Part II
Problem Solving as Searching Path in State Space
Chapter 8
Adversarial Search

“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

Régis Clouard
regis.clouard@ensicaen.fr
ENSICAEN 14050 Caen

Part II: Problem Solving as Searching

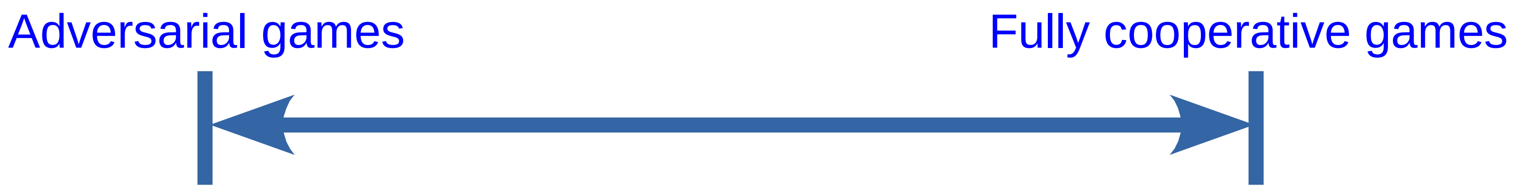
1. Search Problem Formulation
2. Uninformed Search
3. Informed Search
4. Local Search
5. Constraint Satisfaction Search
6. Adversarial Search
    1. Introduction to game search
    2. Game as search
    3. Optimal solution: Minimax algorithm
    4. Pruning to reduce cost: Alpha-Beta algorithm
    5. Approximate solution: Minimax cutoff algorithm
    6. Variations on deterministic 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**
        - Clear criteria for success
        - Rules are fixed
    - Games rise hard problems which require minimal “initial structure”
        - Games often define **very large search spaces**
- They illustrate several important points about AI
    - **perfection is unattainable** → must approximate
    - good idea **to think about what to think about**
- Game is the Drosophila for Artificial Intelligence

Types of Game Problems

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



- 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)**
  - Deterministic: No chance (no dice rolls)
- **Zero-sum / Non-Zero sum**
  - Zero-sum: one players loss is the other’s gain.
- **Asynchronous / Synchronous**
  - Asynchronous: players act alternatively

What games Are Like These?

- Chess, Checkers
  - Asynchronous, zero-sum, deterministic, perfect information
- Battleship
  - Asynchronous, zero-sum, deterministic, imperfect information
- Poker, Bridge, French Tarot
  - Asynchronous, zero-sum, non-deterministic, imperfect information
- Rock-paper-scissors (chifoumi)
  - Synchronous, zero-sum, deterministic, imperfect information
- Prisoner dilemna
  - Synchronous, non zero-sum, deterministic, imperfect information

Game Setup

- Assumptions made for this chapter
  - Asynchronous
  - Zero-sum
  - Perfect information
  - Deterministic
  - Clear rules for legal moves
  - Well-defined outcomes: Win/Loss/Draw
  - Multiplayer
    - Single-players: It's just search.

Part II: Problem Solving as Searching

1. Search Problem Formulation
2. Uninformed Search
3. Informed Search
4. Local Search
5. Constraint Satisfaction Search
6. Adversarial Search
    1. Introduction to game search
    2. Game as search

Search vs. Games

- **Search**
  - No adversary
  - Solution is (heuristic) method for finding goal
  - Heuristics and CSP techniques can find optimal solution
  - Evaluation function: estimate of cost from start to goal through given node
- **Games**
  - Adversary
  - Solution is strategy (aka* policy)
    - strategy specifies move for every possible opponent reply
  - Time limits force an approximate solution
  - Evaluation function: evaluate “goodness” of game position

* aka: also known as

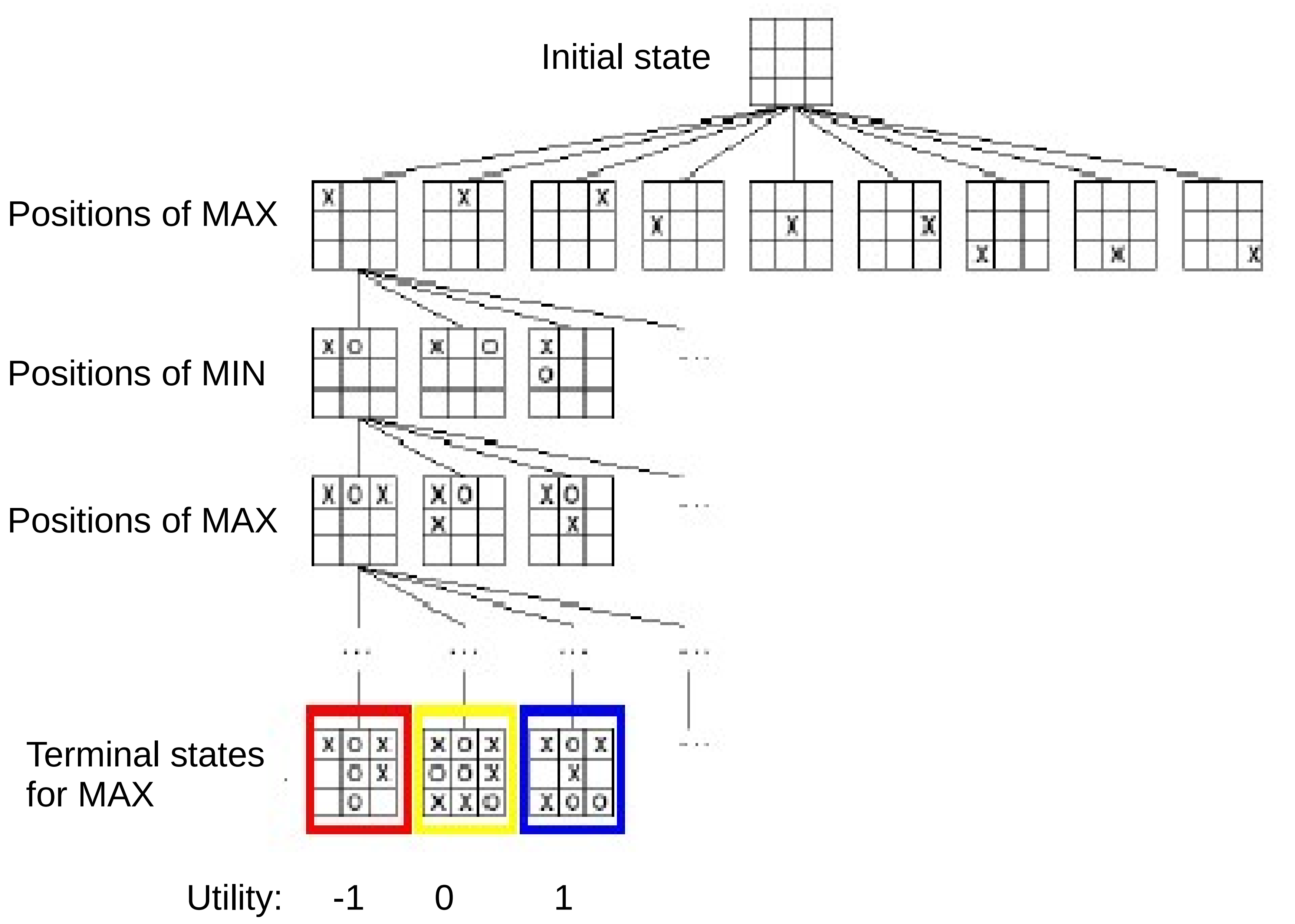
Game Problem

- Game problem formulation
  - **State** S : Board configuration + whose move it is
    - **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): $S \rightarrow$ {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
- Search objective
  - Find the sequence of player’s decisions (moves) maximizing its utility (payoff): $S \rightarrow A$

Game Tree

- Example with two-player game: MAX and MIN
  - The positions are evaluated from the MAX perspective
  - 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 (Zero sum games)
  - Each node is assigned with a **utility**
    - Example in Tic-Tac-Toe
      - +1 for a win, -1 for a loss, 0 for a draw

Example. Tic-Tac-Toe Game Tree

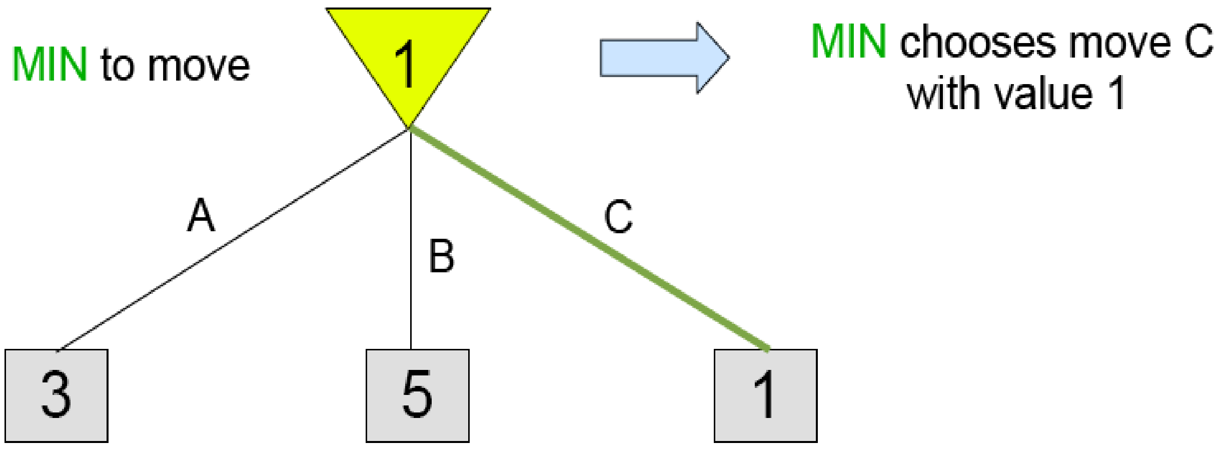


Part II: Problem Solving as Searching

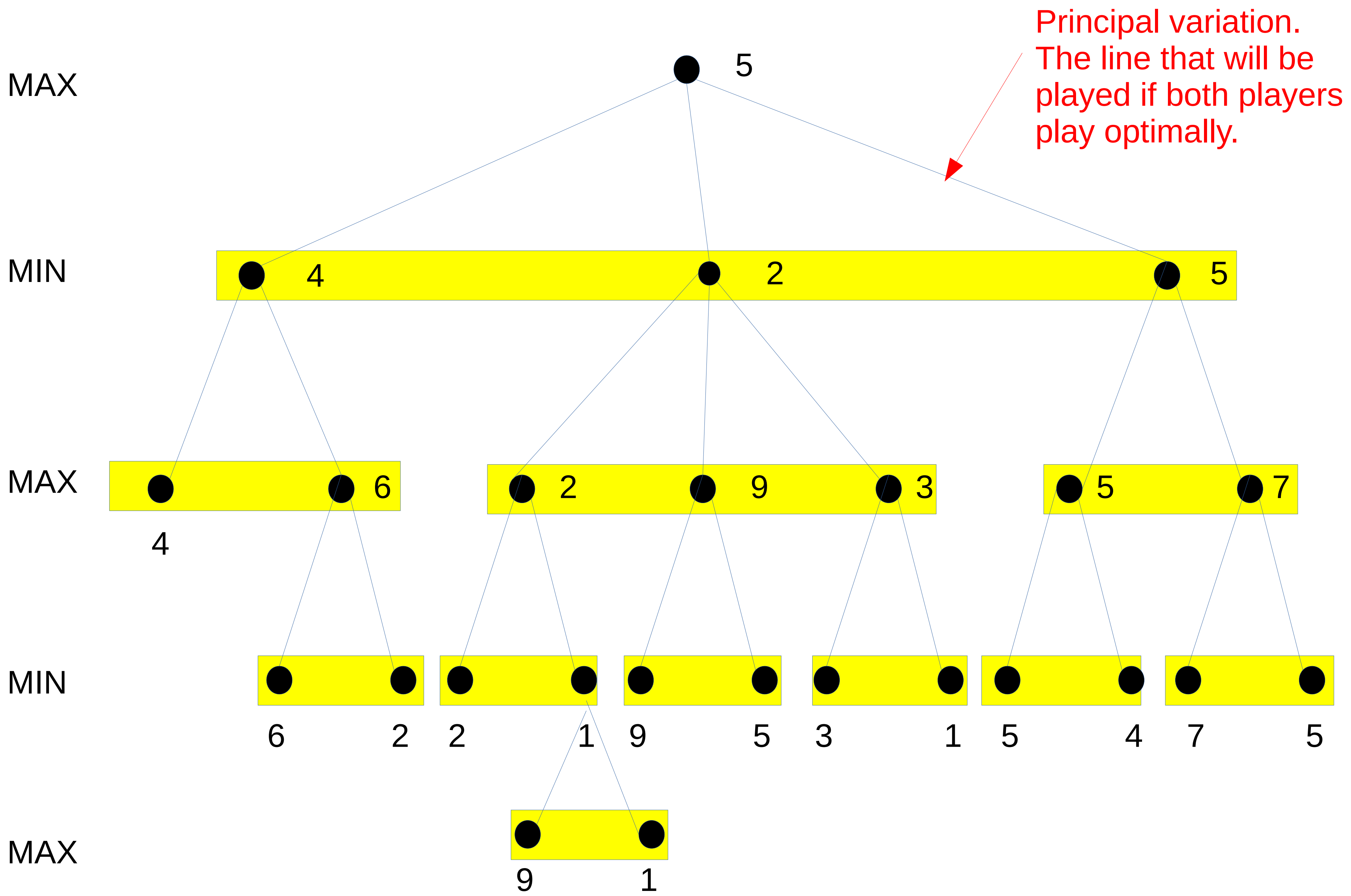
Minimax Algorithm (Von Neumann, 1945)

- Perfect play for deterministic, perfect-information games
  - Find the best strategy for MAX
  - Assumption: Both players play optimally
- 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 SUCCESSORS(state) with value v
end
```

```
function MAX-VALUE(state) 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))
    return v
end
```

```
function MIN-VALUE(state) 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))
    return v
end
```

Properties of Minimax Algorithm

- **Complete?**
  - Yes (if tree is finite)
- **Optimal?**
  - Yes (against an optimal opponent)
- **Time complexity?**
  - $O(b^m)$
- **Space complexity?**
  - $O(bm)$

Examples of Complexity

- The game tree is generally gigantic:
  - Example: Tic-tac-toe.
    - Total number of terminal states: 3^9 = 19683.
  - Example: Chess.
    - Number of actions by turn: b ~ 35.
    - Average number of moves in a game: 50.
    - Total number of states: 35^{50}.
  - Example: Go.
    - Number of actions by turn: 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.

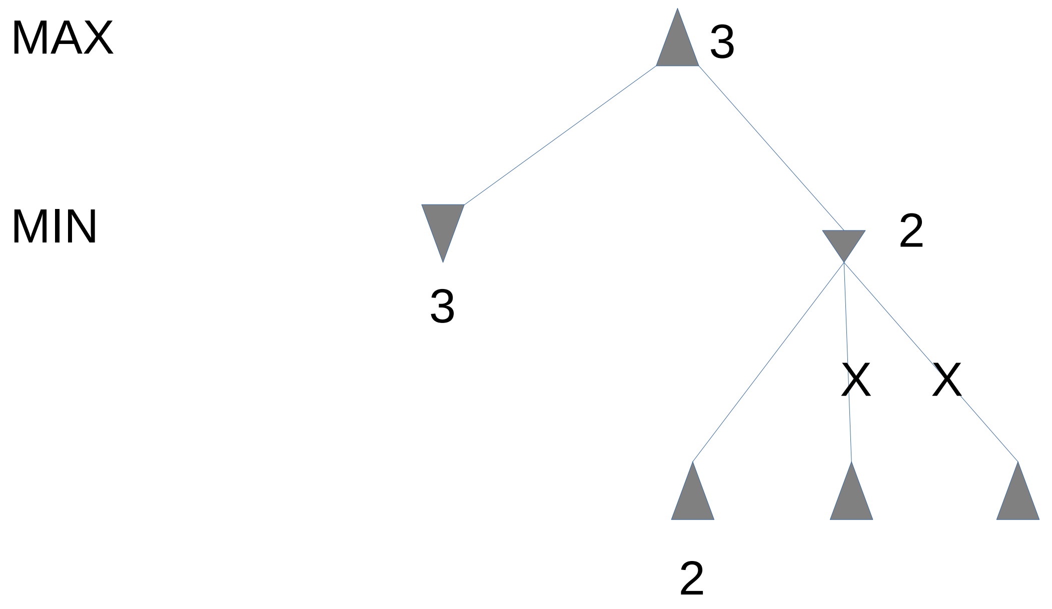
Speed-Up Search

- Number of game states is exponential to the number of moves.
    - Solution: Do not examine every node
- Two complementary approaches
    - Limit the branching factor (prune search tree)
        - Alpha-beta pruning
    - Limit the depth search (cut-off search tree)
        - Minimax cutoff

Part II: Problem Solving as Searching

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

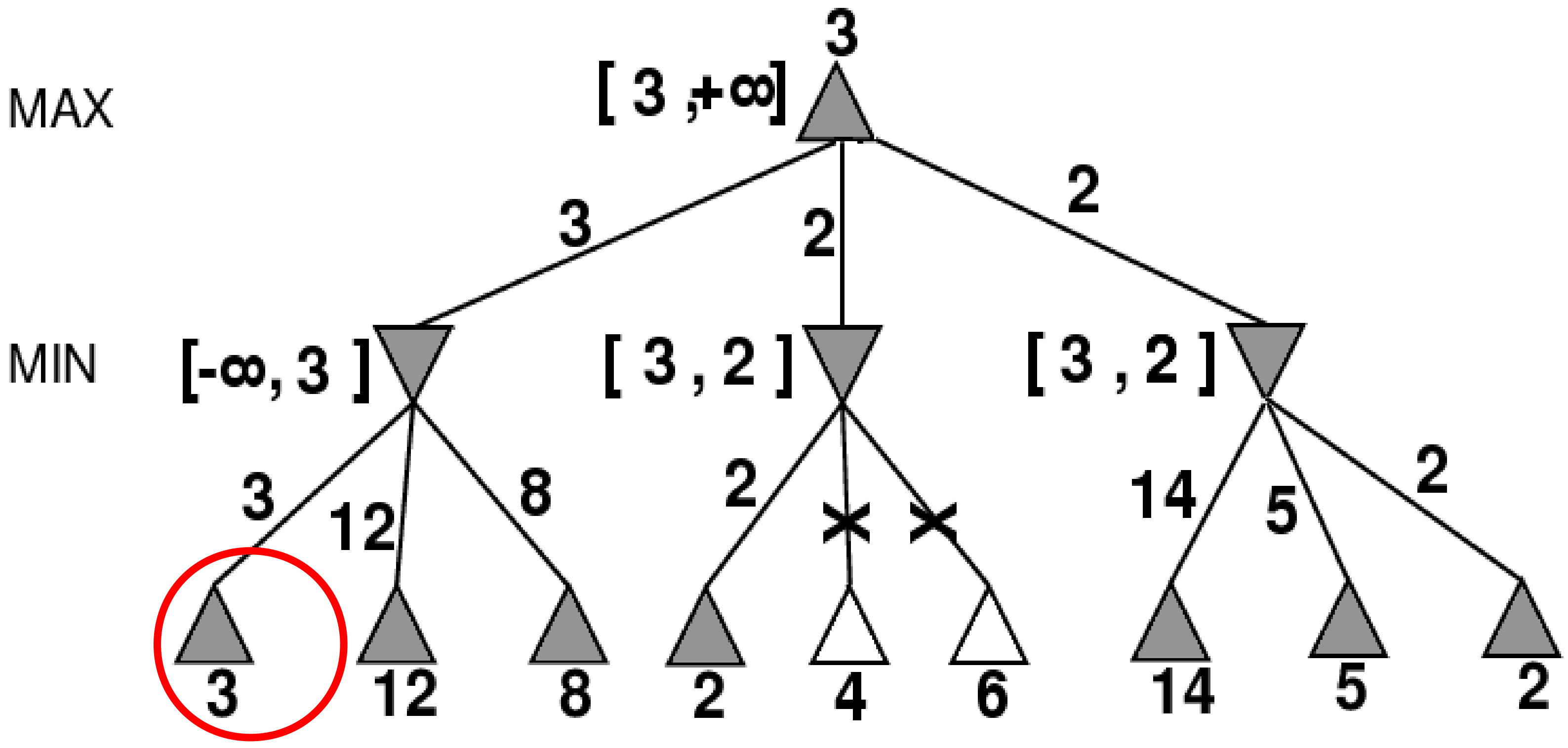
- Maintains two values [α,β] for all nodes in the current path
- Alpha
  - The value of the best choice (i.e., highest value) for the MAX player at any choice node for MAX in the current path
    → MAX can obtain a value of at least α
- Beta
  - The value of the best choice (i.e., lowest value) for the 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

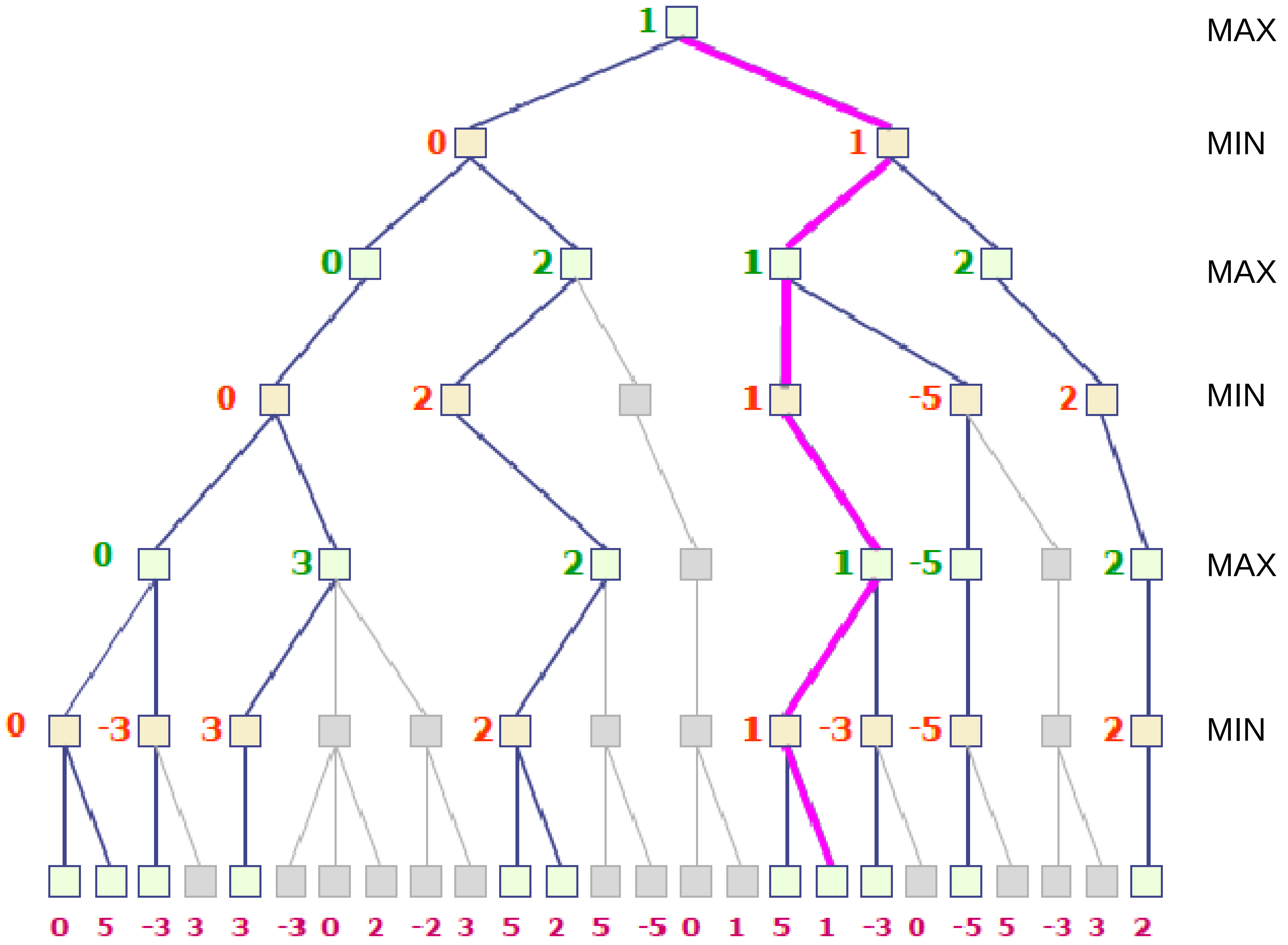
- Alpha and Beta are used for pruning the search tree:
- Alpha-Cutoff:
  - if we find a move with value ≤ α at a MIN node, we do not examine alternatives to this move
  - we already know that MAX can achieve a better result in a different variation
- Beta-Cutoff:
  - if we find a move with value ≥ β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

Alpha-Beta Pruning

(Mc Carthy, 1946)



◆ Note: order of successors has influence on performance!



MAX
MIN
MAX
MIN
MAX
MIN
1
0
1
0
2
1
2
0
2
1
-5
2
0
3
2
1
-5
2
0
-3
3
2
1
-3
-5
2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

The Alpha-Beta Algorithm

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

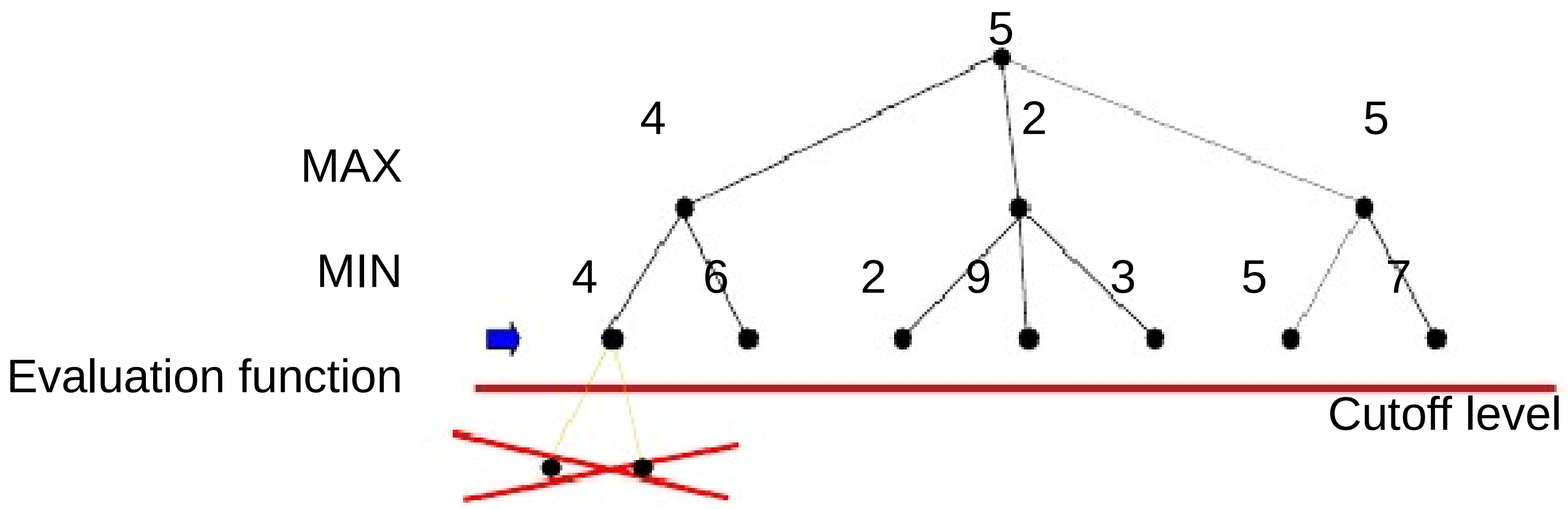
Properties of Alpha-Beta Pruning

- Pruning **does not** affect final result
  - ALPHA-BETA results = MINIMAX results
- Good moves ordering improves effectiveness of pruning
  - But rarely decidable beforehand
- **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!
  - However, in most cases, this allows to add one level to depth search

Part II: Problem Solving as Searching

Minimax Cutoff Algorithm
(Shannon, 1950; Samuel, 1952)

- Resource limits
  - In most games, even with alpha-beta pruning not possible to explore the whole search space e.g. for chess!
- 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**



Cutting Off Implementation

- MINIMAX-CUTOFF is identical to MINIMAX-DECISION except
  - **Terminal test** is completed with **Cutoff test**
  - **Utility** is replaced by **Eval**
- Does it work in practice?
  - If we have room for 10^6 resource and b = 35
    - Can reach depth m = 4 ($O(b^m)$)
- Chess player levels!
  - 4-ply lookahead ≈ human novice
  - 8-ply lookahead ≈ typical PC, human master
  - 12-ply lookahead ≈ Deep Blue, Kasparov

Evaluation of a Position

- Objective
  - Evaluate the goodness of a game position
    - 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
- The zero-sum assumption allows us to use a single evaluation function for each player
  - 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 encodes a certain characteristic of the position
- Advantages
  - Conceptually simple, typically fast to compute
- Disadvantages
  - Tuning of the weights w_i ($\rightarrow$ machine learning)

Evaluation Function. Example

- Tic-Tac-Toe
  - eval(s) = [# 3-lengths open for me] - [# 3-lengths open for you] where 3-lengths is a complete row, column, or diagonal
- Checkers
  - eval(s) = (# my pieces) - (# his/her 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 6000 features
- Current-state of the art uses non-linear functions
  - e.g., different feature weights in different game phases

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)
- 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

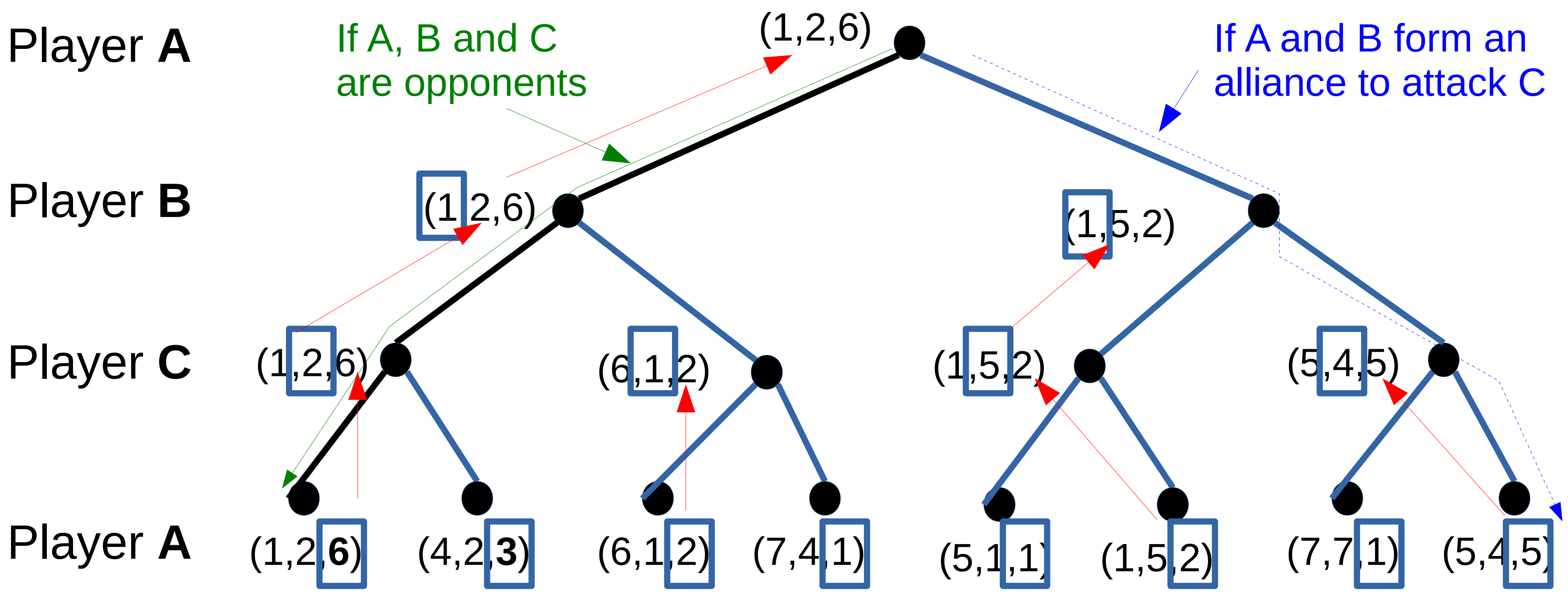
Iterative Deepening

- 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 managements
    - previous iterations provide useful information that allows to guess whether the next iteration can be completed in time
- Frequently used in game-playing programs
  - Quite frequently the total number of nodes searched is smaller than with non-iterative search!

Part II: Problem Solving as Searching

Multiplayer Games (> 2 Players)

- No more zero-sum games, so 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 or alpha-beta algorithms
    - eval(s) = $f(f_A(s), f_B(s), f_C(s))$: cooperation or competition

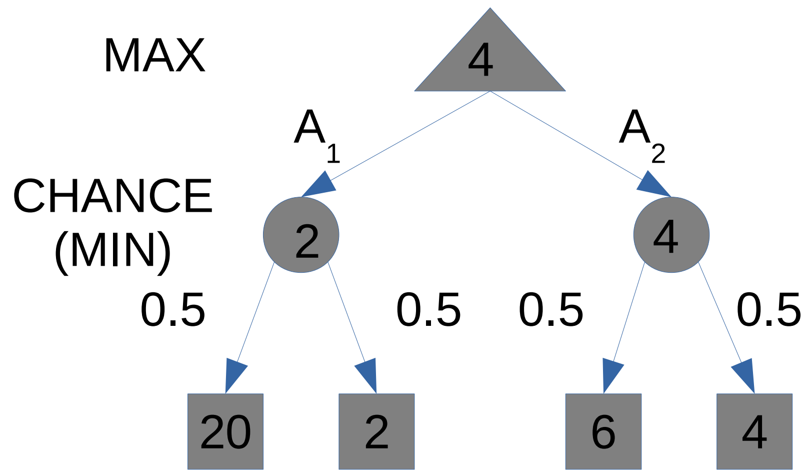


Stochastic Games

- Many games combine skill and chance
  - ie., they contain a random element like the roll of dice
    - e.g., Dice games, Monopoly, Card games, Minesweeper
- Problem
  - Player MAX cannot directly maximize his gain because he does not know what MIN's legal actions will be
    - MIN makes a roll of the dice after MAX has completed his ply
    - and vice versa (MIN cannot minimize)
  - MINIMAX is no longer applicable

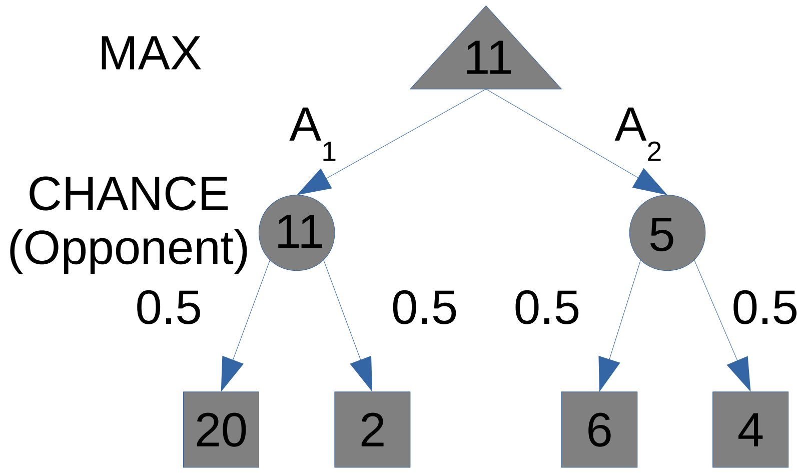
Stochastic Single Player Games

- Idea
  - Add chance level (dice rolls, card deals)
  - Which move would MAX choose ?
    - Minimax
      - A_2



Stochastic Single Player Games

- Idea
  - Add chance level (dice rolls, card deals)
  - Which move would MAX choose ?
    - Expectimax: take average (expectation) of value of children
      - A_1



Stochastic Single Player Games

- Minimax becomes Expertimax
  - Use a weighted average of the value of the chance nodes as the value of the chance node: **expected value**
    - P(s) is the probability of the state *s* among all states.

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

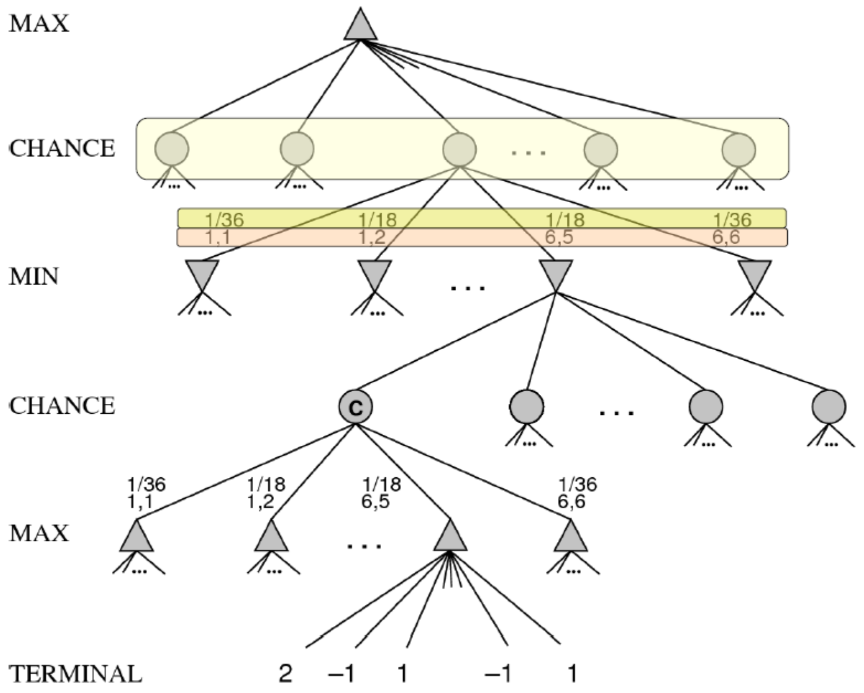
- Complexity
  - Complexity: O($b^m c^m$)

    where b : branching factor and c: chance nodes

Stochastic Multiplayer Games

- Multiplayer stochastic games
  - 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 the adversary 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 we are 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
```

Games of Imperfect Information

- The players do not have access to the entire world state
  - e.g., card games when opponent's cards are unknown
- Idea:
  - compute the minimax value of each action in each deal
  - choose the action with the highest expected value
- Main problem
  - too many possible deals to do this efficiently
    - take a sample of all possible deals
    - e.g., GIB (currently the best Bridge program) generates 100 deals consistent with bidding information (this also restricts)
  - probability of reaching a given node shrinks with depth
    - e.g., TDGammon (best backgammon program) uses depth-2 search + very good eval function

Conclusion

- 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 Samuel method
  - Endgame
    - Database automatically computed in back-office.
      - all finals with less than 8 pawns.

Game Playing State-of-the-Art

- **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. Checkers is now solved!
- **Chess**: 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 and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic but grandmaster remains better. Chess is not solved.
- **Othello**: Human champions refuse to compete against computers, which are too good. Last human winner Takeshi Murakami en 1997. Othello is solved.
- Go ($O(300^{150})$): Human champions are beginning to be challenged by machines. On March 19, 2016, AlphaGo Beat the strongest Go player in the world. However, Go is not solved.