Game Playing

吉建民

USTC
jianmin@ustc.edu.cn

2024 年 3 月 20 日

Used Materials

Disclaimer: 本课件采用了 S. Russell and P. Norvig's Artificial Intelligence –A modern approach slides, 徐林莉老师课件和其他网络课程课件，也采用了 GitHub 中开源代码，以及部分网络博客内容

Table of Contents

Game Playing

Game playing was thought to be a good problem for AI research:

- game playing is non-trivial
- Perfect play（最优策略）
  - players need "human-like" intelligence
  - games can be very complex (e.g., Chess, Go)
  - requires decision making within limited time
- games usually are:
  - well-defined and repeatable
  - fully observable and limited environments
- can directly compare humans and computers

Computers Playing Chess

Computers Playing Go

一些棋类的复杂度

游戏	状态空间复杂度	游戏树复杂度
井字棋	10^4	10^5
国际跳棋	10^21	10^31
国际象棋	10^46	10^123
中国象棋	10^48	10^150
五子棋	10^105	10^70
围棋	10^172	10^360

Games vs. search problems

"Unpredictable" opponent（不可预测的对手）⇒ solution is a strategy specifying a move for every possible opponent reply

Time limits ⇒ unlikely to find goal, must approximate
游戏对于低效率有严厉的惩罚

Plan of attack:

- Computer considers possible lines of play (Babbage, 1846)
- Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)
- Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948;Shannon, 1950)
- First chess program (Turing, 1951)
- Machine learning to improve evaluation accuracy (Samuel, 1952-57)
- Pruning（剪枝）to allow deeper search (McCarthy, 1956)

Types of games

	Deterministic	Stochastic (chance)
perfect information	chess, checkers, Go（围棋）, othello	Backgammon（西洋双陆棋） monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble (拼字游戏) nuclear war

Game Theory

- Models strategic interactions as games
- In normal-form games (matrix games), all players simultaneously select an action, and their joint action determines their individual payoff
  - One-shot interaction
  - Can be represented as an n-dimensional payoff matrix, for n players
- A player's strategy is defined as a probability distribution over his possible actions
- Stochastic games is an extension of normal-form games and MDPs in the sense that they deal with multiple agents in a multiple state situation.

Normal-Form Game

- A normal-form game can be defined as a tuple $(n, A_{1\ldots n}, R_{1\ldots n})$ where:
  - n is the number of agents
  - A_i is the action set for player i
    - $A = A_1 \times \cdots \times A_n$ is the joint action set
  - $R_i : A \to \mathbb{R}$ is the reward function of player i
- Each agent i selects policy $\pi_i : A_i \to [0, 1]$ ($\pi_i \in PD(A_i)$), takes action $a_i \in A_i$ with probability $\pi_i(a_i)$, and receives utility $R_i(a_1, \ldots, a_n)$
- Given policy profile $\langle \pi_1, \ldots, \pi_n \rangle$, expected utility to i is

$$R_i(\pi_1, \ldots, \pi_n) = \sum_{a \in A} R_i(a) \prod_{j=1}^{n} \pi_j(a_j)$$

- Agents want to maximise their expected utilities

Normal-Form Game: Prisoners' Dilemma

Example: Prisoner's Dilemma

- Two prisoners questioned in isolated cells
- Each prisoner can **Cooperate** or **Defect**
- Utilities (row = agent 1, column = agent 2):

	C	D
C	-1,-1	-5,0
D	0,-5	-3,-3

Normal-Form Game: Rock-Paper-Scissors

Example: Rock-Paper-Scissors

- Two players, three actions
- **Rock** beats **Scissors** beats **Paper** beats **Rock**
- Utilities:

	R	P	S
R	0,0	-1,1	1,-1
P	1,-1	0,0	-1,1
S	-1,1	1,-1	0,0

Optimality Concepts

Optimality Concepts in Normal-Form Games:

- Best-Response Function: set of optimal strategies given the other agents current strategies.

$$\begin{array}{ll} \pi_i^* \in BR_i(\pi_{-i}) & \text{iff} \\ \forall \pi_i \in PD(A_i) & R_i(\langle \pi_i^*, \pi_{-i} \rangle) \geq R_i(\langle \pi_i, \pi_{-i} \rangle) \end{array}$$

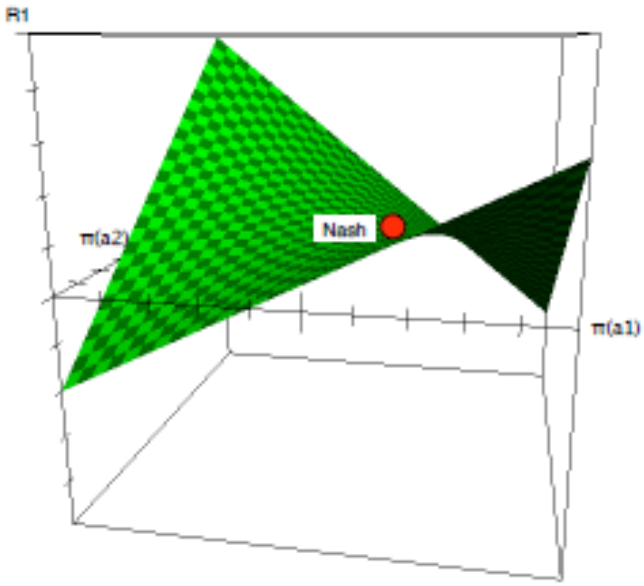
- Nash Equilibria: all agents are using best-response strategies.

$$\forall i = 1 \ldots n \quad \pi_i \in BR_i(\pi_{-i})$$

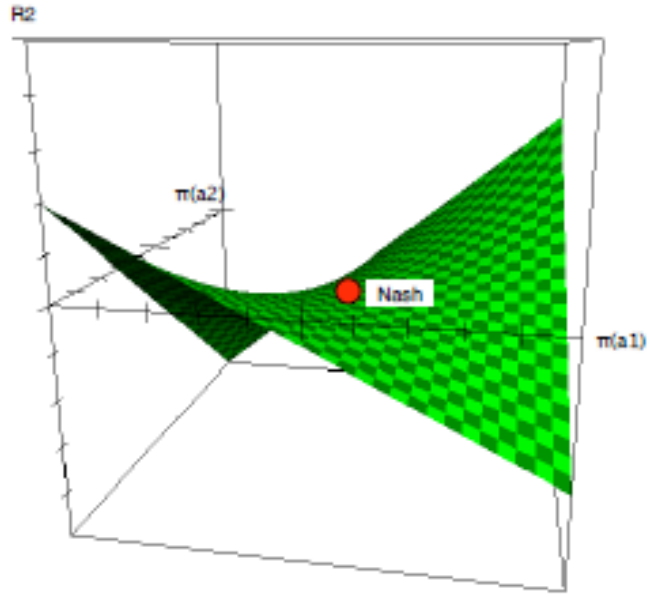
- All Normal-Form Games have at least one Nash Equilibrium

Game Classification: Zero-sum

- 2 players with opposing objectives.
- There is only one Nash equilibrium
  - Minimax to find it.



(a) Reward function for player 1



(b) Reward function for player 2

Two-Player Zero-Sum Games

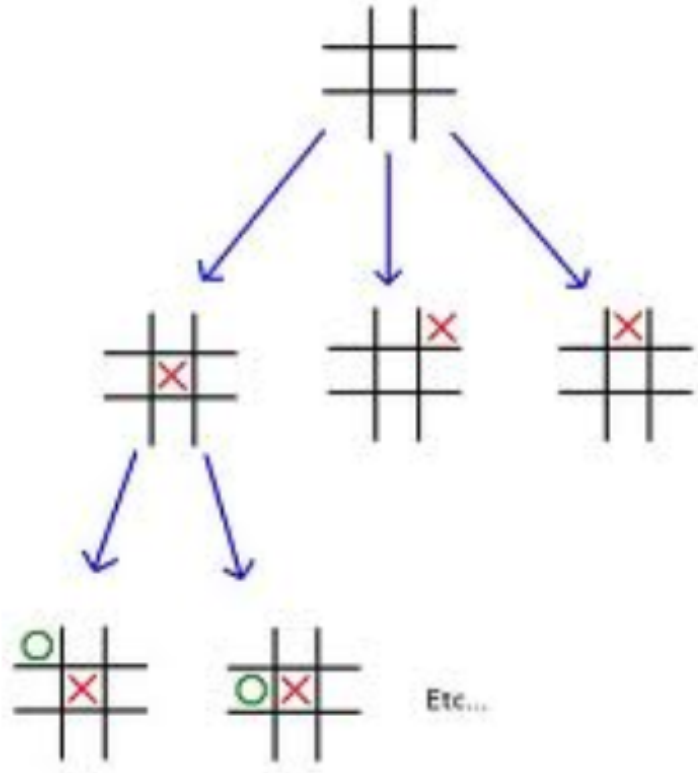
- Characteristics:
  - Two opponents play against each other.
  - symmetrical rewards (always sum zero).
  - Usually only one equilibrium and if more exist they are interchangeable
    - Interchangeable: $\langle \pi_1, \pi_2 \rangle$ 和 $\langle \mu_1, \mu_2 \rangle$ 是两个 Nash equilibria，则 $\langle \pi_1, \mu_2 \rangle$, $\langle \mu_1, \pi_2 \rangle$ 也是 Nash equilibria；并且它们效用都相等
- Minimax to find an equilibrium $(2, A, O, R, -R)$:

$$\max_{\pi \in PD(A)} \min_{o \in O} \sum_{a \in A} \pi(a) R(a, o)$$

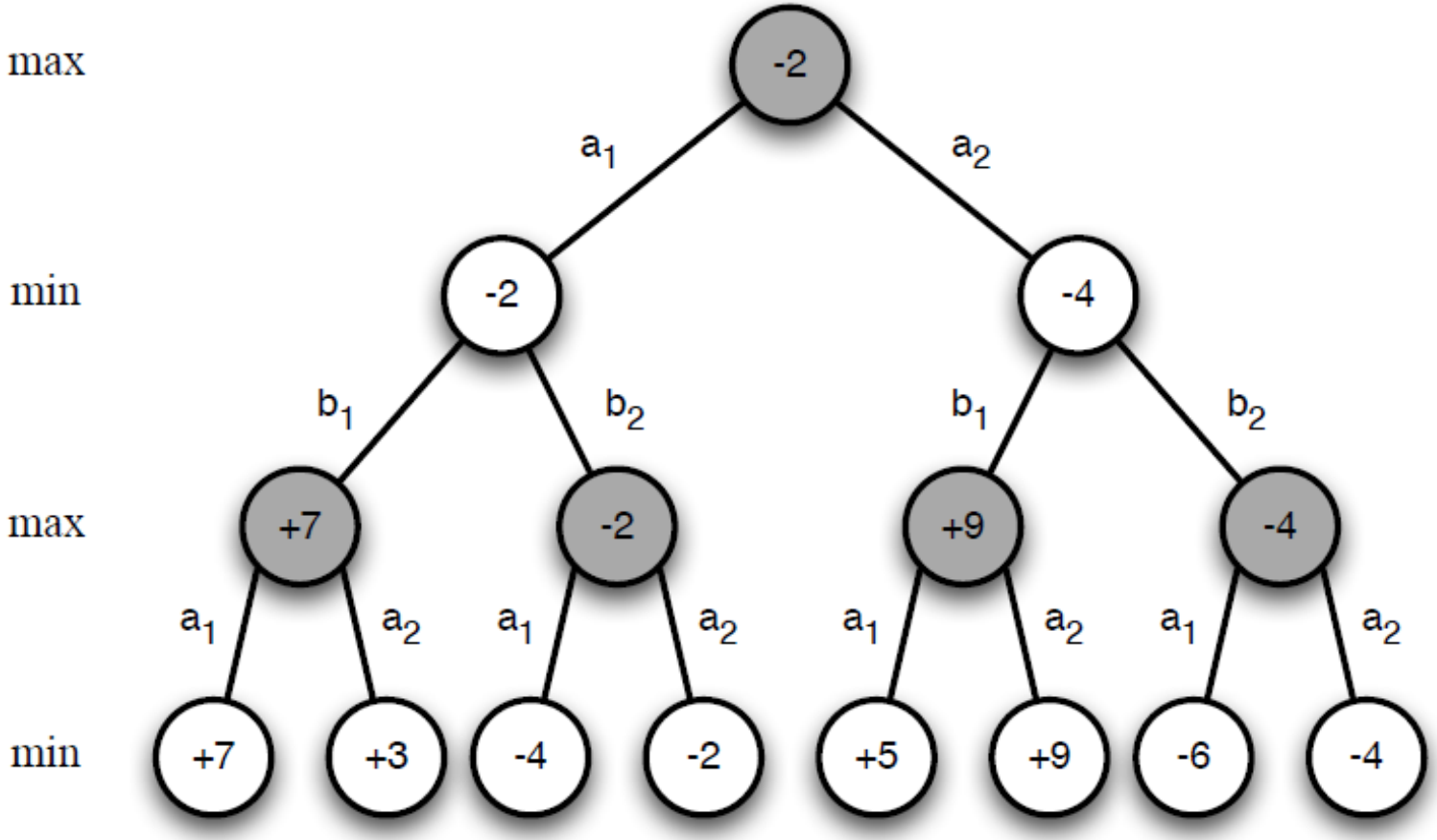
  - Formulated as a Linear Program.
  - Solution in the strategy space: simultaneous playing invalidates deterministic strategies.

Minimax Search

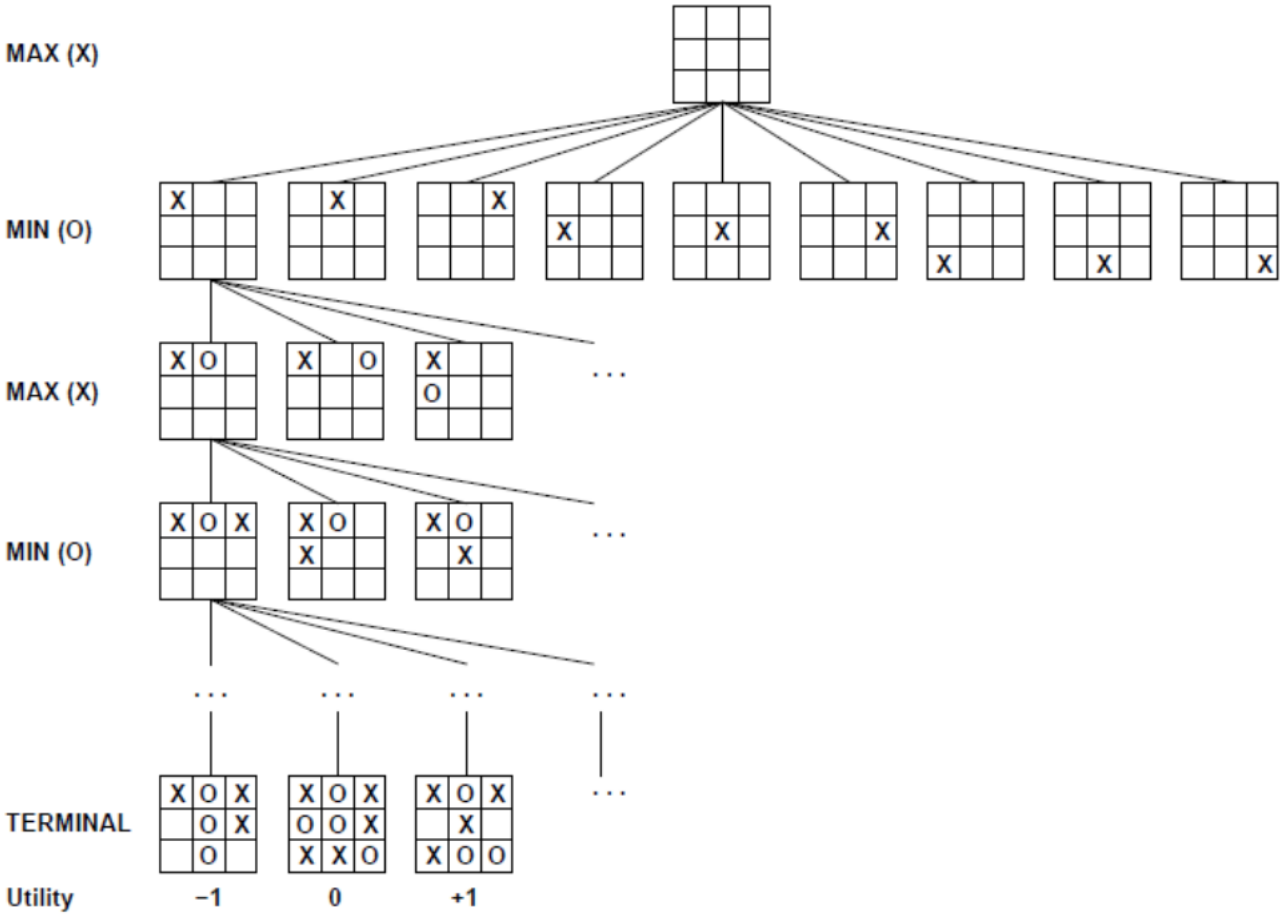
- Minimax values can be found by depth-first game-tree search
- Introduced by Claude Shannon: *Programming a Computer for Playing Chess*
- Ran on paper!



Minimax Search Example



Game tree (2-player, deterministic, turns)



Deterministic Two-Player

- E.g. tic-tac-toe, chess, checkers
- Game search
  - A state-space search tree
  - Players alternate
  - Players Each layer, or ply, consists of a round of moves
  - Choose move to position with highest achievable utility
- Zero-sum games
  - One player maximizes result
  - The other minimizes result

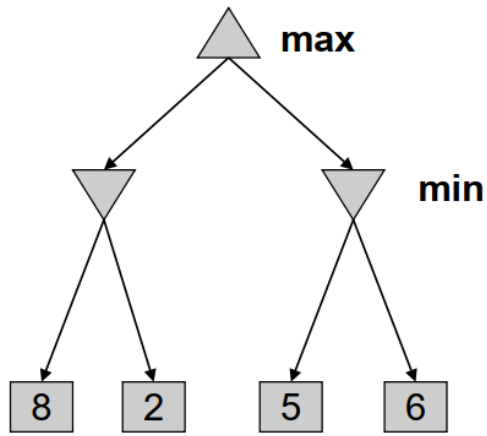
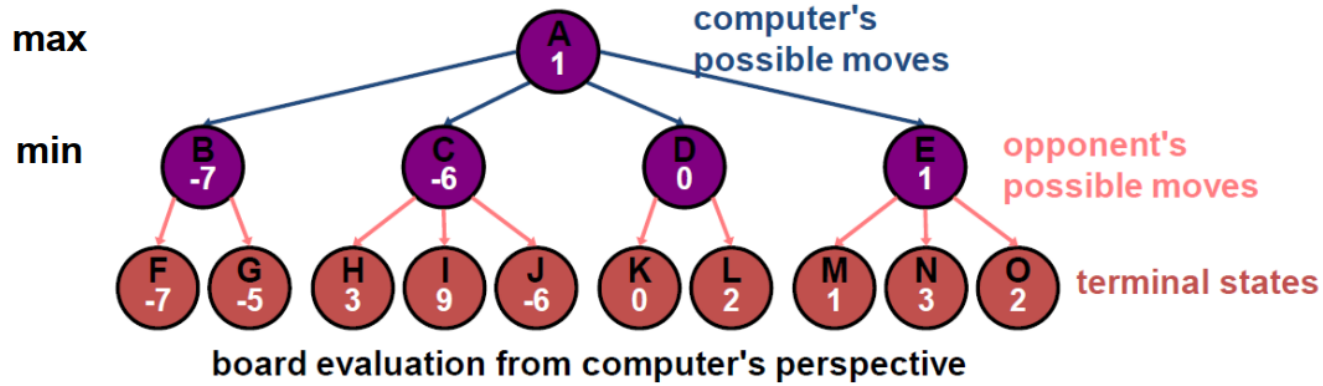


Table of Contents

Minimax Principle

- **Assume both players play optimally**
  - The computer assumes after it moves the opponent will choose the minimizing move
  - The computer chooses the best move considering both its move and the opponent's optimal move



Minimax

Perfect play（最优策略）for deterministic, perfect-information games

Idea: choose move to position with highest minimax value

= best achievable payoff against best play

在对手也使用最优策略的条件下，能导致至少不比其它策略差的结果

假设两个游戏者都按照最优策略进行，那么节点的极小值（MIN 节点）或极大值（MAX 节点）就是对应状态的效用值

- MAX 优先选择有极大值的状态
- MIN 优先选择有极小值的状态

$$\text{MINMAX-VALUE}(n) = \begin{cases} \text{UTILITY}(n) & \text{当}n\text{为终止状态} \\ \max_{s \in Successors(n)} \text{MINMAX-VALUE}(s) & \text{当}n\text{为MAX节点} \\ \min_{s \in Successors(n)} \text{MINMAX-VALUE}(s) & \text{当}n\text{为MIN节点} \end{cases}$$

Minimax

Perfect play（最优策略）for deterministic, perfect-information games

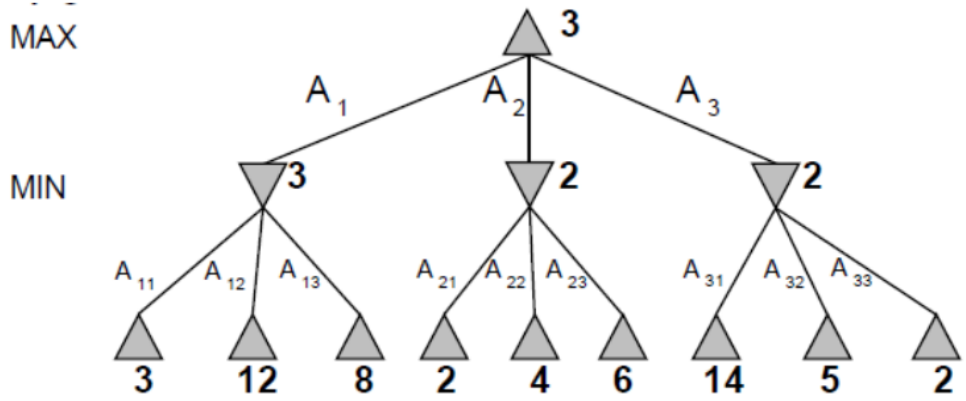
Idea: choose move to position with highest minimax value

= best achievable payoff against best play

在对手也使用最优策略的条件下，能导致至少不比其它策略差的结果

假设两个游戏者都按照最优策略进行，那么节点的极小值（MIN 节点）或极大值（MAX 节点）就是对应状态的效用值

E.g., 2-ply game:



Minimax algorithm

```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game
    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
    return v
```

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)

Optimal?? Yes, against an optimal opponent. Otherwise??

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

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

For chess, b≈35, m≈100 for "reasonable" games
$\Rightarrow$ exact solution completely infeasible

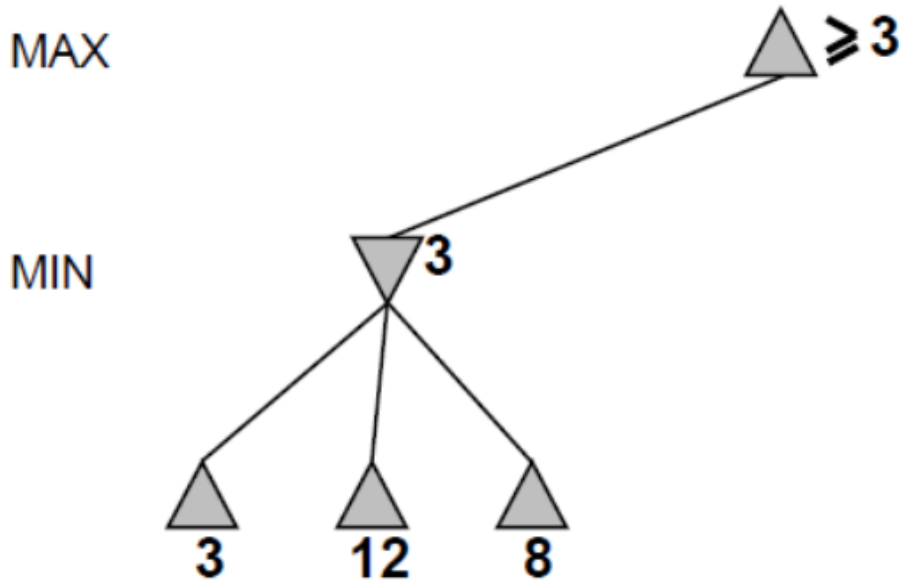
But do we need to explore every path?

$\alpha - \beta$ Pruning

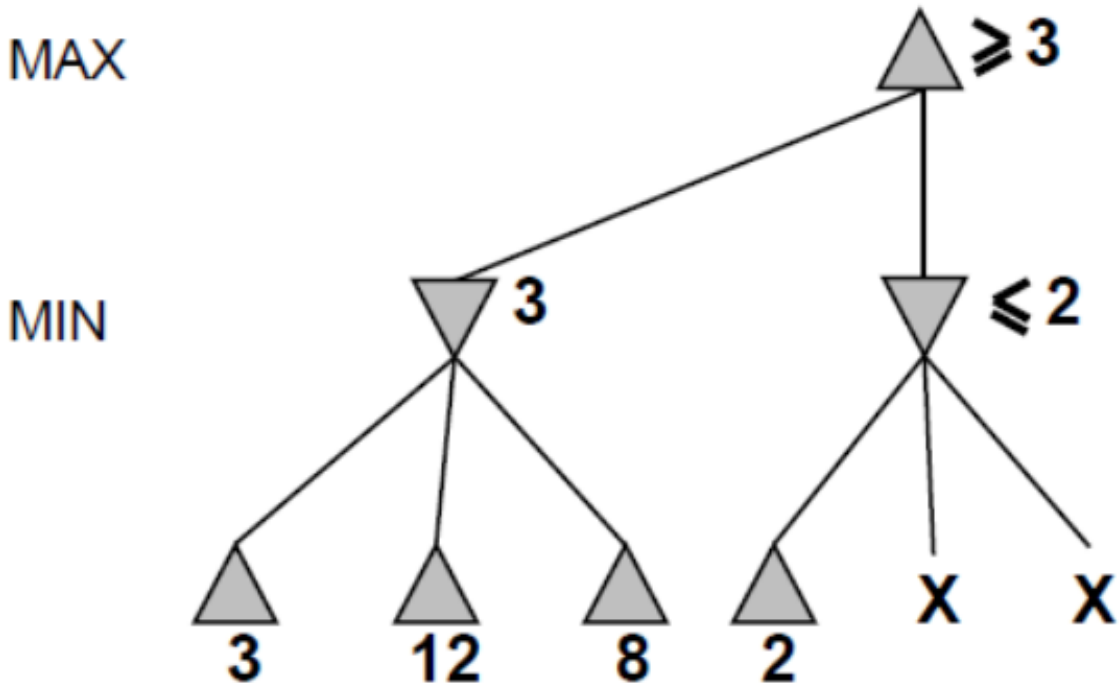
- Some of the branches of the game tree won't be taken if playing against an intelligent opponent
- "If you have an idea that is surely bad, don't take the time to see how truly awful it is."

  – Pat Winston
- Pruning can be used to ignore some branches

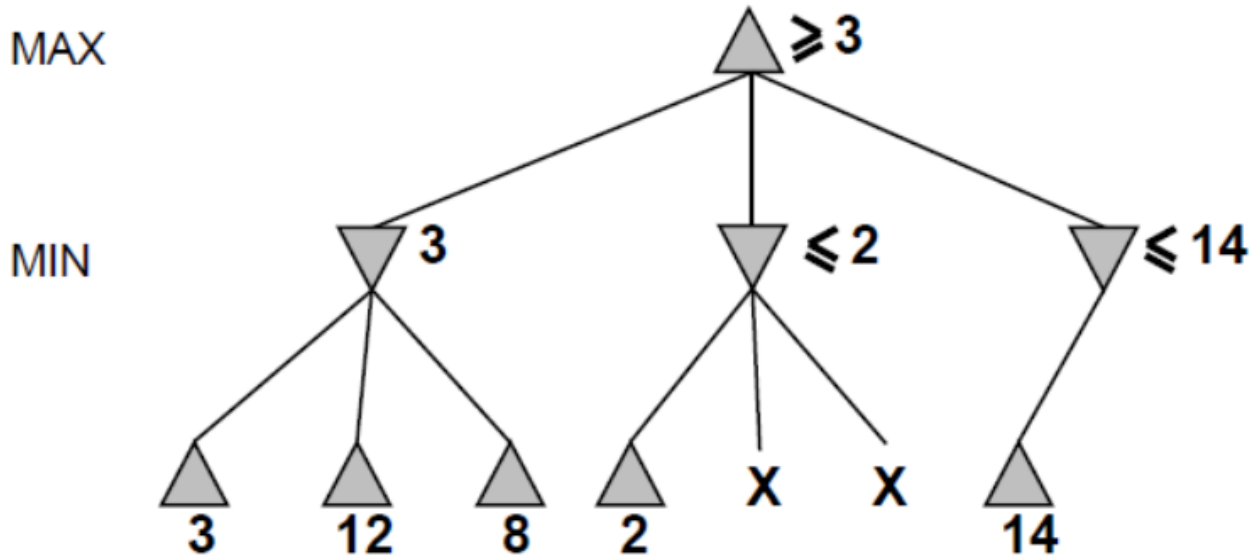
$\alpha - \beta$ pruning example



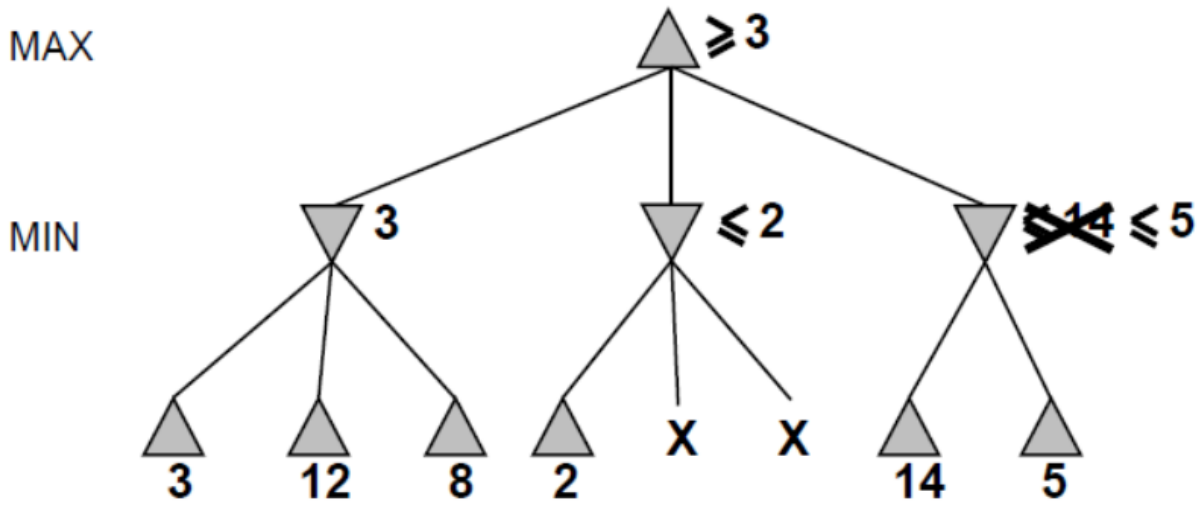
$\alpha - \beta$ example



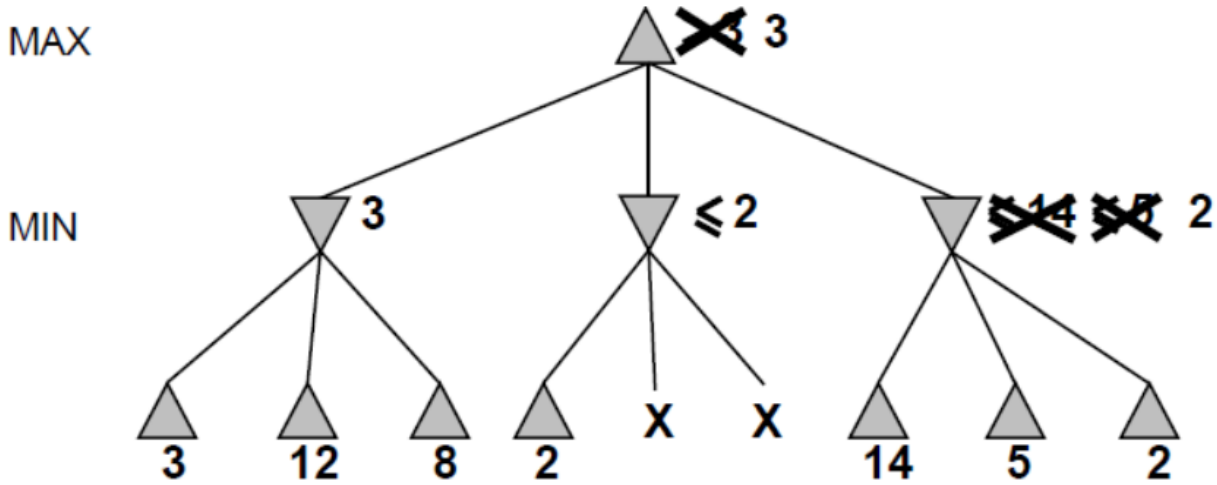
$\alpha - \beta$ pruning example



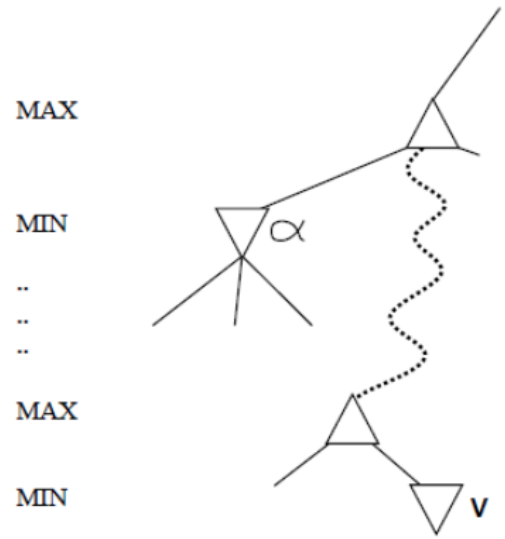
$\alpha - \beta$ pruning example



$\alpha - \beta$ pruning example



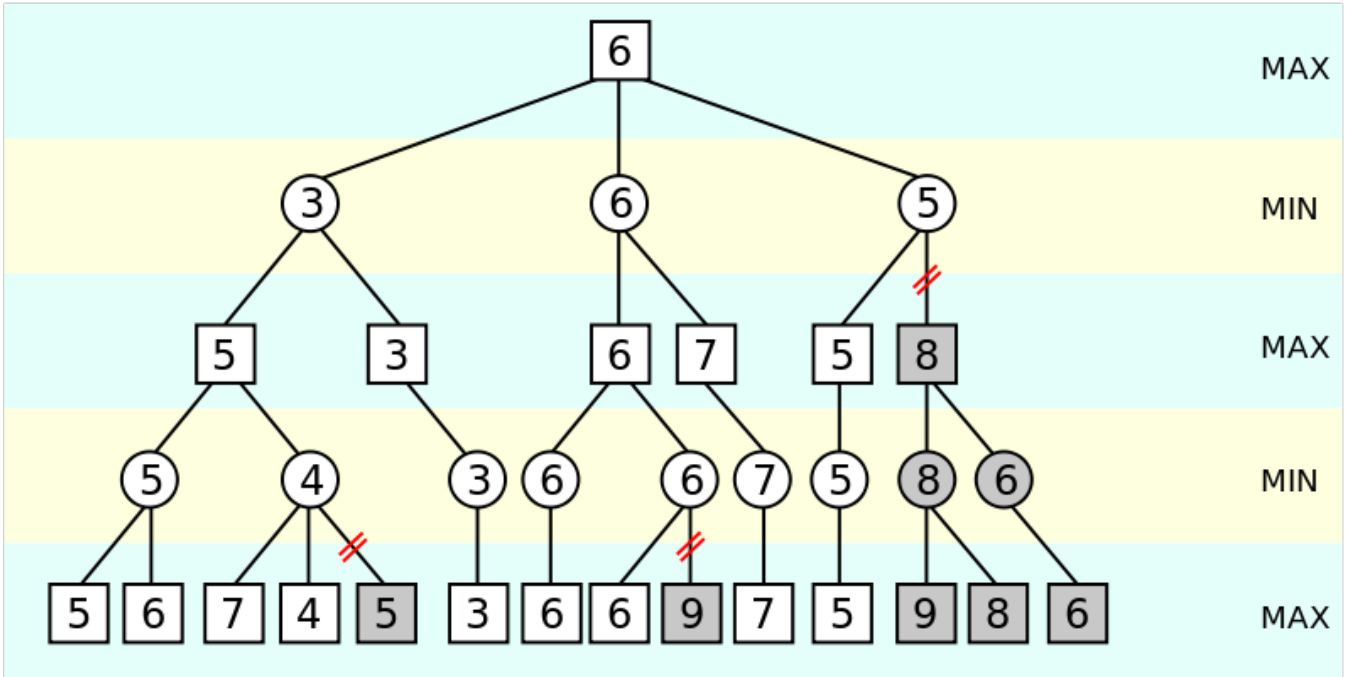
Why is it called $\alpha - \beta$



- α is the best value (to MAX) found so far on the current path
  到目前为止在路径上的任意选择点发现的 MAX 的最佳（即最大值）选择
- If v is worse than α, MAX will avoid it, so can stop considering v's other children $\Rightarrow$ prune that branch
- Define β similarly for MIN

$\alpha - \beta$ pruning

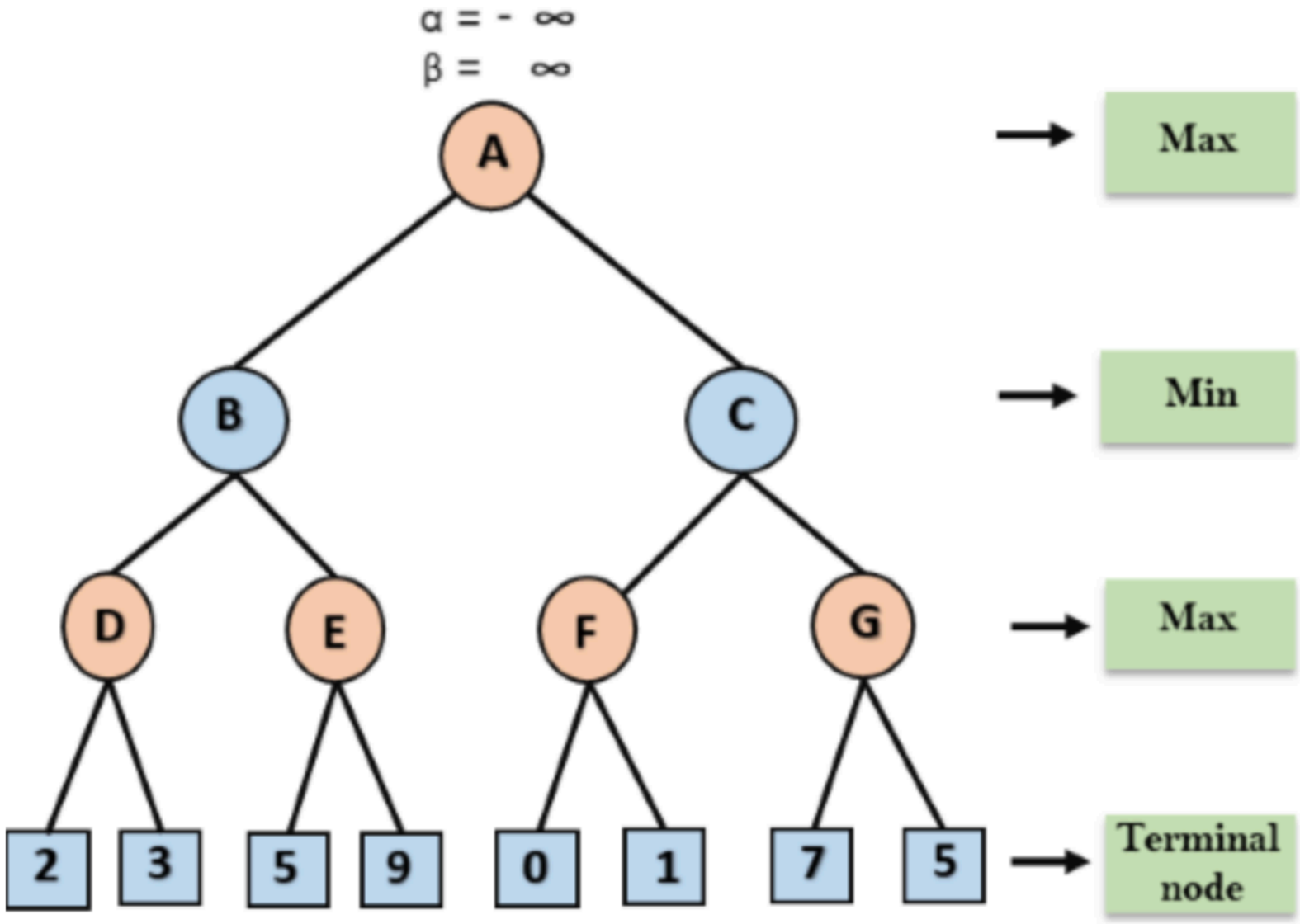
- α: the minimum score that the maximizing player is assured of
- β: the maximum score that the minimizing player is assured of
- Whenever $\beta < \alpha$, the maximizing player need not consider further descendants of this node, as they will never be reached in the actual play.



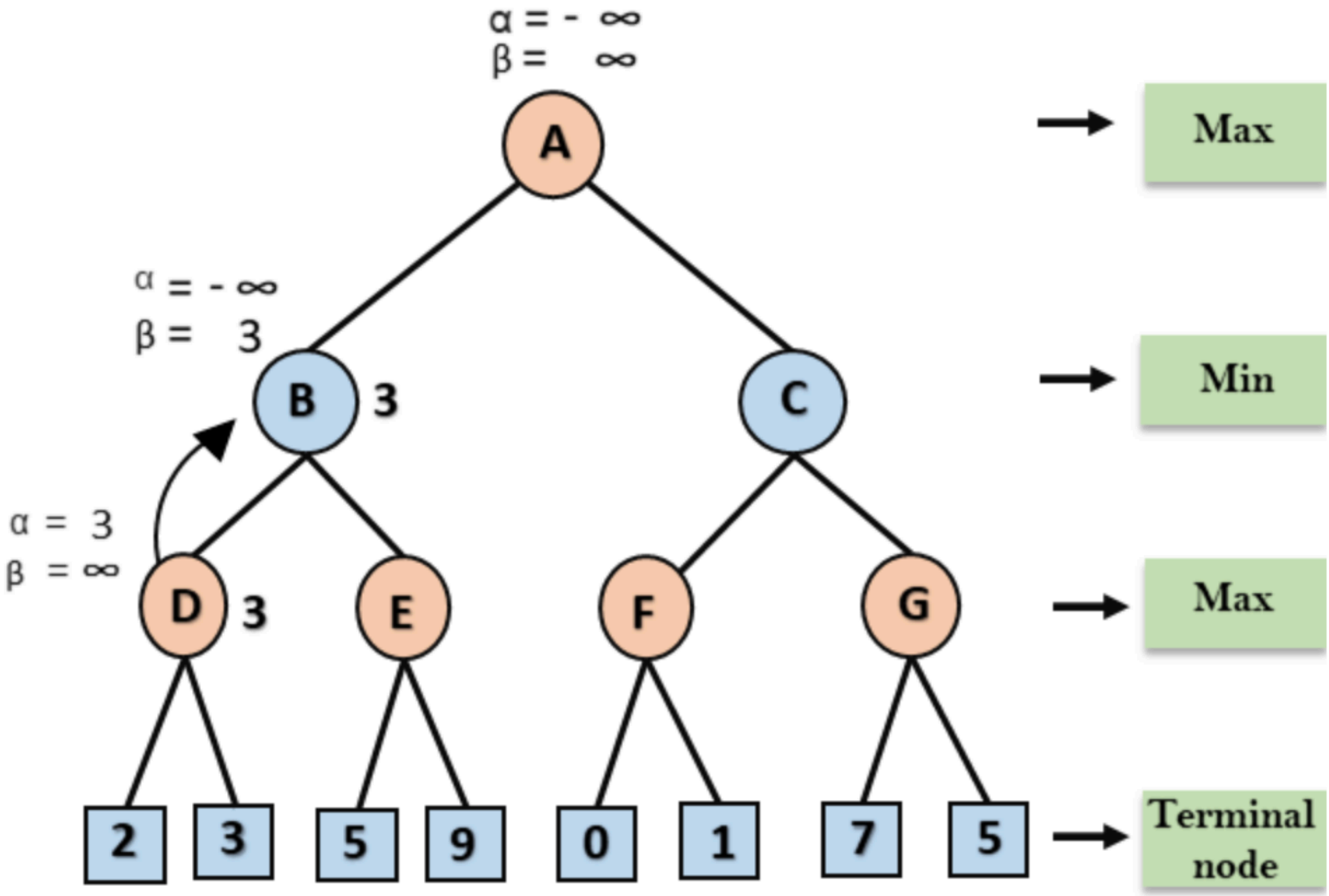
The $\alpha - \beta$ algorithm

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

---

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), 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 TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

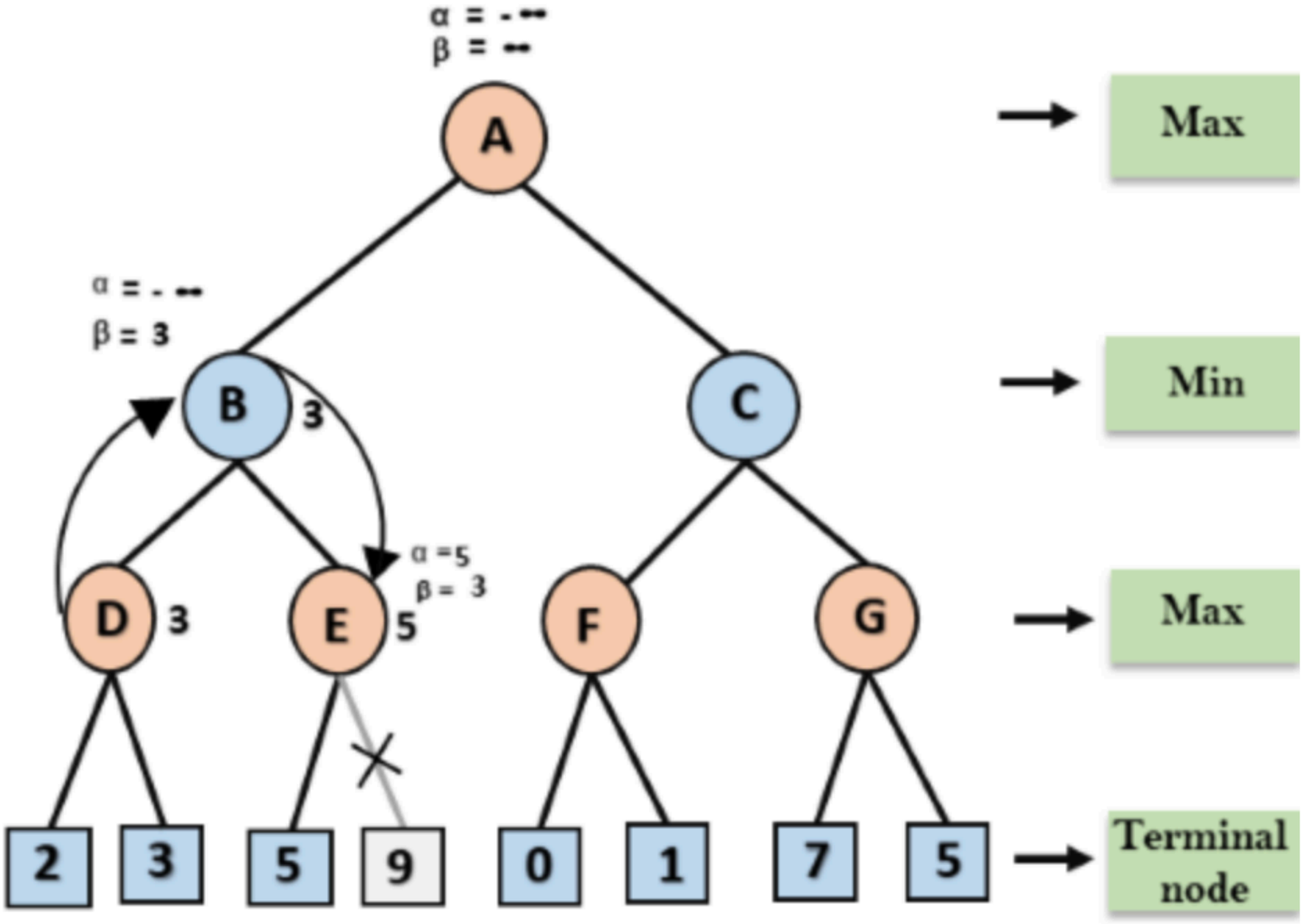
Working of $\alpha - \beta$ pruning



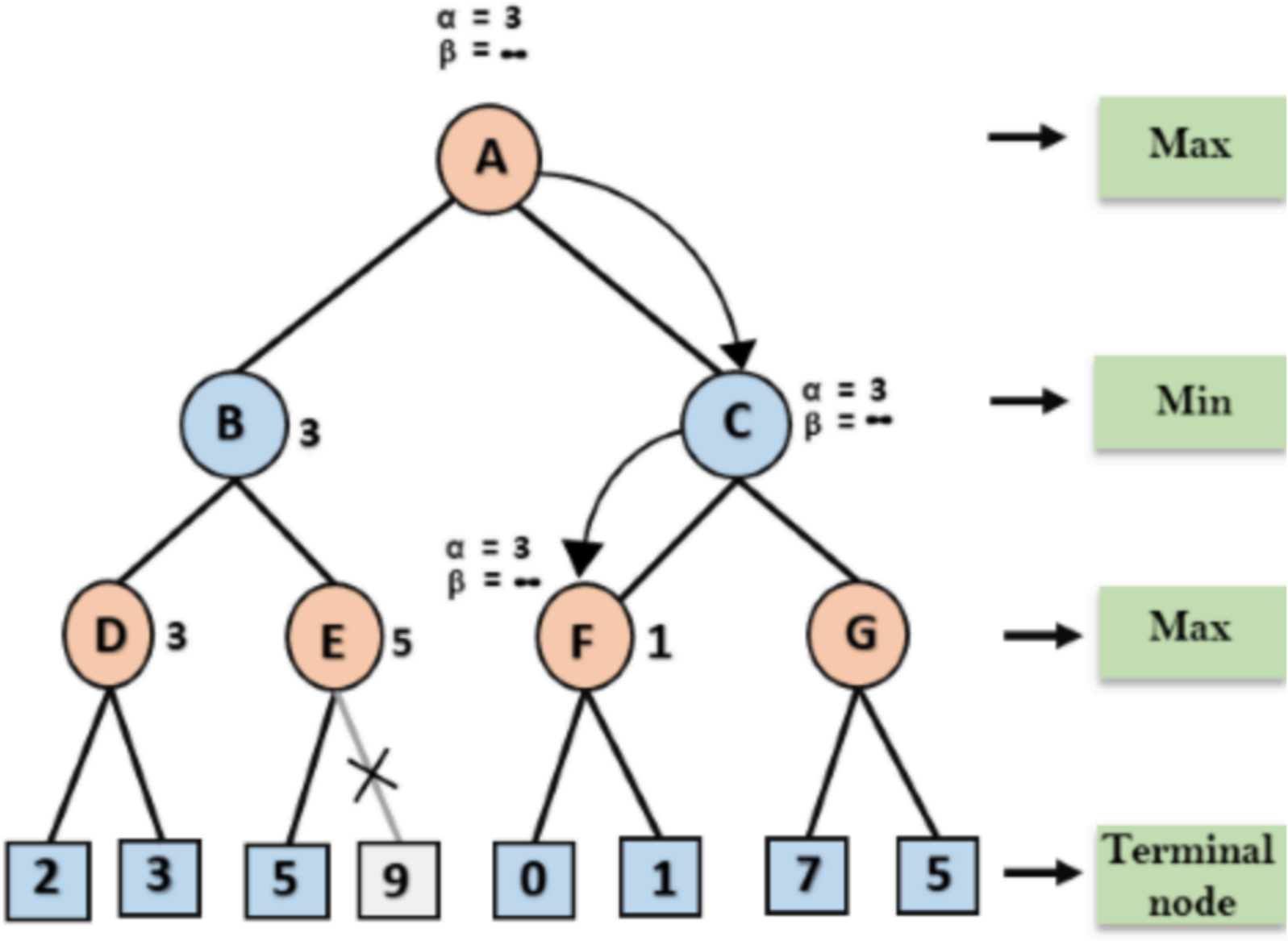
Working of $\alpha - \beta$ pruning



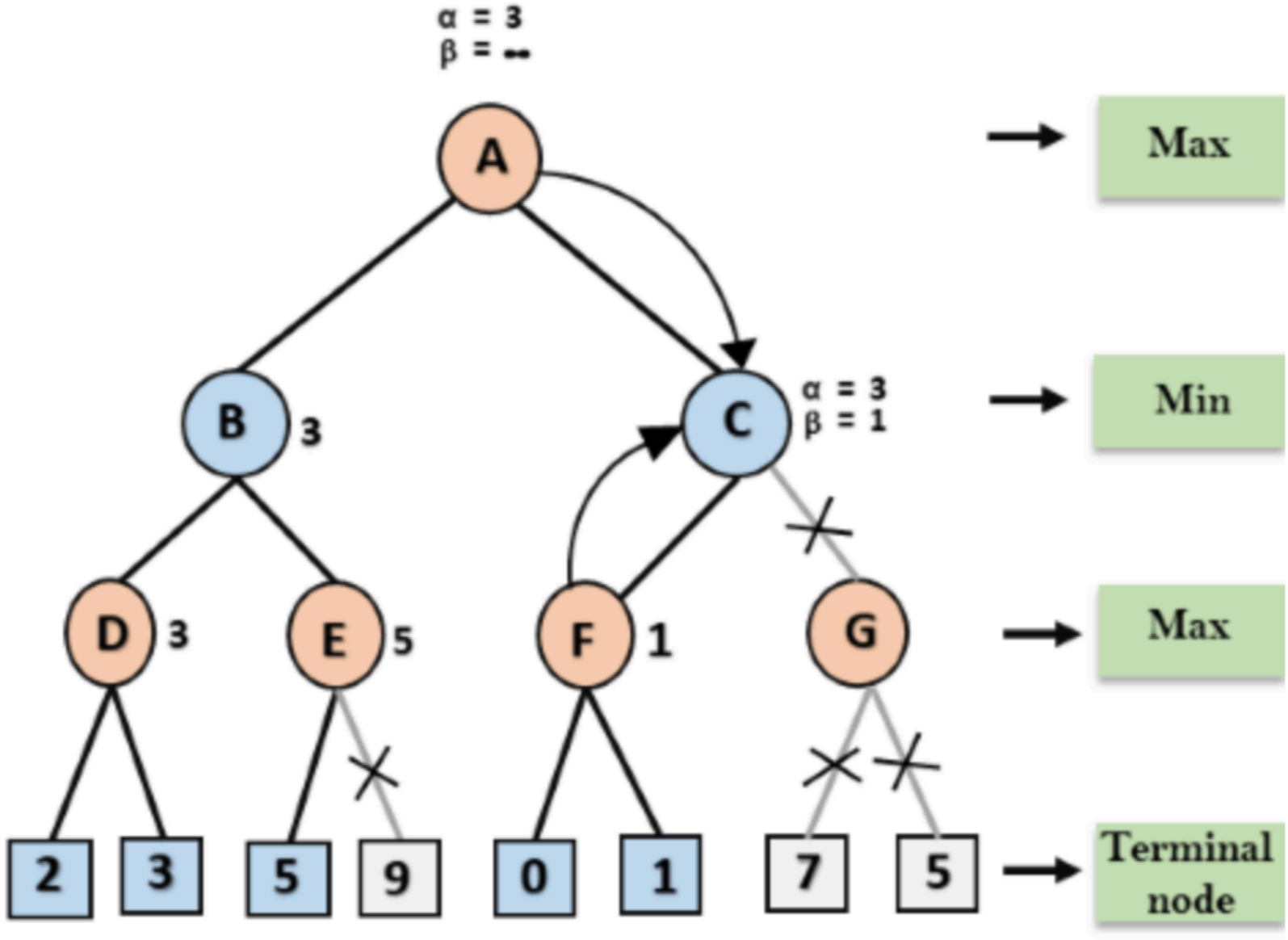
Working of $\alpha - \beta$ pruning



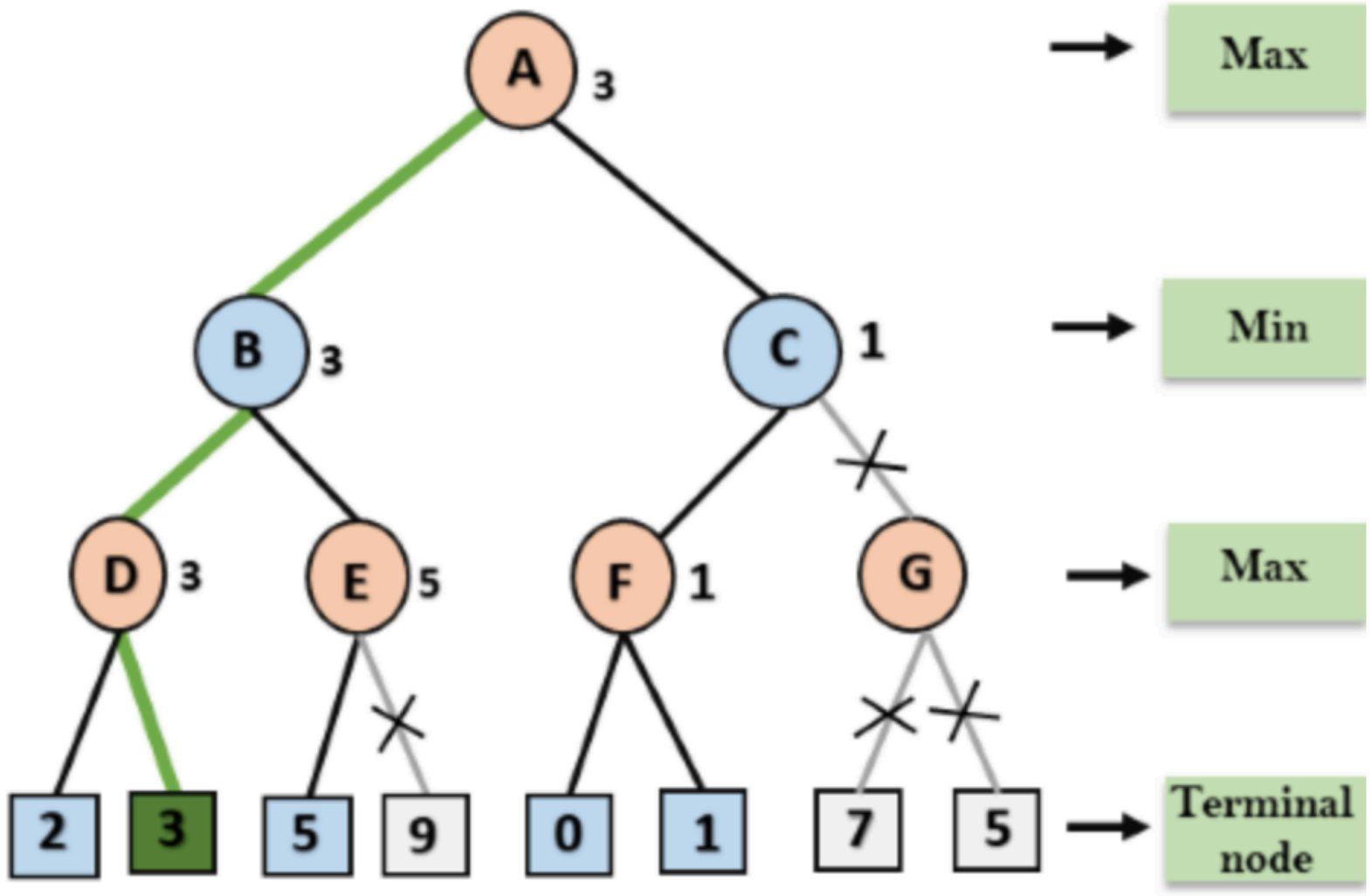
Working of $\alpha - \beta$ pruning



Working of $\alpha - \beta$ pruning



Working of $\alpha - \beta$ pruning



Effectiveness of $\alpha - \beta$ Search

- Effectiveness depends on the order in which successors are examined; more effective if best successors are examined first
- Worst Case:
  - ordered so that no pruning takes place
  - no improvement over exhaustive search
- Best Case:
  - each player's best move is evaluated first
- In practice, performance is closer to best, rather than worst case

Properties of $\alpha - \beta$

Pruning does not affect final result

Good move ordering improves effectiveness of pruning

With "perfect ordering," time complexity = $O(b^{m/2})$
$\Rightarrow$doubles solvable depth

A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

Unfortunately, 35^{50} is still impossible!

Table of Contents

Resource limits

Standard approach: Depth-limited search

- Use CUTOFF-TEST (截断测试) instead of TERMINAL-TEST (终止测试)

  e.g., depth limit (perhaps add quiescence search 静态搜索)

- Use EVAL instead of UTILITY

  用可以估计棋局效用值的启发式评价函数EVAL取代效用函数

  i.e., evaluation function that estimates desirability of position

Suppose we have 100 seconds, explore 10^4 nodes/second

$\Rightarrow 10^6$ nodes per move $\approx 35^{8/2}$

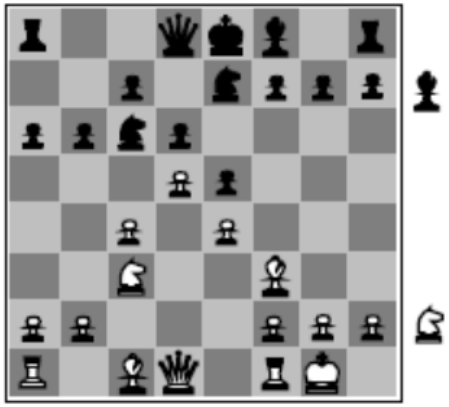
$\Rightarrow \alpha - \beta$ reaches depth 8 $\Rightarrow$ pretty good chess program

4-ply lookahead is a hopeless chess player!

- 4-ply $\approx$ human novice
- 8-ply $\approx$ typical PC, human master
- 12-ply $\approx$ Deep Blue, Kasparov

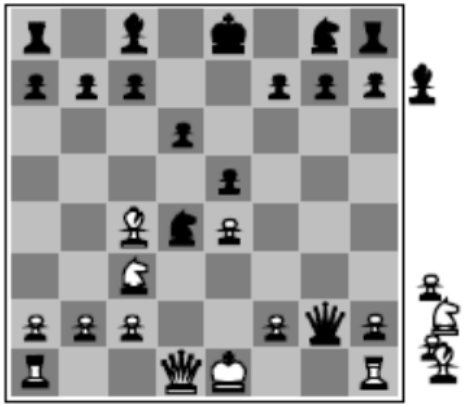
Evaluation functions

- Function which scores non-terminals



Black to move

White slightly better



White to move

Black winning

- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features（特征） :

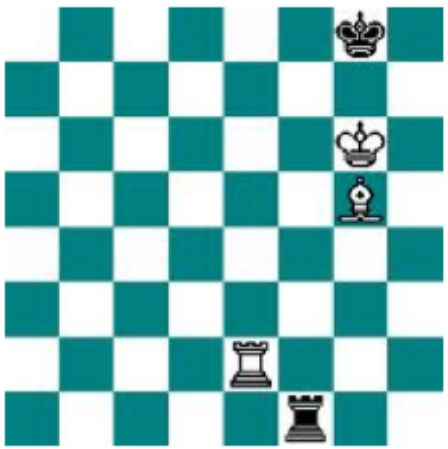
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

e.g., for chess, $w_1 = 9$ with

$f_1(s)$ = (number of white queens) - (number of black queens), etc.

Binary-Linear Value Function

- Binary feature vector $\mathbf{x}(\mathbf{s})$: e.g. one feature per piece
- Weight vector $\mathbf{w}$: e.g. value of each piece
- Position is evaluated by summing weights of active features



$$v(s, \mathbf{w}) = \mathbf{x}(s) \cdot \mathbf{w} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \cdot \begin{bmatrix} +5 \\ +3 \\ +1 \\ -5 \\ -3 \\ -1 \\ \vdots \end{bmatrix}$$

$$v(s, \mathbf{w}) = 5 + 3 - 5 = 3$$

More on Evaluation Functions

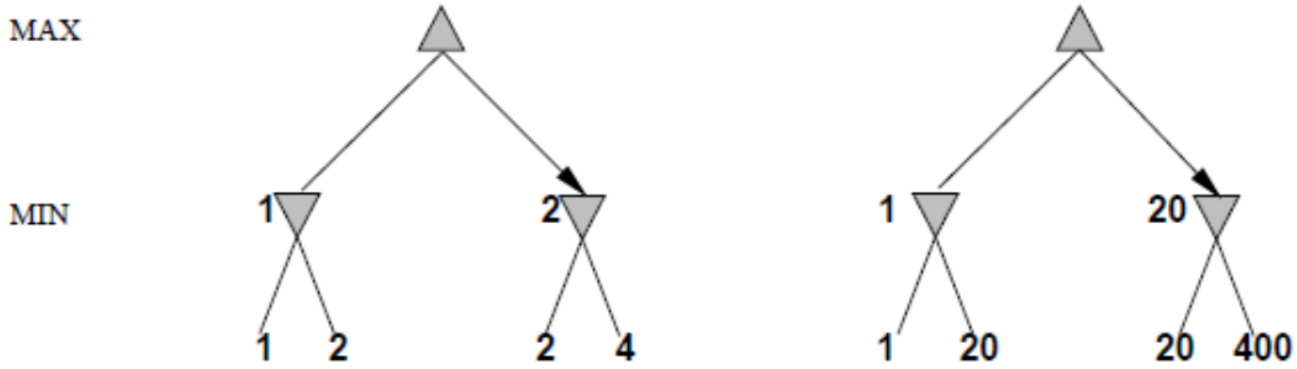
- The board evaluation function estimates how good the current board configuration is

- A linear evaluation function of the features is a weighted sum of f_1, f_2, f_3, ...
  - More important features get more weight

- The quality of play depends directly on the quality of the evaluation function

- To build an evaluation function we have to:
  - construct good features using expert domain knowledge
  - pick or learn good weights

Digression: Exact values don't matter



Behavior is preserved under any monotonic（单调的） transformation of EVAL

Only the order matters:

payoff（结果） in deterministic games acts as an ordinal utility（序数效用） function

Dealing with Limited Time

- In real games, there is usually a time limit T on making a move

- How do we take this into account?

  - cannot stop alpha-beta midway and expect to use results with any confidence

  - so, we could set a conservative depth-limit that guarantees we will find a move in time $< T$

  - but then, the search may finish early and the opportunity is wasted to do more search

Dealing with Limited Time

- In practice, iterative deepening search (IDS) is used

  - run $\alpha - \beta$ search with an increasing depth limit

  -when the clock runs out, use the solution found for the last completed $\alpha - \beta$ search
  (i.e., the deepest search that was completed)

Deterministic games in practice

Chess (国际象棋)：Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

- 计算机能够预见它的决策中的长期棋局序列。机器拒绝走一步有显著短期优势的棋—显示了非常类似于人类的对危险的感觉。

—Kasparov

-Kasparov lost the match 2 wins to 3 wins and 1 tie search

-Deep Blue played by "brute force" (i.e., raw power from computer speed and memory); it used relatively little that is similar to human intuition and cleverness

-Used minimax, $\alpha - \beta$, sophisticated heuristics

Deterministic games in practice

Checkers（西洋跳棋）: Chinook, the World Man-Machine Checkers Champion

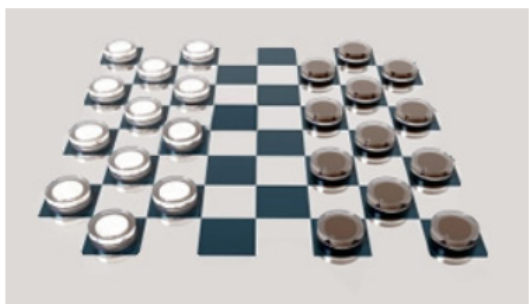
- Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994.

- In 2007, checkers was solved: perfect play leads to a draw

Chinook cannot ever lose

使用了一个提前计算好的存有443,748,401,247个不多于8个棋子的棋局数据库，使它的残局(endgame)走棋没有缺陷

50 machines working in parallel on the problem



Deterministic games in practice

Othello（奥赛罗）: human champions refuse to compete against computers, who are too good.

Go（围棋）: human champions refuse to compete against computers, who are too bad. In go, $b > 300$（棋盘为 19x19）, so most programs use pattern knowledge bases to suggest plausible moves.

A new benchmark for Artificial Intelligence (人工智能新的试金石)

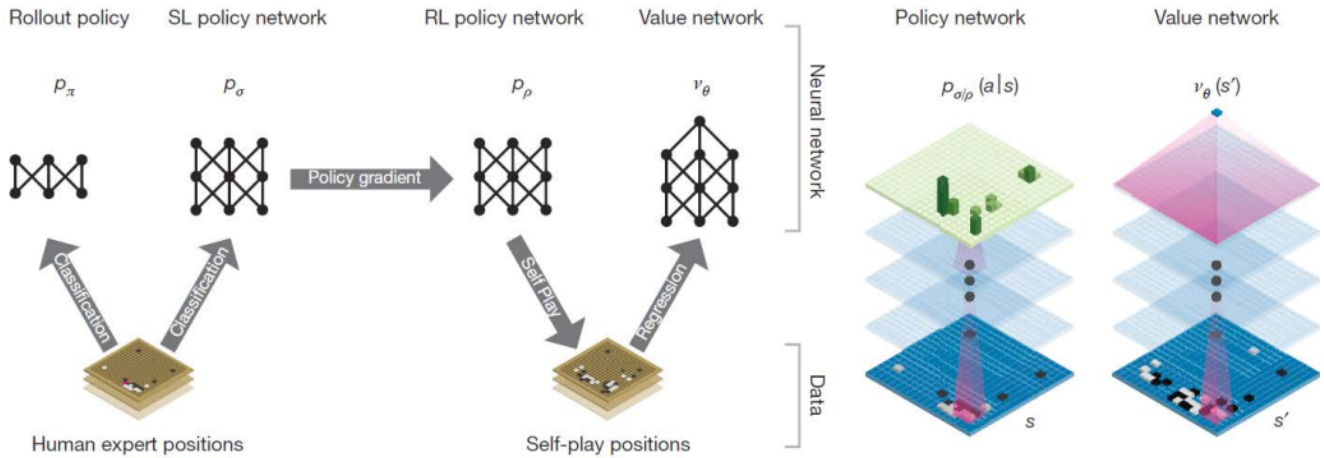
AlphaGo: First to beat human pro in 19x19 Go

- Google DeepMind computer go player
  - deep neural networks:
    - value networks: to evaluate board positions
    - policy networks: to select moves
  - trained by
    - supervised learning
    - reinforcement learning by self-play
  - search algorithm
    - Monte-Carlo simulation + value/policy networks

AlphaGo: Background

- reduction of search space:
  - reduced depth
    - position evaluation
  - reduced branching
    - move sampling based on policy
    - policy = probability distribution $p(a|s)$

Deep Neural Networks in AlphaGo



AlphaGo uses two types of neural networks:

- policy network: what is the next move?
  - learned from human expert moves
- value network: what is the value of a state?
  - learned from self-play using a policy network

SL = supervised learning, RL = reinforcement learning

Deep Neural Networks in AlphaGo

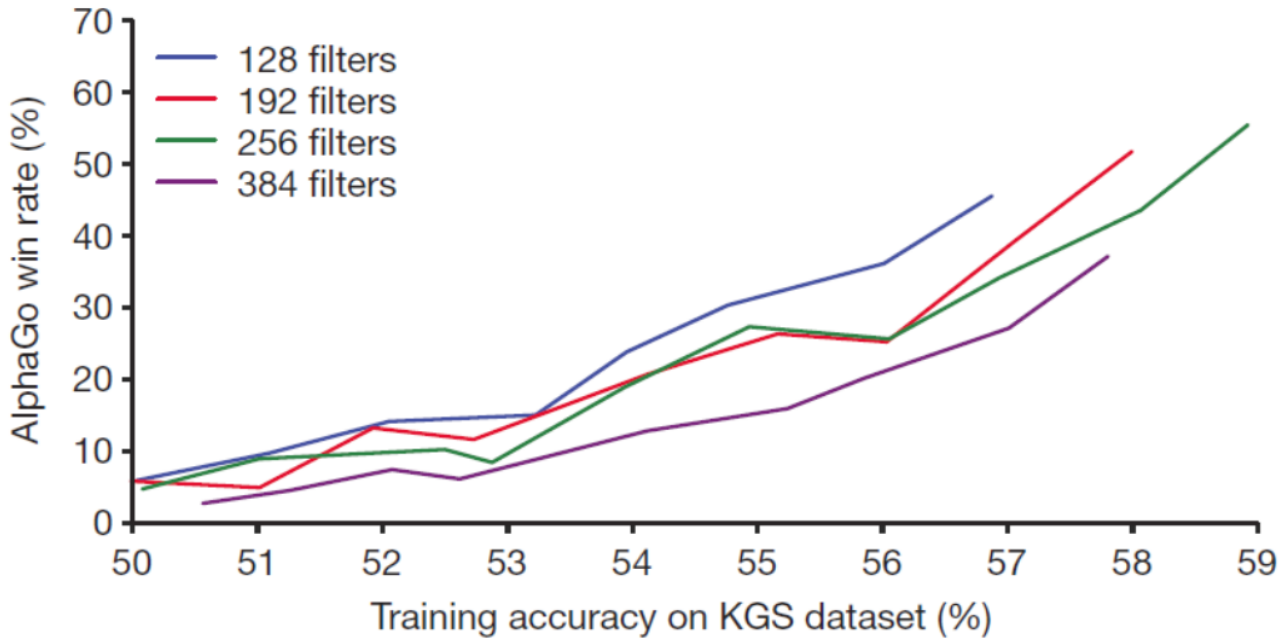
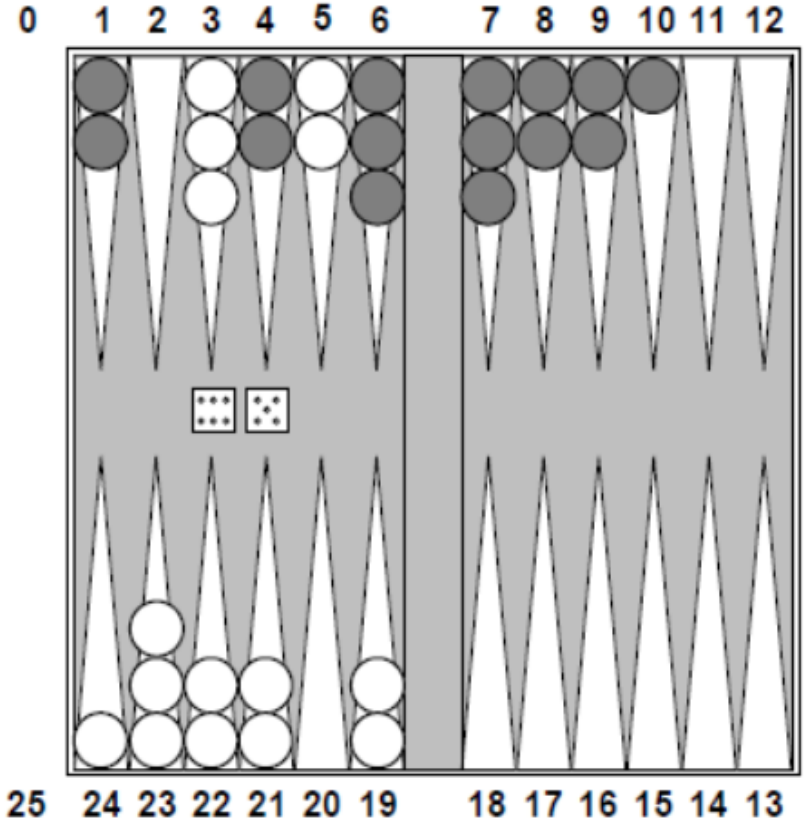


Table of Contents

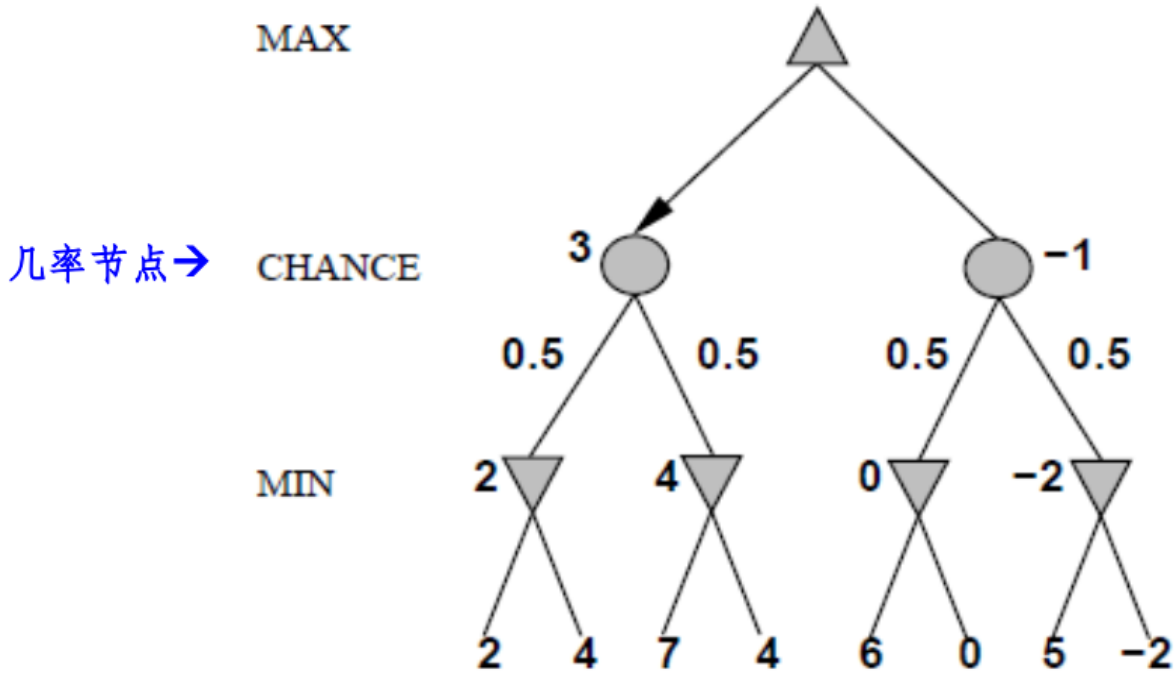
Nondeterministic games:backgammon(西洋双陆棋)



Nondeterministic games in general

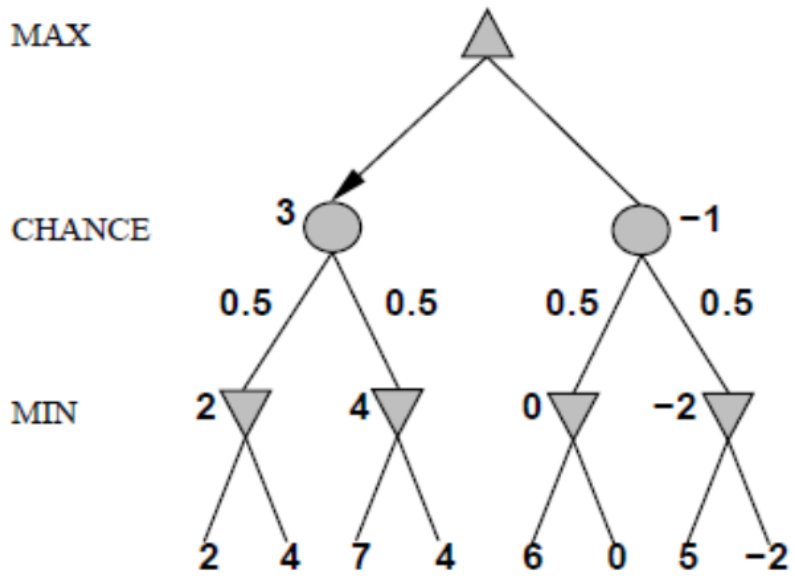
In nondeterministic games, chance introduced by dice, card-shuffling

Simplified example with coin-flipping:



Nondeterministic games in general

- Weight score by the *probability* that move occurs
- Use expected value for move: instead of using max or min, compute the average, weighted by the probabilities of each child
- Choose move with *highest expected value*



Stochastic Game

- Multiple-state / Multiple-agent environment. Like an extension of MDPs and Normal-Form Games.
- Markovian but not from each player's point of view.
- A stochastic game is a tuple $(n, S, A_{1,\ldots,n}, T, R_{1,\ldots,n})$ where:
  - n represents the number of agents
  - S the state set
  - A_i the action set of agent i and $A = A_1 \times \cdots \times A_n$ the joint action set
  - $T : S \times A \times S \to [0, 1]$ is a transition function which depends on the actions of all players
  - $R : S \times A \times S \to \mathbb{R}$ is a reward function representing the expected value of the next reward, which also depends on the actions of all players.
- Each agent i selects policy $\pi_i : S \to PD(A_i)$ (probability $\pi_i(a_i \mid s)$)
- Joint policy $\pi = \langle \pi_i, \pi_{-i} \rangle$

Optimality Concepts in Stochastic Games

Optimality Concepts in Stochastic Games:

- The discounted reward over time is usually considered, as in MDPs:

$$V_i^{\pi}(s) = E\left[\sum_{k=0}^{\infty} \gamma^k r^j_{t+k+1} \mid s_t = s, \pi\right]$$
$$= \sum_a \pi(s,a) \sum_{s'} T(s,a,s') \left(R_i(s,a,s') + \gamma V_i^{\pi}(s')\right)$$
$$Q_i^{\pi}(s,a) = \sum_{s'} T(s,a,s') \left(R_i(s,a,s') + \gamma V_i^{\pi}(s')\right)$$

- Best-response function: defined for policies with the state values as reference.

$$\pi_i^* \in BR_i(\pi_{-i}) \quad \text{iff}$$
$$\forall \pi_i \in S \times PD(A_i), \forall s \in S \quad V_i^{\langle \pi_i^*, \pi_{-i} \rangle}(s) \geq V_i^{\langle \pi_i, \pi_{-i} \rangle}(s)$$

- Nash equilibria: All players are using best-response policy.

$$\forall i = 1 \ldots n \quad \pi_i \in BR_i(\pi_{-i})$$

Maximum Expected Utility

- Why should we average utilities? Why not minimax?
- Principle of maximum expected utility: an agent should chose the action which maximizes its expected utility, given its knowledge
- General principle for decision making
- Often taken as the definition of rationality
- We'll see this idea over and over in this course!

Algorithm for nondeterministic games

EXPECTIMINIMAX gives perfect play

Just like MINIMAX, except we must also handle chance nodes:

...

if *state* is a Max node then

    return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

if *state* is a Min node then

    return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

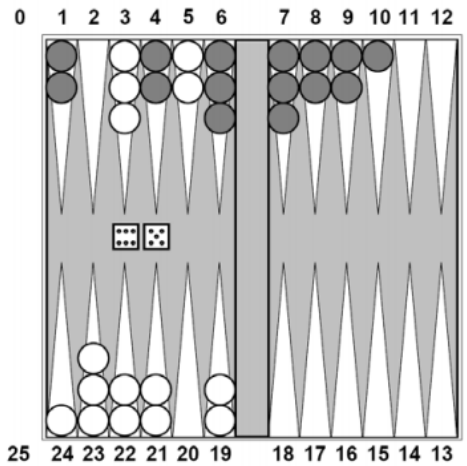
if *state* is a chance node then

    return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

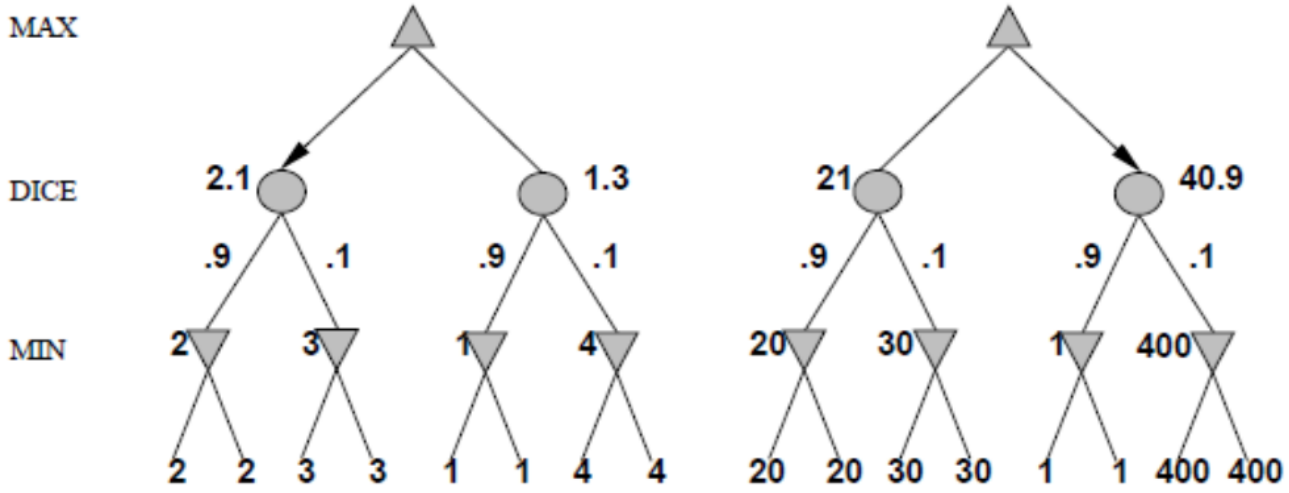
...

Stochastic Two-Player

- Dice rolls increase b: 21 possible rolls with 2 dice
  - Backgammon $\approx$ 20 legal moves
  - Depth 4 $=$ 20 x $(21 \text{ x } 20)^3$ $1.2 \text{ x } 10^9$
- As depth increases, probability of reaching a given node shrinks
  - So value of lookahead is diminished
  - So limiting depth is less damaging
  - But pruning is less possible...
- TDGammon uses depth-2 search + very good eval function + reinforcement learning: world-champion level play



Digression: Exact values DO matter



Behaviour is preserved only by positive linear transformation of EVAL

Hence EVAL should be proportional to the expected payoff

评价函数应该是棋局的期望效用值的正线性变换

Table of Contents

Games of imperfect information

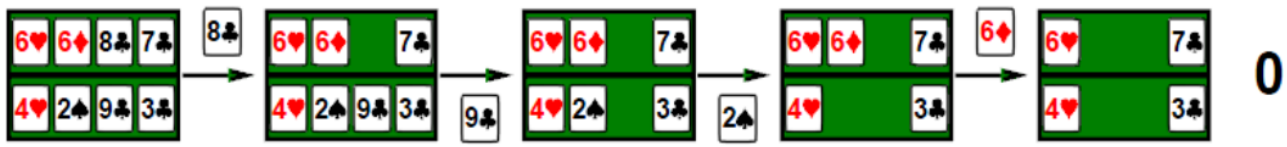
E.g., card games, where opponent's initial cards are unknown
Typically we can calculate a probability for each possible deal
Seems just like having one big dice roll at the beginning of the game

Idea: compute the minimax value of each action in each deal, then choose the action with highest expected value over all deals

在评价一个有未知牌的给定行动过程时，首先计算出每副可能牌的出牌行动的极小极大值，然后再用每副牌的概率来计算得到对所有发牌情况的期望值。

Example

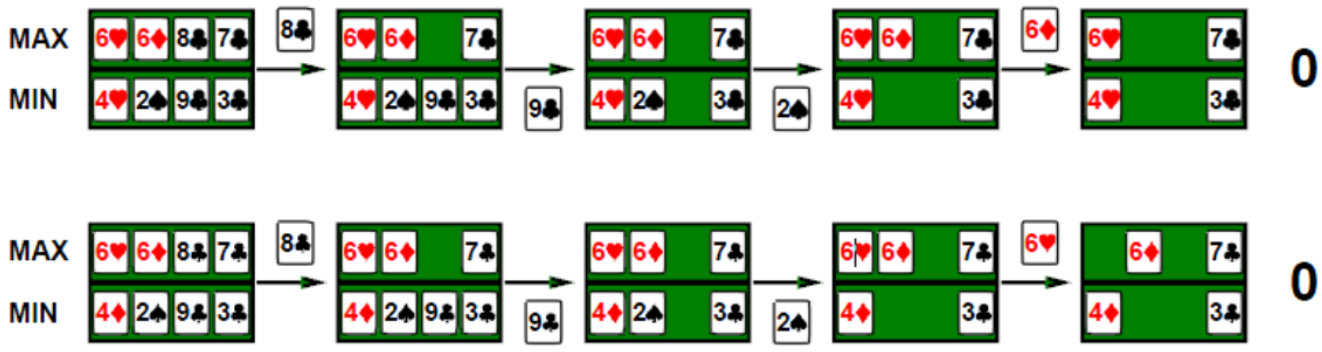
Four-card bridge/whist/hearts hand, MAX to play first



一开始换牌，知道对手其他 3 张牌，除了第一张牌。
简单规则：有同花色，一定要出同花色；比大小；赢一轮得一分，算总数
若第一张牌为红桃 4

Example

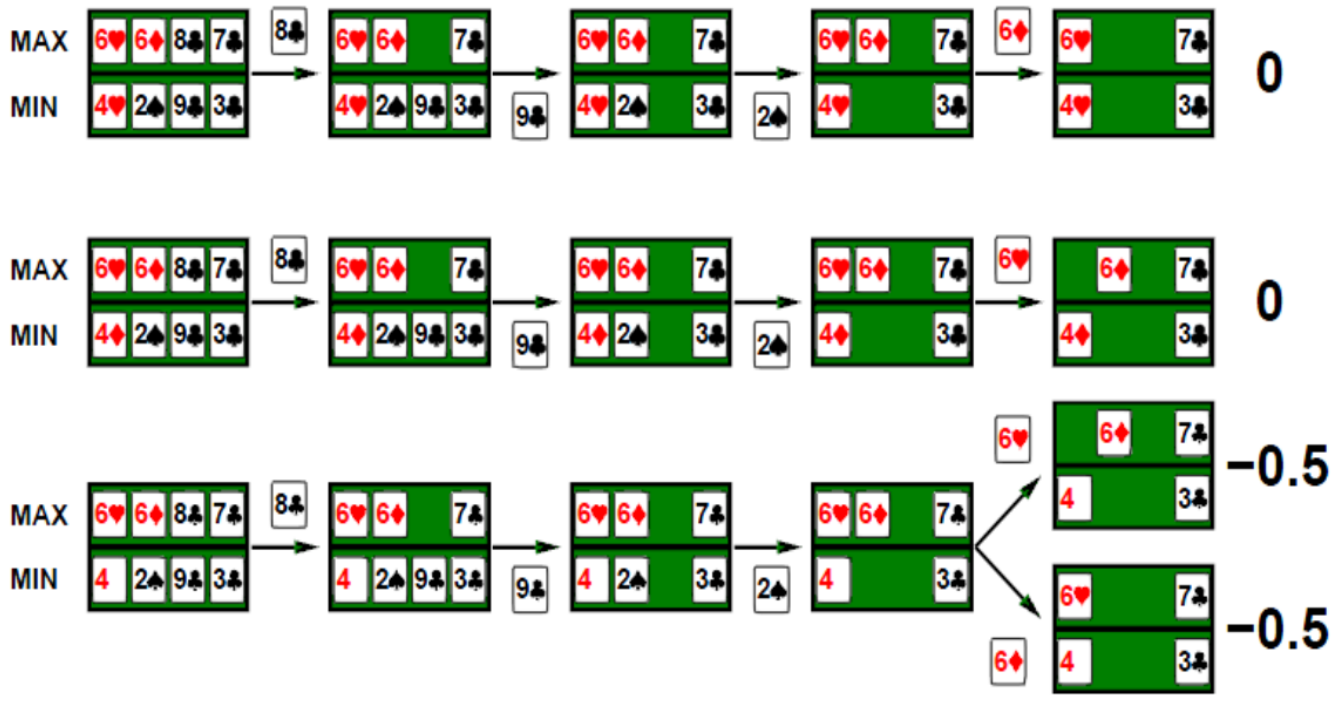
Four-card bridge/whist/hearts hand, MAX to play first



若第一张牌为方块 4

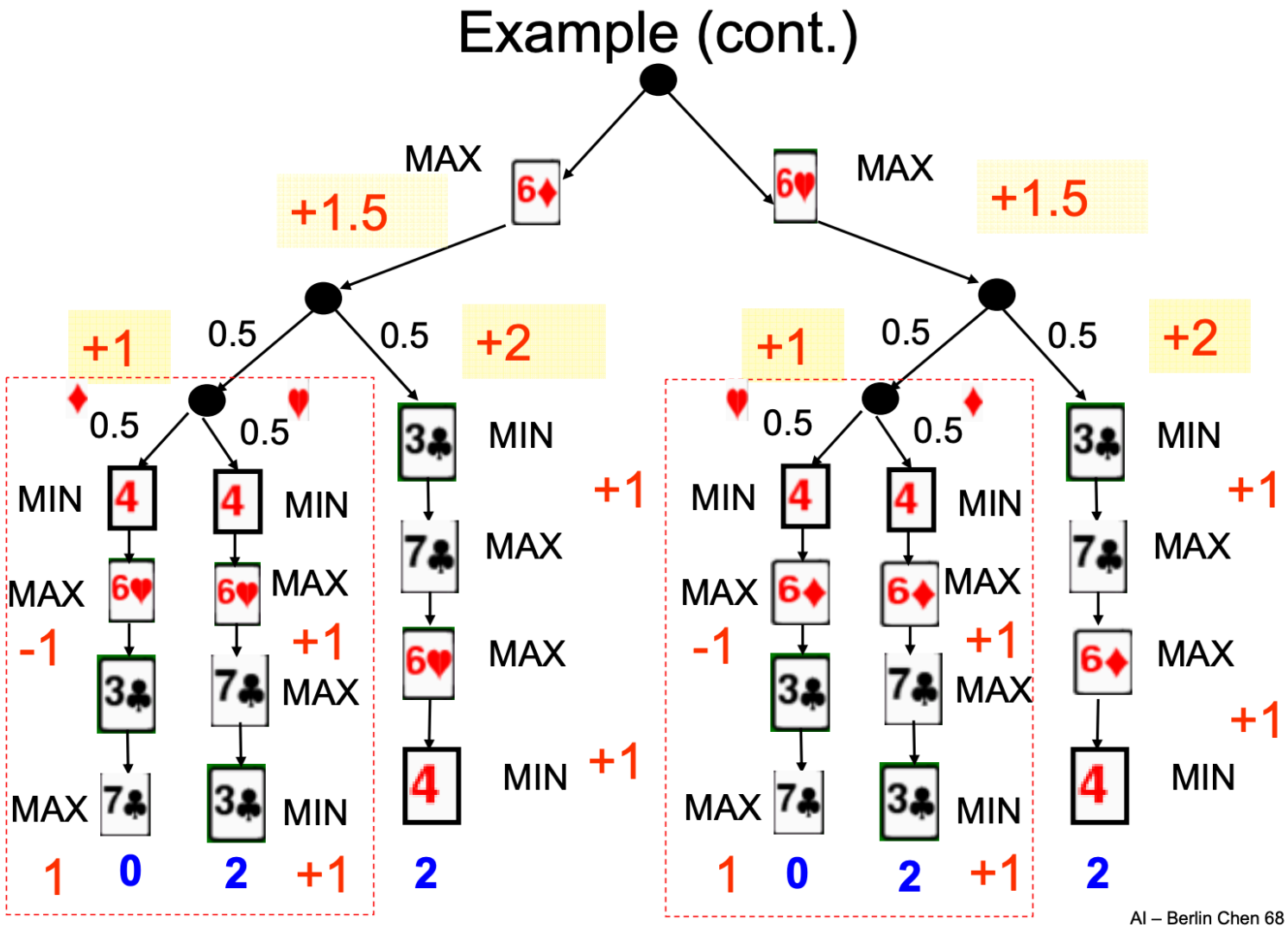
Example

Four-card bridge/whist/hearts hand, MAX to play first



若第一张牌不知花色

Example


Example (cont.)
MAX
+1.5
MAX
+1.5
+1
0.5
0.5
+2
+1
0.5
0.5
+2
MIN
MIN
MAX
MAX
MAX
MAX
MIN
MIN
+1
+1
+1
-1
+1
-1
+1
MAX
MAX
MIN
MIN
1
0
2
+1
2
1
0
2
+1
2

Proper analysis

* Intuition that the value of an action is the average of its values in all actual states is WRONG

With partial observability, value of an action depends on the information state or belief state（信度状态） the agent is in

Can generate and search a tree of information states

Leads to rational behaviors such as

- Acting to obtain information
- Signaling to one's partner
- Acting randomly to minimize information disclosure

Computers Playing Texas Holder

A MYSTERY AI JUST CRUSHED THE BEST HUMAN PLAYERS AT POKER

Professional poker player Jason Les plays against "Libratus," at Rivers Casino in Pittsburgh, on January 11, 2017. ANDREW RUSH/PITTSBURGH POST-GAZETTE/AP

According to the human players that lost out to the machine, Libratus is aptly named. It does a little bit of everything well:

- knowing when to bluff
- and when to bet low with very good cards,
- as well as when to change its bets just to thrown off the competition.

Summary

- Games are fun to work on!
  -perfection is unattainable must approximate
  -Games are to AI as grand prix racing is to automobile design
- Game playing is best modeled as a search problem
  -Search trees for games represent alternate computer/opponent moves
- Evaluation functions estimate the quality of a given board configuration for each player
- Minimax is an algorithm that chooses "optimal" moves by assuming that the opponent always chooses their best move
- Alpha-beta is an algorithm that can avoid large parts of the search tree, thus enabling the search to go deeper —消除无关的子树以提高效率

Summary of Search

- Uninformed search strategies
  Breadth-first search (BFS), Uniform cost search, Depth-first search (DFS),Depth-limited search, Iterative deepening search

- Informed search strategies
  -Best-first search: greedy, A*
  -Local search: hill climbing, simulated annealing etc.

- Constraint satisfaction problems
  -Backtracking = depth-first search with one variable assigned per node
  -Enhanced with: Variable ordering and value selection heuristics, forward checking, constraint propagation

作业

- 第三版: 5.9，5.8，5.13