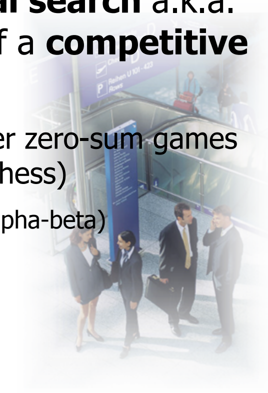
Artificial Intelligence 1

Roman Barták
Department of Theoretical Computer Science and Mathematical Logic

Adversarial Search: Games

- So far we assumed a single-agent environment, but what if there are more agents and some of them „playing“ against us?
- Today we will discuss **adversarial search** a.k.a. **game playing**, as an example of a **competitive multi-agent environment**.
  - deterministic, turn-taking, two-player zero-sum games of perfect information (tic-tac-toe, chess)
    - optimal (perfect) decisions (minimax, alpha-beta)
    - imperfect decisions (cutting off search)
  - stochastic games (backgammon)



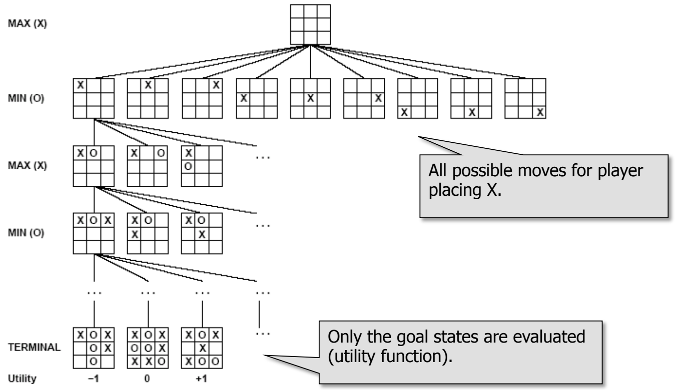
- **Mathematical game theory** (a branch of economics) views any multi-agent environment as a game, provided that the impact of each agent on others is significant.
  - environments with many agents are called economies (rather than games)
- AI deals mainly with **turn-taking**, **two-player zero-sum** games (one player wins, the other one loses).
  - **deterministic** games vs. stochastic games
  - **perfect** information vs. imperfect information
  - Why games in AI? Because games are:
    - hard to play
    - easy to model (not that many actions)
    - funny

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information		bridge, poker, scrabble nuclear war

- We consider two players **MAX** and **MIN**
  - MAX moves first, and then the players take turns moving until the game is over
  - we are looking for the strategy of MAX
- Again, we shall see game playing as a search problem:
  - **initial state**: specifies how the game is set up at the start
  - **successor function**: results of the moves (move, state)
    - the initial state and the successor function define the **game tree**
  - **terminal test**: true, when the game is over (a goal state)
  - **utility function**: final numeric value for a game that ends in terminal state (win, loss, draw with values +1, 0, -1)
    - higher values are better for MAX, while lower values are better for MIN

Game tree – tic-tac-toe

- Two players place X and O in an empty square until a line of three identical symbols is reached or all squares are full.



Optimal strategy

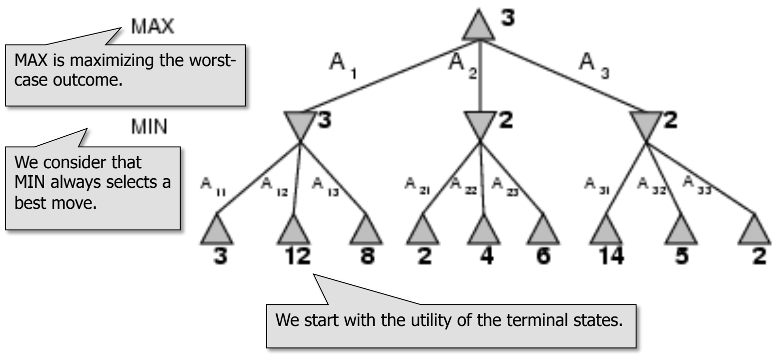
- Classical search is looking a **(shortest) path to a goal state**.
- Search for games is looking for a **path to the terminal state with the highest utility**, but MIN has something to say about it.
- MAX is looking for a contingent strategy, which specifies
  - MAX's move in the initial state
  - MAX's moves in the states resulting from every possible response by MIN
  - an **optimal strategy** leads to outcomes at least as good as any other strategy when one is playing an infallible opponent

Minimax value

- The optimal strategy can be determined from the **minimax value** of each node computed as follows:

MINIMAX-VALUE(n)=

UTILITY(n)	if n is a terminal state
$\max_{s \in successors(n)}$ MINIMAX-VALUE(s)	if MAX plays in n
$\min_{s \in successors(n)}$ MINIMAX-VALUE(s)	if MIN plays in n



Algorithm minimax

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

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
```

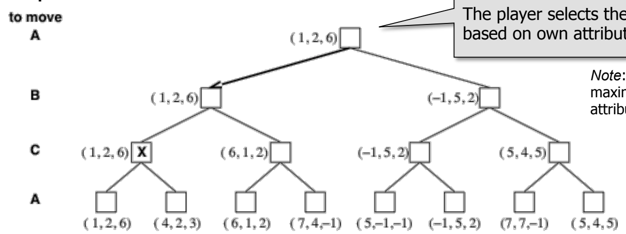
The algorithm assumes that the player plays optimally. Otherwise, the utility is even higher

- **Time complexity O(b^m)**
- **Space complexity O(bm)**

(b - #actions in states, m - #moves)

Minimax for more players

- For multiplayer games we can use a **vector of utility values** – this vector gives the utility of the state from each player's viewpoint.



The player selects the best move based on own attribute in vector.

Note: each player is maximizing a value of own attribute in the vector.

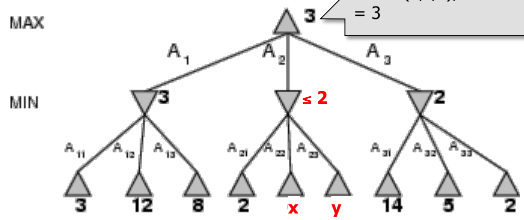
- Multiplayer games usually involve **alliances**, whether formal or informal, among the players.
  - Alliances seems to be a natural consequence of optimal strategies for each player.
  - For example, suppose A and B are in weak positions and C is in a stronger position. Then it is often optimal for both A and B to attack C rather than each other.

    Of course, as soon as C weakens under the joint onslaught, the alliance loses its value.

Improving minimax

- The minimax algorithm always finds an optimal strategy, but it has to explore a complete game tree.
- Can we speed-up the algorithm?
  - YES!
    We do not need to explore all states, if the are "very bad".
  - α-β **pruning** eliminates branches that cannot possibly influence the final decisions.

= max(min(3,12,8),min(2,x,y),min(14,5,2))
= max(3,min(2,x,y),2)
= max(3,z,2), where z ≤ 2
= 3



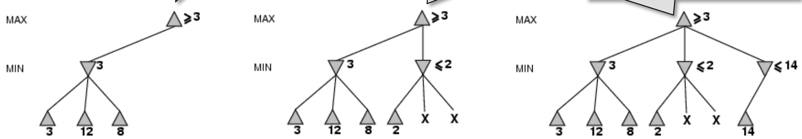
MINIMAX value of the root does not depend on values x and y and hence it is not necessary to explore these sub-trees.

α-β pruning - example

The first estimate of the MINIMAX value of root.

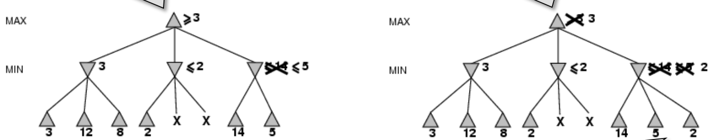
We can stop evaluation of the MIN node when its MINIMAX value is worse (smaller) than in the parent.

For the third MIN node we can still find a better solution.



We can still find a better value for the MIN node in the range ⟨3,5⟩.

Hmm, it was a false hope, the optimum is 3.



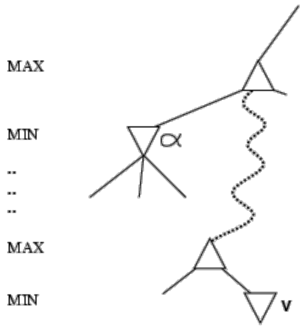
If we explored the nodes in the order 2,5,12, it would be enough to evaluate node 2.

Algorithm α-β

```
function ALPHA-BETA-SEARCH(state) returns an action
   inputs: state, current state in game

   v ← MAX-VALUE(state, −∞, +∞)
   return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
   inputs: state, current state in game
           α, the value of the best alternative for MAX along the path to state
           β, the value of the best alternative for MIN along the path to state

   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for a, s in 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
   inputs: state, current state in game
           α, the value of the best alternative for MAX along the path to state
           β, the value of the best alternative for MIN along the path to state

   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← +∞
   for a, s in SUCCESSORS(state) do
      v ← MIN(v, MAX-VALUE(s, α, β))
      if v ≤ α then return v
      β ← MIN(β, v)
   return v
```

Why α-β?



- α is the value of best (i.e. the highest-value) choice we have found so far at any choice point along the path for MAX
  - if α is not worse (smaller) than v, MAX will never play in the direction to v and hence the sub-tree below v does not need to be explored
- β is the value of best (i.e. the lowest-value) choice we have found so far at any choice point along the path for MIN
  - we can similarly prune the sub-trees for MIN

Properties:

- By cutting off the sub-trees we do not miss optimum.
- By „perfect ordering" we can decrease time complexity to $O(b^{m/2})$, which gives a branching factor $\sqrt{b}$ (b for minimax), so we can solve a tree roughly twice as deep as minimax in the same amount of time.

„Imperfect" strategies

- Both minimax and α-β have to **search all the way to terminal states**.
  - This is not practical for bigger depths (depth = #moves to reach a terminal state).
- We can **cut off search** earlier and apply a heuristic evaluation function to states in the search.
  - does not guarantee finding an optimal solution, but
  - can finish search in a given time
- **Realisation:**
  - terminal test → cutoff test
  - utility function → heuristic evaluation function EVAL

Evaluation function

- Returns an estimate of the expected utility of the game from a given position (similar to the heuristic function h).
- Obviously, quality of the algorithm depends on the quality of evaluation function.

Properties:

- terminal states must be ordered in the same way as if ordered by the true utility function
- the computation must not take too long
- for nonterminal states, the evaluation function should be strongly correlated with the actual chances of winning
  - given the limited amount of computation, the best the algorithm can do is make a guess about the final outcome

- How to construct such a function?

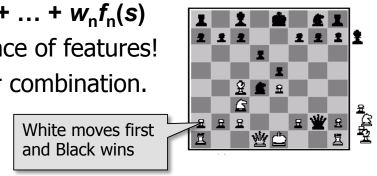
Evaluation function - examples

Expected value

- based on selected features of states, we can define various *categories* (equivalence classes) of states
- each category is evaluated based on the proportion of winning and losing states
  - EVAL = $(0.72 \times +1) + (0.20 \times -1) + (0.08 \times 0) = 0.52$

„Material" value

- estimate the numerical contribution of each feature
  - chess: pawn = 1, knight = bishop= 3, rook = 5, queen = 9
- combine the contributions (e.g. weighted sum)
  - **EVAL(s) = $w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$**
  - The sum assumes independence of features!
  - It is possible to use non-linear combination.



Problems with cut off

- The situation may change dramatically by assuming one more move after the cut-off limit.

Identical material value (better for Black) for both states,
but **White wins the right position** by capturing the queen.

  - **quiescent**: if the opponent can capture a chess-man then the estimate is not stable and it is better to explore a few more moves (for example only selected moves)
- **horizon effect**
  - the unavoidable bad situation can be delayed after the cut-off limit (horizon) and hence it is not recognized as a bad state

Black has a better material value, but if White changes a pawn to a queen, then **White wins**.
Black may consider checking the white king so the situation does not look so bad.

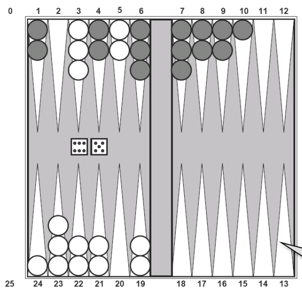
Possible improvements

- **Singular extension**
  - explore the sequence of moves that are "clearly better" than all other moves
  - a fast way to explore the area after the depth limit (quiescent is a special case)
- **Forward pruning**
  - some moves at a given state are not assumed at all (a human approach)
  - dangerous as it can miss the optimal strategy
  - safe, if symmetric moves are pruned
- **Transposition tables**
  - similarly to classical search, we can remember already evaluated states for the case when they are reached again by a different sequence of moves

Stochastic games

- In real life, many unpredictable external events can put us into unforeseen situations.
- Games mirror **unpredictability** by including a random element, such as throwing of dice.

Backgammon

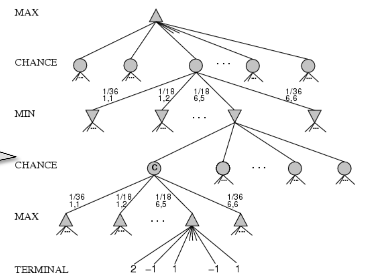


- the goal is to move all one's pieces off the board (clockwise)
- who finishes first, wins
- dice are rolled to determine the legal moves
  - the total travelled distance

There are four legal moves for White:
(5-10,5-11), (5-11,19-24), (5-10,10-16), (5-11,11-16)

Playing stochastic games

- Game tree is extended with **chance nodes** (in addition to MAX and MIN nodes) describing all rolls of dice.
  - 36 results for two dice,
    21 without symmetries (5-6 and 6-5)
  - chance for double is 1/36,
    other results 1/18



Chance nodes are added to each layer, where the move is influenced by randomness.
MAX rolls the dice here.

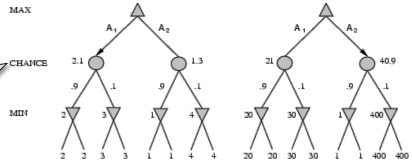
- Instead of the MINIMAX value, we use **expected MINIMAX value** (based on probability of chance actions):
  EXPECTIMINIMAX-VALUE(n)=

UTILITY(n)	if n is a terminal node
$\max_{s \in successors(n)}$ EXPECTMINIMAX-VALUE(s)	if MAX plays in n
$\min_{s \in successors(n)}$ EXPECTMINIMAX-VALUE(s)	if MIN plays in n
$\sum_{s \in successors(n)} P(s)$. EXPECTMINIMAX(s)	if n is a chance node

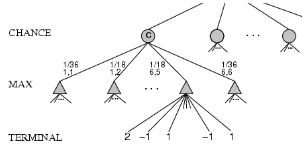
Stochastic games - discussion

- **Beware of the evaluation function** (for cut-off)
  - the absolute value of nodes may play a role
  - the values should be a linear transformation of expected utility in the node

The left tree is better for A_1 while the right tree is better for A_2, though the order of nodes is identical.



- **Time complexity** $O(b^m n^m)$, where n is the number of random moves
  - it is not realistic to reach a bigger depth especially for larger random branching
- **Using cut-off à la α-β**
  - we can cut-off the chance nodes if the evaluation function is bounded
  - the expected value can be bounded when the value is not yet computed



Card Games

- Card games may look like the stochastic games, but the dice are rolled just once at the beginning!
- Card games are an example of games with **partial observability** (we do not see opponent's cards).

Example: card game "higher takes" with open cards

Situation 1: MAX: ♥6 ♦6 ♣9 8 MIN: ♥4 ♠2 ♣10 5

1. MAX gives ♣9, MIN confirms colour ♣10 — MIN wins
2. MIN gives ♠2, MAX gives ♦6 — MIN wins
3. MAX gives ♥6, MIN confirms colour ♥4 — MAX wins
4. MIN gives♣5, MAX confirms colour ♣8 — MAX wins

– ♣9 is the optimal first move for MAX

Situation 2: MAX: ♥6 ♦6 ♣9 8 MIN: ♦4 ♠2 ♣10 5

– a symmetric case, ♣9 is again the optimal first move for MAX

Situation 3: MIN hides the first card (♥4 or ♦4), what is the optimal first move for MAX now?

– Independently of ♥4 and ♦4 the optimal first move was ♣9, so it is the first optimal move now too.

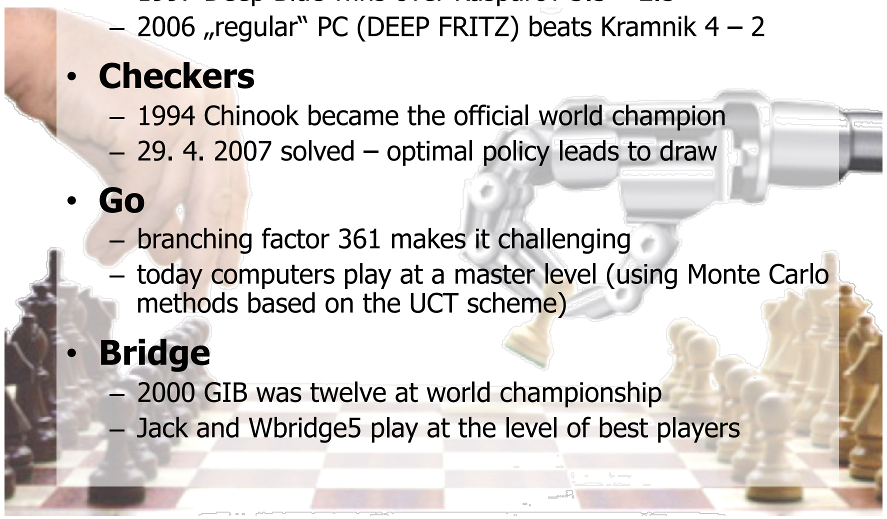
– **Really?**

Incomplete information

Example: how to become rich (a different view of cards)

- **Situation 1**: Trail A leads to a gold pile while trail B leads to a road-fork. Go left and there is a mound of diamonds, but go right and a bus will kill you (diamonds are more valuable than gold). Where to go?
  - the best choice is B and left
- **Situation 2**: Trail A leads to a gold pile while trail B leads to a road-fork. Go right and there is a mound of diamonds, but go left and a bus will kill you. Where to go?
  - B a right
- **Situation 3**: Trail A leads to a gold pile while trail B leads to a road-fork. Select the correct side and you will reach a mound of diamonds, but select a wrong side and a bus will kill you. Where to go?
  - a reasonable agent (not risking the death;-) goes A
- This is the same case as in the previous slide. We do not know what happens at the road-fork B. In the card game, we do not know which card (♥4 or ♦4) the opponent has, 50% chance of failure.
- **Lesson learnt:** We need to assume information that we will have at a given state (the problem of using ♣9 is that MAX plays differently when all cards are visible).

Computer games– the state of the art



- **Chees**
  - 1997 Deep Blue wins over Kasparov 3.5 – 2.5
  - 2006 „regular" PC (DEEP FRITZ) beats Kramnik 4 – 2
- **Checkers**
  - 1994 Chinook became the official world champion
  - 29. 4. 2007 solved – optimal policy leads to draw
- **Go**
  - branching factor 361 makes it challenging
  - today computers play at a master level (using Monte Carlo methods based on the UCT scheme)
- **Bridge**
  - 2000 GIB was twelve at world championship
  - Jack and Wbridge5 play at the level of best players