Games

Outline

I. Game as adversarial search

II. The minimax algorithm

* Figures/images are from the textbook site (or by the instructor) . Otherwise, the source is specifically cited unless citation would make little sense due to the triviality of generating such an image.

Games

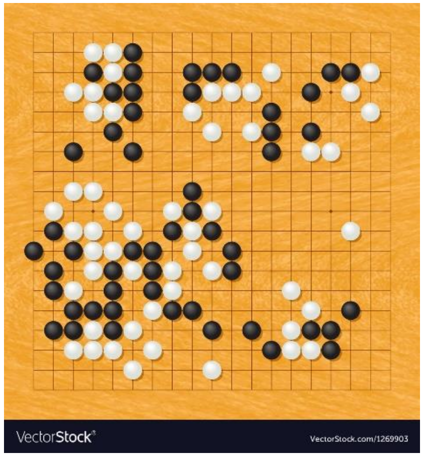
Competitive environments: goals are in conflict.

Adversarial search problems (games)

Games

Competitive environments: goals are in conflict.

Adversarial search problems (games)



Why Study Games?

- Multiagent environment
  - Aggregate of a large number of agents for predictions (e.g., price rise).
  - Nondeterminism made by adversarial agents.
  - Introduction of new modeling techniques.

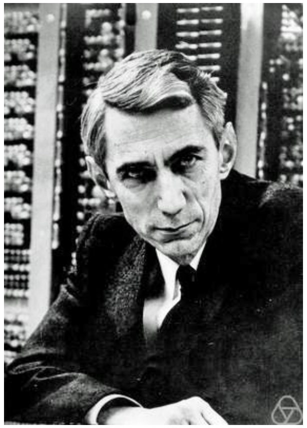
Why Study Games?

- Multiagent environment
  - Aggregate of a large number of agents for predictions (e.g., price rise).
  - Nondeterminism made by adversarial agents.
  - Introduction of new modeling techniques.
- Mathematical game theory – an important branch of economics.

Why Study Games?

- Multiagent environment
  - Aggregate of a large number of agents for predictions (e.g., price rise).
  - Nondeterminism made by adversarial agents.
  - Introduction of new modeling techniques.
- Mathematical game theory – an important branch of economics.
- Appealing subject for study in AI.
  - Fun and entertaining.
  - Hard – engaging the intellectual faculties of humans.
  - Abstract nature – easy to represent with small number of actions.

History of Computer Games



Claude Shannon (MIT)

"Father of information theory"
National Medal of Science (1966)

1950 Claude Shannon, *Programming a Computer for Playing Chess.*

1956 John McCarthy conceives alpha-beta search.

1982 BELLE becomes the first chess program to achieve master status.

1984 Judea Pearl, *Heuristics*.

1997 Deep Blue (IBM) defeats world chess champion Garry Kasparov.

2017 AlphaGo (Alphabet) defeats world's no. 1 Go player Ke Jie.

- Visual pattern recognition
- Reinforcement learning
- Neural networks
- Monte Carlo tree search

2018 AlphaZero (Alphabet) defeats top programs in Go, chess, shogi.

2019 Pluribus (CMU) defeats top-ranked players in Texas hold'em games with six players.

* Photo from https://en.wikipedia.org/wiki/Claude_Shannon.

Types of Games

- ♣ Games with deterministic, perfect information (e.g., chess, go, checkers)
- ♣ Stochastic games (e.g., backgammon)
- ♣ Partially observable games (e.g., bridge, poker)

Two-Player Game

- Perfect information – fully observable.
- Zero sum – what is good for one player is just as bad for the other.

move ⇔ action
position ⇔ state

MAX and MIN: two players.

Formal Definition of a Game

- s_0: initial state – game setup.

Formal Definition of a Game

- s_0: initial state – game setup.

At a state s:

- TO-MOVE(s): the player to move in the state.

Formal Definition of a Game

- s_0: initial state – game setup.

At a state s:

- TO-MOVE(s): the player to move in the state.
- ACTIONS(s): the set of legal moves in the state.

Formal Definition of a Game

- s_0: initial state – game setup.

At a state s:

- TO-MOVE(s): the player to move in the state.
- ACTIONS(s): the set of legal moves in the state.
- RESULT(s, a): the transition model defining the next state from taking action a.

Formal Definition of a Game

- s_0: initial state – game setup.

At a state s:

- TO-MOVE(s): the player to move in the state.
- ACTIONS(s): the set of legal moves in the state.
- RESULT(s, a): the transition model defining the next state from taking action a.
- IS-TERMINAL(s): to test if the game is over, i.e., if s is a terminal state.

Formal Definition of a Game

- s_0: initial state – game setup.

At a state s:

- TO-MOVE(s): the player to move in the state.
- ACTIONS(s): the set of legal moves in the state.
- RESULT(s, a): the transition model defining the next state from taking action a.
- IS-TERMINAL(s): to test if the game is over, i.e., if s is a terminal state.
- UTILITY (s, p): a utility function to return a value to the player p if the game ends in terminal state s.

Formal Definition of a Game

- s_0: initial state – game setup.

At a state s:

- TO-MOVE(s): the player to move in the state.
- ACTIONS(s): the set of legal moves in the state.
- RESULT(s, a): the transition model defining the next state from taking action a.
- IS-TERMINAL(s): to test if the game is over, i.e., if s is a terminal state.
- UTILITY (s, p): a utility function to return a value to the player p if the game ends in terminal state s.

e.g., in chess, win (1), loss (0), draw (1/2)

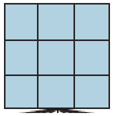
Total payoff for all players is constant (zero-sum game):

$$1 + 0 = 0 + 1 = \frac{1}{2} + \frac{1}{2} = 1$$

State Space Graph (Tic-Tac-Toe)

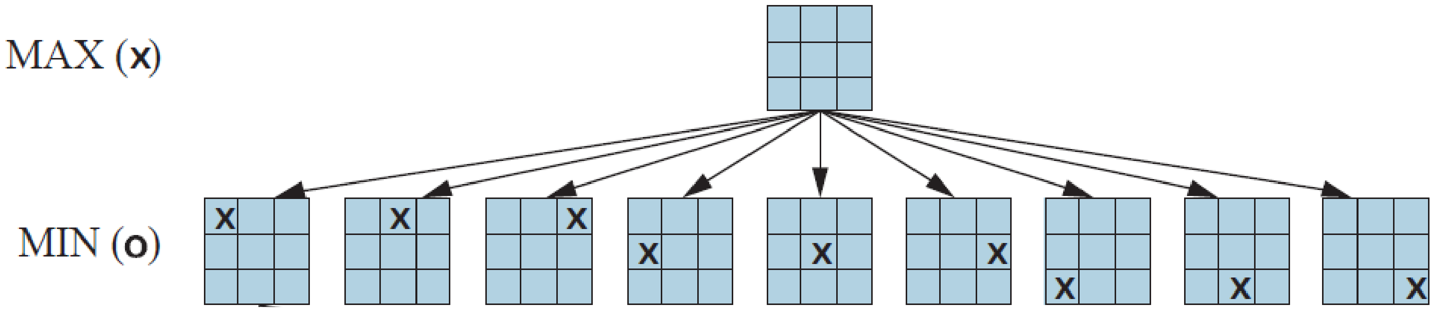
Vertices ↔ states and edges ↔ moves

MAX (**x**)



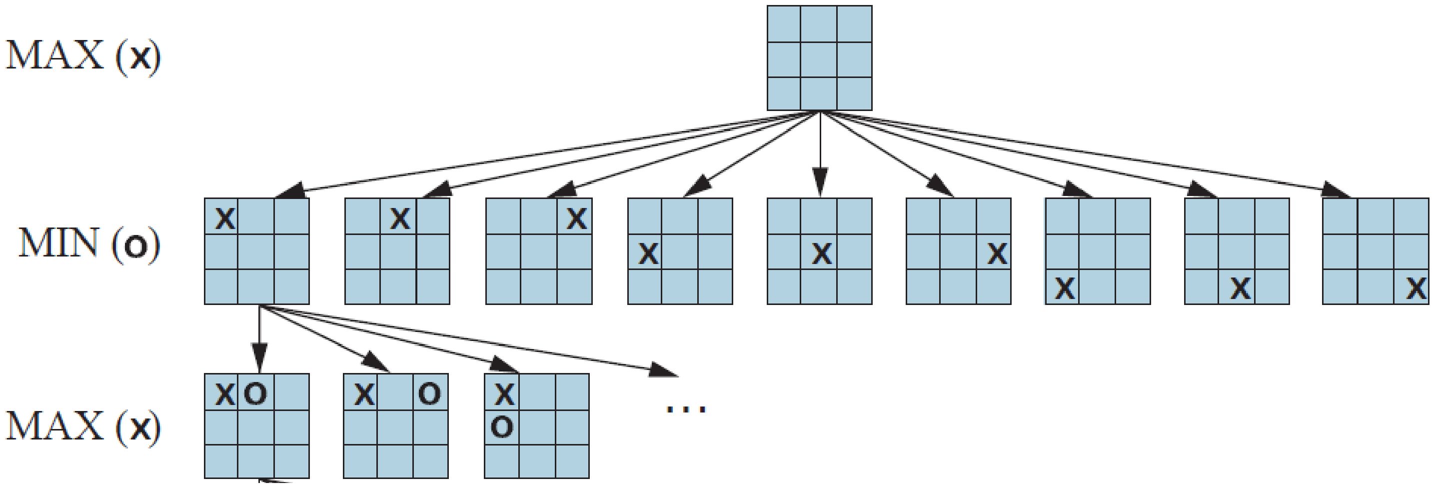
State Space Graph (Tic-Tac-Toe)

Vertices ↔ states   and   edges ↔ moves



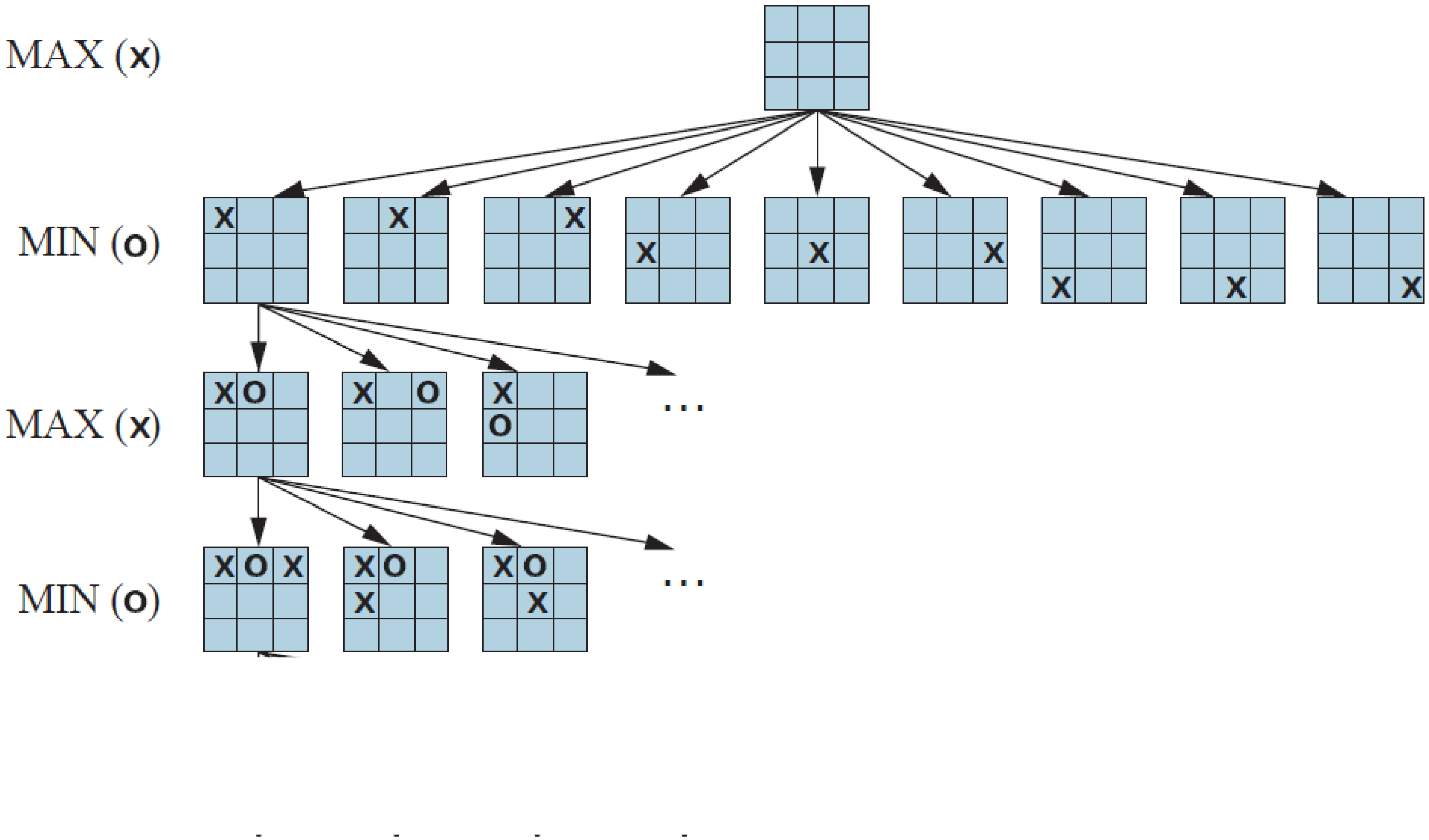
State Space Graph (Tic-Tac-Toe)

Vertices ⟷ states and edges ⟷ moves



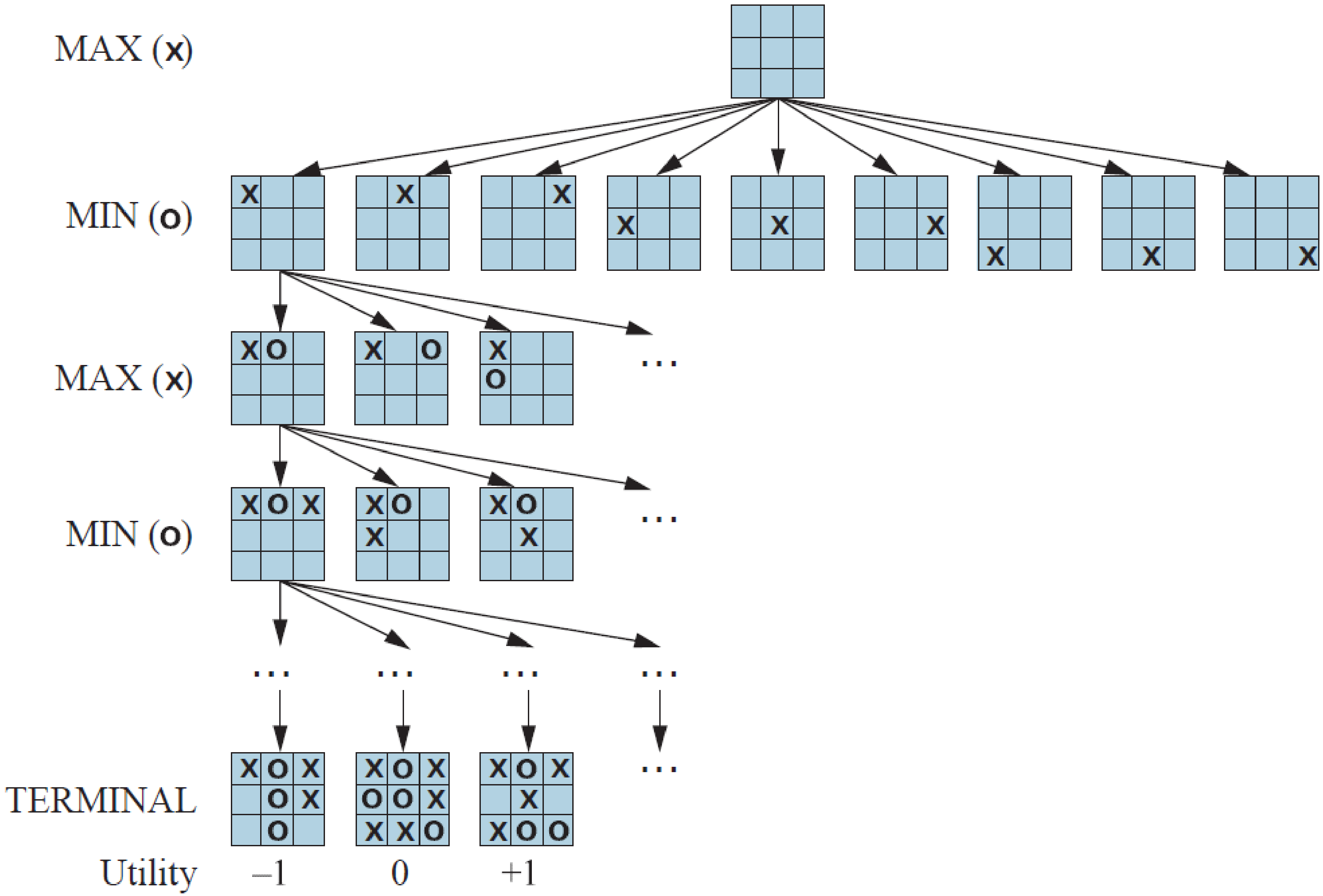
State Space Graph (Tic-Tac-Toe)

Vertices ↔ states   and   edges ↔ moves



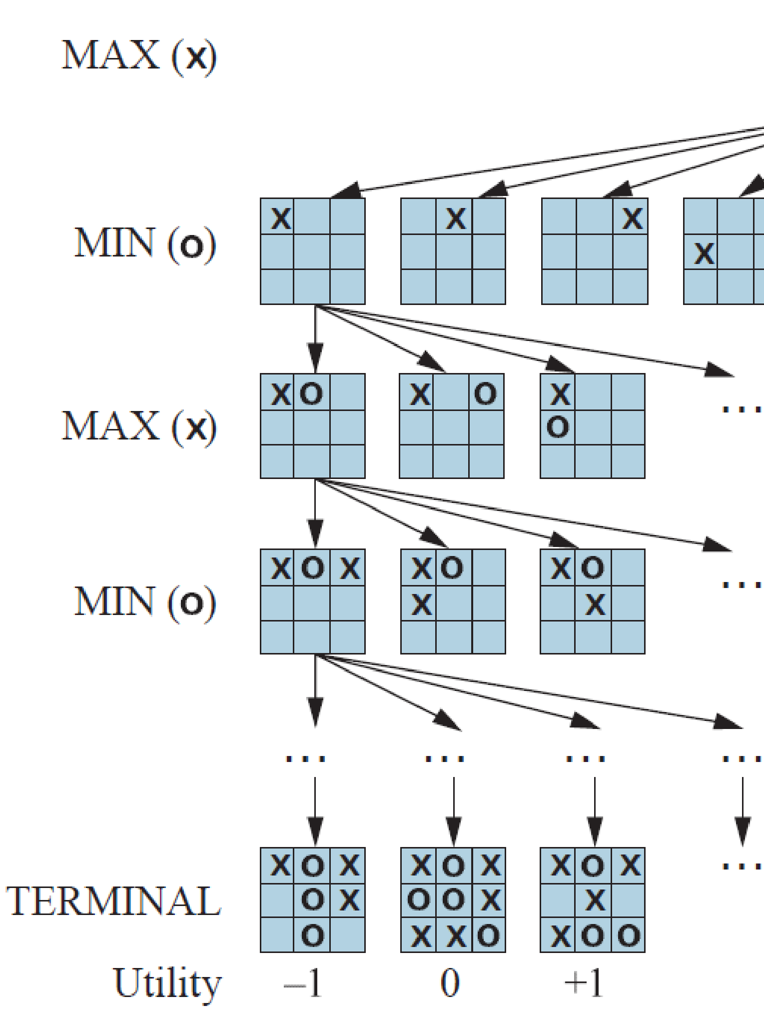
State Space Graph (Tic-Tac-Toe)

Vertices ⟷ states and edges ⟷ moves



State Space Graph (Tic-Tac-Toe)

Vertices ⟷ states and edges ⟷ moves

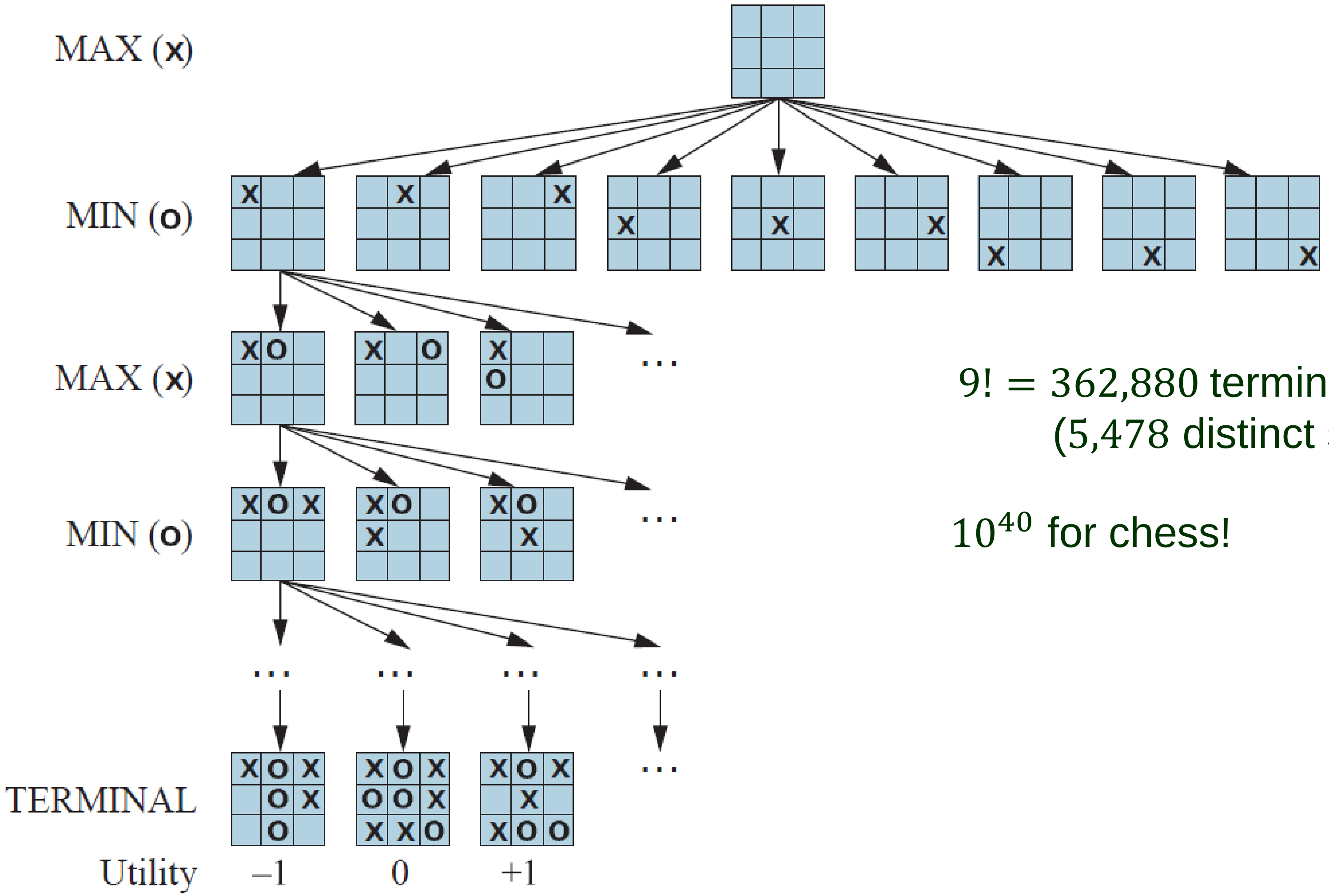


$9! = 362{,}880$ terminal nodes
(5,478 distinct states)

10^{40} for chess!

State Space Graph (Tic-Tac-Toe)

Vertices ⟷ states and edges ⟷ moves

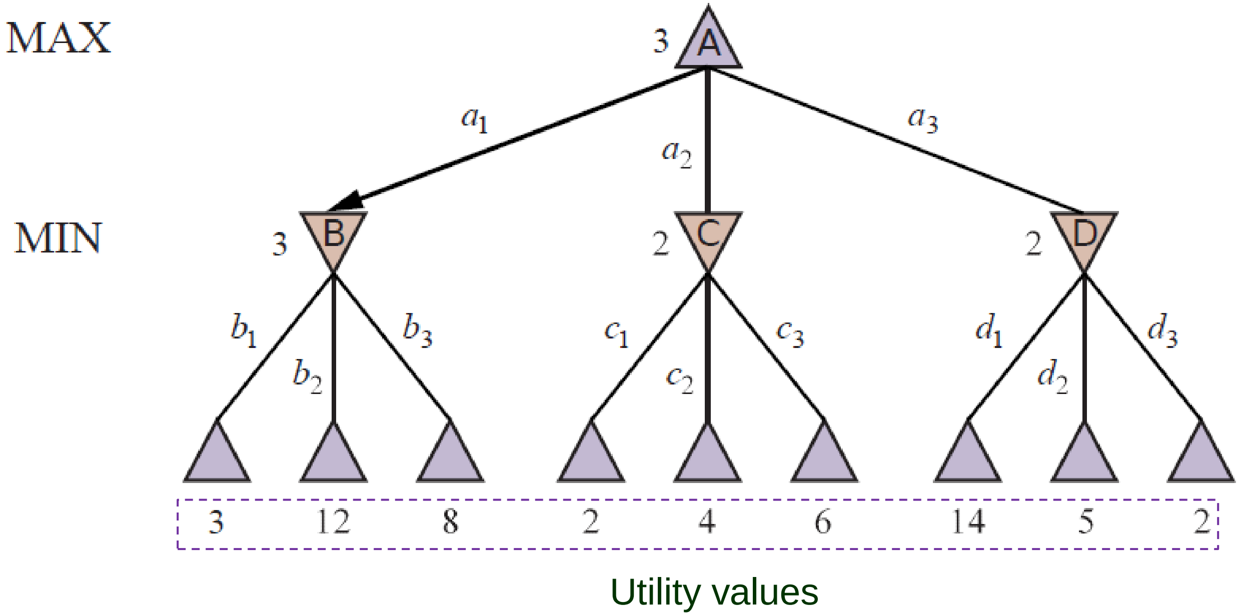


9! = 362,880 terminal nodes
(5,478 distinct states)

10^{40} for chess!

Two-Ply Game Tree

Ply: one move by a player



Optimal Strategy

Work out the minimax value of every state s in the tree,

$$\text{MINIMAX}(s)$$

assuming both players play optimally:

- MAX moves to a state of maximum value at its turn;
- MIN moves to a state of minimum value at its turn.

Optimal Strategy

Work out the minimax value of every state s in the tree,

$$\text{Minimax}(s)$$

assuming both players play optimally:

- Max moves to a state of maximum value at its turn;
- Min moves to a state of minimum value at its turn.

$$\text{Minimax}(s) =$$

Optimal Strategy

Work out the minimax value of every state s in the tree,

$$\text{MINIMAX}(s)$$

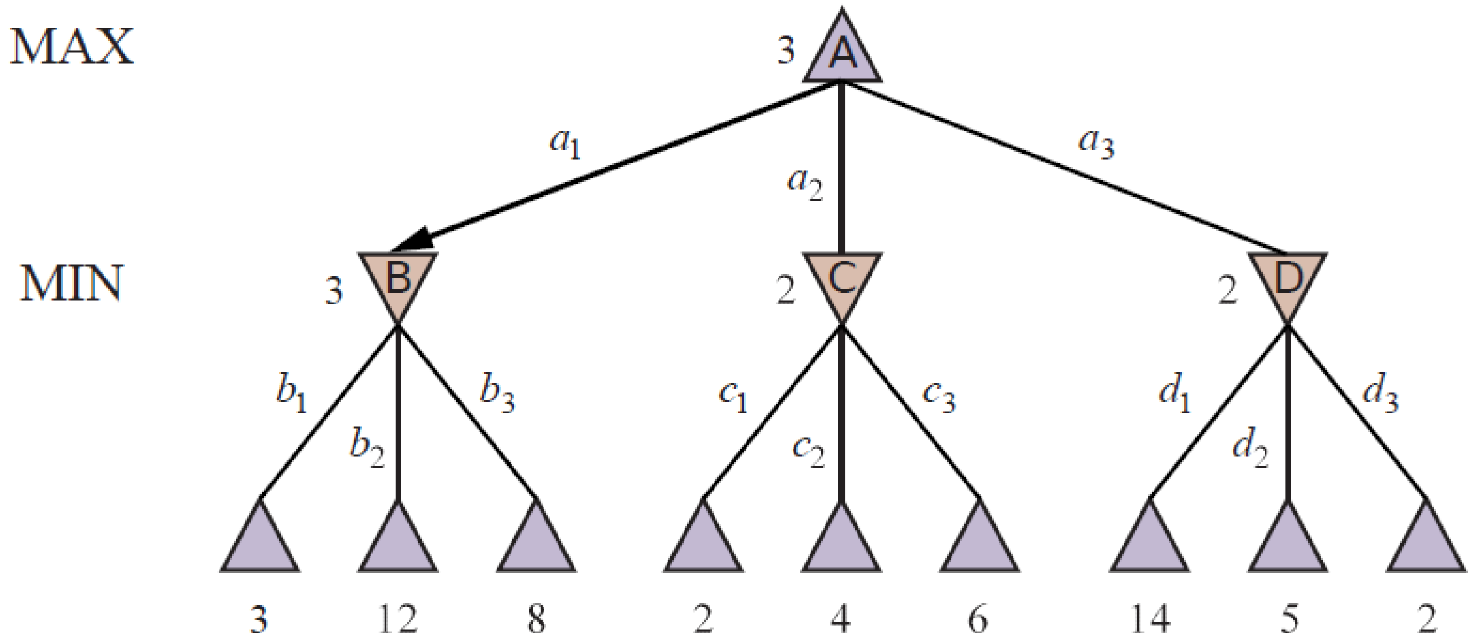
assuming both players play optimally:

- MAX moves to a state of maximum value at its turn;
- MIN moves to a state of minimum value at its turn.

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

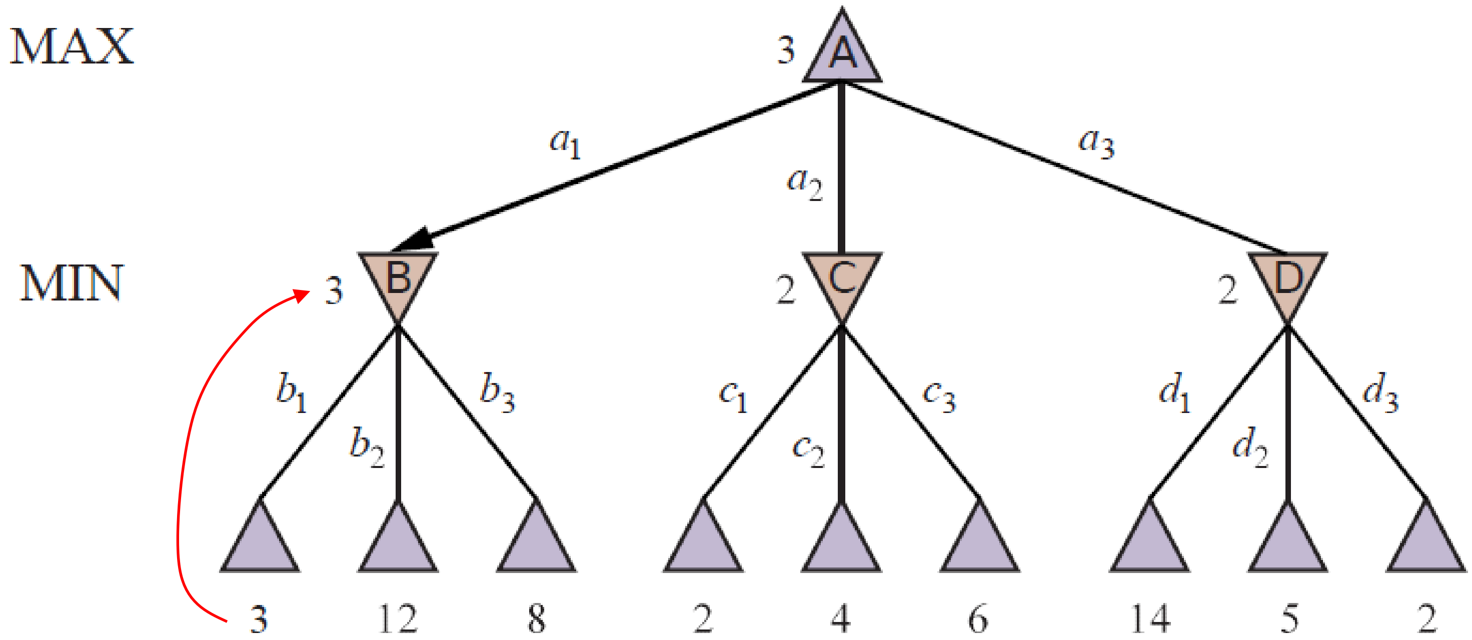
Minimax Value at Min Nodes

MIN: choose a move to a MAX node with the lowest value.



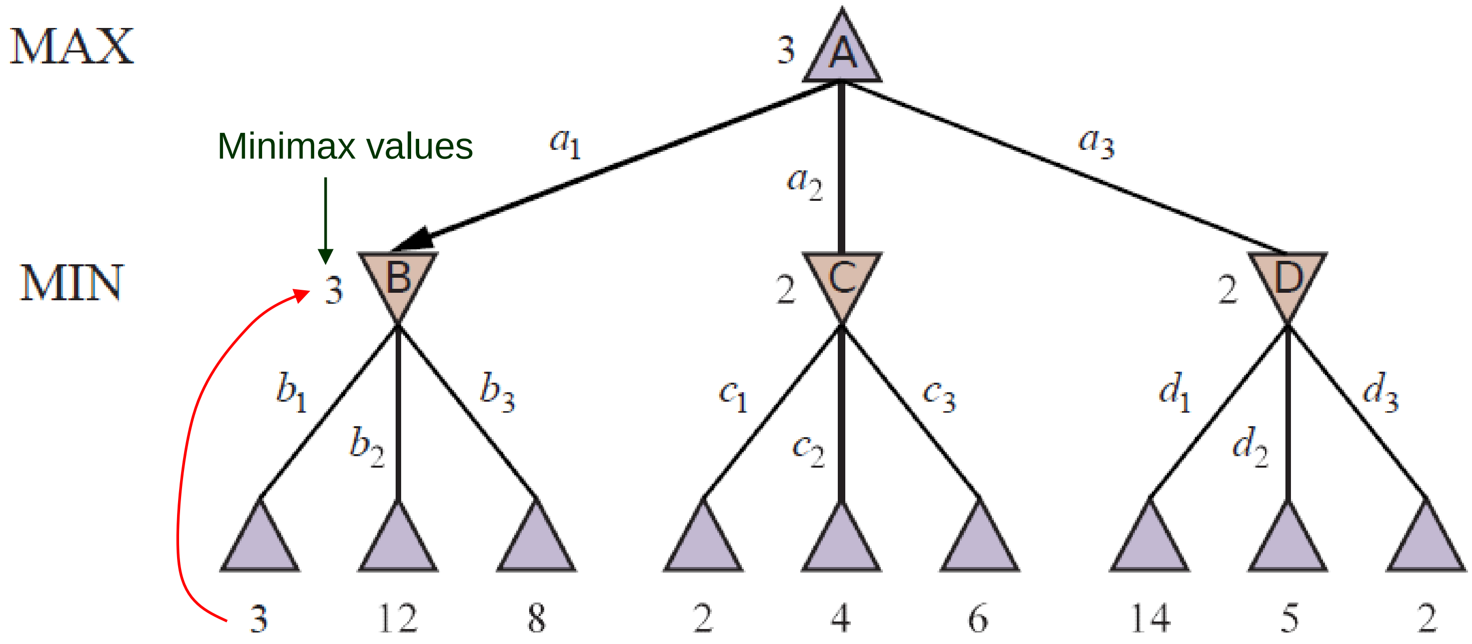
Minimax Value at Min Nodes

MIN: choose a move to a MAX node with the lowest value.



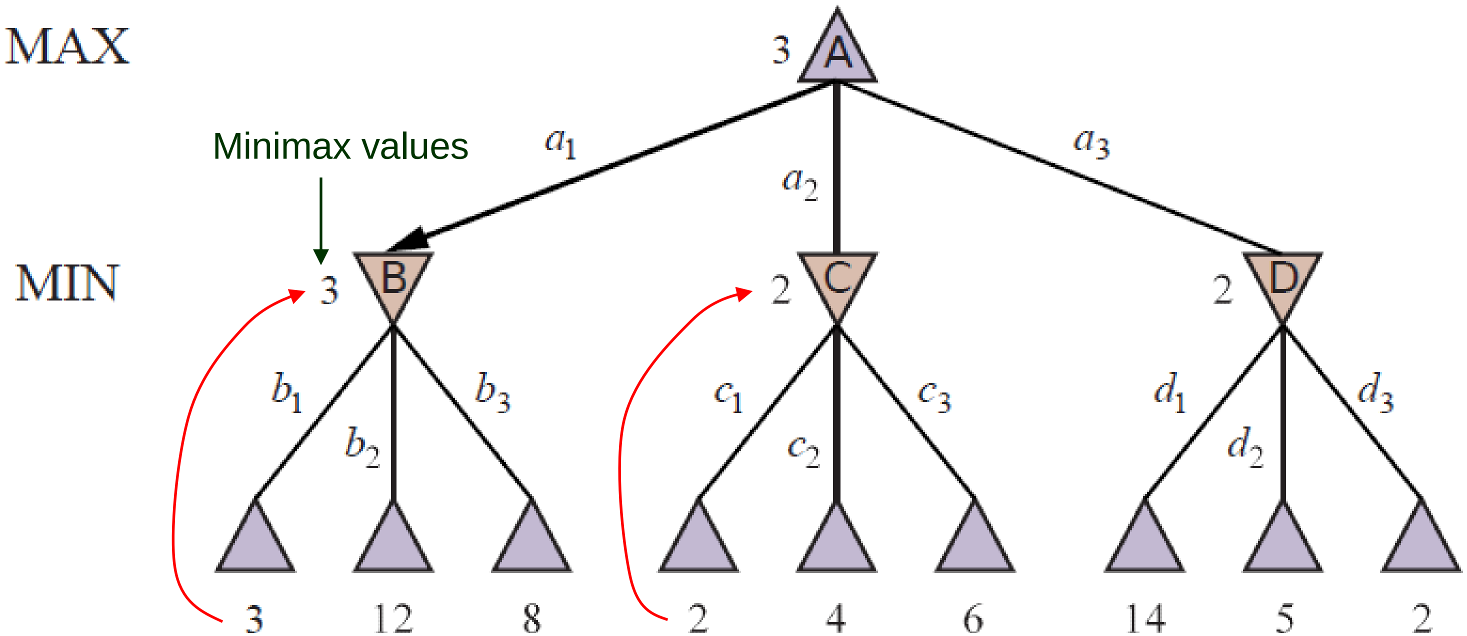
Minimax Value at Min Nodes

MIN: choose a move to a MAX node with the lowest value.



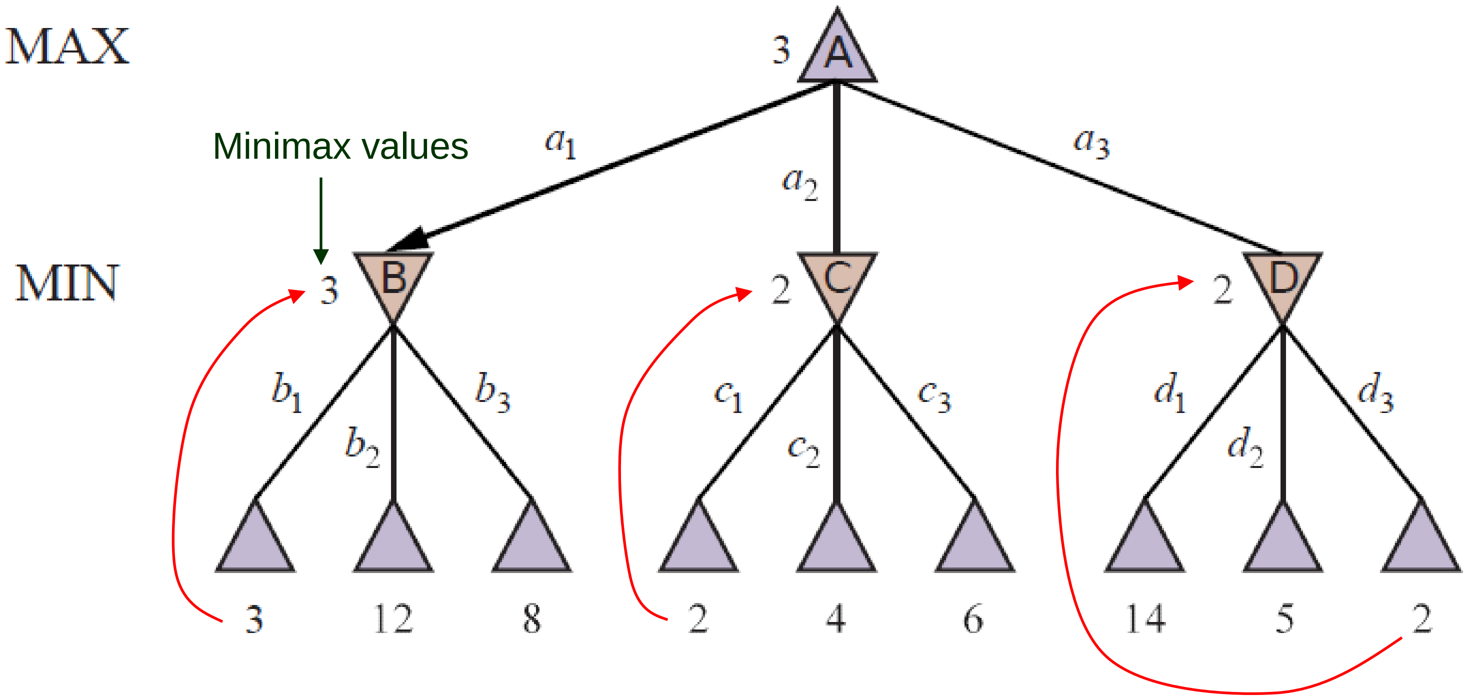
Minimax Value at Min Nodes

MIN: choose a move to a MAX node with the lowest value.



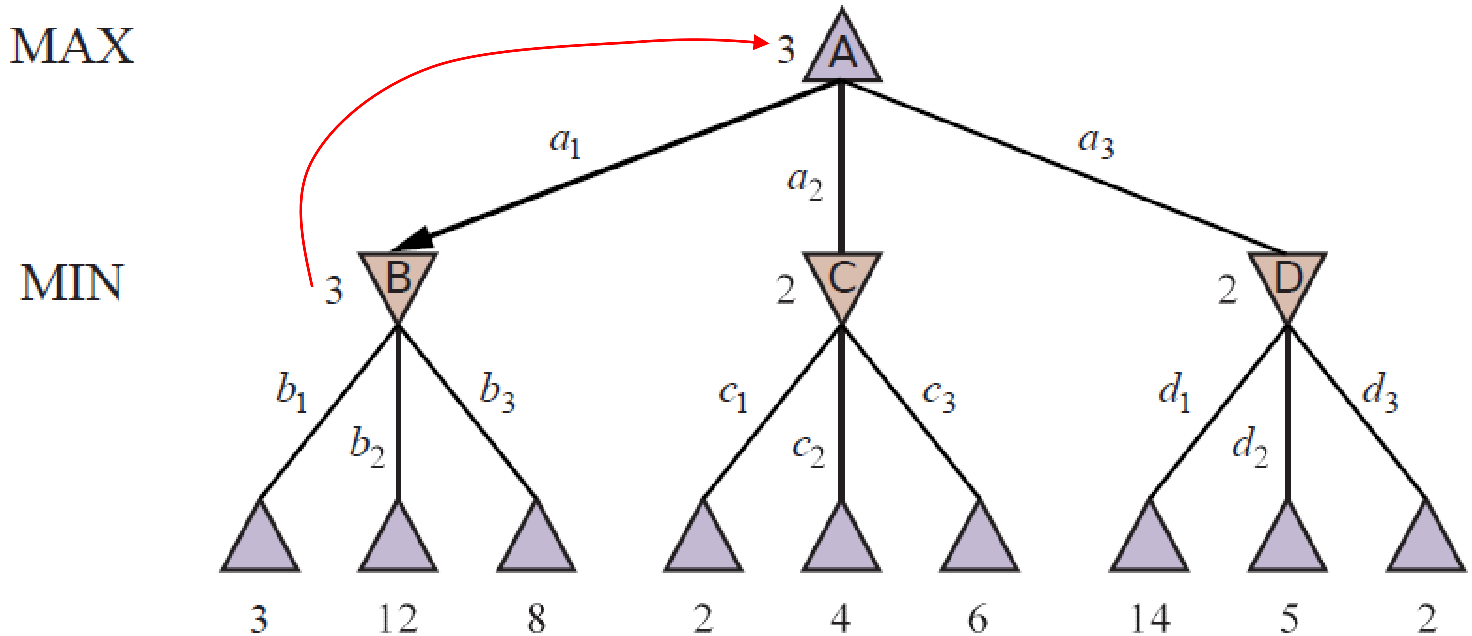
Minimax Value at Min Nodes

MIN: choose a move to a MAX node with the lowest value.

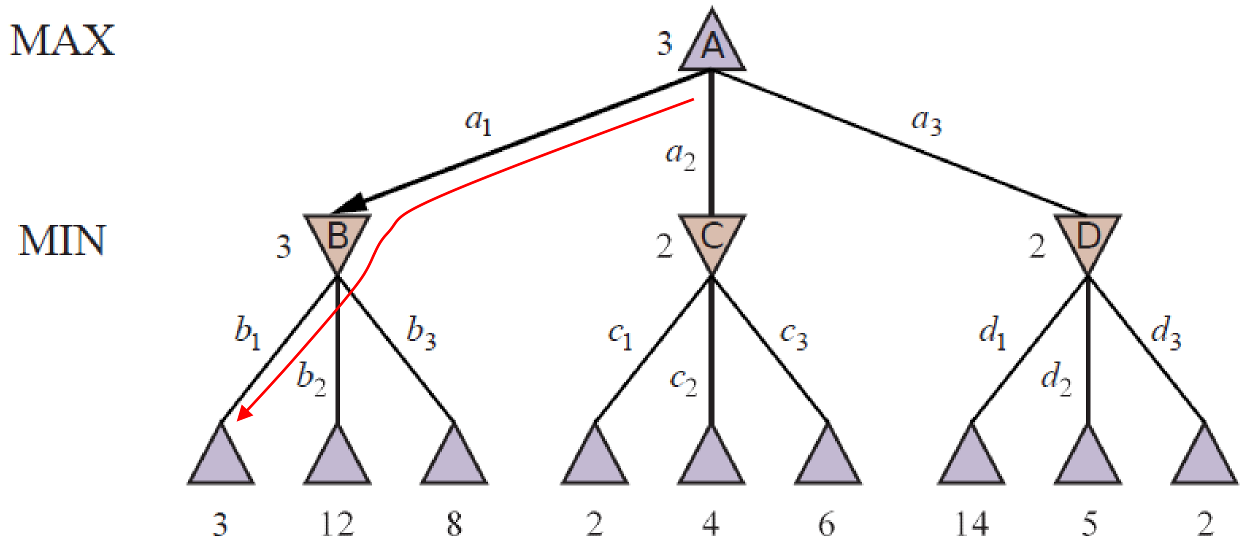


Minimax Value at a Max Node

MAX: choose a move to a MIN node with the highest value.



Solution of the Game



Best move for MAX: a_1

Best move for MIN in response: b_1

The Minimax Search Algorithm

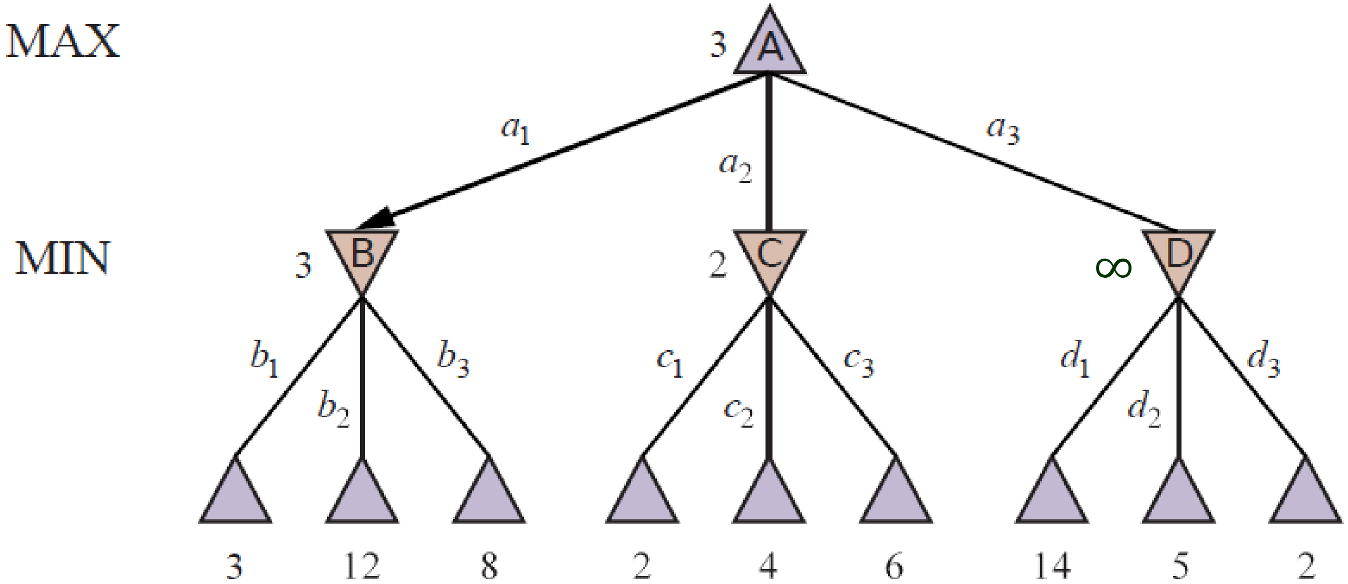
```
function MINIMAX-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state)
  return move

function MAX-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
    if v2 > v then
      v, move ← v2, a
  return v, move

function MIN-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
    if v2 < v then
      v, move ← v2, a
  return v, move
```

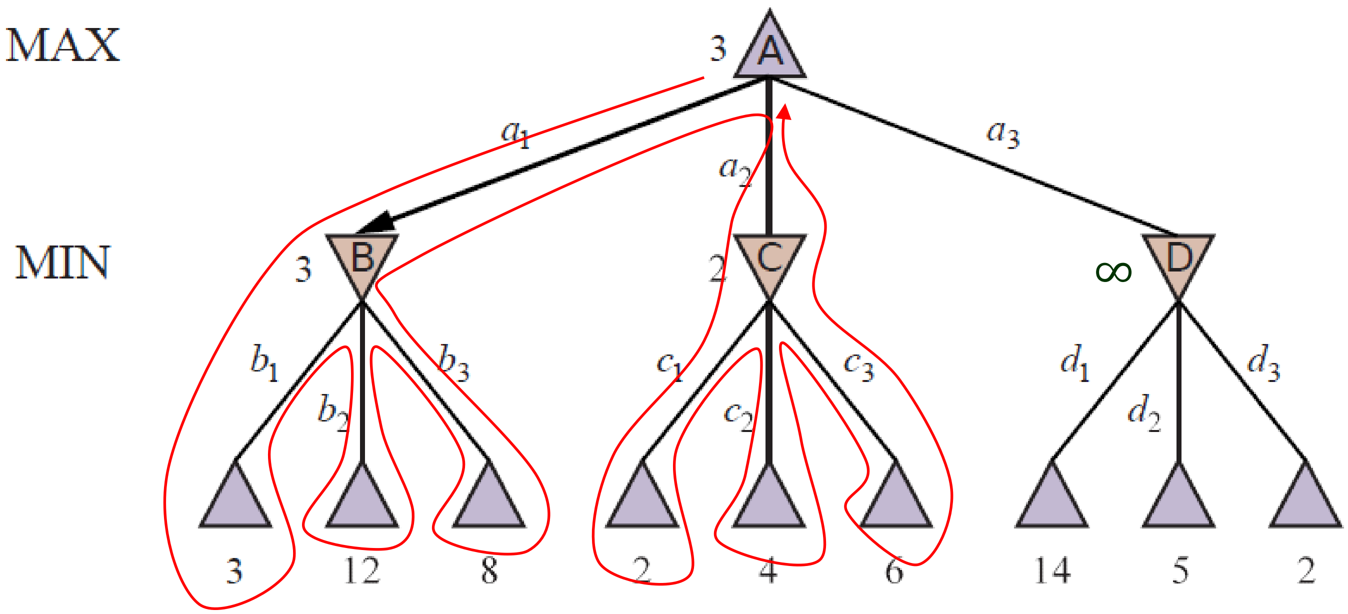
Algorithm Execution

Depth-first search with backed-up value on return from a node.



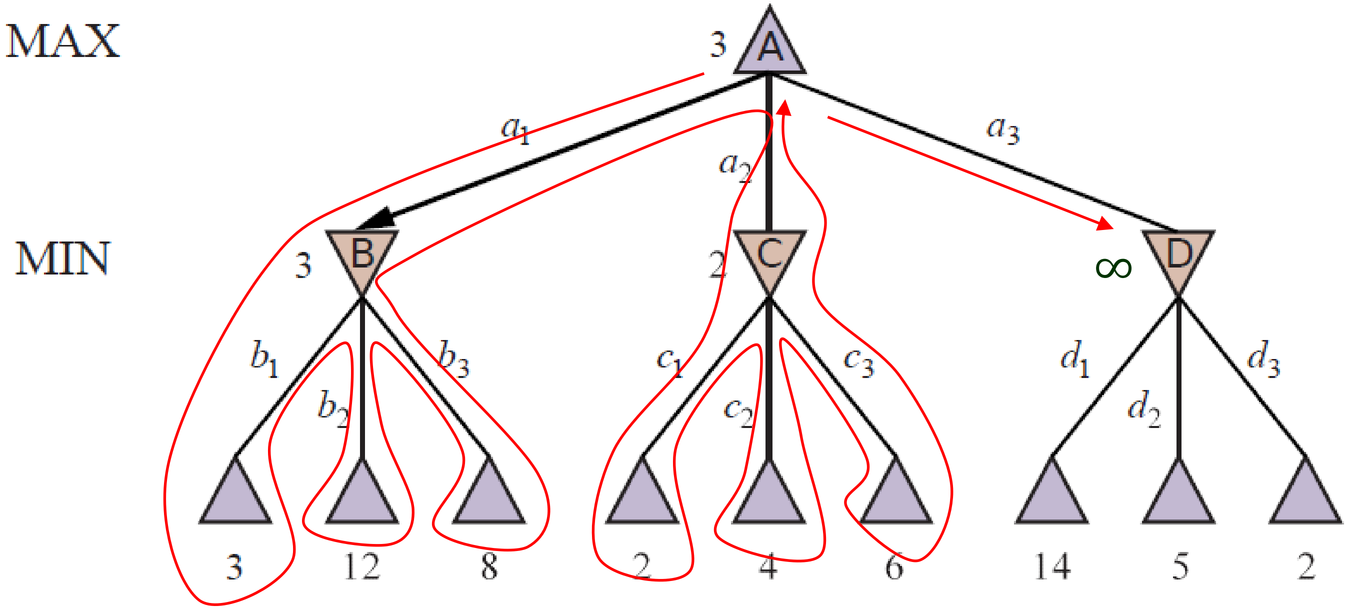
Algorithm Execution

Depth-first search with backed-up value on return from a node.



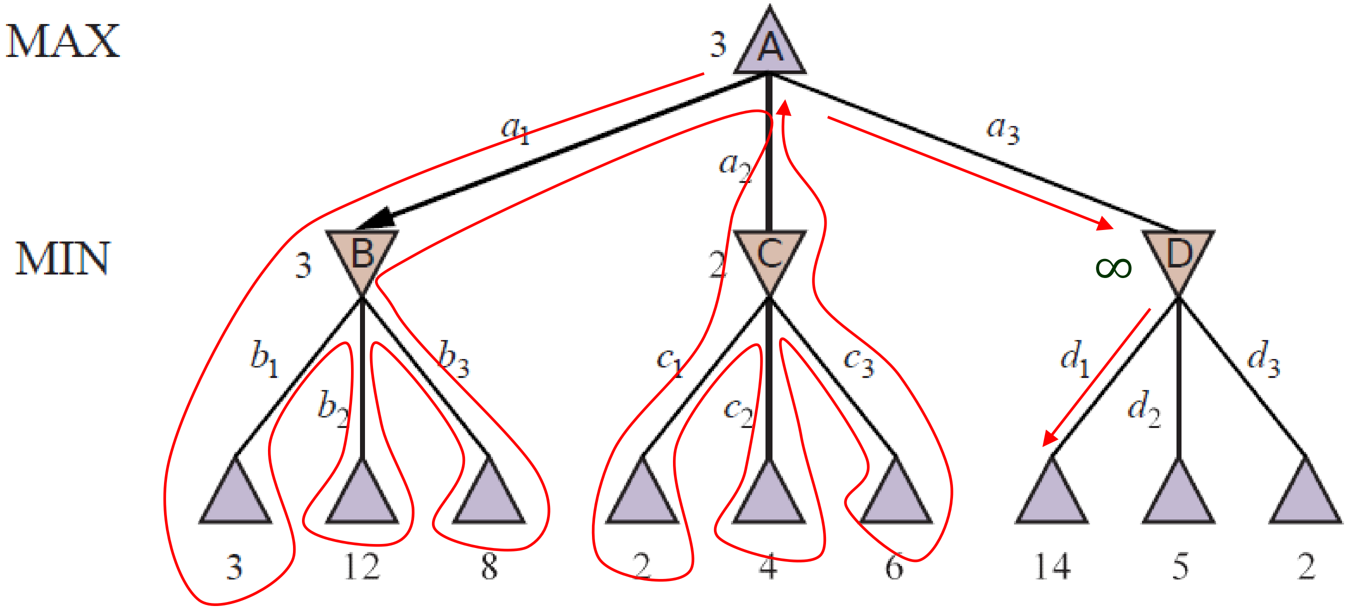
Algorithm Execution

Depth-first search with backed-up value on return from a node.



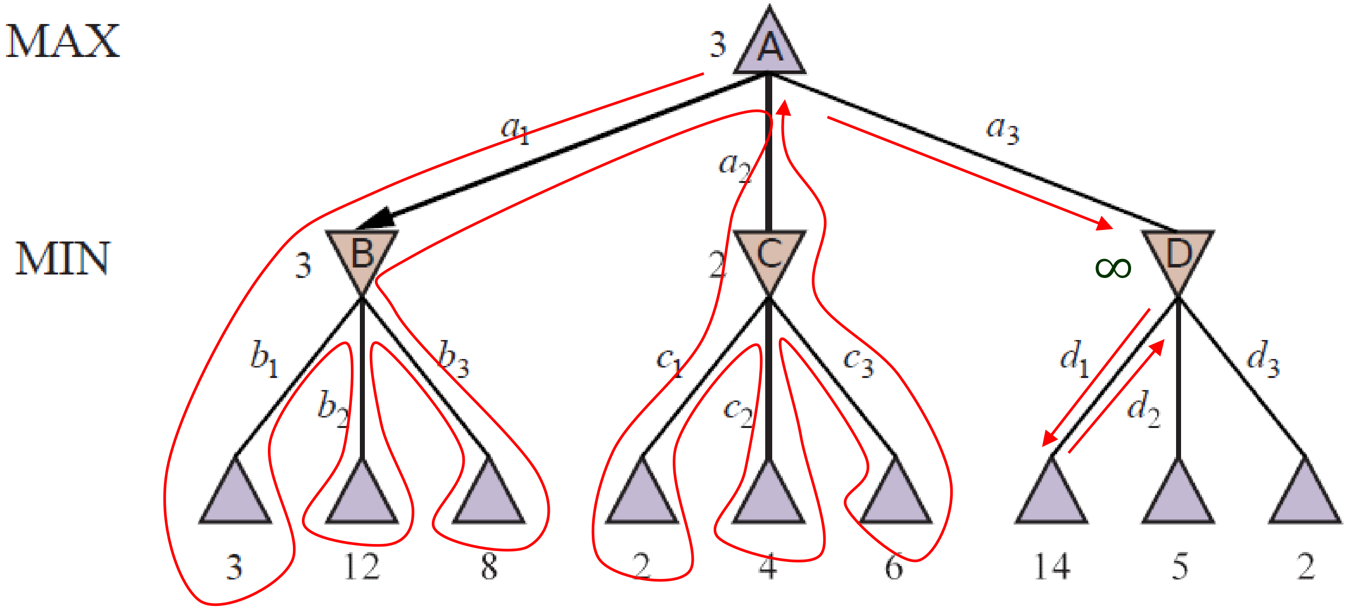
Algorithm Execution

Depth-first search with backed-up value on return from a node.



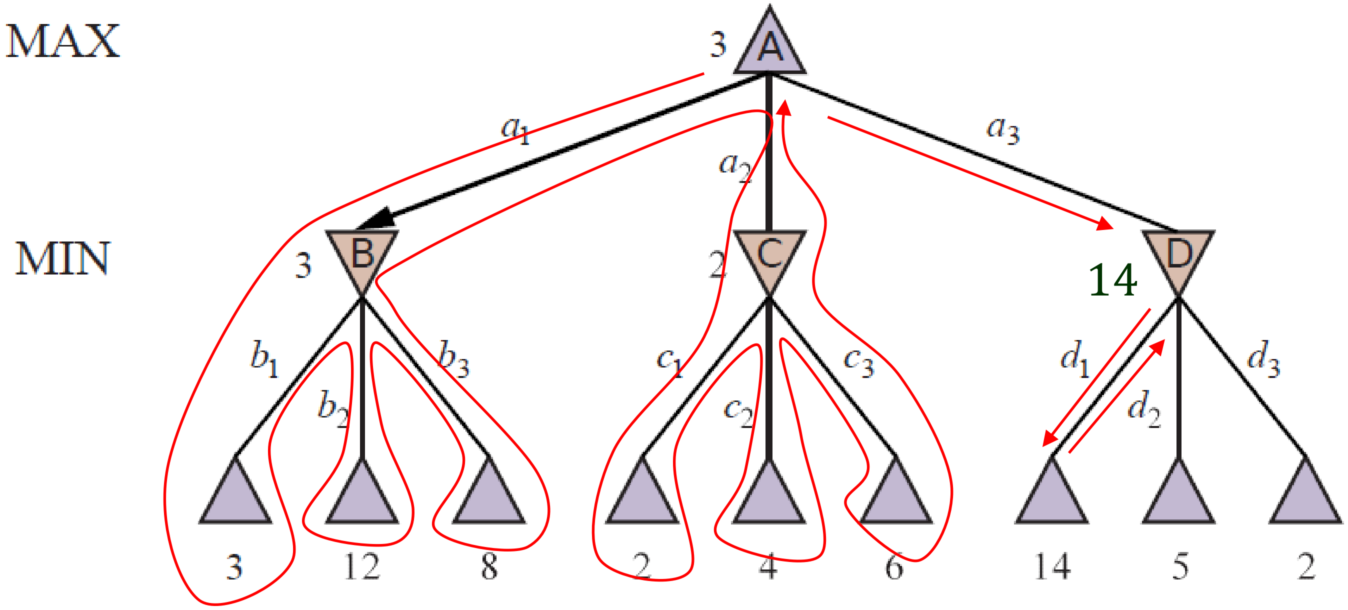
Algorithm Execution

Depth-first search with backed-up value on return from a node.



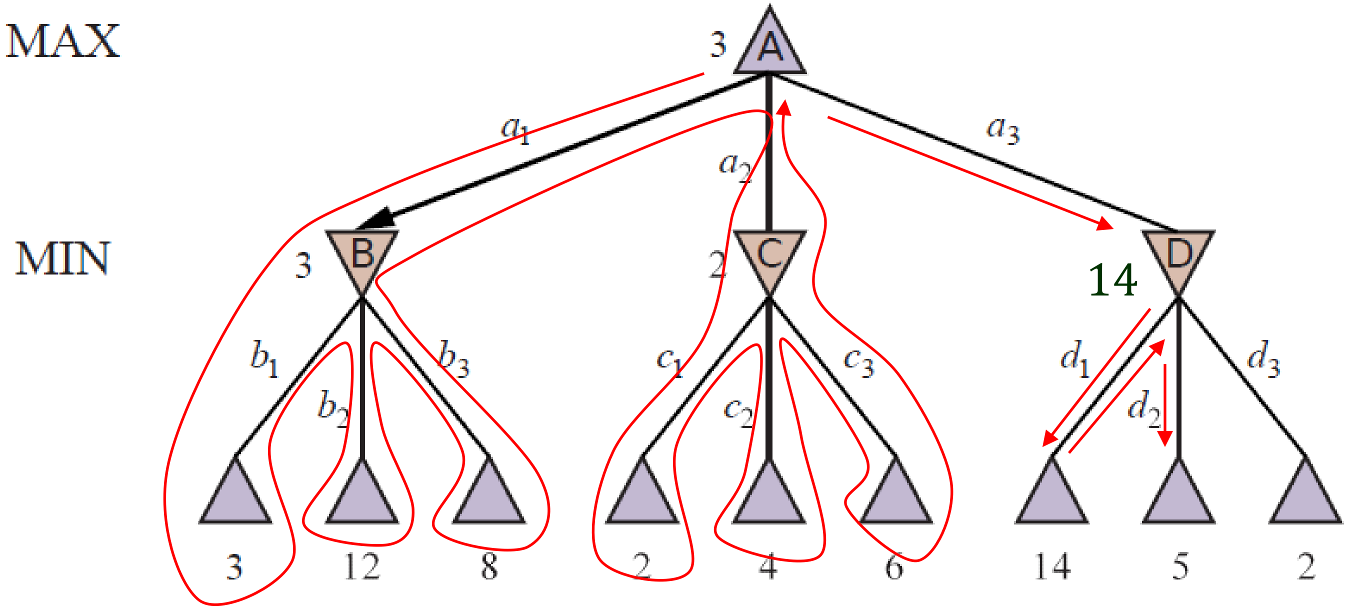
Algorithm Execution

Depth-first search with backed-up value on return from a node.



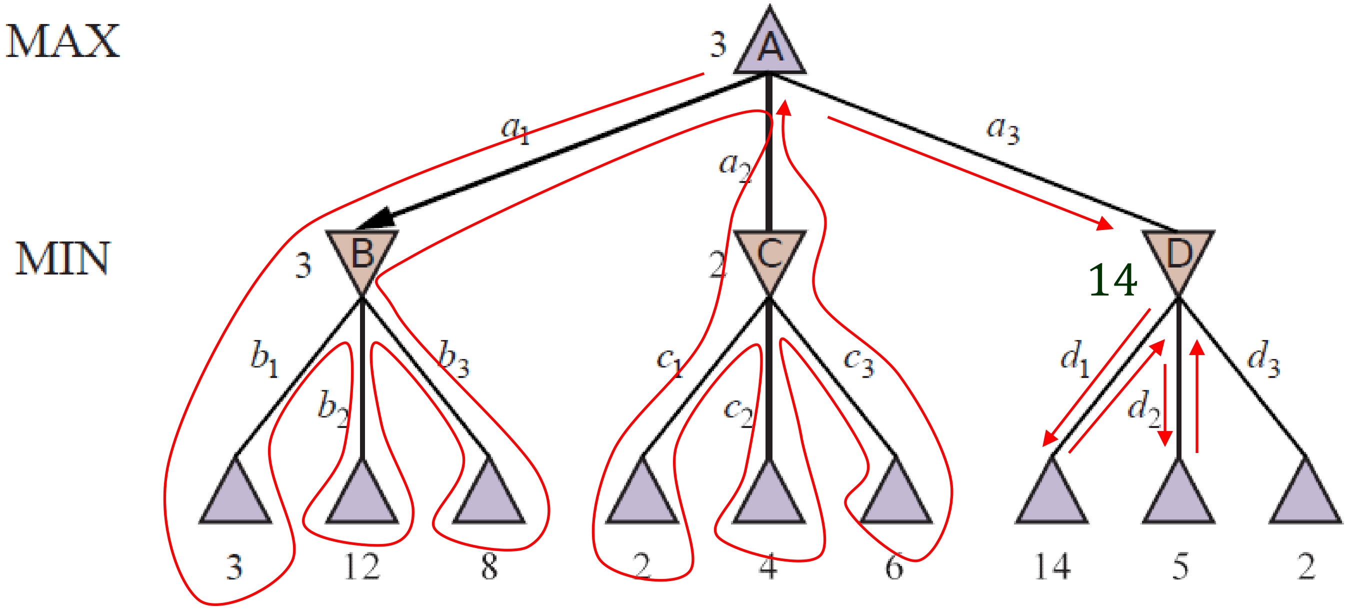
Algorithm Execution

Depth-first search with backed-up value on return from a node.



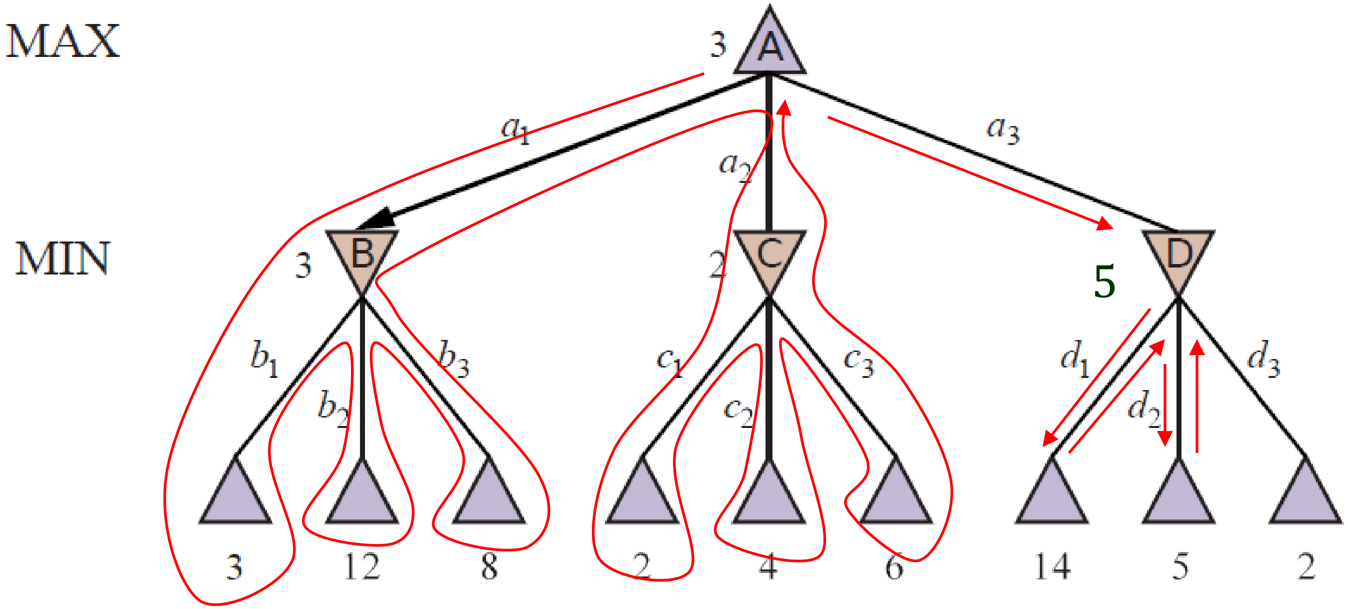
Algorithm Execution

Depth-first search with backed-up value on return from a node.



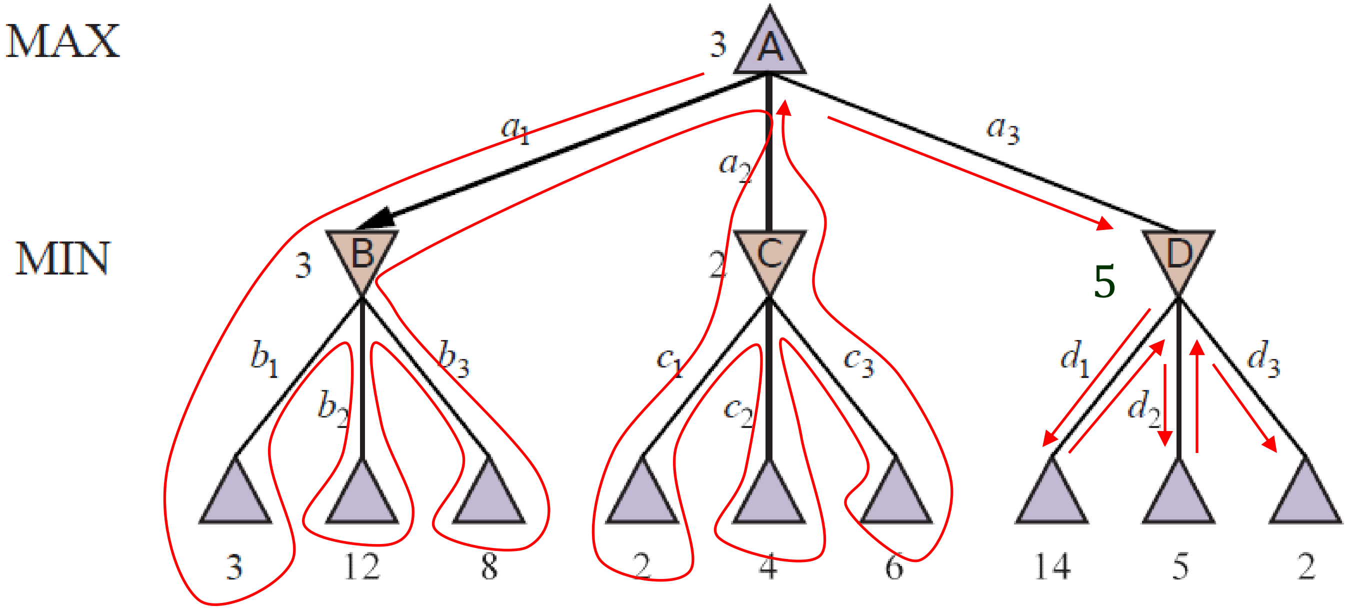
Algorithm Execution

Depth-first search with backed-up value on return from a node.



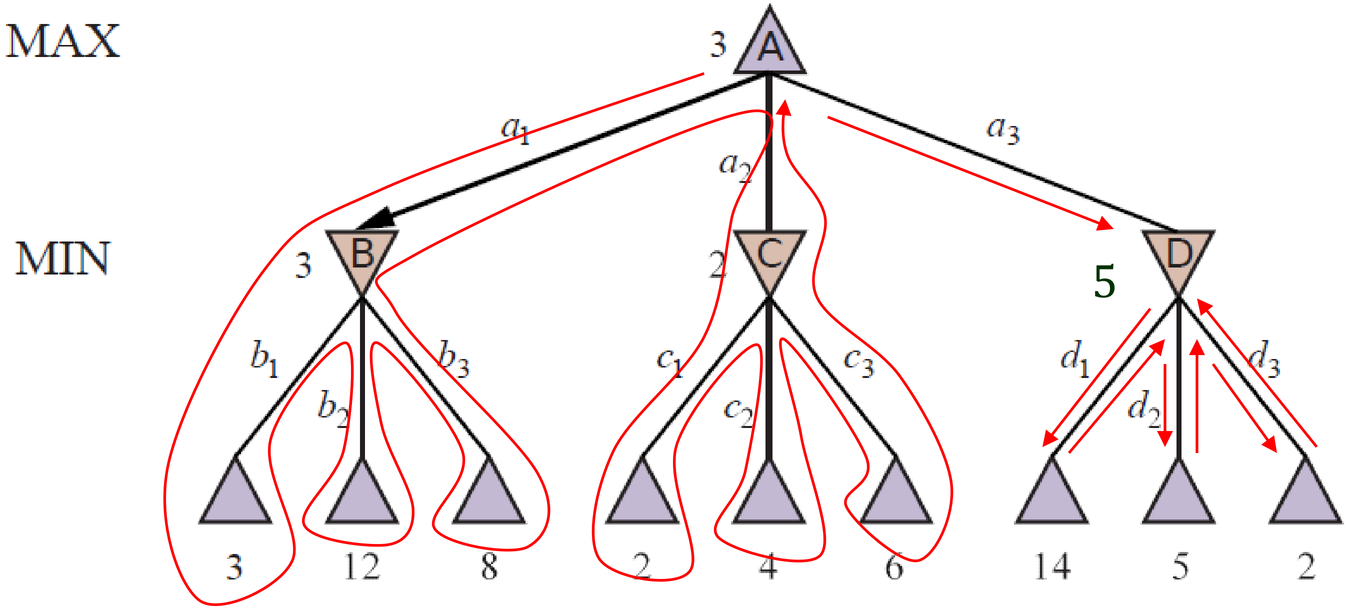
Algorithm Execution

Depth-first search with backed-up value on return from a node.



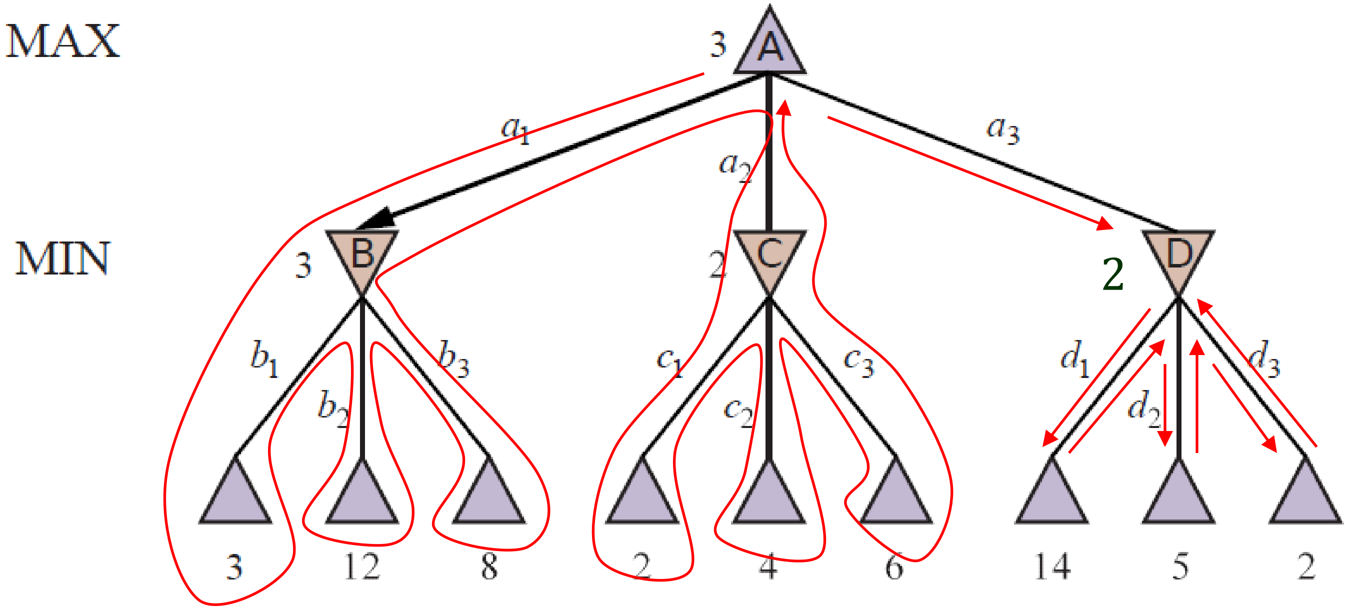
Algorithm Execution

Depth-first search with backed-up value on return from a node.



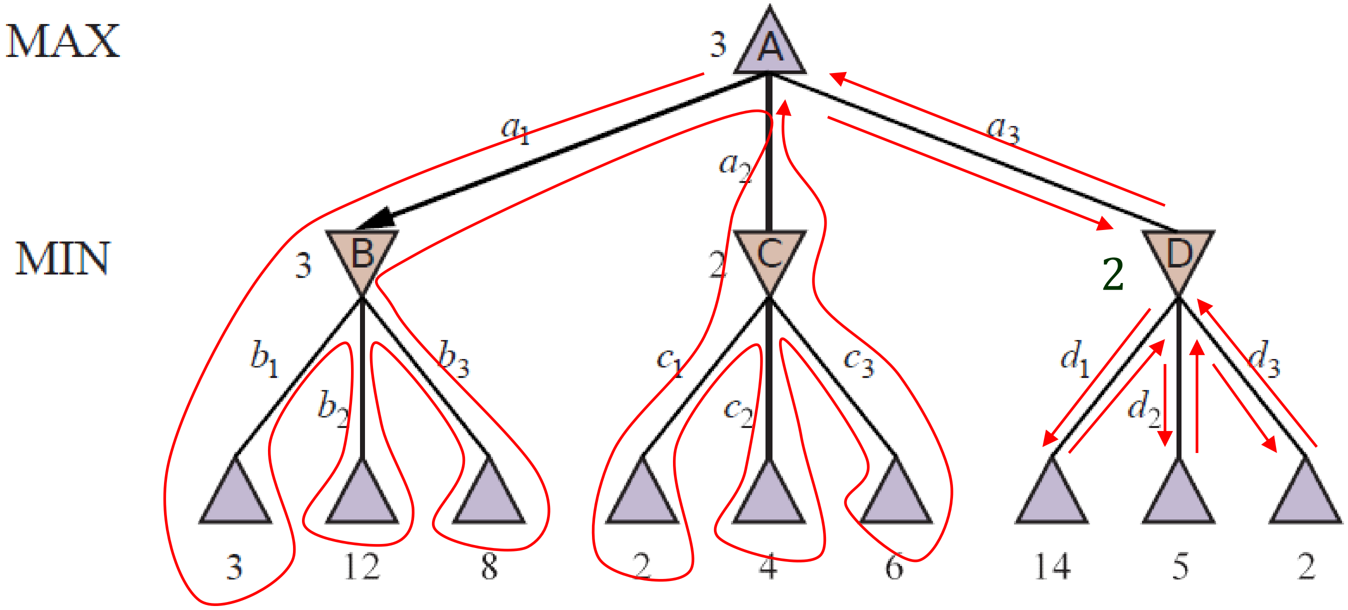
Algorithm Execution

Depth-first search with backed-up value on return from a node.



Algorithm Execution

Depth-first search with backed-up value on return from a node.



Summary on Minimax

- Complete if the game tree is finite.

Summary on Minimax

- Complete if the game tree is finite.
- Optimal against an optimal opponent.

Summary on Minimax

- Complete if the game tree is finite.
- Optimal against an optimal opponent.

  If MIN does not play optimally,

  1) MAX will play at least as well as an optimal player;
  2) but there may be a better strategy against the suboptimal MIN.

Summary on Minimax

- Complete if the game tree is finite.

- Optimal against an optimal opponent.

  If MIN does not play optimally,
  1) MAX will play at least as well as an optimal player;
  2) but there may be a better strategy against the suboptimal MIN.

- Complexities:

  Time: $O(b^m)$

Summary on Minimax

- Complete if the game tree is finite.
- Optimal against an optimal opponent.

  If MIN does not play optimally,
  1) MAX will play at least as well as an optimal player;
  2) but there may be a better strategy against the suboptimal MIN.

- Complexities:

  branching factor
  max depth

  Time: $O(b^m)$

Summary on Minimax

- Complete if the game tree is finite.

- Optimal against an optimal opponent.

  If MIN does not play optimally,
  1) MAX will play at least as well as an optimal player;
  2) but there may be a better strategy against the suboptimal MIN.

- Complexities:

  branching factor
  max depth

  Time: $O(b^m)$ Space: $O(bm)$

Summary on Minimax

- Complete if the game tree is finite.

- Optimal against an optimal opponent.

  If MIN does not play optimally,
  1) MAX will play at least as well as an optimal player;
  2) but there may be a better strategy against the suboptimal MIN.

- Complexities:

  branching factor → b, max depth → m

  Time: $O(b^m)$ Space: $O(bm)$

  Chess: $b \approx 35$ and $m \approx 100$ for a reasonable game.
  Exact optimal solution infeasible!

Multiplayer Games

Extend the minimax algorithm:

- Every node now has a *vector* of values.

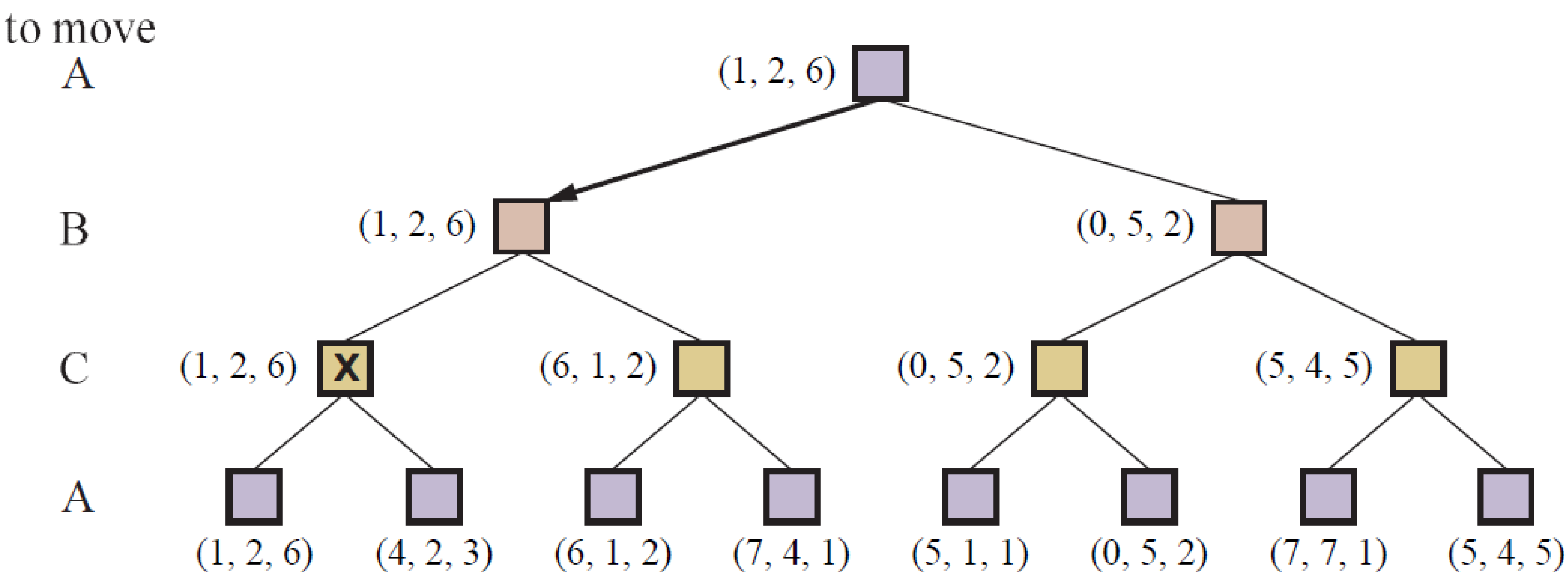
  $\langle v_A, v_B, v_C \rangle$ for three players A, B, C

Multiplayer Games

Extend the minimax algorithm:

- Every node now has a *vector* of values.

  $\langle v_A, v_B, v_C \rangle$ for three players A, B, C

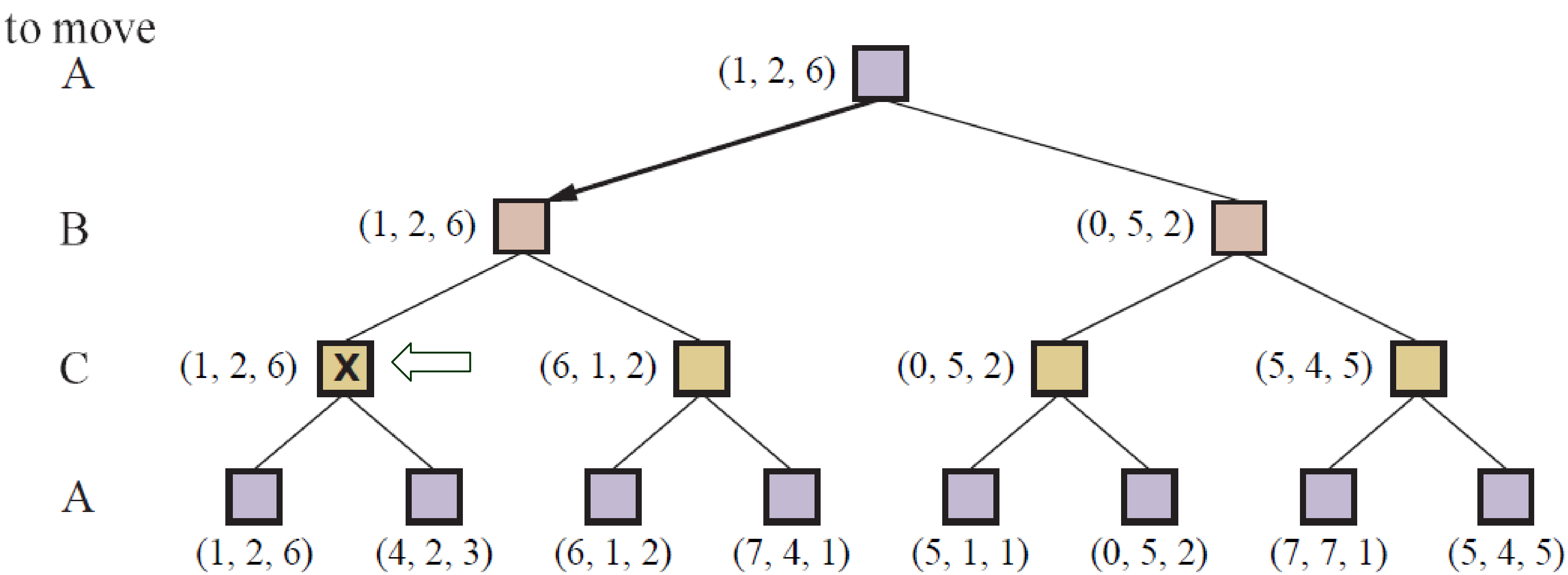


Multiplayer Games

Extend the minimax algorithm:

- Every node now has a *vector* of values.

  $\langle v_A, v_B, v_C \rangle$ for three players A, B, C

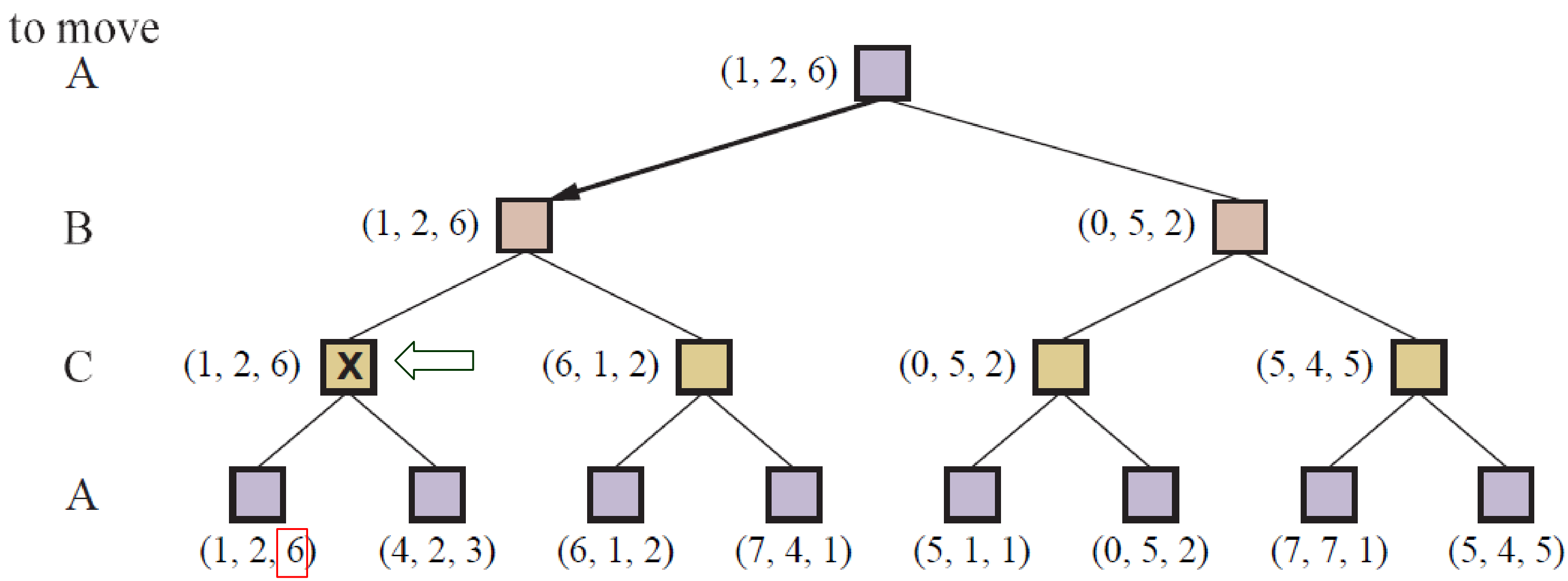


Multiplayer Games

Extend the minimax algorithm:

- Every node now has a *vector* of values.

  $\langle v_A, v_B, v_C \rangle$ for three players A, B, C

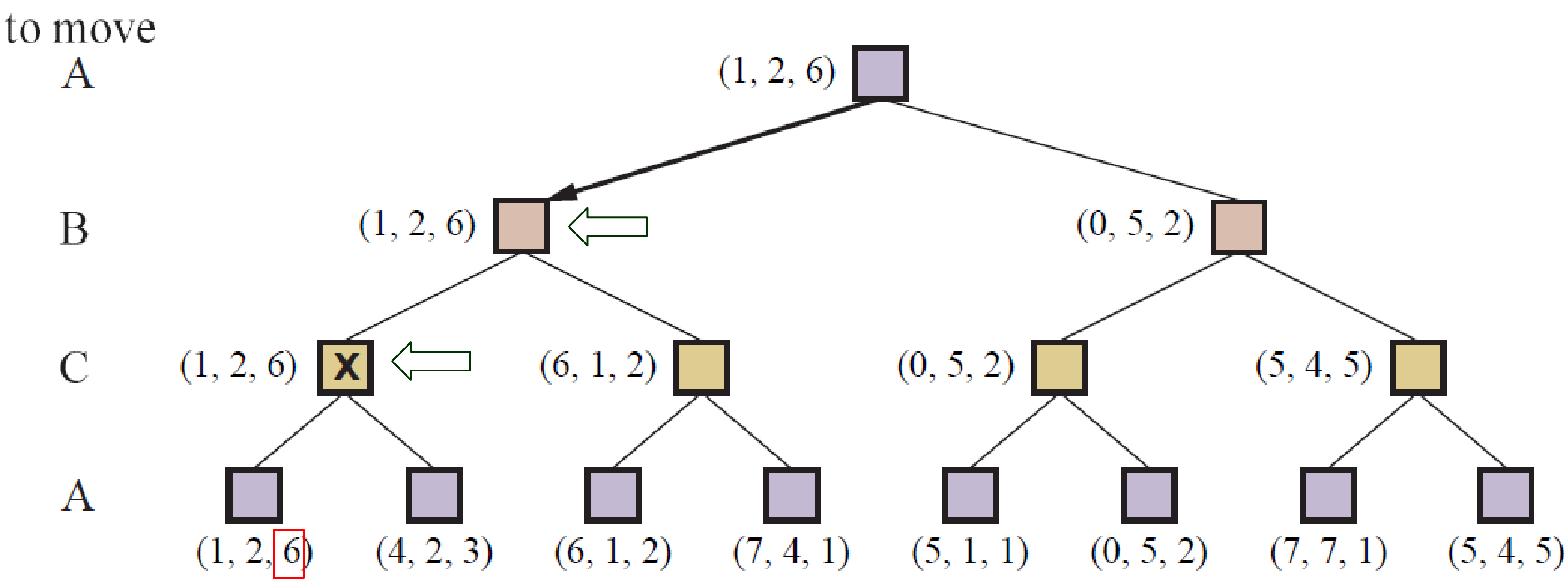


Multiplayer Games

Extend the minimax algorithm:

- Every node now has a *vector* of values.

  $\langle v_A, v_B, v_C \rangle$ for three players A, B, C

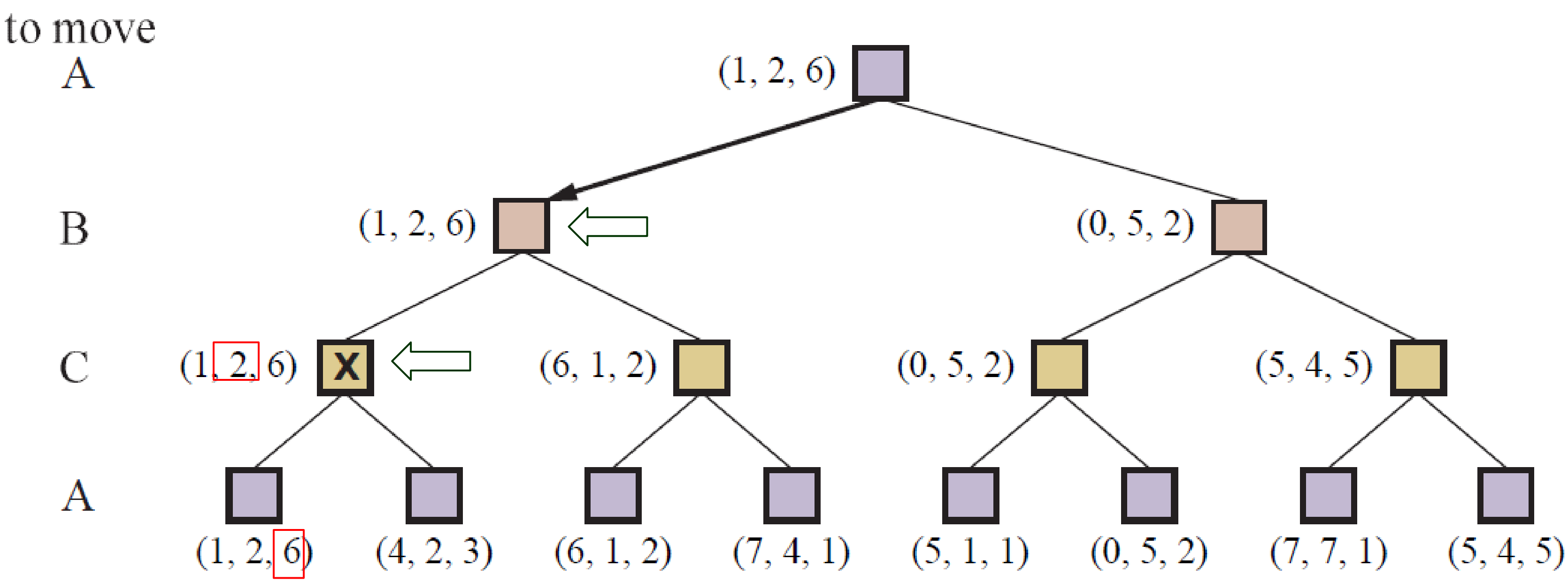


Multiplayer Games

Extend the minimax algorithm:

- Every node now has a *vector* of values.

  $\langle v_A, v_B, v_C \rangle$ for three players A, B, C

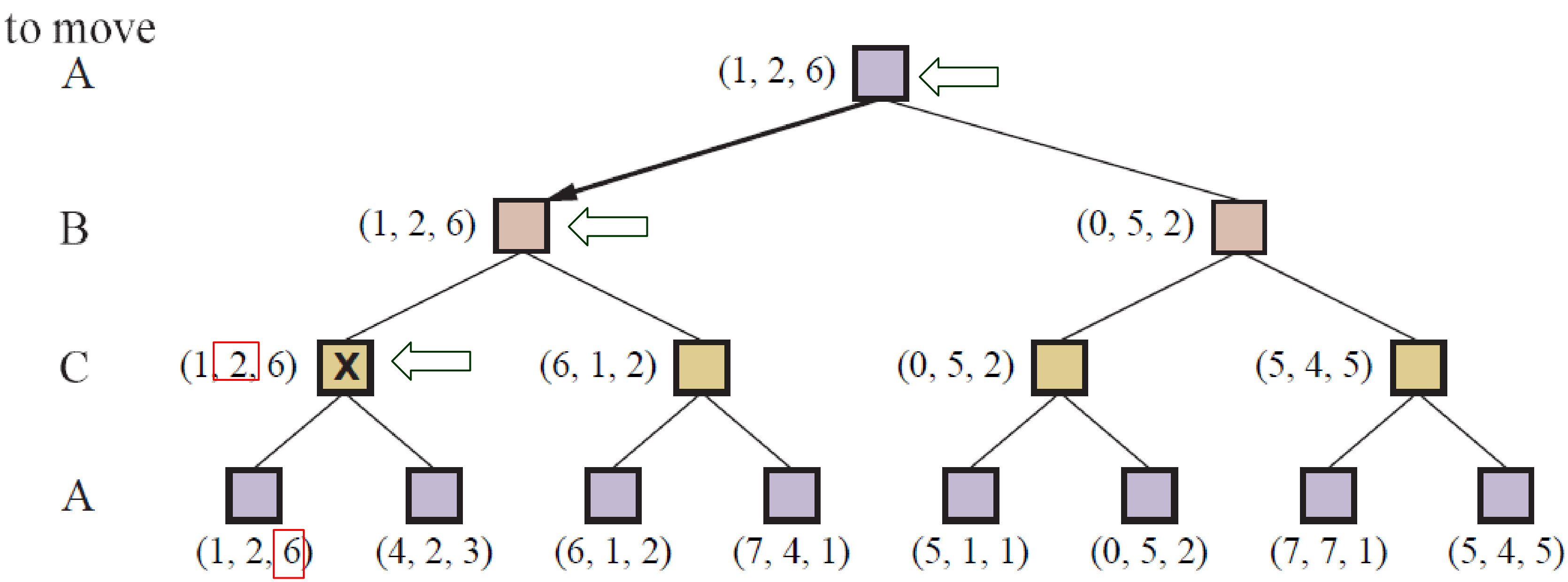


Multiplayer Games

Extend the minimax algorithm:

- Every node now has a *vector* of values.

  $\langle v_A, v_B, v_C \rangle$ for three players A, B, C

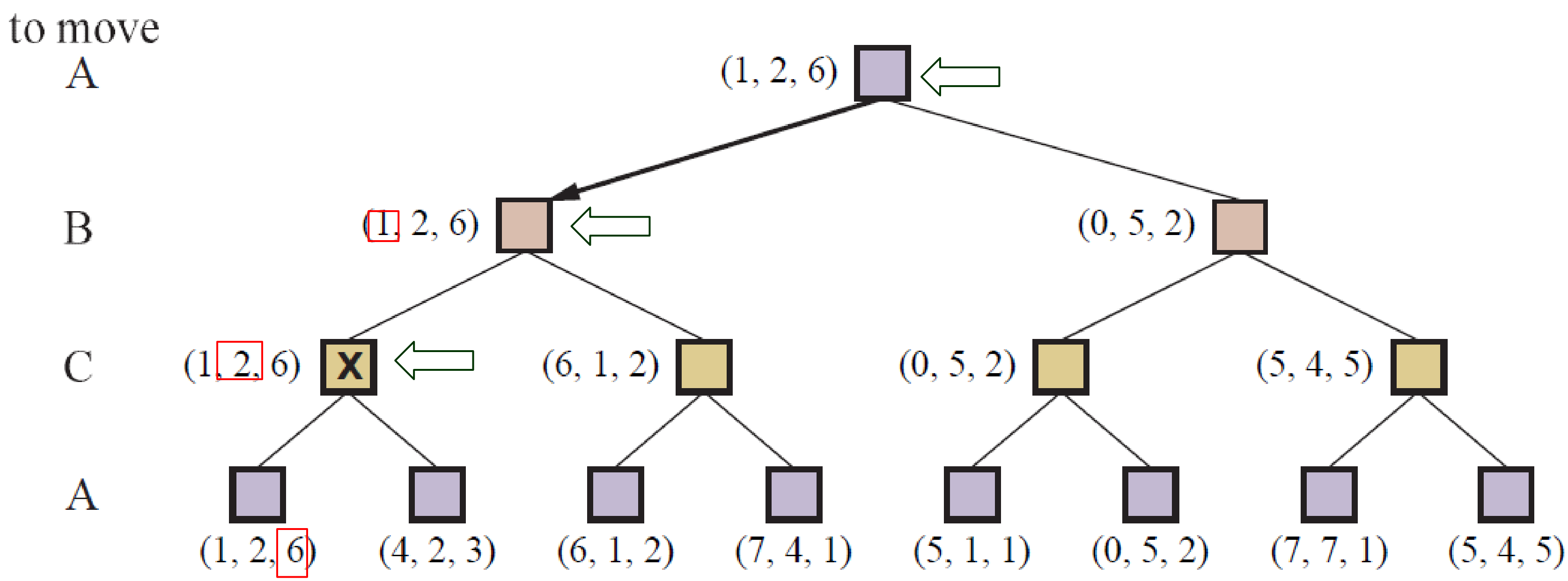


Multiplayer Games

Extend the minimax algorithm:

- Every node now has a *vector* of values.

  $\langle v_A, v_B, v_C \rangle$ for three players A, B, C

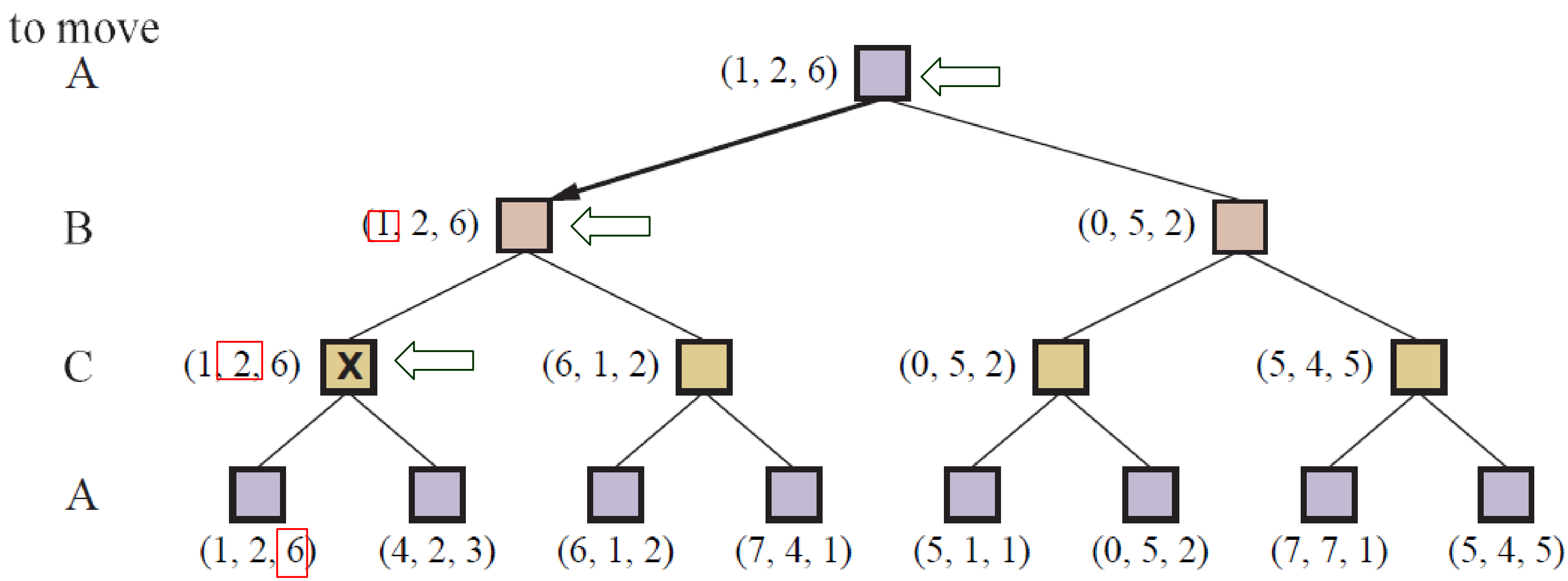


Multiplayer Games

Extend the minimax algorithm:

- Every node now has a *vector* of values.

  $\langle v_A, v_B, v_C \rangle$ for three players A, B, C



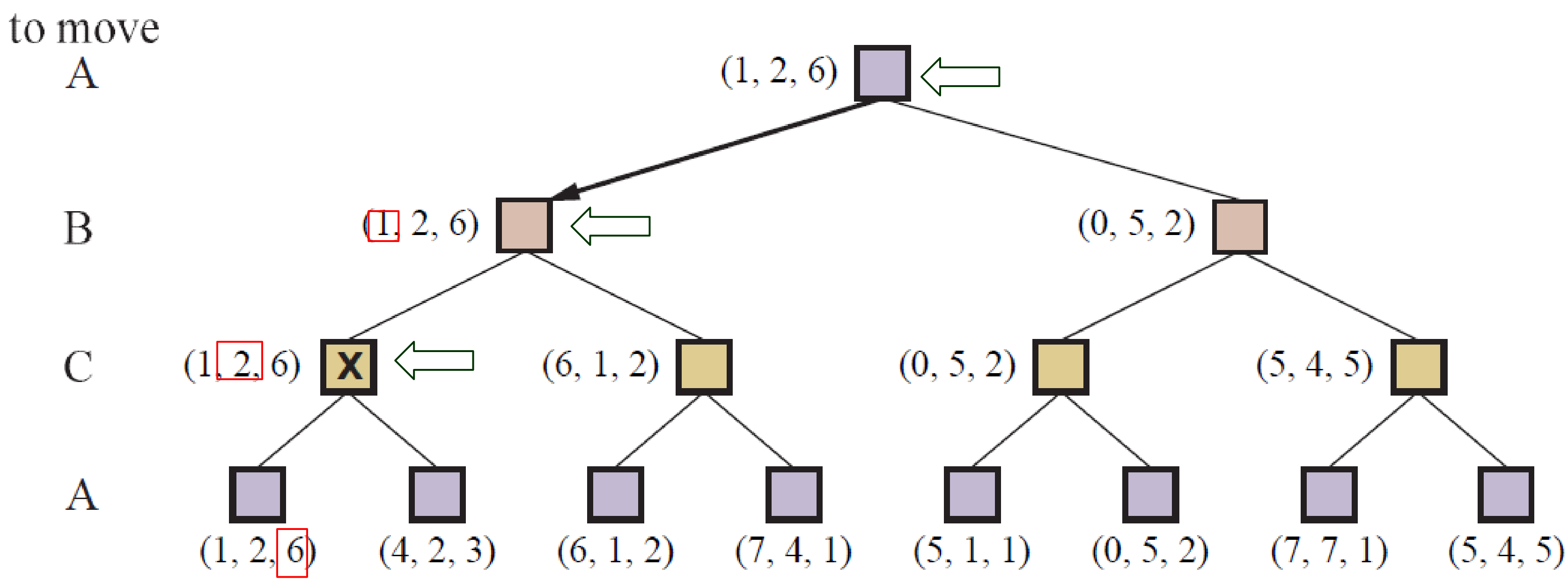
Backed-up value at a node n = utility vector of the successor state with the highest value for the player choosing at n

Multiplayer Games

Extend the minimax algorithm:

- Every node now has a *vector* of values.

  $\langle v_A, v_B, v_C \rangle$ for three players A, B, C



Backed-up value at a node n = utility vector of the successor state with the highest value for the player choosing at n