ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

06
Chapiter

Adversarial Search

2I1AE1: Artificial Intelligence

Régis Clouard, ENSICAEN - GREYC

"Chess is the drosophila of AI. However, computer chess has developed much as genetics might have if the geneticists had concentrated their efforts starting in 1910 on breeding racing drosophila. We would have some science, but mainly we would have very fast fruit flies."
John McCarthy

In this chapter

- Adversarial Search Algorithms
  - In which we examine the problems that arise when we try to plan ahead in a world where other agents are planning against us.

1) Introduction to game Adversarial Search
2) Game as search
3) Optimal solution: Minimax algorithm
4) Pruning to reduce cost: Alpha-Beta algorithm
5) Approximate solution: Hminimax algorithm
6) Variations on deterministic perfect information zero-sum games

Why Studying Game?

- Games allow us to experiment with easier versions of real-world situations.
  - Games have **a finite set of moves**.
  - Games are **well-formalized.**
    - Rules are fixed.
    - Clear criteria for success.
  - Games rise hard problems with **very large search spaces**.
- They illustrate several important points about AI.
  - **Perfection is unattainable** → must approximate.
  - **Lot of works on models of thinking**: strategy, policy, tactics.
- Game is the Drosophila of Artificial Intelligence.

Application of game theory

- Not only games, but everything dealing with concurrency:
  - Economy
  - Traffic / Transport
  - Elections
  - Auctions
  - Military
  - Security

Types of Game Problems

- A spectrum of game problems between:
  - Adversarial games
    - Win of one player is the loss of the other.
  - Cooperative games
    - Players have a common interest and utility function.

- In this course, we focus on adversarial games only.

Game Classification

- **Perfect information** / **Imperfect information**
  - Perfect: both players have access to complete information about the state of the game. No information is hidden from either player.
- **Deterministic** / **Non-deterministic (Stochastic)**
  - Non deterministic: behavior has a degree of uncertainty and is somewhat unpredictable (eg., use of dice rolls).
- **Zero-sum** / **Non-Zero sum**
  - Zero-sum: one player's loss is the other's gain.
- **Asynchronous** / **Synchronous**
  - Asynchronous: players act alternatively.

- This classification leads to various classes of solution.

What games Are Like These?

	Perfect information	Deterministic	Zero-sum	Asynchronous
Chess, Checkers	✓	✓	✓	✓
Battleship	✗	✓	✓	✓
Poker, French Tarot	✗	✗	✓	✓
Rock-paper-scissors (Chifoumi)	✗	✓	✓	✗

Game Setup

- Assumptions made for this section:
  - Perfect information
  - Deterministic
  - Zero-sum (only consider MAX utility)
  - Asynchronous
- This includes:
  - Clear rules for legal moves.
  - Well-defined outcomes.
    - eg. Win/Loss/Draw.
  - Multiplayer
    - Single-players: It's just search!

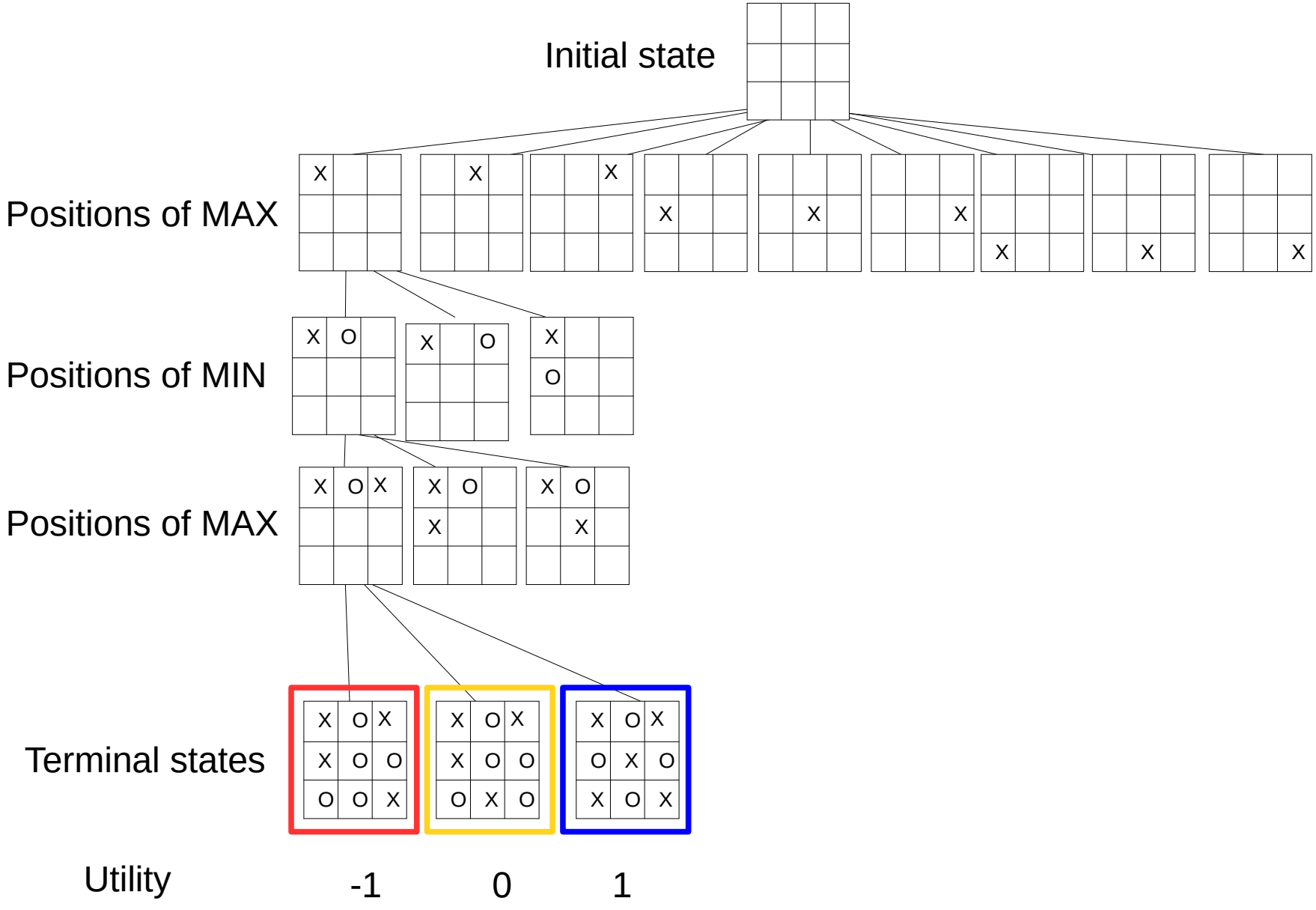
Game Problem

- Game problem formulation (path finding)
  - **State** S : Board configuration + whose move it is (from **Players** P = {1, …, n})
  - **Initial state** S_0 : Initial board position + first player to move.
  - **Actions** A : Legal moves a player can make.
  - **Goal test** (terminal test): S → {True, False}
    - Determines when the game is over.
  - **Utility function** (cost/payoff function): $S \times P \rightarrow \Re$
    - Assign a numerical value to terminal states.
  - **Type**: path finding
- Search objective
  - Find the sequence of player's decisions (moves) maximizing a utility function (payoff):
    S → A

Game Tree

- Example with two-player game: MAX and MIN
  - The positions are evaluated from the MAX perspective (Zero-sum games).
  - MAX and MIN take turns until the game is over.
    - ply: a half-move by one of the players.
    - move: two plies, one by MAX and one by MIN.
- A **game tree** is a directed graph whose nodes are positions in a game and whose edges are plies.
  - The **root** is the initial position.
  - The terminal nodes are evaluated from the **MAX's perspective**.
    - Example in Tic-Tac-Toe
      - +1 for a win, -1 for a loss, 0 for a draw
    - Example in Reversi (Othello)
      - Number of whites – number of blacks

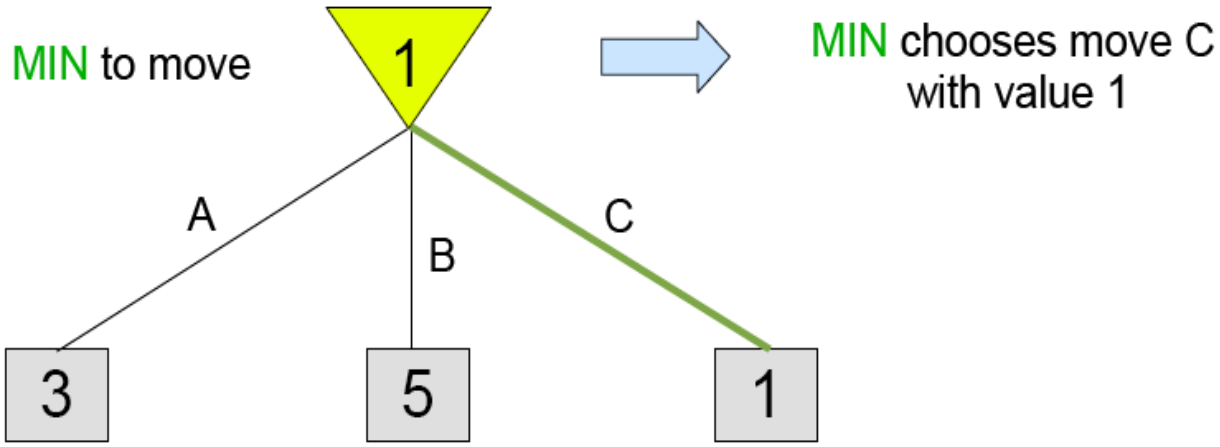
Example. Tic-Tac-Toe Game Tree



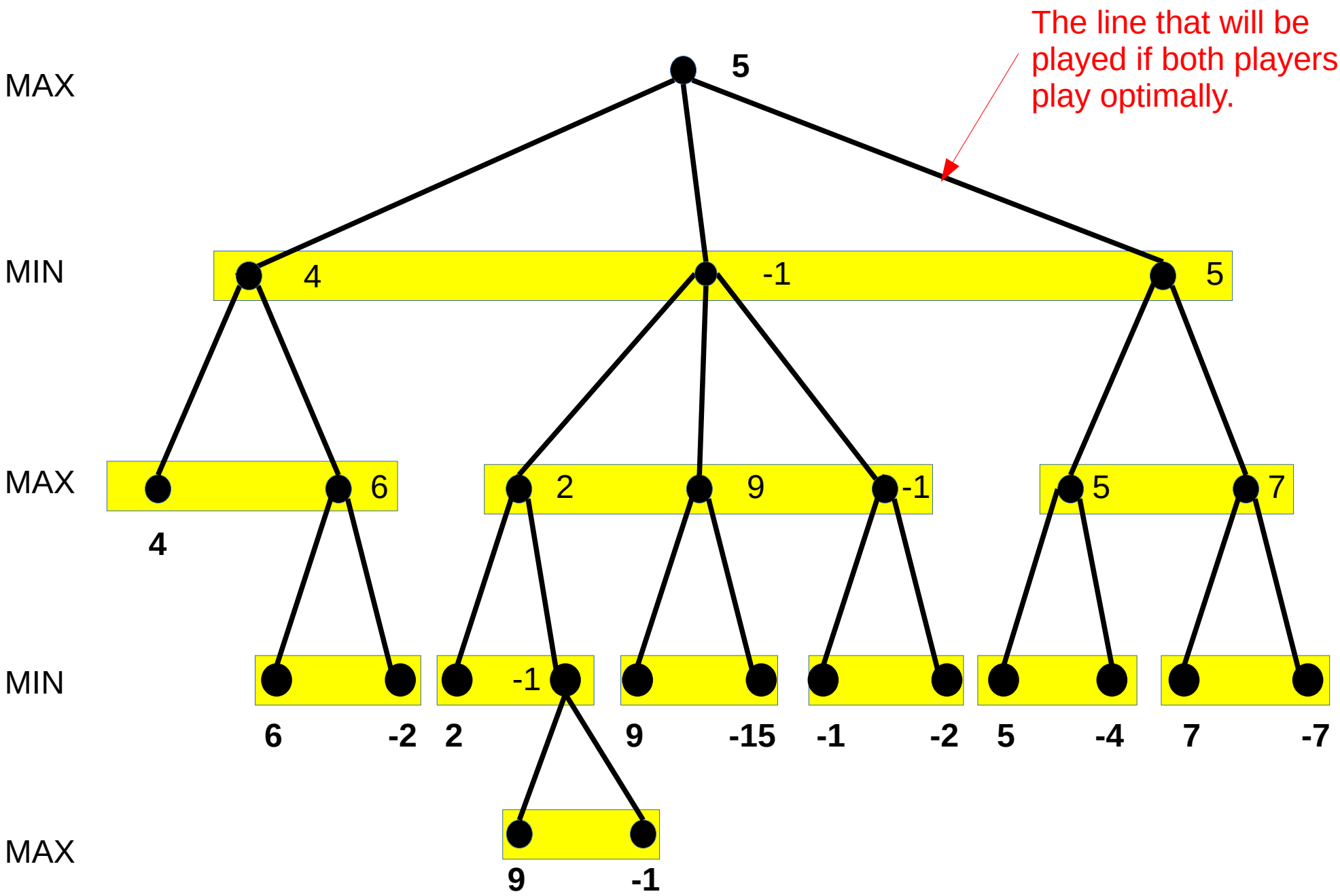
Minimax Algorithm
(Von Neumann, 1945)

- Perfect play for deterministic and perfect-information games.
  - Find the best strategy for MAX.
  - Assumptions: both players play optimally and zero sum game.
- Algorithm: evaluation of a node according to minimax algorithm

$$\text{MINIMAX}(n) = \begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal node} \\ \max_{s \in \text{successors}}(\text{MINIMAX}(s)) & \text{if } n \text{ is a MAX node} \\ \min_{s \in \text{successors}}(\text{MINIMAX}(s)) & \text{if } n \text{ is a MIN node} \end{cases}$$



Minimax Algorithm. Example



Minimax Algorithm


```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in GET-SUCCESSORS(state) with value v
end
```

```
function MAX-VALUE(state) returns a utility value
    IF IS-TERMINAL(state) THEN return UTILITY(state)
    v ← −∞
    FOREACH a in GET-SUCCESSORS(state) DO
        v ← MAX(v, MIN-VALUE(NEXT-STATE(state, a))
    return v
end
```

```
function MIN-VALUE(state) returns a utility value
    IF IS-TERMINAL(state) THEN return UTILITY(state)
    v ← +∞
    FOREACH a in GET-SUCCESSORS(state) DO
        v ← MIN(v, MAX-VALUE(NEXT-STATE(state, a)))
    return v
end
```

Properties of Minimax Algorithm

- **Complete**?
  - Yes (if tree is finite)
- **Optimal**?
  - Yes (against an optimal opponent)
- **Time complexity**?
  - $O(b^m)$ – m: depth of the search tree
- **Space complexity**?
  - O(b.m) – eqv. DFS by the way of recursivity

Examples of Complexity

- The game tree is generally gigantic:
  - Example: Tic-tac-toe
    - Total number of states: $3^9 = 19683$ (3 states for each of the 9 cells)
  - Example: Chess
    - Number of actions by ply: b ~ 35.
    - Average number of moves in a game: 50.
    - Total number of states: 35^{50}.
  - Example: Go
    - Number of actions by ply: b ~ 300.
    - Average number of moves in a game: 150.
    - Total number of states: 300^{150}.
- For most of interesting games, the exact solution is completely infeasible.

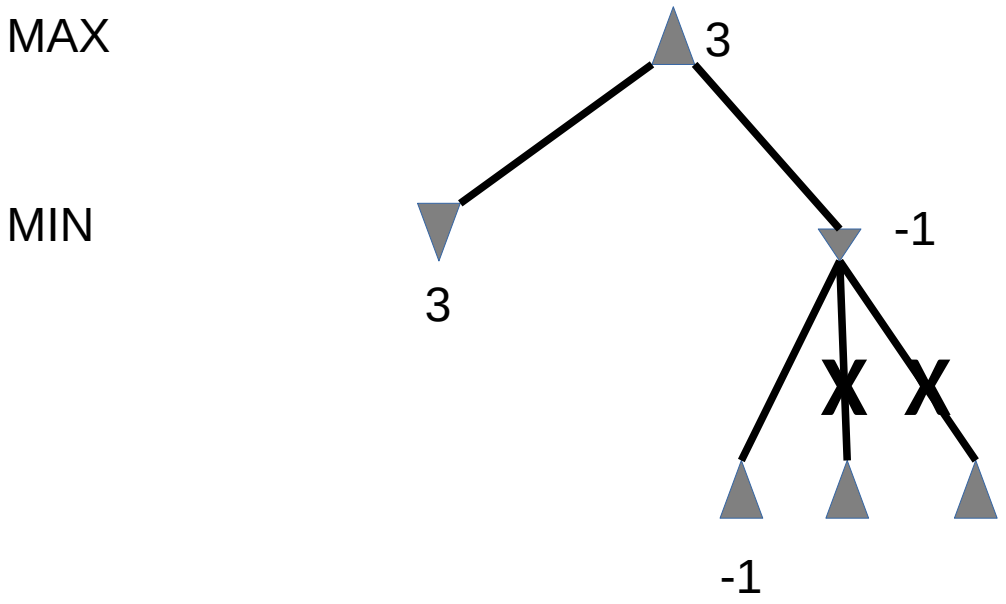
[go]

Speed-Up Search

- Number of paths in the game tree is exponen
- tial
- First solution: take care of the real states
  - eg: tic-tac-toe
    - A simple upper bound for the number of paths is 9! = 362,880
      - 9 positions for the first ply, 8 for the second, etc
    - This includes illegal and redundant states (eg 2 states for same position like X|O when X is placed before and after the O).
      - A more careful count gives 255,168 possible paths.
    - When rotations and symmetries of positions are considered, the number of paths in the game tree is 26,830.
- Most of the time, it is not enough
  - More drastic solution: Don’t examine all tree nodes
    - Two complementary approaches:
      - Limit the branching factor (prune search tree): Alpha-beta pruning.
      - Limit the depth search (cut-off search tree) : Minimax cutoff.

Alpha-Beta Pruning
(Mc Carthy, 1946)

- Some branches will never be played by rational players since they include sub-optimal decisions (for either player).



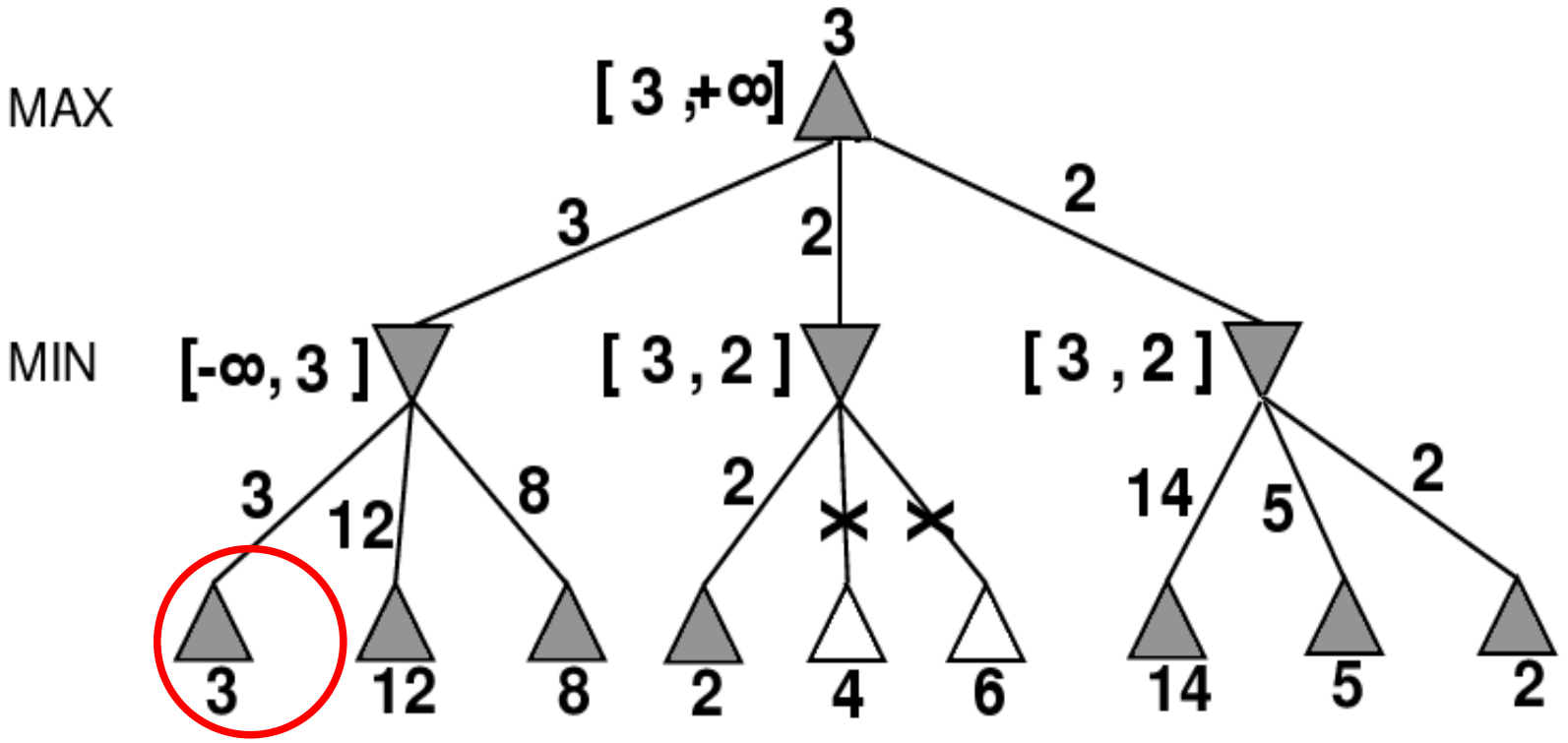
Alpha-Beta Pruning

- Maintains two values [α,β] for all nodes in the current path.
- Alpha
  - The value of the best choice (i.e., highest value) for the MAX player at any choice node for MAX in the current path.
    → MAX can obtain a value of at least α.
- Beta
  - The value of the best choice (i.e., lowest value) for the MIN player at any choice node for MIN in the current path.
    → MIN can make sure that MAX obtains a value of at most β.
- The values are initialized with [−∞, +∞].

Alpha-Beta Pruning

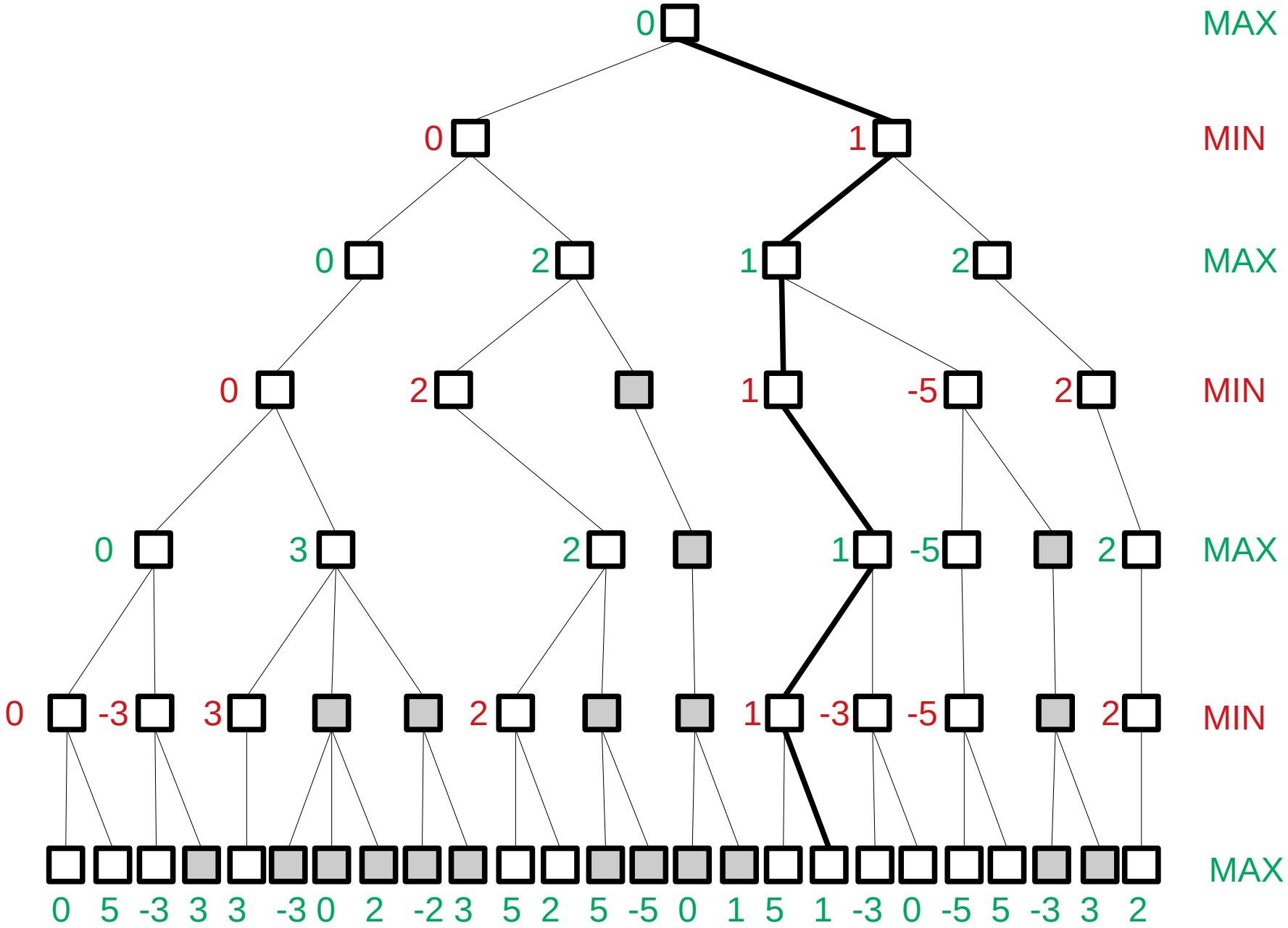
- Alpha and Beta are used for pruning the search tree:
- Alpha-Cutoff:
  - if we find a move with value ≤ α at a MIN node, we do not examine alternatives to this move.
  - we already know that MAX can achieve a better result α in a different variation.
- Beta-Cutoff:
  - if we find a move with value ≥ β at a MAX node, we do not examine alternatives to this move.
  - we already know that MIN can achieve a better result β in a different variation.

Alpha-Beta Pruning
(Mc Carthy, 1946)



- Note: order of successors has influence on performance!

Alpha-Beta Pruning. Example



The Alpha-Beta Algorithm


```
function ALPHA-BETA-SEARCH(state) returns an action
   v ← MAX-VALUE(state, -∞, +∞)
   return the action in GET-SUCCESSORS(state) with value v
```

```
function MAX-VALUE(state, α, β) returns a utility value
   IF IS-TERMINAL(state) THEN return UTILITY(state)
   v ← −∞
   FOREACH s in GET-SUCCESSORS(state) DO
      v ← MAX(v, MIN-VALUE(s, α, β))
      IF v ≥ β THEN return v
      α ← MAX(α, v);
   return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
   IF IS-TERMINAL(state) THEN return UTILITY(state)
   v ← +∞
   FOREACH s in GET-SUCCESSORS(state) DO
     v ← MIN(v, MAX-VALUE(s, α, β))
     IF v ≤ α THEN return v
     β ← MIN(β, v);
   return v
```

Properties of Alpha-Beta Pruning

- Pruning **does not** affect final result.
  - Alpha-Beta results = Minimax results.
- **Time complexity**
  - Worst case: same as the Minimax algorithm: $O(b^m)$.
  - Best case, with "perfect ordering": $O(b^{m/2})$: can double depth of search!
  - In most cases, this allows at least to **add one level** to depth search.
- Good moves ordering improves effectiveness of pruning.
  - For MAX, if the successors are ordered by decreasing evaluation values, the number of cutoffs is maximum.
  - For MIN, if the successors are ordered by increasing evaluation values, the number of cutoffs is maximum.
  - But rarely the order of successors at each level cannot be decidable beforehand.
    - However, how can this optimization be taken into account?

Improvement: Iterative Deepening

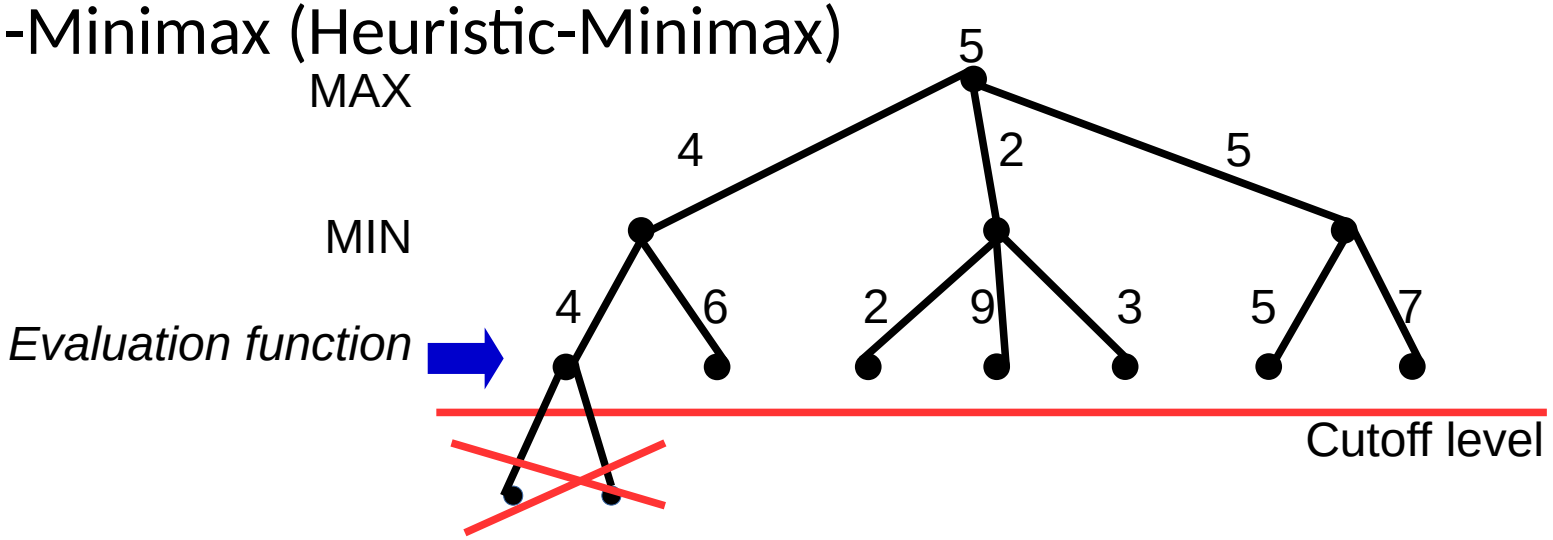
- Repeated fixed-depth searches for depths *d* = 1, …, D.
- Advantages:
  - Improved dynamic move-ordering in alpha-beta.
    - What worked well in the previous iteration is tried first in the next iteration.
  - Simplifies time management.
    - Previous iterations provide useful information that allows to guess whether the next iteration can be completed in time.
  - Cost: slightly more expensive in execution time
    - Recall chapter 3: the most part of the nodes is at the last level.
      - Previous level are explored multiple times. The total number of explored nodes:

        $$d.b + (d-1).b^2 + \ldots + (2)b^{d-1} = [b^{d+1} + d(b-1)] / [b - 1]^2 = \mathbf{O(b^{d-1})}$$

      - Last level is explored once: $\mathbf{O(b^d)}$
      - So, the last level have more nodes to explore than all the previous levels even if they are explored several times.
- Frequently used in game-playing programs.
  - Quite frequently the total number of nodes searched is smaller than with non-iterative search!

H-Minimax Algorithm
(Shannon, 1950; Samuel, 1952)

- Resource limits
  - In most games, even with alpha-beta pruning, it is not possible to explore the whole search space e.g. chess!
  - Minimax is unusable in practice
- Idea
  - Cutoff the search tree before the terminal state is reached.
    - **Depth limit**
  - Use imperfect estimate of the minimax value at the leaves.
    - **Evaluation function**
- → H-Minimax (Heuristic-Minimax)



Cutting Off Implementation

- H-Minimax is identical to Minimax except:
  - **Terminal test** is completed with **Cutoff test**.
  - **Utility** is replaced by **Evaluation**.
- Chess player levels:
  - 4-ply look-ahead (2-move) ≈ human novice
  - 8-ply look-ahead (4-move) ≈ typical PC, human master
  - 12-ply look-ahead (6-move) ≈ Deep Blue, Kasparov ($\rightarrow 35^{12}$)

Evaluation of a Position

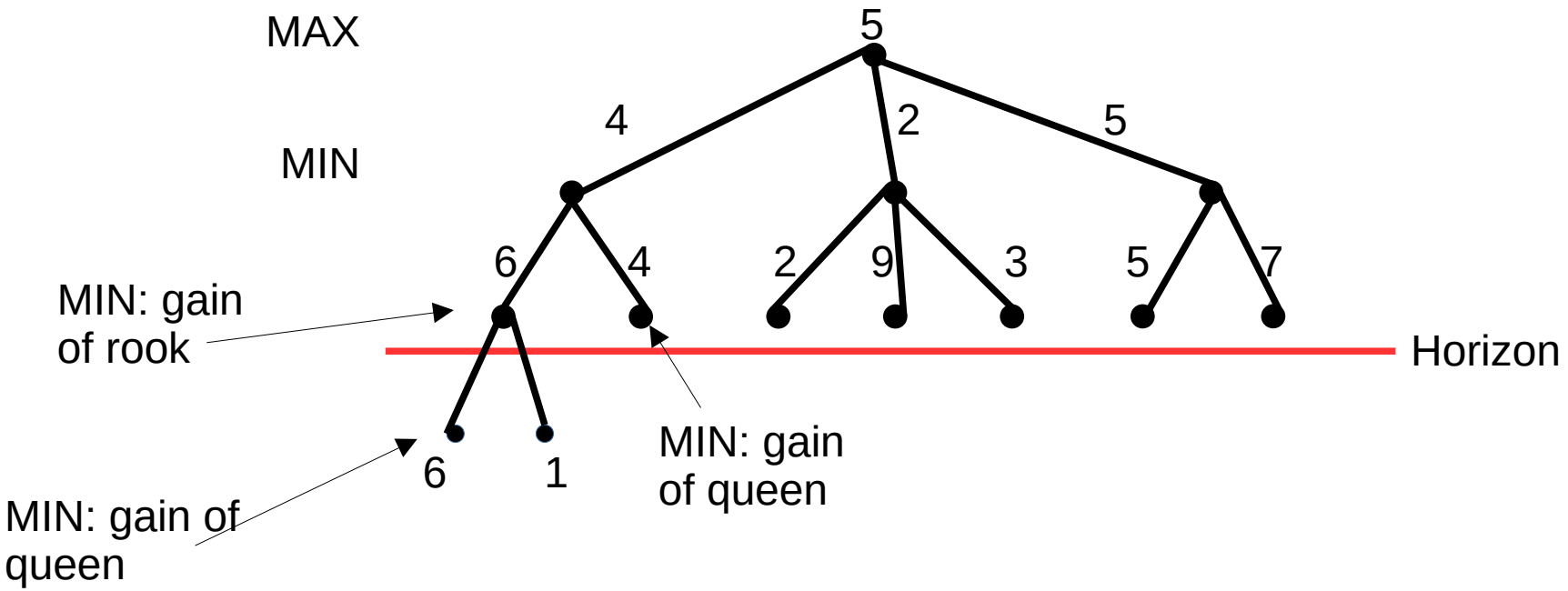
- Objective
  - Evaluate the goodness of a game position (*heuristic evaluation*).
  - Contrast with heuristic search where the evaluation function was a non-negative estimate of the cost from the start to a goal and passing through the given node.
- Most evaluation functions are linear combinations of features
  - $Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
  - $f_i(s)$ encodes a certain characteristic value of the position (state) and w_i the related weight.
  - Note: Addition assumes independence and same domain of variation.
- Advantages
  - Conceptually simple, typically fast to compute.
- Disadvantages
  - Tuning of the weights w_i ($\rightarrow$ reinforcement learning)

Examples of Evaluation Functions

- Tic-Tac-Toe
  - eval(s) = f_1(s): [# 3-lengths open for me] - [# 3-lengths open for you]
    where 3-lengths is a complete row, column, or diagonal.
- Checkers
  - eval(s) = f_1(s): (# my pieces) - (# adversary pieces)
- Chess
  - eval(s) = f_1(s): (# white queens) – (# black queens)
    f_2(s): total value of pieces on both sides with
    queen: 9; rook: 5; bishop: 3; knight: 3; pawn: 1
  - Note: Deep blue has about 6,000 features.

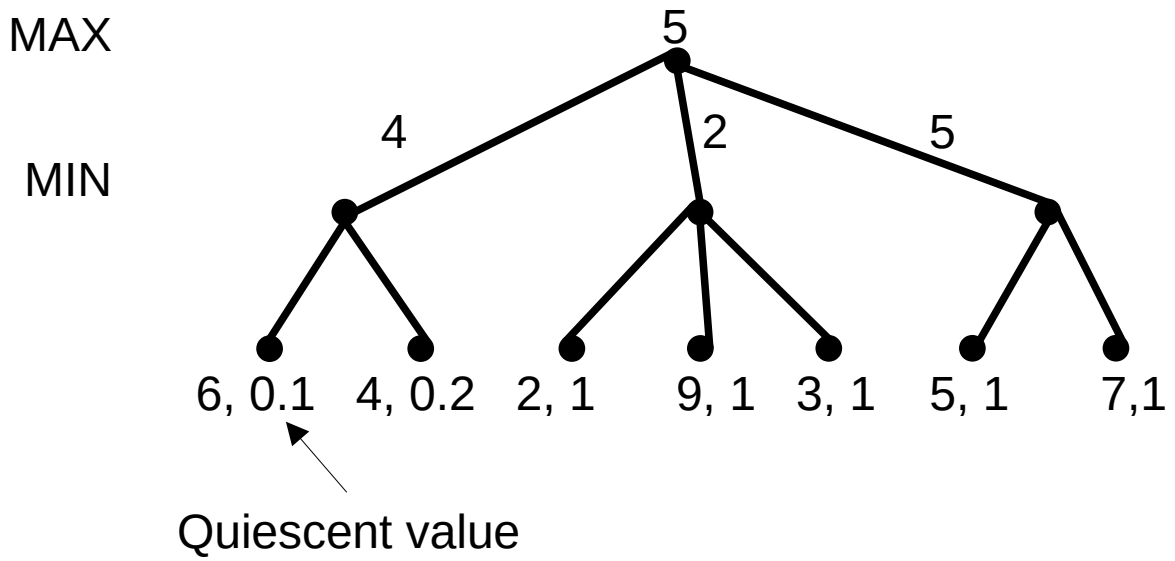
Deficiencies of Cutting Off Search

- The horizon effect
  - Due to fixed-depth search, search has a horizon beyond which it cannot see.
    - Inevitable losses are postponed.
    - Unachievable goals appear achievable.
    - Short-term gains mask unavoidable consequences (traps).



Deficiencies of Cutting Off Search

- How to counter the horizon effect?
  - Evaluation function is only useful for quiescent states.
  - Do not cut off search at non-quiescent states.
    - e.g., states in the middle of an exchange are not quiet.
    - e.g., king in danger is not quiescent position.
  - Give the search algorithm ability to look beyond its horizon for a certain class of moves of major importance to the game state.
    - Go down the tree one level less than the possible maximum and allow to go down deeper for certain states.

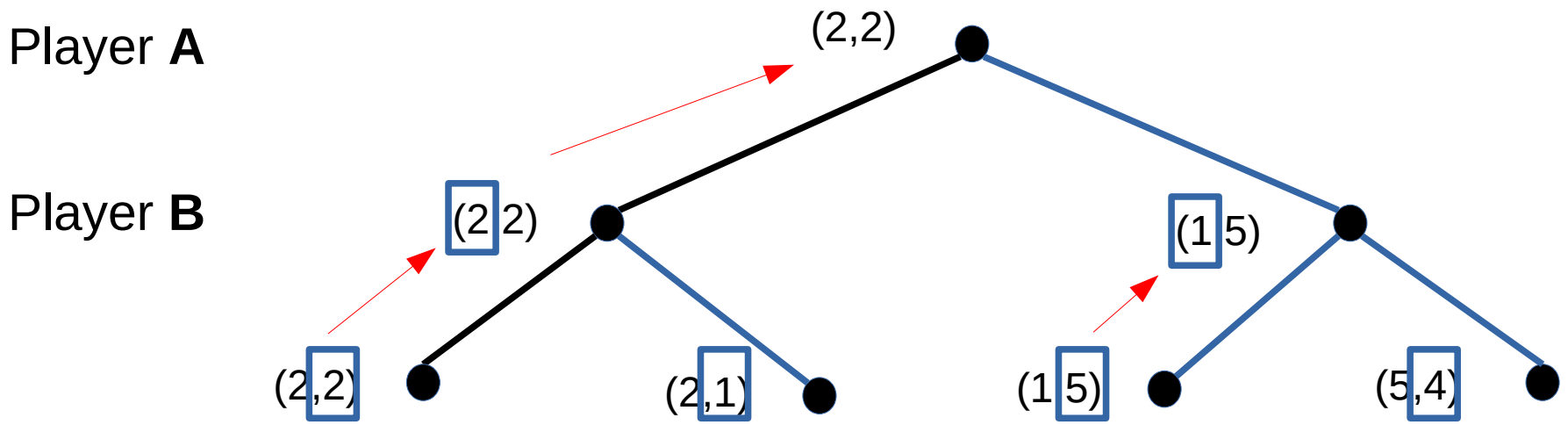


Other classes of problem

- What to do in case of
  1) Non Zero-sum games
  2) Multiplayer games
  3) Non-deterministic (ie stochastic) games
  4) Imperfect information games

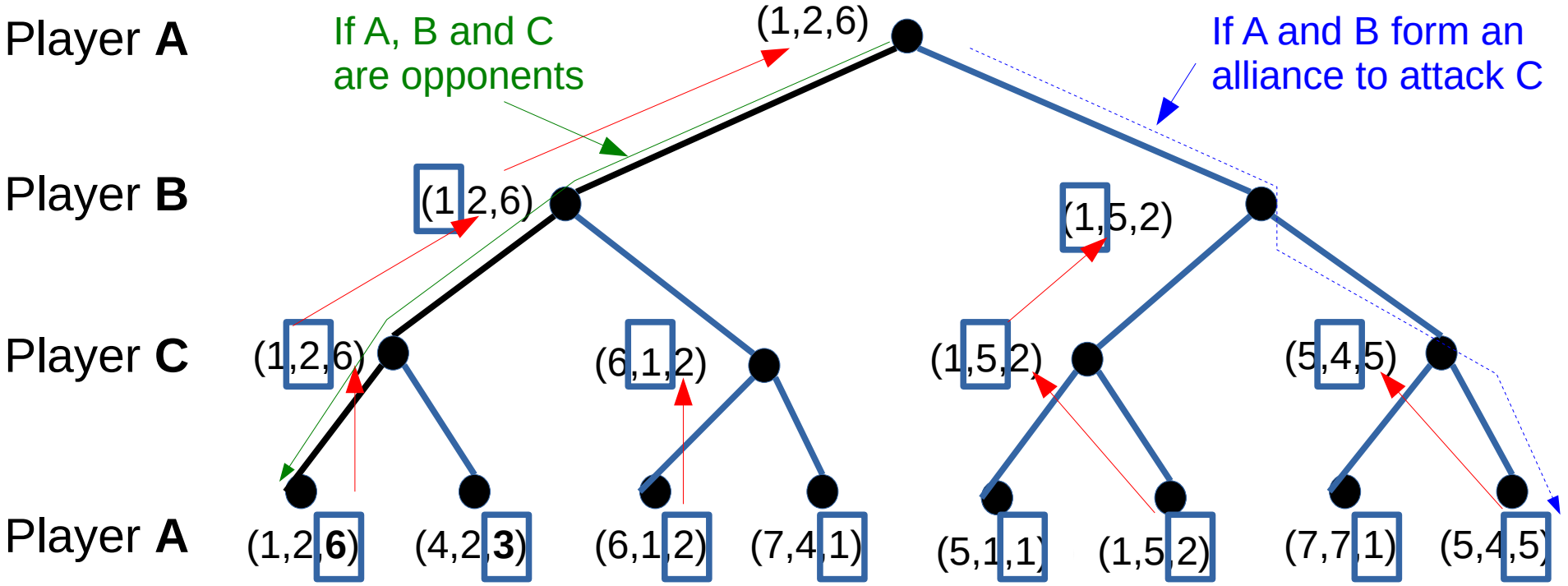
Variation 1: Non-Zero-Sum Games

- The idea is:
  - Store a **vector of utilities** for each terminal node.
  - Each player maximizes its own utility and propagates the related vector up.
  - Use Minimax (H-Minimax) or Alpha-Beta algorithms.



Variation 2: Multiplayer

- Same solution:
  - Store a **vector of utilities** for each terminal node.
  - Each player maximizes its own utility and propagates the related vector up.
  - Use Minimax or Alpha-Beta algorithms.
    - $eval(s) = f(f_A(s), f_B(s), f_C(s))$
  - Multiplayer games usually involve alliances, whether formal or informal, among the players. Alliances are made and broken as the game proceeds.

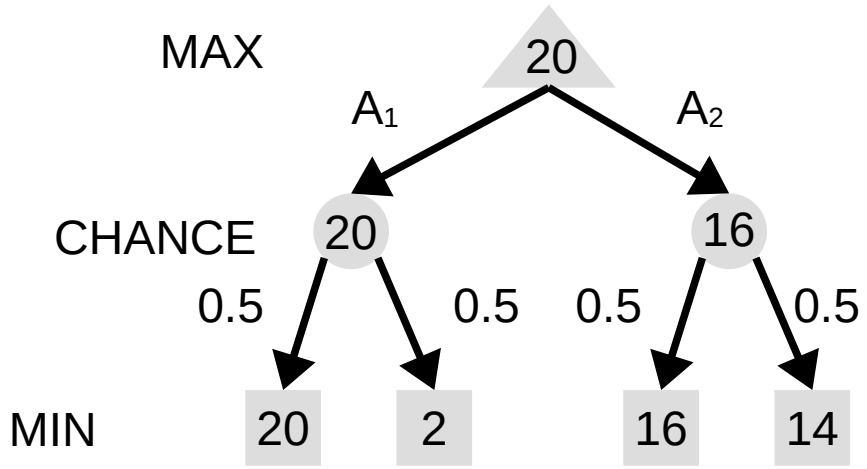


Variation 3: Stochastic Games

- No more deterministic games
  - They contain a random element like the roll of dice.
    - e.g., Dice games, Monopoly, Card games, Minesweeper.
- Problem
  - MIN makes a roll of the dice after MAX has completed his ply.
    - Player MAX cannot directly maximize his gain because he does not know what MIN's legal actions will be.
  - Minimax is no longer applicable.

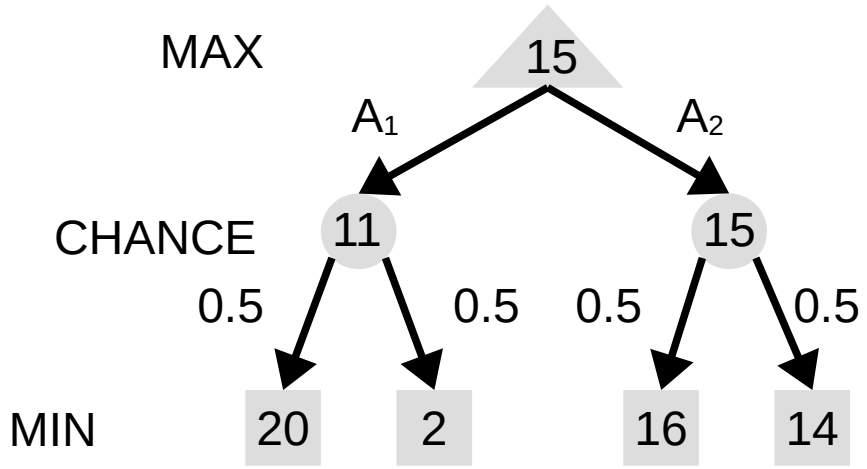
Stochastic Single Player Games

- What if the player doesn't known what the result of an action will be?
  - Solution: add a chance level
  - Which move would MAX choose ?
    - (1) Minimax: take maximum of maximum value
      - A_1



Stochastic Single Player Games

- What if the player doesn't known what the result of an action will be?
  - Solution: add a chance level
  - Which move would MAX choose ?
    - (2) Expectimax: take expectation of value of children.
      - A_2



Stochastic Single Player Games

- Minimax becomes Expectimax
  - Use a weighted average of the value at chance level: **expected value.**
    - P(s) is the probability of the state *s* among all states.

$$\text{EXPECTIMAX(n)} = \begin{cases} UTILITY(n) & \text{if n is a terminal node} \\ max_{s \in SUCCESSORS} EXPECTIMAX(n) & \text{if n is a MAX node} \\ \sum_{s \in SUCCESSORS} P(s).EXPECTIMAX(n) & \text{if n is a CHANCE node} \end{cases}$$

- Complexity
  - $O(b^m c^m)$

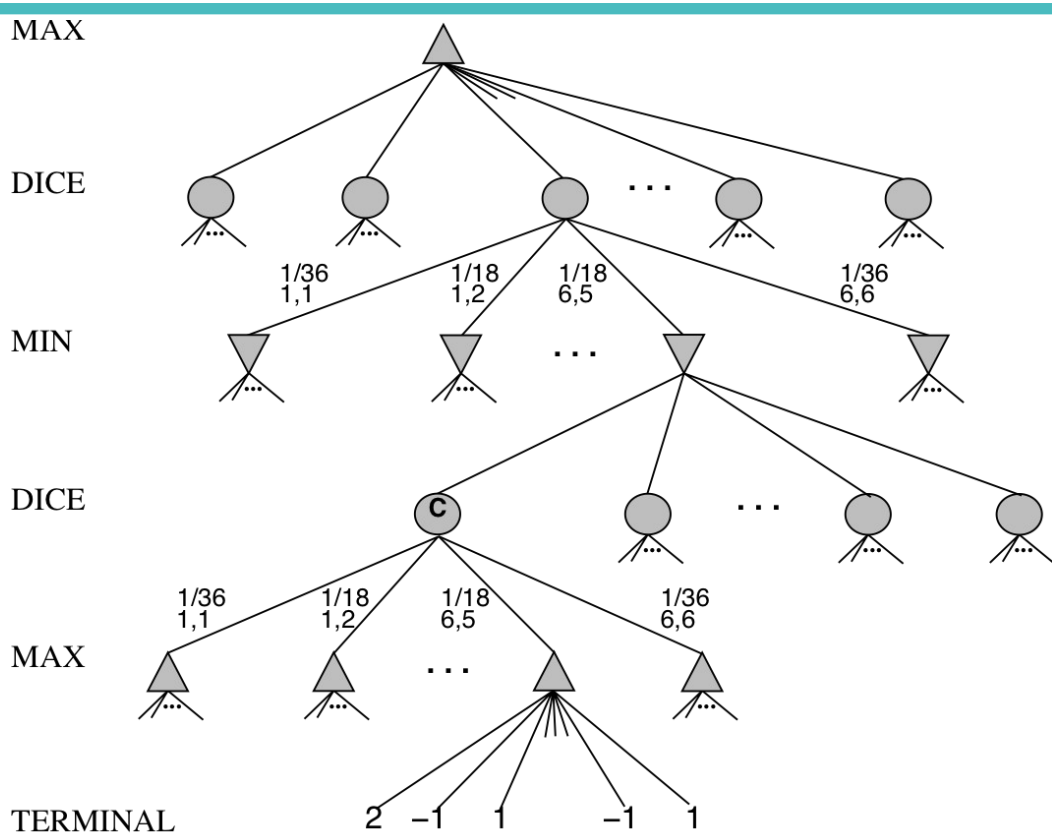
    where b: branching factor

    c: number of chance nodes
- **Caveat** : There no pruning for Expectimax (→ incompatible with Alpha-Beta).

[demo pacman]

Stochastic Multiplayer Games

- Multiplayer stochastic games
  - Example “Monopoly” or “Backgammon”
    - Players throw 2 dices before deciding what to do according to the dice values.
  - What is the best play for Max after throwing the dices?
- Add a choice level for each player



- Expectimax becomes Expertiminimax

$$\text{EXPECTIMINIMAX(n)}=\begin{cases} UTILITY(n) & \text{if n is a terminal node} \\ max_{s \in SUCCESSORS} EXPECTIMINIMAX(n) & \text{if n is a MAX node} \\ min_{s \in SUCCESSORS} EXPECTIMINIMAX(n) & \text{if n is a MIN node} \\ \sum_{s \in SUCCESSORS} P(s).EXPECTIMINIMAX(n) & \text{if n is a CHANCE node} \end{cases}$$

EXPECTIMINIMAX Algorithm


```
function EXPECTIMINIMAX(node, depth)
    IF node is a terminal node or depth = 0
    THEN return the heuristic value of node
    ELSE IF MIN is to play at node
    THEN // get value of minimum-valued child node
         α ← +∞
         FOREACH child of node DO
            α ← min(α, EXPECTIMINIMAX(child, depth - 1))
    ELSE IF MAX is to play at node
    THEN // get value of maximum-valued child node
         α ← -∞
         FOREACH child of node DO
            α ← max(α, EXPECTIMINIMAX(child, depth - 1))
    ELSE IF random event at node
    THEN // get weighted average of all child node values
         α ← 0
         FOREACH child of node DO
            α ← α + (Probability[child]
                     * EXPECTIMINIMAX(child, depth - 1))
    return α
end
```

Variation 4: Games of Imperfect Information

- The players do not have access to the entire world state.
  - Typical games : card games when opponent's cards are unknown (Poker, French Tarot).
- Solution: We have good perfect information-solvers
  - How can we use them for imperfect information games?
    - Solution: sample all unknown information in order to build multiple perfect information worlds.

Games of Imperfect Information

- Monte Carlo Sampling
  - For each move, sample possible continuation using a randomized playing policy for both players.
  - Solve perfectly with Alpha-Beta (or Expectiminimax for stochastic games).
  - Take the average best move of all continuations.
- Main problems
  - Too many possible deals to do this efficiently.
    - e.g., GIB (currently the best Bridge program) generates only 100 deals consistent with bidding information.
  - Probability of reaching a given node shrinks with depth (*search tree becomes less representative*).
    - e.g., TDGammon (best backgammon program) uses only depth-2 search + very good eval function.

Conclusion

- Solutions for adversarial games are based on game tree exploration.
  - Minimax
- Game tree size is exponential. To limit the game tree size:
  1) Remove illegal positions, take care of reflections, rotations, etc.
  2) Alpha-Beta
  3) H-Minimax, H-Alpha-Beta (*in practice, minimax is never use → h-minimax*)
- Variations
  - Non zero-sum games: use of tuples.
  - Stochastic games: Expectiminimax algorithm (No more Alpha-Beta pruning)
  - Game of imperfect information: Monte Carlo sampling
- Classical game conception: eg. checkers.
  - Opening: Database automatically computed in back-office up to 5 levels.
  - Middle game: Kingdom of Alpha-Beta with evaluation function based on weighed sum of game position features (weights are learned using Reinforcement Learning.)
  - Endgame: Database automatically computed in back-office (all finals with less than 8 pawns.)

Game Playing: State-of-the-Art

- **Checkers** (5×10^{20}): 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. Checkers is now solved (18 years of calculus)!
- **Othello (Reversi)**: Human champions refuse to compete against computers, which are too good. Last human winner Takeshi Murakami en 1997. Othello is solved.
- **Chess** ($O(10^{40})$): Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue examined 200 million positions per second, used very sophisticated evaluation. Chess is not solved.
- **Go** ($O(300^{150})$): Human champions are challenged by machines. On March 19, 2016, AlphaGo (deep learning) beat the strongest Go player in the world. However, Go is not solved.

[reversi.sh]