Properties of Expectimax

Time Complexity:

Space Complexity:

Properties of Expectimax

Time Complexity: $O(b^m)$

Space Complexity: O(b*m)

Applications: Useful in an environment where **<u>actions of one of the agents</u>** are random. Following are the examples,

- For example, in **Minesweeper** can be modeled by player agent as the maximizer and the mines as the chance nodes.