# MIN-MAX algorithm — history, variants, refinements

Przemysław Klęsk
*pklesk@wi.zut.edu.pl*

Department of Methods of Artificial Intelligence and Applied Mathematics

# Table of contents

# Table of contents

# Game theory

## Game theory

A branch of mathematics dealing with situations of conflict (strategic situations), where a result of a participant depends on choices made by himself and others. Sometimes, also called the *theory of rational behaviours*. Apart from computer science, applied in the fields of sociology, economics, military (historically earlier).

Important historical works:

- Émil Borel, *Applications for random games* (fr. *Applications aux Jeux de Hasard*), 1938.
- John von Neuman and Oskar Morgenstern, *Theory of Games and Economic Behavior*, 1944.

# Notions

## Game

A situation of conflict, where:

- at least two players can be indicated,
- every player has a certain number of possible *strategies* to choose from (a strategy precisely defines the way the game shall be played by the player),
- result of the game is a direct consequence of the combination of strategies chosen by players.

## Strategy

**Complete** set of decisions (about choices or moves) that a player has to make for all possible states the game can reach.

It is often impossible to write down (memorize) a strategy because of its size (for typical games).

# Notions

## Finite game

A game for which it is guaranteed that the game shall finish.

## Zero-sum game

A game in which *payoffs* for all players (determined by result of game) sum up to zero.

For chess the convention is: 0 (loss), 1 (win), $\frac{1}{2}$ (draw); zero sum can be obtained by a linear transformation: $2x - 1$).

# Minimax Theorem

## Minimax Theorem (von Neuman, 1928)

For every finite two-person zero-sum game there exists at least one optimal *mixed strategy*. Therefore, there exists a game value $v$, such that by applying the optimal strategy the first player guarantees for himself a payoff not worse than $v$, while the second player guarantees for himself a payoff not worse than $-v$.

For zero-sum games the minimax solution is identical with *Nash equilibrium* (a broader notion).

# Example for theorem

| | $B$ chooses $b_1$ | $B$ chooses $b_2$ | $B$ chooses $b_3$ |
|---|---|---|---|
| $A$ chooses $a_1$ | 3 | −2 | 3 |
| $A$ chooses $a_2$ | −1 | 0 | 4 |
| $A$ chooses $a_3$ | −4 | −3 | 1 |

- Matrix of payoffs: $W = \{w_{ij}\}$ for zero-sum game, where players $A$ and $B$ make **simultanous moves**.
- What are minimax choices for $A$ and $B$?
- Is it a stable solution?
- Do there exist dominated choices?

# Example for theorem

| | $B$ chooses $b_1$ | $B$ chooses $b_2$ | $B$ chooses $b_3$ |
|---|---|---|---|
| $A$ chooses $a_1$ | 3 | $-2$ | 3 |
| $A$ chooses $a_2$ | $-1$ | 0 | 4 |
| $A$ chooses $a_3$ | $-4$ | $-3$ | 1 |

- The minimax choice for $A$ is $a_2$, because the worst possible result for $A$ is then $-1$:

$$\max_i \min_j w_{ij} = -1. \tag{1}$$

- The minimax choice for $B$ is $b_2$, because the worst possible result for $B$ is then 0:

$$\min_j \max_i w_{ij} = 0. \tag{2}$$

- Solution $(a_2, b_2)$ is **not stable**, because if $B$ believes that $A$ chooses $a_2$ then $B$ shall choose $b_1$ in order to obtain the payoff $-1$; then, if $A$ believes that $B$ chooses $b_1$ then $A$ shall choose $a_1$ to obtain the payaoff 3, etc.
- **Dominated choices**: $a_3$ and $b_3$ — regardless of opponent's choice, the other choices are better (more precisely: not worse). Hence, the matrix of payoffs can be reduced by deleting third row and third column.

# Example for theorem

| | $B$ chooses $b_1$ | $B$ chooses $b_2$ |
|---|---|---|
| $A$ chooses $a_1$ | 3 | $-2$ |
| $A$ chooses $a_2$ | $-1$ | 0 |

$$\begin{pmatrix} p & 1-p \end{pmatrix} \cdot W \cdot \begin{pmatrix} q \\ 1-q \end{pmatrix} = -q - 2p + 6pq. \tag{3}$$

- **Mixed strategy** is a combination of choices — pure strategies — with certain probabilities (frequencies).
- The presence of $pq$ term indicates the existence of **saddle point**.
- By demanding $\frac{\partial \cdot}{\partial p} = 0$ and $\frac{\partial \cdot}{\partial q} = 0$, one obtains the solution:

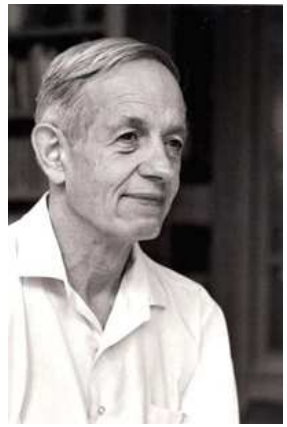
$$p = \frac{1}{6}, \quad q = \frac{1}{3}. \tag{4}$$

- **Game value**: $v = -\frac{1}{3}$.
- Formally, when $P$ and $Q$ represent mixed strategies (as vectors of probabilities), then:

$$\max_P \min_Q P^T \cdot W \cdot Q = \min_Q \max_P P^T \cdot W \cdot Q = v. \tag{5}$$

- If there exist more than one optimal mixed strategy then infinitely many of them exist.

# Nash Equilibrium (NEQ)



John Nash, born 1928, Nobel prize winner in 1994 (in economics).

## Informally

In a multi-player game, we say that a certain set of strategies from particular players constitutes the *Nash equilibrium*, if and only if each of those strategies is the best response for all remaining ones and none of the players can gain by changing its own strategy with other strategies kept fixed.

# Nash Equilibrium (NEQ)

## Formally

- In a game with $n$ players, let $S_i$ denote the set of possible strategies for $i$-th player.
- Let $S$ denote the **space of all strategies** as the cartesian product of sets of strategies from particular players:

$$S = S_1 \times S_2 \times \cdots \times S_n$$

- For any set of strategies $(s_1, \ldots, s_n)$ from particular players, let $W_i(s_1, \ldots, s_n)$ determine the **payoff** for the $i$-th player. Therefore, $W_i$ is a function:

$$W_i \colon S \to \mathbb{R}.$$

  Let $W$ denote a vector function: $W = (W_1, \ldots, W_n)$.
- A **game** can be understood as a pair $(S, W)$.
- We say that a set of strategies $(s_1^*, \ldots, s_n^*)$ constitutes the **Nash equilibrium** if and only if:

$$\forall i \quad \forall s_i \in S_i : \qquad W_i(s_1^*, \ldots, s_i^*, \ldots, s_n^*) \geqslant W_i(s_1^*, \ldots, s_i, \ldots, s_n^*). \tag{6}$$

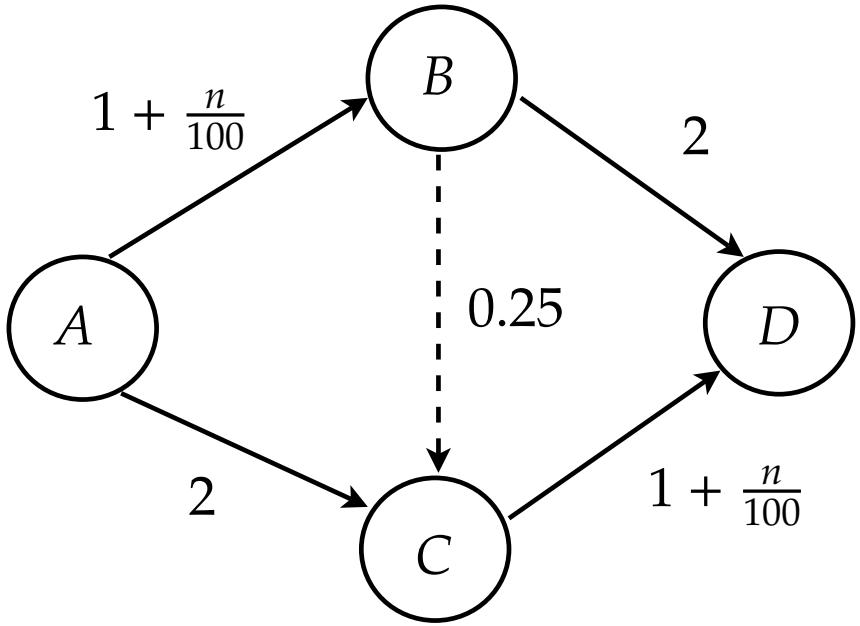
# Nash Equilibrium (NEQ)

- Another way to define NEQ is that $s_i^*$ can be viewed as the solution of:

$$\max_{s_i \in S_i} W_i(s_1^*, \ldots, s_i, \ldots, s_n^*) \tag{7}$$

  for all $i$.
- The idea of NEQ can be applied to analyze or predict what happens when several players (parties, institutions) must make decisions simultaneously, and when the outcome depends on all those decisions. The outcome cannot be predicted when analyzing the decisions seperately (in isolation).
- NEQ does not have to indicate the best result for the group (the best sum of results) and may seem irrational for an outside observer (e.g. *prisonner dilemma*, *Braess paradox*).
- In many cases, players could improve their group result if they agree strategies different from NEQ (e.g. business cartels instead of free-market competition).

# Braess Paradox




## Problem

Assuming selfishness and rationality of drivers, find the expected traffic flow (the NEQ) for 100 drivers travelling from $A$ to $D$ in two cases: (1) when edge $BC$ does not exist, (2) when edge $BC$ exists.
Treat the problem as a game where every player (driver) has 2 or 3 strategies, respectively: $ABD$, $ACD$ and possibly $ABCD$. The payoff is the time of travel for road selected. For $AB$ and $BD$ edges, the $n$ parameter denotes the number of players who selected the edge as a road fragment.

# Braess paradox — solution

$p, q, r$ — number of drivers choosing strategies, respectively: $ABD$, $ACD$, $ABCD$.

**Case 1**

$$\begin{cases} 1 + \frac{p}{100} + 2 & = 1 + \frac{q}{100} + 2; \\ p + q & = 100. \end{cases} \tag{8}$$

Solution: $p = q = 50$, road cost (game value) $v = 3.5$.

# Braess paradox — solution

**Case 2**

$$\begin{cases} 1 + \frac{p+r}{100} + 2 & = 1 + \frac{q+r}{100} + 2 = 1 + \frac{p+r}{100} + 0.25 + 1 + \frac{q+r}{100}; \\ p + q + r & = 100. \end{cases} \tag{9}$$

Solution: $p = q = 25$, $r = 50$, road cost (game value) $v = 3.75$.

Drivers would travel shorter if they agreed no to use the $BC$ fragment.

# Table of contents

# Game trees

For a certain state $s$ assume there exists $n$ possible choices (moves, manipulations, actions): $a_1, a_2, \ldots, a_n$ causing new states to arise from $s$, respectively: $s_1, s_2, \ldots, s_n$. For each of those states there again exist further possible choices. By coninuing this procedure a *tree structure* arises naturally.

# Search problems — difficulties

- **too large search space** — when generation of complete tree is impossible or too expensive computationally or memory-wise (exponential growth).
- **games of imperfect information** — players do not have complete information about the state of game (e.g. cards held by opponents, opponents' letters in Scrabble, opponents' military units, etc.).
- **random factors** — when random generators are an element of game (e.g. rolling dice, shuffling cards, random events of environment).

# Game complexity measures

1. **State-space complexity** — the number of legal positions in a game that can be reached from the initial position. Often, this number can be upperbounded, when it is easire to make an estimate by taking into account illegal positions.
2. **Game tree size** — the number of different games that can be played; equivalently, the number of leaves in the tree with the root representing the initial position. The same states reached by different paths (different moves order) are calculated multiple times. The number can be upperbounded by taking into account the tree growth with illegal moves included.

# Game complexity measures

3. **Decision complexity** — the number of leaves in the smallest *decision tree* which is capable to determine the game value for the initial position.
4. **Game tree complexity** (ang. *game tree complexity*) — the number of leaves in the smallest *decision tree of full width* capable to determine the gamve value for the initial position. Such a tree takes into account all possibilities for both players. The number corresponds to the required number of operations in a MIN-MAX search.

**Decision tree (from the first player perspective)**

A subtree (of the full game tree) in which all states have labels: *win*, *draw*, *loss*.

- A state becomes labeled as a *win* when *any* of its decendants is labeled a *win*.
- A states becomes labeled as a *loss* when *all* its decendants are labeled a *loss*.
- A state becomes labeled as a *draw* when at least one of its decendants is labeled a *draw* and all remaining ones are labeled a *loss*.

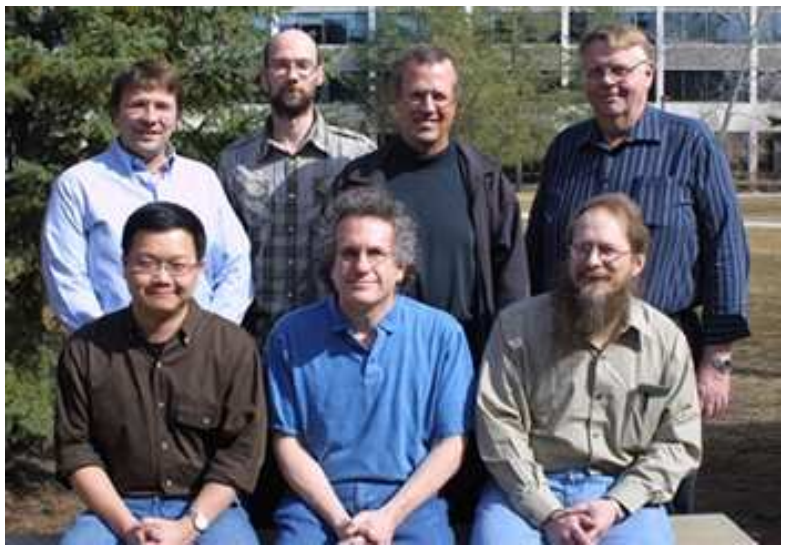
A decision tree takes into account all possibilities for one player and only one possibility (the best response) for the second player (corresponds roughly to the number of operations in a $\alpha$-$\beta$ pruning optimisitc search).

# Complexity estimates for some games

| game | board size (number of cells) | state-space complexity | game tree complexity |
|---|---|---|---|
| *tic-tac-toe* | 9 | $10^3$ | $10^5$ |
| *connect 4* | 42 | $10^{13}$ | $10^{21}$ |
| *English checkers* | 32 | $10^{20}$ | $10^{31}$ |
| *hex* also: *Nash* or *John* | 121 | $10^{56}$ | ? |
| *chess* | 64 | $10^{47}$ | $10^{123}$ |
| *connect 6* | 361 | $10^{172}$ | $10^{140}$ |
| *backgammon* | 28 | $10^{20}$ | $10^{144}$ |
| *Go* | 361 | $10^{171}$ | $10^{360}$ |

# Chinook project (checkers)

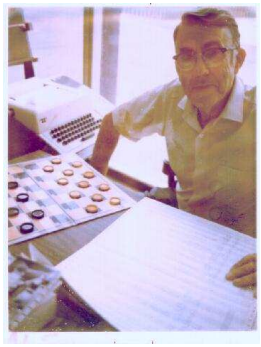
- *English checkers* ($8 \times 8$, kings move in any direction but by 1 cell only).
- Project started in 1989. **Goal: beating human in world championships of checkers.**
- **Authors:** (back, from the left) Yngvi Bjørnsson, Neil Burch, Rob Lake, Joe Culberson (from, from the left) Paul Lu, Jonathan Schaeffer, Steve Sutphen. **Universities:** Rejkjavik (Iceland), Alberta (Kanada), Hakodate (Japan).



# Chinook project (checkers)

- In 1990 the program was given the right to participate in championships and playing against human.
- The program lost the championships in 1992, but won in 1994. In 1996, the possibility of Chinook's participation was withdrawn (program was much stronger than any other human player).
- Search space of order: $5 \cdot 10^{20}$. Database (library) with information about the best move (continuation) for many states.
- **29.04.2007 — authors of the project anounce English checkters a solved game!** Black (starting the game) have a draw guarantee with a perfect play. White is also guaranteed with a draw, regardless of the first move by black.
- Until today, it is the "largest" solved mind game.

# Chinook project — Samuel legacy



- **Arthur Samuel** wrote a checkers engine program in 1950 under a project sponsored by IBM.
- In 1952, a **genetic element of self-training** was added — two instances of the program were playing against one another with repetitions. The thus evolved program was beating amateurs and intermediate players.
- After program presentation in 1956 for IBM stakeholders, the IBM stock quotes rose by 15 points.
- In 1962 the program played a public match against Robert Nealy (a blind checkers master), in which the program won. The win was given much publicity. Nealy was not a world-class master.
- **In effect, a false belief was spread, that English checkers were already a solved game at the time.** Bjørnsson had troubles obtaining his grant for Chinook research in the 80s because of that.
- A year later, Samuel's program lost a rematch: 1 loss, 5 draws. In 1966 the program lost 8 consecutive games against top level players: Derek Oldbury and Walter Hellman.

# Table of contents

# MIN-MAX algorithm

**Procedure mmEvaluateMax$(s, d, D)$**

1. If $s$ is a terminal then return $h(s)$ (position evaluation).
2. $v := -\infty$.
3. For all states $t$ being descendants of $s$:
   1. $v := \max\{v, \text{mmEvaluateMin}(t, d + \frac{1}{2}, D)\}$.
4. Return $v$.

**Procedure mmEvaluateMin$(s, d, D)$**

1. If $s$ is a terminal then return $h(s)$ (position evaluation).
2. $v := \infty$.
3. For all states $t$ being descendants of $s$:
   1. $v := \min\{v, \text{mmEvaluateMax}(t, d + \frac{1}{2}, D)\}$.
4. Return $v$.

# $\alpha$-$\beta$ *pruning* algorithm

- Many independent discoverers: Samuel (1952), McCarthy (1956), Newell and Simon (1958).
- During analysis two values are propagated down and up the tree:
  $\alpha$ — **guaranteed (so far) payoff for maximizing player**,
  $\beta$ — **guaranteed (so far) payoff for minimizing player**.
- Out-most execution for root forces $\alpha = -\infty$, $\beta = \infty$.
- Children-nodes (and their subtrees) are analyzed while $\alpha < \beta$.
- Whenever $\alpha \geqslant \beta$, one should stop considering successive children (and their subtrees) — they will not affect the outcome for the whole tree; they would be a result of a non-optimal play by some of players.
- In optimistic case, the **gain in complexity** with respect to MIN-MAX is from $O(b^D)$ to $O\left(b^{D/2}\right) = O\left(\sqrt{b^D}\right)$, where $b$ — *branching factor* (constant or average). E.g. for chess $b \approx 40$.
- Owing to the gain one may search deeper.

# $\alpha$-$\beta$ *pruning* algorithm ( *fail-hard* version returns a result within $[\alpha, \beta]$)

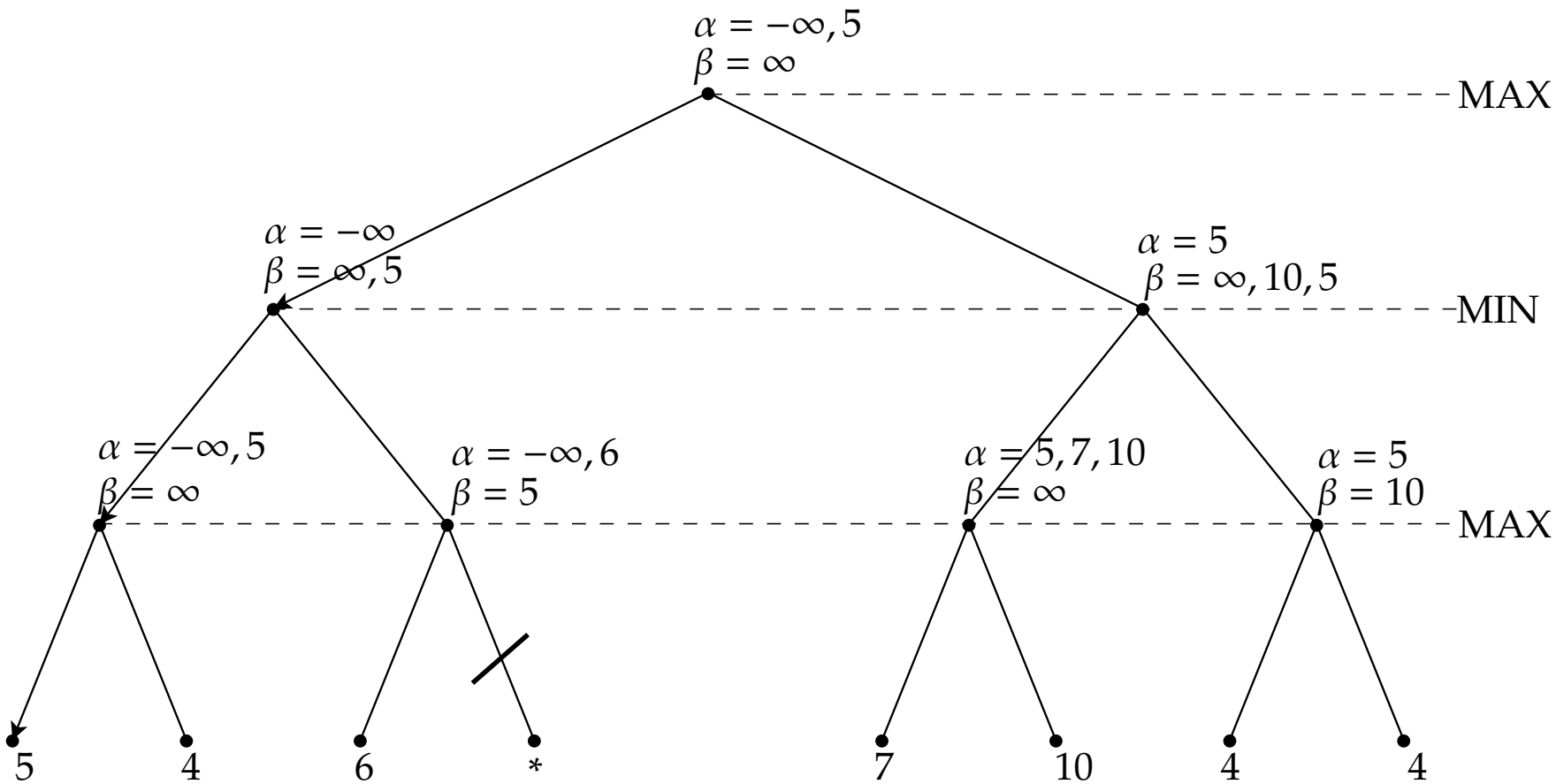
**Procedure alphaBetaEvaluateMax$(s, d, D, \alpha, \beta)$**

1. If $s$ is a terminal then return $h(s)$ (position evaluation).
2. For all states $t$ being descendants of $s$:
   1. $v :=$ alphaBetaEvaluateMin$(t, d + \frac{1}{2}, D, \alpha, \beta)$.
   2. If $\beta \leqslant v$ then return $\beta$. (cut-off)
   3. $\alpha := \max\{\alpha, v\}$.
3. Return $\alpha$.

**Procedure alphaBetaEvaluateMin$(s, d, D, \alpha, \beta)$**

1. If $s$ is a terminal then return $h(s)$ (position evaluation).
2. For all states $t$ being descendants of $s$:
   1. $v :=$ alphaBetaEvaluateMax$(t, d + \frac{1}{2}, D, \alpha, \beta)$.
   2. If $v \leqslant \alpha$ then return $\alpha$. (cut-off)
   3. $\beta := \min\{\beta, v\}$.
3. Return $\beta$.

# Illustration for *α-β pruning* — example 1



# Why is optimistic complexity $O(b^{D/2})$?

- In traditional MIN-MAX:

$$O(\underbrace{b \cdot b \cdots b}_{D\text{-times } b}) = O(b^D). \tag{10}$$

- In $\alpha$-$\beta$ with even number of tree levels, **optimistically** we have:

$$O(\underbrace{b \cdot 1 \cdot b \cdot 1 \cdots b \cdot 1}_{D/2\text{-times } b}) = O\left(b^{D/2}\right). \tag{11}$$

**Explanation:** we need to build all possible children for the first player, but **we assume moves are optimally ordered** therefore, in each child, alread the first move of the second player causes a cut-off ($\alpha \geqslant \beta$) and further moves are discarded as non-optimal ones. And so forth recursively.

- There exist estimates for the average case (random order of children), yielding $O(b^{\frac{3}{4}D})$.
- In chess for: $b = 40$ and $D = 12$ (12 half-moves), the proportion of visited states for pessimistic ordering to visited states for optimistic ordering is $40^6$, i.e. of $10^9$ order.

# $\alpha$-$\beta$ *pruning* algorithm (*fail-soft* version, result can fall outside $[\alpha, \beta]$)

## Procedure fsAlphaBetaEvaluateMax$(s, d, D, \alpha, \beta)$

1. If $s$ is a terminal then return $h(s)$ (position evaluation).
2. For all states $t$ being descendants of $s$:
   1. $v :=$ fsAlphaBetaEvaluateMin$(t, d + \frac{1}{2}, D, \alpha, \beta)$.
   2. $\alpha := \max\{\alpha, v\}$.
   3. If $\alpha \geqslant \beta$ then return $\alpha$. (cut-off)
3. Return $\alpha$.

## Procedure fsAlphaBetaEvaluateMin$(s, d, D, \alpha, \beta)$

1. If $s$ is a terminal then return $h(s)$ (position evaluation).
2. For all states $t$ being descendants of $s$:
   1. $v :=$ fsAlphaBetaEvaluateMax$(t, d + \frac{1}{2}, D, \alpha, \beta)$.
   2. $\beta := \min\{\beta, v\}$.
   3. If $\alpha \geqslant \beta$ then return $\beta$. (cut-off)
3. Return $\beta$.

# Knuth-Moore theorem (1975)

Article: **Knuth D.E., Moore R.W., „*An Analysis of Alpha-Beta Pruning*", Artificial Intelligence, 1975.**

## „Theorem about $\alpha$-$\beta$ window"

*Let $v^*$ denote the true (exact) game result obtained via MIN-MAX procedure. Let $v$ denote the result of fsAlphaBeta (fail-soft) procedure executed for a root node with parameters $\alpha$, $\beta$. Then, possible are three cases:*

1. $\alpha < v < \beta \quad \Rightarrow \quad v = v^*$,
2. $v \leqslant \alpha$ *(failing low)* $\Rightarrow \quad v^* \leqslant v \quad$ *($v$ is an upper bound for $v^*$)*,
3. $\beta \leqslant v$ *(failing high)* $\Rightarrow \quad v \leqslant v^* \quad$ *($v$ is a lower bound fo $v^*$).*

In particular, a consequence: $fsAlphaBeta(\text{root}, -\infty, \infty) = v^*$.

The theorem is useful for building more advanced search algorithms: *Negascout*, *MTD-f* based on so called *zero search windows*.

# *Quiescence* algorithm

- Tries to mimic the intuition of human players by: **expanding *loud* nodes/leaves, and not expanding *quiet* nodes/leaves (instead, immediate return of position evaluation)**.
- Partially solves the *horizon effect* problem.
- We call a position **quiet** if no sudden changes of position evaluation between given state and its descendants occur (e.g. takes/captures).
- An assesment if given state is quiet or not may not be easy; it may require a heuristic in itself. Importantly, such an assement must be faster than expanding a new tree level.
- *Quiescence* does not have to be applied only at leaves level, but already sooner. Current depth can be used as an element for assesment of quietness, i.e. **the deeper we are the larger tendency to leave quiet states not expanded**.
- Exact description e.g. in: D. Laramée, *Chess Programming Part V: Advanced Search*, 2000.

# Sorting children nodes in $\alpha$-$\beta$ *pruning*

It is worth to sort children nodes, especially "at the top of the tree", where it is fairly cheap (there are fewer states at higher levels) and might result in greater savings deeper.

## Sorting heuristics

- in chess: ***„captures first"***,
- in many card games e.g. bridge: ***„extreme cards first, middle ones later"***; e.g. a hand $A, D, 8, 6, 5, 2$ can be sorted to: $A, 2, D, 5, 8, 6$; the order can be also arranged accoring to the position of the player within a trick (e.g. the last player typically plays a high card first, the second player typically plays a low card first, etc.),
- ***sorting according to position evaluation*** — evaluate and sort children immediately based on their position evaluation, before running the recurrence downwards.

# Sorting children nodes in $\alpha$-$\beta$ pruning

## Sorting heuristics (. . . continued)

- ***„refutation table"*** — a table memorizing so called (*refutation moves*) — the ones causing cut-offs, usually at shallow levels; this allows to consider these moves in first order in next search iterations (*progressive search* or *iterative search*). Historically, first chess programs with serious memory limitations typicall kept $b^2$ refutation moves for root grand children, or possibly $b^2 + b^4$ by considering two more levels. Also called *best (principal) continuation table*.
- ***„killer heuristic"*** — memorizes for each level a short list of *killer moves* (usually 2, 3 moves) causing cut-offs and taking those into account in first order for other states at the same level; intuition: if some move is good for some state it might be also good for another (similar) state.
  If a *non-killer* moved caused a cut-off it is introduced to the list in the place of the "weakest" killer-move so far (counting approaches).

# Transposition table

- The name comes from chess and represents the possibility of obtaining the same position (state) by different sequences of moves.
- If the downwards recurrence for such a state has already been calculated then one can save time by using a ready-made result.
- Often, implemented as a *hash map* (time efficience, analogical to *Closed* set in $A^*$, BFS, etc.). The keys in hash map are states themselves or their abbreviations — hash codes (e.g.. in chess positions of at most 32 pieces are required plus information about castling and *en passant* capture possibilities).
- Conditions for reusing a state from the transposition table:
  - Depth of the state in transposition table not deeper than for the tested state (so that the ready score comes from an analysis of a tree of equal or greater depth).
  - $\alpha$-$\beta$ window for the state in transposition table must be not more narrow than current ones (so that the ready score was not affected by more cut-offs)
- Sometimes applied as the book of openings or endgames (chess, checkers).

# *Scout* algorithm)

- Historically, an idea due to **J. Pearl**'a (1980): for reconnaissance, one may preliminarily and less expensively test if the current payoff can be improved. Two recursive procedures eval(·) and test(·); the second returns a boolean inficating if an improvement is possible.
- The idea developed further by **A. Reinefeld**, *„An Improvement to the Scout Tree Search Algorithm"*, ICCA Journal, (1983); introduced are so called ***zero*** $\alpha$-$\beta$ ***windows*** (also: *null windows*, *scout windows*).
- If payoffs are integers, a *zero window* takes place when

$$\alpha + 1 = \beta. \tag{12}$$

- The idea "cooperates" with the Knuth-Moore theorem.

# *Scout* algorithm

**Definition**

We say that a given $\alpha$-$\beta$ window *succeeded* if $v$ returned by the *fsAlphaBeta* procedure (fail-soft) is such that: $\alpha < v < \beta$. It implies (Knuth-Moore) that the true game value $v^*$ equals $v$.

**Definition**

We say that a given $\alpha$-$\beta$ window ***failed low*** if $v$ returned by the *fsAlphaBeta* procedure (fail-soft) is such that: $v \leqslant \alpha$. It implies (Knuth-Moore) that $v$ is an upperbound on the true game value: $v^* \leqslant v$.

**Definition**

We say that a given $\alpha$-$\beta$ window ***failed high*** if $v$ returned by the *fsAlphaBeta* procedure (fail-soft) is such that: $\beta \leqslant v$. It implies (Knuth-Moore) that $v$ is a lowerbound on the true game value: $v \leqslant v^*$.

# *Scout* algorithm

- **The more narrow the imposed window is, the greater chance to generate more cut-offs.**
- Only the first child of each state is analyzed by a full $\alpha$-$\beta$ window, the **second and successive children are analyzed by a zero window**, i.e. $\alpha$-$(\alpha+1)$ or $(\beta-1)$-$\beta$, respectively for MAX, MIN states.
- A **zero window must fail either way.**
- If a zero window imposed on a child of MAX state failed low then we do not have to care — the payoff for the maximizing player could not be improved within this child (**computational gain**, probably greater number of cutoffs should appear within the subtree of that child).
- If a zero window imposed on a child of MAX failed high then we have to repeat the search for that child (**computational loss**) with a wider window $v$-$\beta$ in order to obain an exact result for given subtree. Remark: still, the window (for the repeated calculation) is more narrow than the original one: $\alpha$-$\beta$.
- The last two remarks are suitably opposite for MIN states.

# *Scout* algorithm

**Procedure scoutMax$(s, d, D, \alpha, \beta)$**

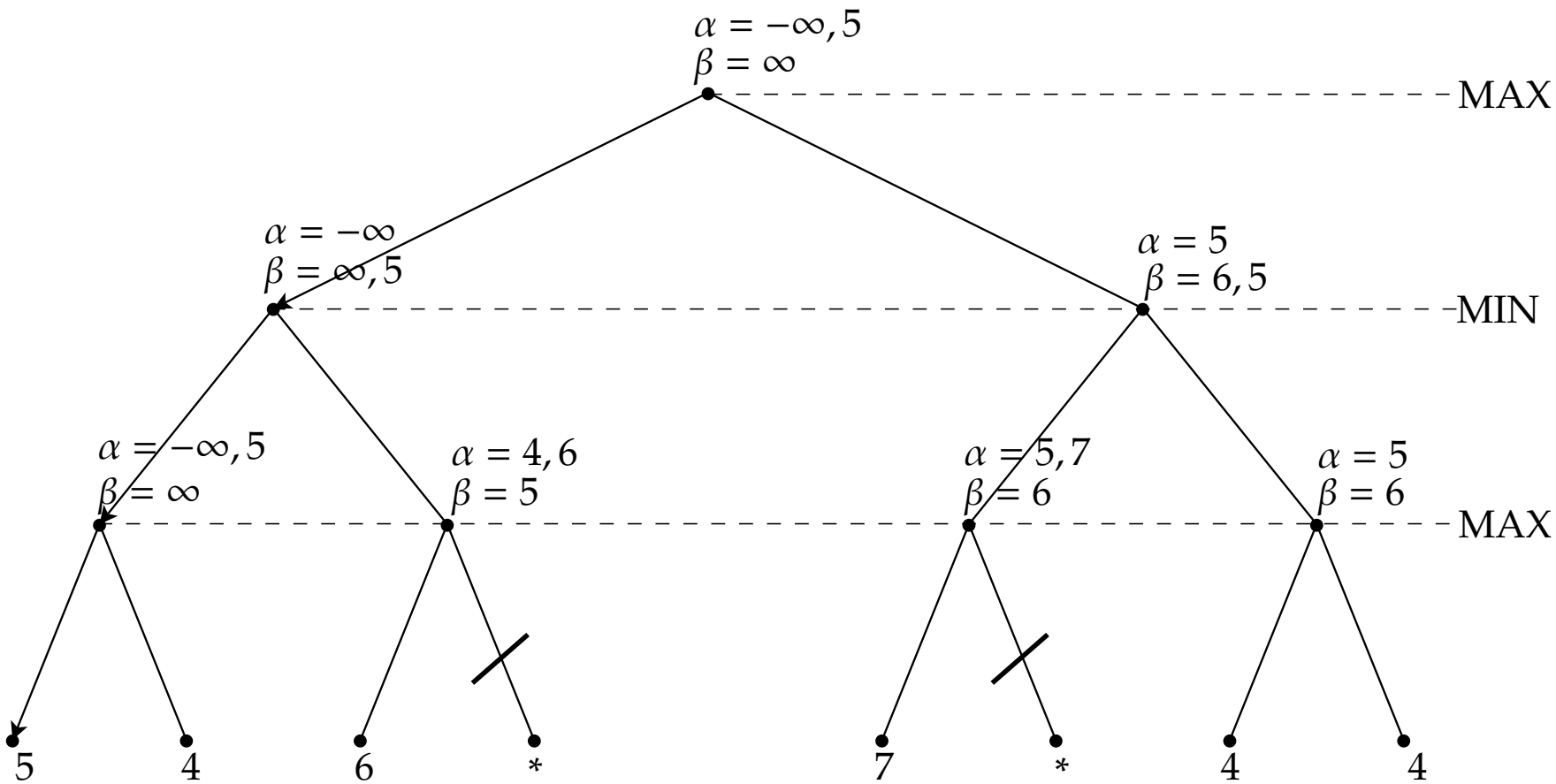
1. If $s$ is a terminal then return $h(s)$ (position evaluation).
2. $b := \beta$.
3. For all states $t$ being descendants of $s$:
   1. $v :=$ scoutMin$(t, d + \frac{1}{2}, D, \alpha, b)$.
   2. If $t$ is not the first child and $D - d > 2 \cdot \frac{1}{2}$ and $b \leqslant v$ (failing high) then:
      1. $v :=$ scoutMin$(t, d + \frac{1}{2}, D, v, \beta)$. (repeat search with a wider window)
   3. $\alpha := \max\{\alpha, v\}$.
   4. If $\alpha \geqslant \beta$ then return $\alpha$. (cut-off)
   5. $b := \alpha + 1$.
4. Return $\alpha$.

# *Scout* algorithm

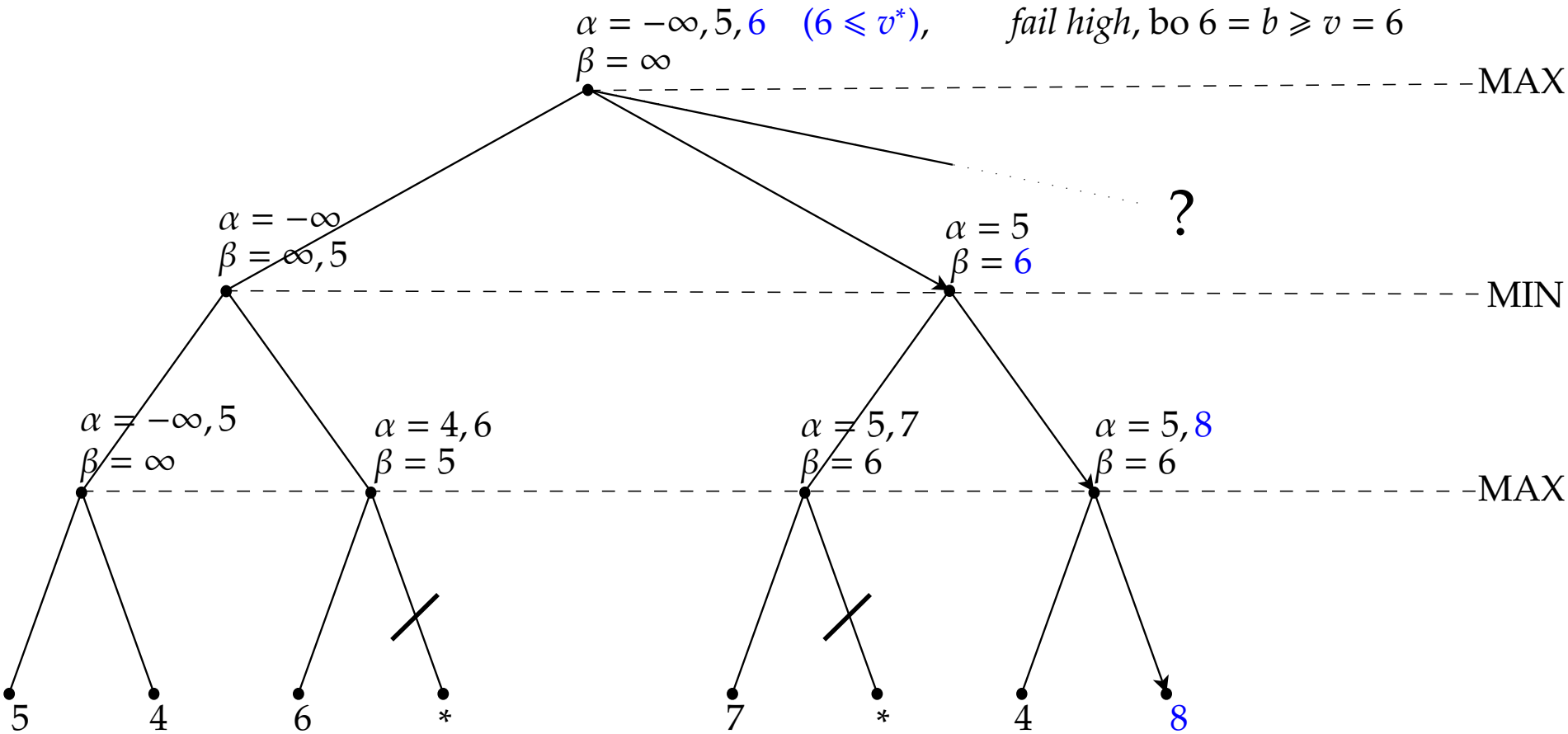
**Procedure scoutMin$(s, d, D, \alpha, \beta)$**

1. If $s$ is a terminal then return $h(s)$ (position evaluation).
2. $a := \alpha$.
3. For all states $t$ being descendants of $s$:
   1. $v :=$ scoutMax$(t, d + \frac{1}{2}, D, a, \beta)$.
   2. If $t$ is not the first child and $D - d > 2 \cdot \frac{1}{2}$ and $v \leqslant a$ (failing low) then:
      1. $v :=$ scoutMax$(t, d + \frac{1}{2}, D, \alpha, v)$. (repeat search with wider window)
   3. $\beta := \min\{\beta, v\}$.
   4. If $\alpha \geqslant \beta$ then return $\beta$. (cut-off)
   5. $a := \beta - 1$.
4. Return $\beta$.

# Illustration for *Scout* — example 1



# Illustration for *Scout* — example 2



# *Scout* algorithm

- The condition $D - d > 2 \cdot \frac{1}{2}$ checks if we are at a depth of 2 halfmoves away or deeper from the search horizon. If so then it is not necessary to repeat the search despite *fail* situation, because the algorithm works accurately at such depths.
- The algorithm works well in the *progressive search* scenario when cooperating with children-sorting heuristics, especially with *killer heuristic*, when the best move (and best path) are often considered first. Due to that fact, the scenario is often called ***Principal Variation Search***.
- Experiments indicate that **computational gains implied by zero windows are more frequent cut-offs are typically greater than computational losses induced by repeated searches**.
- Reinefeld's experiments showed that for trees with a branching factor within $b \in \{20, \ldots, 60\}$ (e.g. chess), *Scout* visitis on average about 20% fewer tree leaves. Tests for depths: $4, 5$ halfmoves.

# *Negamax* algorithm

- Fact:

$$\forall n \in \mathbb{N} \quad \forall a_1, a_2, \ldots, a_n \qquad \min\{a_1, a_2, \ldots, a_n\} = -\max\{-a_1, -a_2, \ldots, -a_n\}.$$

- Owing to the above one may simplify implementation by replacing two twin-procedures by a single one in all algorithms: *Negamax* (in fact *negaAlphaBeta*), *Negascout*, etc.

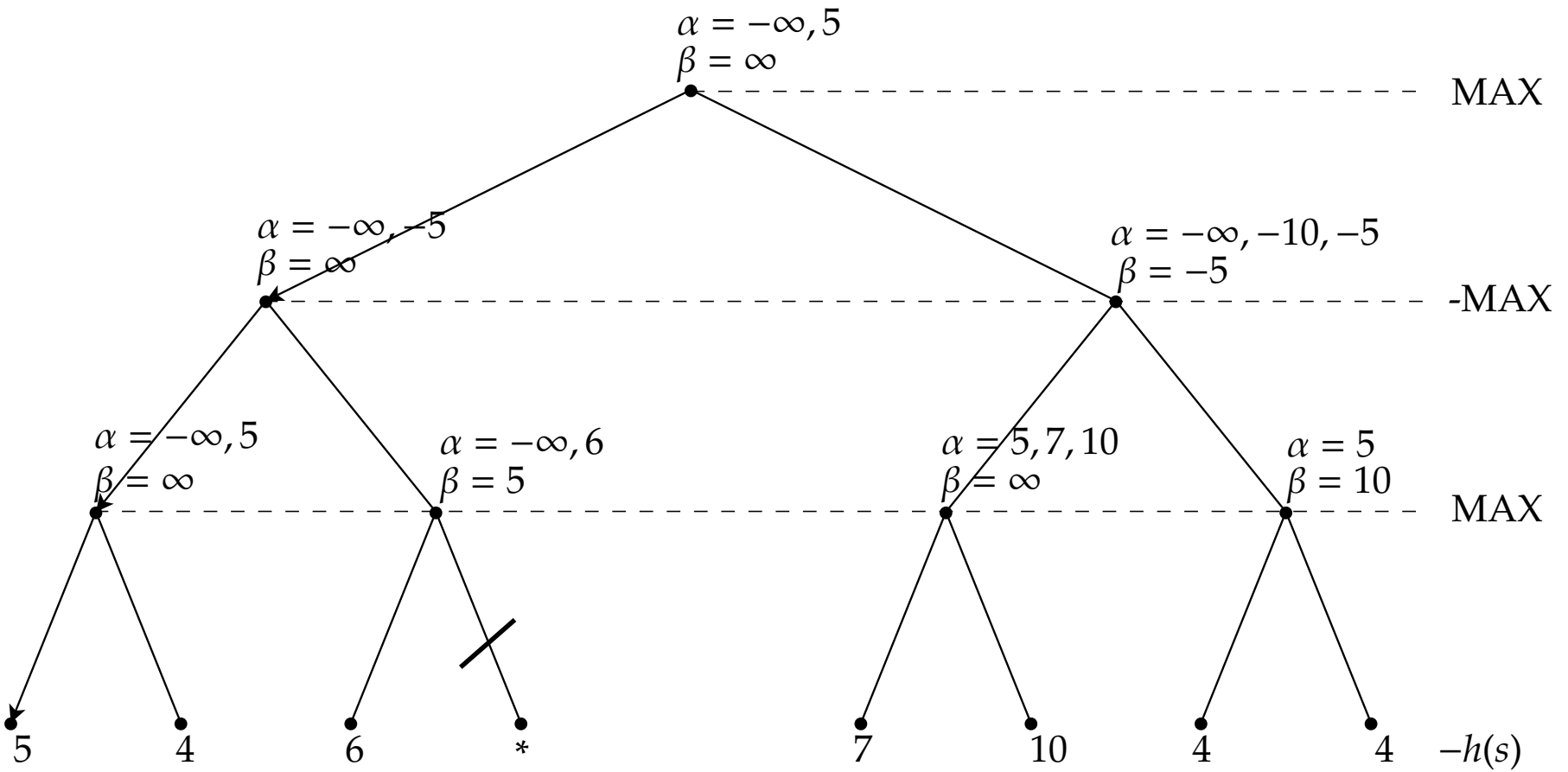
# *Negamax* algorithm

**Procedure negaMax($s, d, D, \alpha, \beta,$ color)**

1. If $s$ is a terminal then return color $\cdot\, h(s)$ (position evaluation).
2. For all states $t$ being descendants of $s$:
   1. $\alpha := \max\{\alpha, -\text{negaMax}(t, d + \frac{1}{2}, D, -\beta, -\alpha, -\text{color})\}$.
   2. If $\alpha \geqslant \beta$ then return $\alpha$.
3. Return $\alpha$.

The out-most call for the root is:: negaMax(root, $0, D, -\infty, \infty, 1$).

# Illustration for *Negamax* — example 1



# *Negascout* algorithm

**Procedure negascout($s, d, D, \alpha, \beta$, color)**

1. If $s$ is a terminal then return color $\cdot\, h(s)$ (position evaluation).
2. $b := \beta$.
3. For all states $t$ being descendants of $s$:
   1. $v := -\text{negascout}(t, d + \frac{1}{2}, D, -b, -\alpha, -\text{color})$.
   2. If $t$ is not the first child and $D - d > 2 \cdot \frac{1}{2}$ and $b \leqslant v$ (failing high) then:
      1. $v := -\text{negascout}(t, d + \frac{1}{2}, D, -\beta, -v)$. (repeated search with a wider window)
   3. $\alpha := \max\{\alpha, v\}$.
   4. If $\alpha \geqslant \beta$ then $\alpha$. (cut-off)
   5. $b := \alpha + 1$.
4. Return $\alpha$.

# *Warcabnik* (Mateusz Bożykowski) (1)

Master thesis: **Mateusz Bożykowski, *Implementation of self-teaching program for checkers*, WI, 2009.**

- Checkers: **international (a.k.a. Polish)** (100 squares, players have 20 pawns each), **Brazilian** (64 squares, 12 pawns per player), **English** (64 squares, 12 pawns, kings moving by 1 square).
- Implementation of $\alpha$-$\beta$ *pruning* including transpostion table and *Quiescence*.
- Multiple programs playing checkers according to different heuristics (position evaluation functions) and competing withing a **genetic evolution**.
- **Individuals can be identified with heuristics** evaluation positions (various AIs). The simplest heuristics **materialistic, symmetrical**:

$$h = w_1 P_p + w_2 K_p - w_1 P_o - w_2 K_o, \tag{13}$$

where $P, K$ denote the number of pawns and kings, respectively; whereas indexes $p, o$ stand for player and opponent, respectively. The parameters under genetic optimization are $w_1, w_2$.

- Parameters coded as integers, initially picked on random from $\{-100, \ldots, 100\}$. During evolution, the parameter have not gone outside the initial range.

# *Warcabnik* (Mateusz Bożykowski) (2)

- It is difficult to fairly assign a numeric fitness or rank to such due to the possibility of *three-ways-draw*: $A$ wins against $B$, $B$ wins agains $C$, $C$ wins against $A$.
- **Tournament selection** comes to mind. Difficulties: (1) frequent draws (who should be then selected to next population?), (2) possibility of losing the best individual, (3) pointing out the best individual in the final population.
- Final approach: **tournaments for population sizes being powers of** 2 $n = 2^m$. Individuals paired randomly into matches, $n/2$ of population filled up with winners (in case of a draw added was 1 child of crossed parents), the rest of population filled up iteratively with winners of matches between winners added before.
- The winner of the very last match (the winner among winners) considered the best individual in the final population.

# *Warcabnik* (Mateusz Bożykowski) (3)

- **Linear cross-over** — draw a randome number $\alpha \in (0, 1)$, a child $C$ of parents $A$ and $B$ is obtained as:
$$w_i(C) = \alpha w_i(A) + (1 - \alpha) w_i(B). \tag{14}$$
- **Uniform cross-over** — for all $w_i$ we randomly decide from which parent it comes (and copy it unchanged). Additionally, mutation is suggested.
- **Mutation** of constant radius — each gene (weights) is added a random value from $\{-20, \ldots, 20\}$. The probability of mutation is lineaerly decreased from 0.9 in the first iteration to 0.3 in the last.
- Different depths for tree analysis were set up based on the GA iteration (the later iteration the more accurate the analysis should be).

# *Warcabnik* (Mateusz Bożykowski) (4)

**Studied heuristics:**

- **materialistic, unsymmetrical**

$$h = w_1 P_p + w_2 K_p + w_3 P_p + w_4 K_p \tag{15}$$

- **materialistic-positional, symmetrical** (in general $h = h_p - h_o$)

$$h_s = w_1 P_s + w_2 K_s + w_3 C_s + w_4 S_s + w_5 OT_s + w_6 PT_s + w_7 KD_s, \tag{16}$$

where: $C$ — number of pieces in the board center, $S$ — number of pieces at sides, $OT$ — number of pawns at promotion row for opponent (defence of promotion), $PT$ — number of pawns one square away from own promotion, $KD$ — number of kings on main diagonal.

- **extended materialistic-positional, symmetrical** (in general $h = h_p - h_o$)

$$h_s = w_1 P_s + \cdots + w_7 KD_s + w_8 M_s + \left\lfloor w_9 \frac{D_s}{P_s + K_s} \right\rfloor + w_{10} KD2_s, \tag{17}$$

where: $M \in \{0, 1\}$ — extra reward for turn to move, $D$ — number of doubled pieces (touching by corners), $KD2$ — number of kings on opposite double diagonal.

# *Warcabnik* (Mateusz Bożykowski) (5)

**Studied heuristics:**

- **materialistic-row-wise** — pawns have different values according to rows they occupy.

$$h_s = \sum_{i=1}^{N-1} w_i P_i + w_N K, \tag{18}$$

where: $N$ — number of rows on the board, $P_i$ — number of pawns in $i$-th row.

- **materialistic-structural, symmetrical** — created based on an observation that row-wise heuristics was significantly weaker than extended materialistic-positional (an attempt to improve the row-wise heuristics), and an observation that the number of doubled pieces had a negative impact on evaluation:

$$h_s = w_1 P_s + w_2 K_s + w_3 OT_s + w_4 PT_s + w_5 I_s + w_6 F_s \tag{19}$$

where: $I$ — number of immortal (non-capturable) pieces, $F$ — number of frozen pieces.

# *Warcabnik* (Mateusz Bożykowski) (6)

**Obtained results of optimization:**

- **materialistic, unsymmetrical**:

$$P_p = 5, K_p = 12, P_o = -7, K_o = 10.$$

Comment: aggressive play with pawns — capturing opponent's pawn increases the evaluation by 2. Careful play with kings in endgame, because own kings are worth more than opponents'.

- **materialistic-positional, symmetric**:

$$P_s = 24, K_s = 65, C_s = 1, S_s = 1, OT_s = -11, PT_s = 27, KD_s = 0.$$

Comment: surprisingly the pawns defending promotion line are evaluated negatively, and kings on main diagonal as immaterial.

# *Warcabnik* (Mateusz Bożykowski) (7)

**Obtained optimization results:**

- **extended materialistic-positional, symmetrical**:

$$P_s = 5, K_s = 16, C_s = 0, S_s = 0, OT_s = 0, PT_s = 6, KD_s = 0, M_s = 0, D_s = -7, KD2_s = 0.$$

Comment: surprisingly, most of parameters were zeroed; of relevance seem to be: pawns just before promotion and penalty for doubled pieces.

- **extended materialistic-row-wise, symmetrical**:

$$P_1 = 2, P_2 = 1, P_3 = 2, P_4 = 2, P_5 = 2, P_6 = 2, P_7 = 1, P_8 = 3, P_9 = 6, K = 12.$$

Comment: interestingly, values in rows from 3 to 6 are equal; a pawn in 8-th row already starts to be worth more.

- **extended materialistic-structural, symmetrical**:

$$P_s = 13, K_s = 85, OT_s = 0, TT_s = 6, I_s = 1, F_s = -1.$$

Comment: one can note that immortal pieces have a positive impact, frozen ones a negative impact (before both those elements were 'hidden' in the doubling parameter $D$).

# *Warcabnik* (Mateusz Bożykowski) (7)

**Comparison against free checkers programs:**

- **Dam 2.0** — highly assessed program, developed since 1987. Lack of possibility to set up the tree depth — **inaccurate comparison**. Test: best AI from *Warcabnik* with 7 halfmoves horizon vs. successive levels of *Dam 2.0*. *Warcabnik* winse agains levels Beginner A, B, draws with Beginner C, D, starts losing against Beginner E (in fact, it loses endgames involving kings).
- **Little polish v0.7** — a program by Piotr Beling for Brazilian checkers. The opponent was set up limited to 1 *s* per move. *Warcabnik* drew, despite the opponent was at times analyzing even 18 halfmoves deep. In matches against stronger AIs ($> 1$ *s*) *Warcabnik* was losing.
- **Windames 3D** — allows to set up 9 difficulty levels. *Warcabnik* wins with first three and loses against next ones. Since times to move at fourth level are about equal, one may suspect that the depth of analysis is then similar.

# *Warcabnik* (Mateusz Bożykowski) (8)

**Comparison against free checkers programs:**

- **Warcaby v.1.2** — a program by Marcin Żółtkowski for Polish and Brazilian checkers. It allows to set up the tree depth (fair comparison). Tests for 6 halfmoves (max. allowed for opponent). *Warcabnik* won in both variants of checkers and was much faster (approx. 2 *s* per move, while opponent was taking 30 *s* per move). By setting 1 halfmove less *Warcabnik* was still able to draw in Brazilian checkers.
- **Gui Checkers 1.05+** — a program by Jon Kreuzer for English checkers. It allows to set up both depth and time. Tests for 10 halfmoves — *Warcabnik* was losing even after allowing him for 1 halfmove more. Conclusion: studied heuristics are not sufficient.

# Bridge — "double dummy" problem

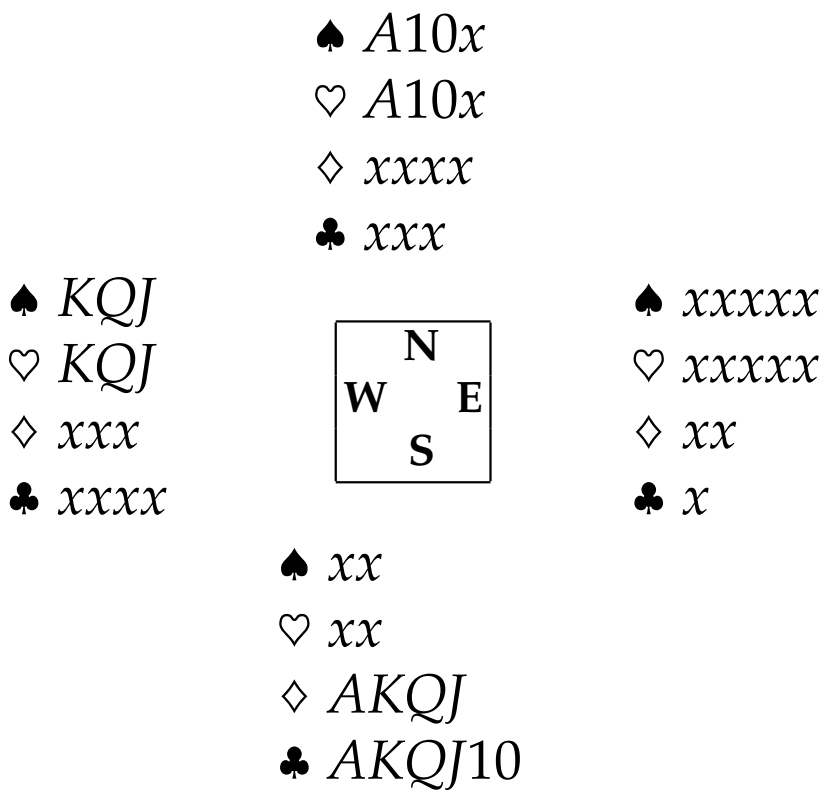
## Double dummy

A version of bridge as a game of perfect information. Helpful for analysis of optimal bridge play at the moment the whole board (all four hands) is known. Existing programs: *Deep finesse*, *GIB*.

Bachelor thesis: **Katarzyna Kubasik,** ***Application of game tree searching algorithms for finding minimax points in "double dummy" bridge*****, WI, 2011.**

- Implementation of $\alpha$-$\beta$ *pruning* with the use transposition table.
- Despite 4 players: N, E, S, W; alternate players (N, S) and (E, W) constitute pairs which can be identified with two players: maximizing and minimizing.
- A play by each player constitutes a new level in the search tree. The full tree has $4 \cdot 13 = 52$ levels.
- When searching **MAX, MIN do not have to alternate** — one has to check which side took the last trick (this side will again have the next move).
- Improving elements: **checking current sequences** (e.g. a configuraiton $6, 4, 2$, becomes a sequence once other players have used the cards 5 and 3); **sorting moves according to heuristics:** ***„extreme cards first, middle ones later"***.

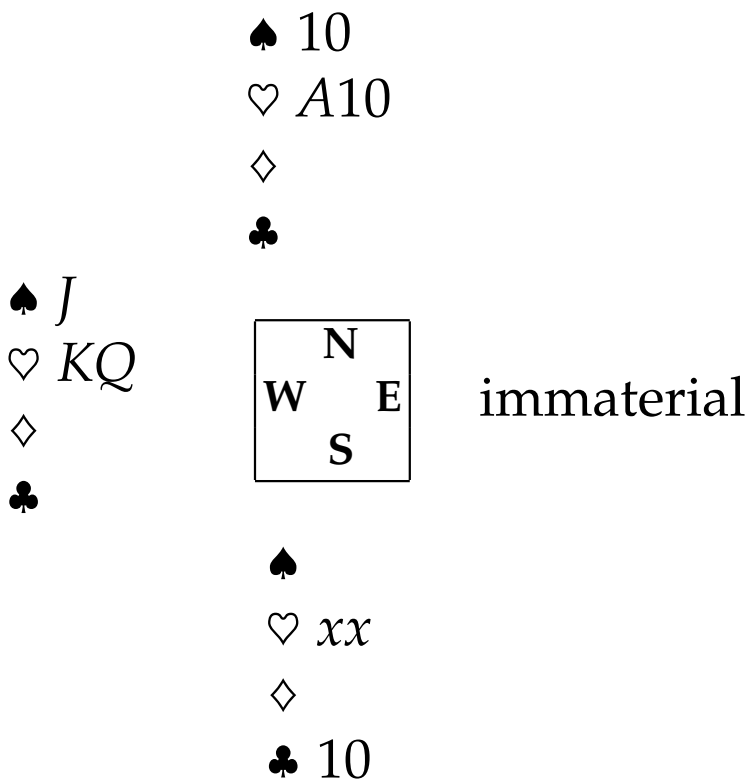
# Example of "double dummy" problem




Contract: 6 no trump by NS. First lead by W: $K\spadesuit$. How NS should play to take 12 tricks?

# "Double dummy" problem — solution (1)

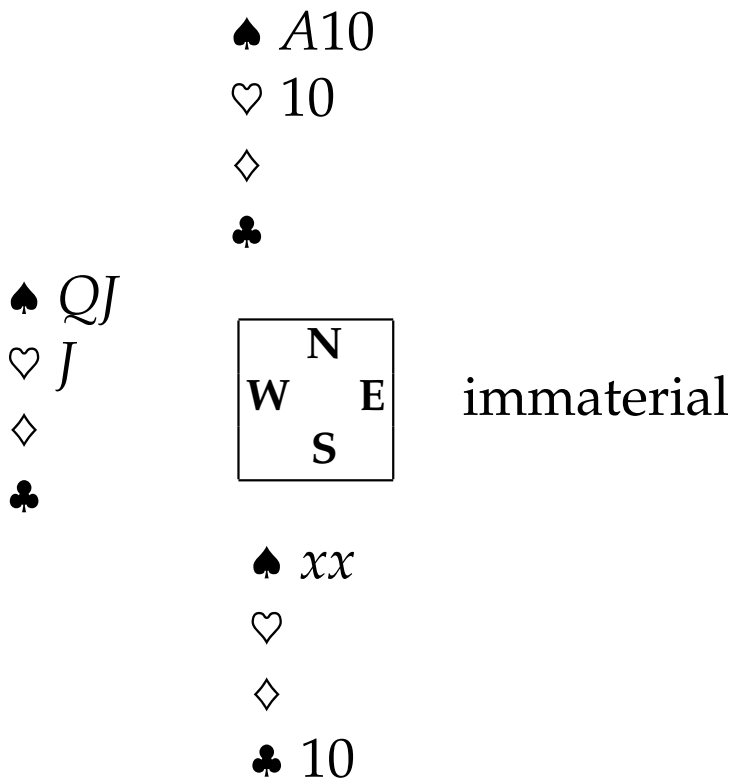
Player N should **duck** (not take with ace) the first trick. After any continuation by W, a **squeeze** in spades and clubs shall take place in the endgame. E.g. after a continutation of $Q\spadesuit$.




S now playes 10♣ and W is squeezed. **Without the initial duck at trick one the squeeze won't take place — the first play by N has a consequence 40 levels deeper in the tree!**

# “Double dummy” problem — solution (2)

After another continuation (an other second lead by W), e.g. $K\heartsuit$ the following endgame occurs:



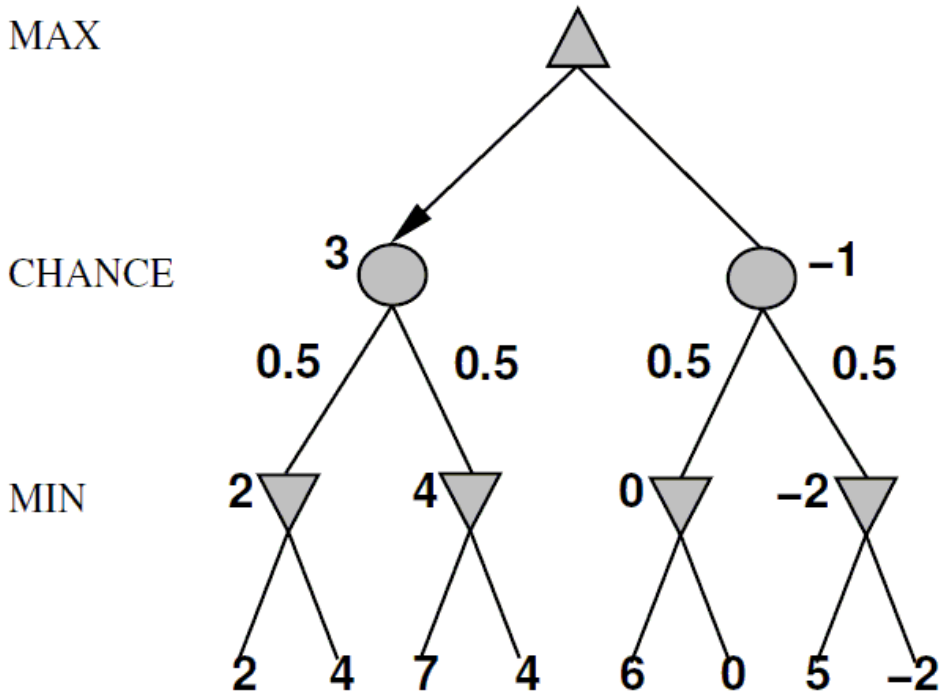

Again, S plays 10♣ and W is squeezed.

# Table of contents

# *Expectiminimax* algorithm (Michie, 1966)

- Dedicated for games where result partially depends on players' choices and partially on **random elements**: e.g. dice roll, coin toss, polling a card from a shuffled deck, etc.
- Idea: apart from regular nodes performing MIN, MAX operations, one should introduce **CHANCE** nodes, which perform **arithmetic average** or **weighted average** operations.
- Places of occurence of CHANCE nodes depends on game rules. In particular, if e.g. everly player rolls dice before his move than the following interlace shall take place: CHANCE - MAX - CHANCE - MIN - . . . .

# Illustration for *Expectiminimax*

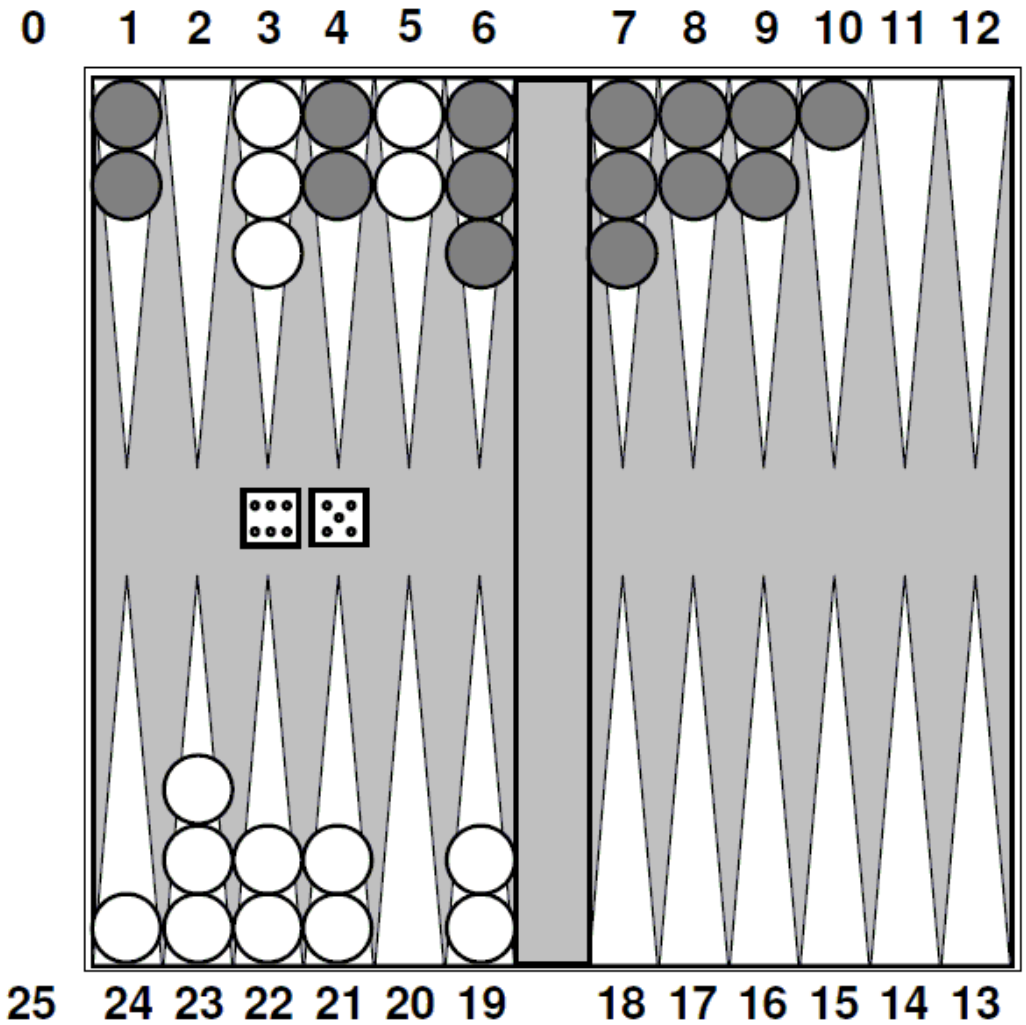


# *Expectiminimax* algorithm

**evaluateState($s, d, D$) (mixed recursive procedure)**

1. Calculate position evaluation $h := h(s)$.
2. If $h = \pm\infty$ or $d = D$ then return $h$.
3. If $s$ is a **MIN** state:
   1. $v := \infty$.
   2. For all $t$ being descendants of $s$: $v := \min\{v, \text{evaluateState}(t, d + \frac{1}{2}, D)\}$.
4. If $s$ is a **MAX** state:
   1. $v := -\infty$.
   2. For all $t$ being descendants of $s$: $v := \max\{v, \text{evaluateState}(t, d + \frac{1}{2}, D)\}$.
5. If $s$ is a **CHANCE** state:
   1. $v := 0$.
   2. For all $t$ being descendants of $s$: $v := v + P(t) \cdot \text{evaluateState}(t, d + \frac{1}{2}, D)$.
6. Return $v$.

# Example: *backgammon*



# Example: *backgammon*

- Two dices are rolled — number of possibilities: $21 = \binom{2+6-1}{2}$. Branching at CHANCE level: $b = 21$.
- For $n = 15$ pawns, in effect of a „typical roll" (non "double") a player can either select one pawn to move (outcomes sum), or select two pawns (individual outcomes). Number of possibilities: $n(n-1) = 210$.
- In case of "doubles" (the same outcome on both dices) a player has 4 single moves (of same value) at disposal. Number of possibilities: $\binom{4+n-1}{4} = 3060$. Doubles occur with $\frac{1}{6}$ probability.
- Field blockages significantly reduce the number of possibilities. The average branching is estimated to be approx. 400.
- As depth increases the probability of some state decreases exponentially fast — therefore, long-term forcasts are of little value.
- The *TDGammon* program visits only 4 halfmoves of depth but has a very complex heuristics for position evaluation (sufficient for a play at master's level).

# Table of contents

# Imperfect information

- E.g. in card games, where we do not know opponents' cards.
- Theoretically, one might calculate the probability of every possible hand (situation) — "a dice with a great number of faces, rolled once at game start".
- **Idea**: calculate payoffs for all players on all hands and choose the play (move) with the greatest *expected* payoff.
- Example: *GIB* — currently, strongest bridge playing problem; generates at least 100 hands (Monte-Carlo sampling) compliant with the so-far information and chooses the play which on average leads to the greates number of taken tricks.
- Special case: **if there exists a play which on every hand leads to the greatest payoff than this play is optimal**.
- **Remark**: the above intuion can be false and may lead to errors if 'the inequality' is not strict (i.e. there exist other playes of equally good average) and if we do not observe variance.
- In fact, it may be necessary to run **multiple repetitions of Monte-Carlo sampling**, each repetition made before successive plays (before each successive search).

# False intuition — example 1

- Exemplary 4-card endgame. “No trump” contract played. Players must follow the suit played.
- Ceratin (known for sure) cards for **player A:** $K\spadesuit, Q\spadesuit, A\diamondsuit, A\clubsuit$
- Ceratin (known for sure) cards for **player B:** $A\spadesuit, J\spadesuit, A\heartsuit$.
- Assume the **fourth card for player B is unknown**: $K\diamondsuit$ or $K\clubsuit$.
- Evaluate the play of $K\spadesuit$ from the perspective of A.

# False intuition — example 1

**Possible hand 1**: (player A) $K\spadesuit, Q\spadesuit, A\diamond, A\clubsuit : A\spadesuit, J\spadesuit, A\heartsuit, K\diamond$ (player B).

Exemplary sequence of correct plays:

1. A: $K\spadesuit$, B: $A\spadesuit$. Remaining: (A) $Q\spadesuit, A\diamond, A\clubsuit : J\spadesuit, A\heartsuit, K\diamond$ (B).
2. B: $A\heartsuit$, A: $A\clubsuit$. Remaining: (A) $Q\spadesuit, A\diamond : J\spadesuit, K\diamond$ (B).
3. B: $J\spadesuit$, A: $Q\spadesuit$. Remaining: (A) $A\diamond : K\diamond$ (B).
4. A: $A\diamond$, B: $K\diamond$

Both sides take 2 tricks. **Score:** 0 **(?)**.

# False intuition — example 1

**Possible hand 2**: (A) $K\spadesuit, Q\spadesuit, A\diamond, A\clubsuit : A\spadesuit, J\spadesuit, A\heartsuit, K\clubsuit$ (B).

Exemplary sequence of correct plays:

1. A: $K\spadesuit$, B: $A\spadesuit$. Remaining: $Q\spadesuit, A\diamond, A\clubsuit$ (A) : $J\spadesuit, A\heartsuit, K\clubsuit$ (B).
2. B: $A\heartsuit$, A: $A\diamond$. Remaining: $Q\spadesuit, A\clubsuit$ (A) : $J\spadesuit, K\clubsuit$ (B).
3. B: $J\spadesuit$, A: $Q\spadesuit$. Remaining: $A\clubsuit$ (A) : $K\clubsuit$ (B).
4. A: $A\clubsuit$, B: $K\clubsuit$

Both sides take 2 tricks. **Score:** 0 **(?)**.

# False intuition — example 1

**In fact, we have**: (player A) $K\spadesuit, Q\spadesuit, A\diamondsuit, A\clubsuit : A\spadesuit, J\spadesuit, A\heartsuit, K*$ (player B).

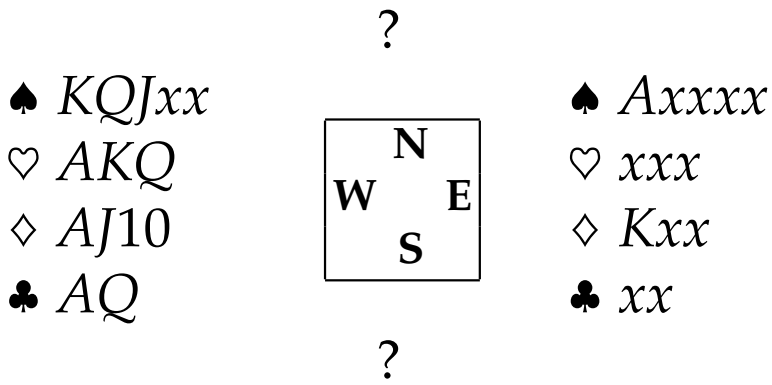
Problem — it is necessary to guess which ace to dispose of at the moment B plays his $A\heartsuit$.

1. A: $K\spadesuit$, B: $A\spadesuit$. Remaining: $Q\spadesuit, A\diamondsuit, A\clubsuit$ (A) : $J\spadesuit, A\heartsuit, K*$ (B).
2. B: $A\heartsuit$, A: $A\diamondsuit/A\clubsuit$ (?) Remaining: $Q\spadesuit, A\clubsuit$ (A) : $J\spadesuit, K*$ (B).

**Score:** $\frac{1}{2} \cdot (-1) + \frac{1}{2} \cdot 0 = -\frac{1}{2}$.

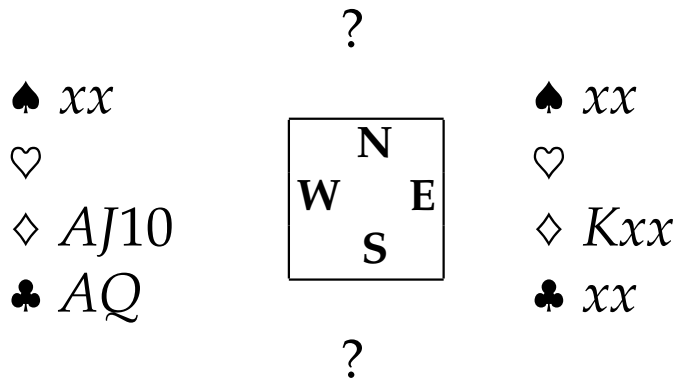
**Remark:** Other first plays by A: $A\diamondsuit$ i $A\clubsuit$ **are not associated with positive variation while also leading to the payoff** 0.

# False intuition — example 2




Contract: 6 spades played by WE. First lead N: $x\spadesuit$. Key missing cards: $Q\diamondsuit$, $K\clubsuit$. **Does there exist a play guaranteeing** 12 **tricks?**

# False intuition — example 2 — solution




**Optimal play:** We trump out opponents — spades (by playing three rounds of spades if need be), we play three rounds of hearts. An endgame as above shall take place. Now, we play $A\clubsuit$ and $Q\clubsuit$ and we give up $Q\clubsuit$ voluntarily! Regardles of continuation by N or S, 12 tricks is guaranteed.

An other play based on attempts to catch $K\clubsuit$ at S (odds: $\approx 50\%$) and to catch $Q\diamondsuit$ at S or N (odds: $\approx 50\%$) leads to the following expected number of tricks: $\approx \frac{1}{4}11 + \frac{2}{4}12 + \frac{1}{4}13 = 12$, but with variance: $\approx \frac{1}{4}(12-11)^2 + \frac{2}{4}(12-12)^2 + \frac{1}{4}(12-13)^2 = \frac{1}{2}$.

# Some references


1. J. von Neuman and O. Morgenstern, *Theory of Games and Economic Behavior*, 1944 (see: `http://press.princeton.edu/titles/7802.html`).
2. *Chinook* project wesite: `http://webdocs.cs.ualberta.ca/~chinook`.
3. D.E. Knuth, R.W. Moore, *An Analysis of Alpha-Beta Pruning*, Artificial Intelligence, 1975 (patrz: `http://www.eecis.udel.edu/~ypeng/articles/An Analysis of Alpha-Beta Pruning.pdf`).
4. A. Reinefeld, *„An Improvement to the Scout Tree Search Algorithm"*, ICCA Journal, 1983 (see: `http://www.top-5000.nl/ps/An improvement to the scout tree search algorithm.pdf`)
5. D. Larameé, *Chess Programming I-V*, 2000. `http://www.gamedev.net/page/resources/_/reference/programming/artificial-intelligence/gaming/chess-programming-part-i-getting-started-r1014`)
6. M. Bożykowski, *Implementation of self-teaching program for checkers*, master thesis, WI ZUT, 2009.
7. K. Kubasik, *Application of game tree searching algorithms for finding minimax points in "double dummy" bridge*, bachelor thesis, WI ZUT, 2011.
8. P. Beling, *Practical aspects of logical games programming*, master thesis, Polytechnics of Łódź, 2006.
9. *Expectiminimax tree*, Wikipedia, (see: `http://en.wikipedia.org/wiki/Expectiminimax_tree`).
10. Materials on *GIB* — bridge playing program, (see: `http://www.greatbridgelinks.com/gblSOFT/Reviews/SoftwareReview090301.html` and `http://www.gibware.com/`).