# Sztuczna Inteligencja

Soma Dutta

Wydział Matematyki i Informatyki, UWM w Olsztynie
soma.dutta@matman.uwm.edu.pl

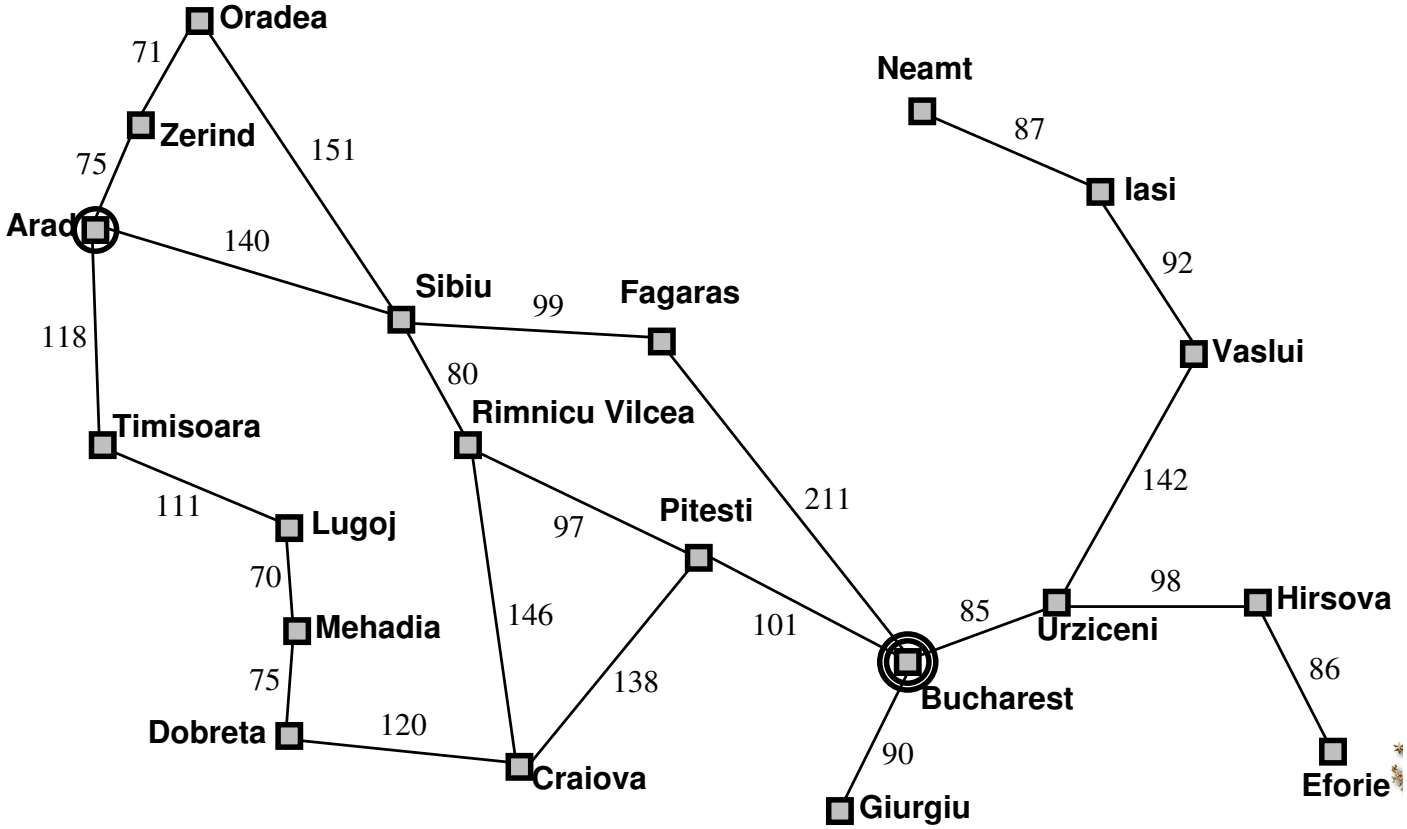
Wykład - 3: Heurystyczne algorytmy przeszukiwania i gry

Semestr letni 2022

# Przeszukiwanie heurystyczne

Ogólne podejście nazywa się 'best-first search'.

- Jest podobne do przeszukiwania 'uniform cost search'.
- Algorytm przeszukiwania grafu, w którym węzeł jest wybierany do ekspansji na podstawie funkcji oceny.
- Evaluation function (Funkcja oceny): $f(n)$ - oszacowanie kosztów dotarcia do węzła celu.
- Różne definicje dla $f$ dają różne strategie.
- Często $f(n)$ zawiera jako składnik funkcję heurystyczną (heuristic function) $h(n)$.
- Przy wykorzystaniu funkcji heurystycznej, kodowana jest w algorytmie przeszukiwania pewna dodatkowa wiedza na temat problemu.


Oradea
71
Neamt
87
Zerind
151
75
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
97
Pitesti
211
Lugoj
98
70
85
Hirsova
Mehadia
146
101
Urziceni
75
86
138
Bucharest
Dobreta
120
90
Craiova
Giurgiu
Eforie

# Greedy best-first search

- Wybiera rozwinięcie tego węzła, który jest najbliżej węzła celu bo próbuje szybko znaleźć rozwiązanie.
- Na przykład dla problemu wyznaczania trasy jako $h$ traktuje heurystyczną odległość w linii prostej (straight-line distance $h_{SLD}$).
- Nieoptymalny: Wybiera minimalną liczbę kroków do osiągnięcia celu. Może to nie prowadzić do minimalnych kosztów.
- Niekompletny: Ponieważ wybiera tylko węzeł najbliżej węzła docelowego, jeśli sam węzeł jest ślepym zaułkiem, algorytm nigdy nie osiągnie celu.
- W najgorszym przypadku złożoność czasowa i pamięciowa dla wersji drzewa jest $O(b^m)$.

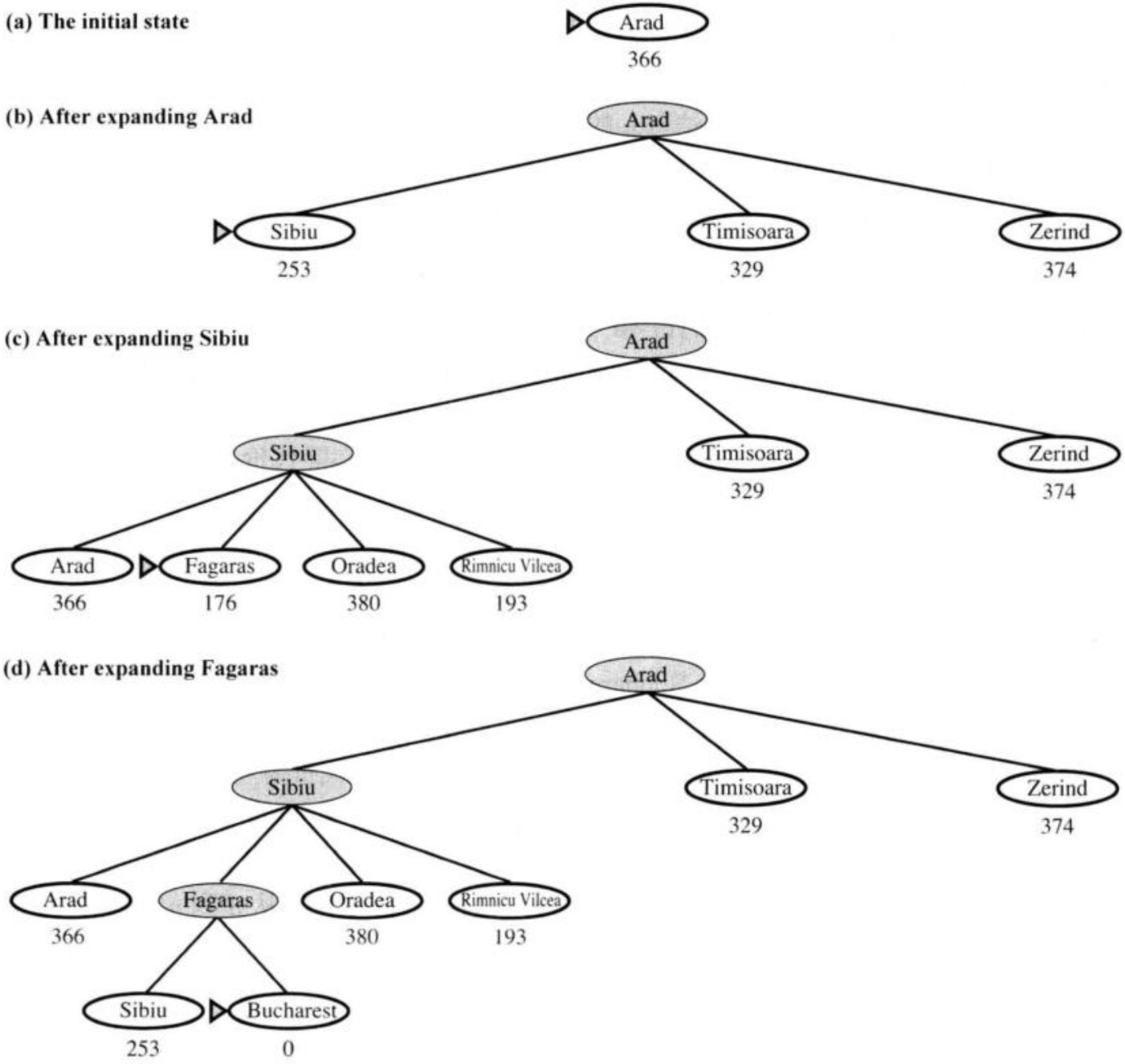
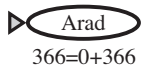



图 3.23　使用直线距离启发式 $h_{SLD}$ 的贪婪最佳优先树搜索。结点上都标明了该结点的 $h$ 值

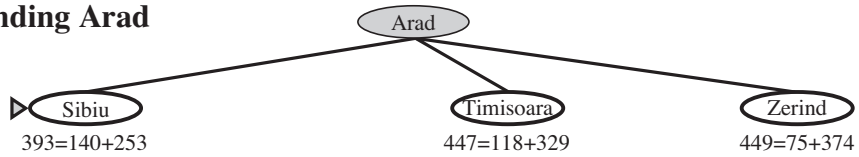
# Przeszukiwanie $A^*$

- Bazuje na oszacowaniu minimalnego całkowitego kosztu rozwiązania (minimum estimated solution cost)
- $f(n) = g(n) + h(n)$,
  gdzie $g(n)$ = koszt dotarcia do węzła $n$ i $h(n)$ = szacowany koszt dotarcia do węzła docelowego od $n$.
- Jest kompletny i optymalny pod warunkiem, że $h$ spełnia określone warunki.
  - (Admissible) (Warunek dopuszczalności): $h$ powinno być dopuszczalne, to znaczy być taką funkcją, która nigdy nie zawyża kosztów osiągnięcia celu. (np. $h_{SLD}$)
  - (Consistency/Monotonicity) (Warunek niesprzeczności / monotoniczności): Ten warunek dotyczy tylko przeszukiwania grafów. $h$ jest niesprzeczny, jeśli dla każdego węzła $n$ i każdego następnika $n'$ wygenerowanego z $n$ przez akcję $a$, mamy $h(n) \leq c(n, a, n') + h(n')$.
  - Twierdzenie: Każda niesprzeczna heurystyka jest dopuszczalna.
  - $A^*$ użyte w przeszukiwaniu drzewa jest optymalne, jeśli $h(n)$ jest dopuszczalne, a $A^*$ użyte w przeszukiwaniu grafów jest optymalne, jeśli $h(n)$ jest niesprzeczna.

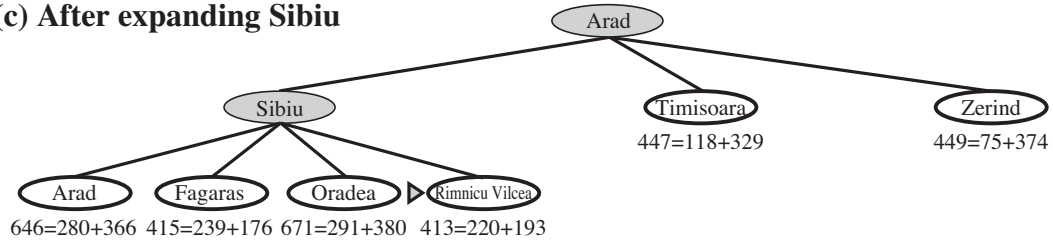
**(a) The initial state**



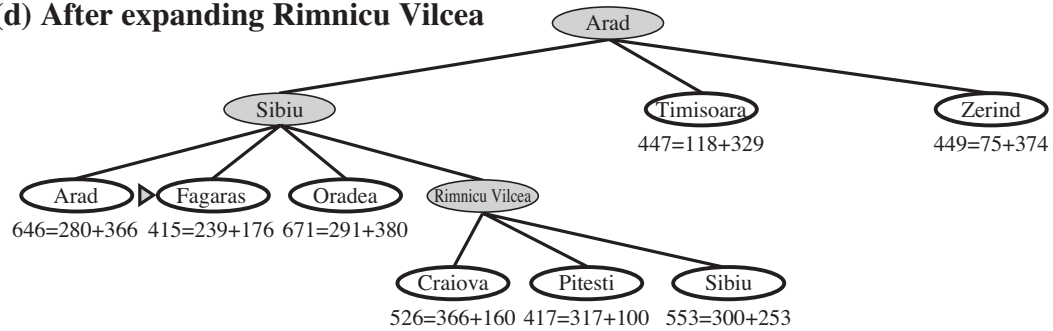

**(b) After expanding Arad**



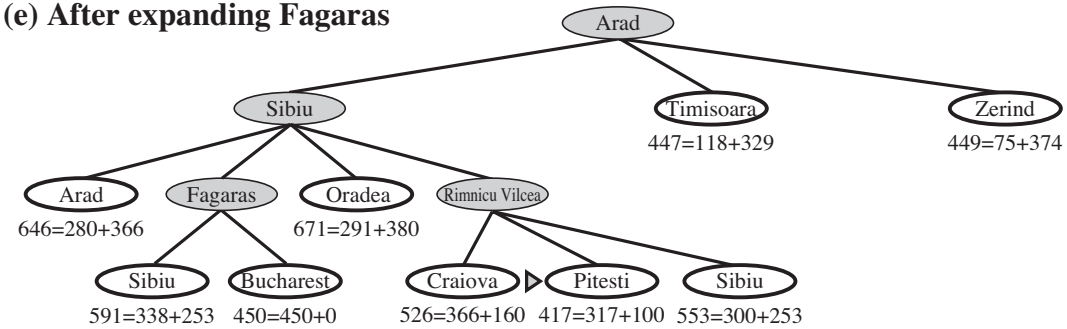

**(c) After expanding Sibiu**




**(d) After expanding Rimnicu Vilcea**




**(e) After expanding Fagaras**



# Optymalna wydajność (optymalna efektywność)

- $A^*$ jest optymalnie wydajny/efektywny (optimally efficient) - oznacza to, że dla każdej spójnej funkcji heurystycznej nie ma innego optymalnego algorytmu, który mógłby rozwinąć mniej węzłów niż $A^*$.
- Złożoność $A^*$ często sprawia, że $A^*$ nie jest stosowalny w praktyce.
- Niektóre warianty $A^*$ koncentrują się tylko na znalezieniu szybkiego rozwiązania nieoptymalnego lub wykorzystują dokładniejsze, ale niedopuszczalne, funkcje heurystyczne.

# Przeszukiwanie heurystyczne z ograniczona pamięcią

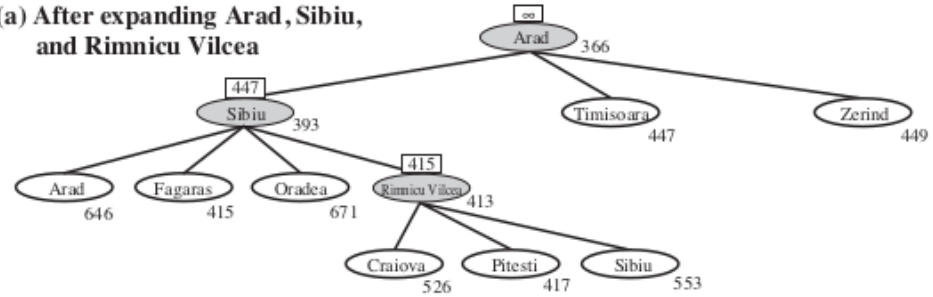
## Memory-bounded heuristic search

- Iterative deepening $A^*$ (IDA$^*$): Stosowane jest ograniczenie kosztu f, a ograniczenie dla każdej iteracji jest najmniejszym kosztem f osiąganym na węzłach, który to koszt przekracza limit poprzedniej iteracji.
- rozwiązanie to jest praktyczne dla wielu problemów z kosztami jednostkowymi i pozwala uniknąć znacznych kosztów ogólnych związanych z utrzymywanie posortowanej kolejki węzłów.

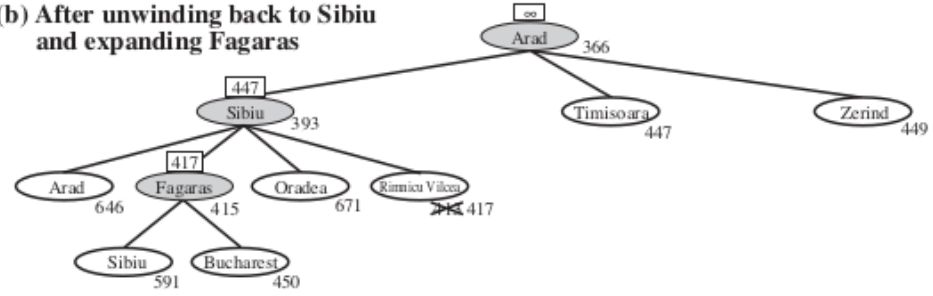
## Recursive best-first search (RBFS):

- Jest podobny do 'recursive depth-first search'. Używa zmiennej 'f-limit' do zachowania śladu wartości f dla najlepszej alternatywnej ścieżki dostępnej od dowolnego przodka obecnego węzła.
- dla aktualnego węzła przekroczona jeśli zostanie to ograniczenie (limit), to rekurencja rozwija się ponownie dla alternatywnej ścieżki. Gdy rekurencja się rozwija, RBFS zastępuje f-wartość każdego węzła na ściezce przez najlepszą wartość f swoich dzieci. Tak więc pamiętana jest f-wartość najlepszego liścia w zapamiętanym poddrzewie i może w razie potrzeby zdecydować o ponownym rozwinięciu.

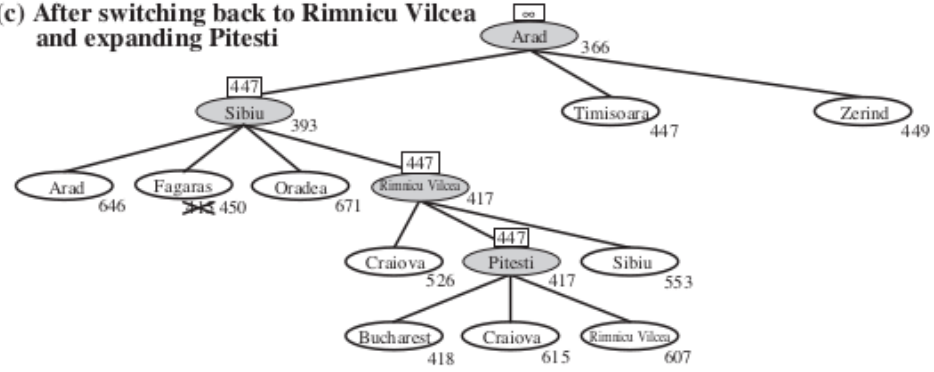
**(a) After expanding Arad, Sibiu, and Rimnicu Vilcea**




**(b) After unwinding back to Sibiu and expanding Fagaras**

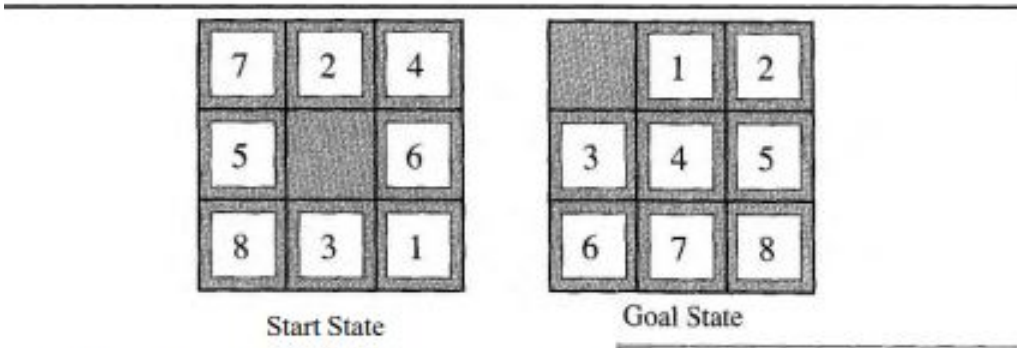



**(c) After switching back to Rimnicu Vilcea and expanding Pitesti**



- Podobnie jak przeszukiwanie drzewa $A^*$, RBFS jest optymalnym algorytmem, o ile tylko jest to funkcja heurystyczna $h(n)$ dopuszczalna. Jego złożoność pamięciowa jest liniowa względem głębokości najgłębszego optymalnego rozwiązania.
- Zarówno IDA$^*$, jak i RBFS zużywają bardzo mało pamięci. Pomiędzy iteracjami IDA$^*$ trzeba zachować tylko jedną liczbę: aktualny limit kosztów f. RBFS przechowuje więcej informacji w pamięci, ale wykorzystuje tylko przestrzeń liniową: nawet jeśli dostępna jest większa pamięć, RBFS nie używa jej.

# Funkcje heurystyczne




- $h_1$ = liczba płytek na niewłaściwym miejscu
  (Na rysunku wszystkie osiem płytek jest na niewłaściwym pozycji, więc stan początkowy miałby $h_1 = 8$.)
- $h_2$ = suma odległości płytek od ich pozycji celowych. Ponieważ płytki nie mogą poruszać się po przekątnych, odległość, którą policzymy, jest sumą odległości poziomie i pionowie. Jest to czasami nazywane odległością miejską (city block distance) lub odległością Manhattanu (Manhattan distance).
  (Na rysunku $h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$.)

# Poza klasycznym przeszukiwaniem

- Algorytmy omówione do tej pory dotyczą tych problemów, które są obserwowalne, deterministyczne i gdzie możemy zorganizować systematyczne przeszukiwanie, które w rezultacie zwraca sekwencję działań, tzn. rozwiązanie.
- Co dzieje się, gdy problem nie jest w pełni obserwowalny i deterministyczny? Co dzieje się w sytuacjach, gdy agent nie jest w stanie dokładnie przewidzieć, jakie percepty otrzyma?
- Omówimy niektóre algorytmy, które zamiast systematycznego przeszukiwania ścieżek od stanu początkowego, przeprowadzają przeszukiwanie lokalne w przestrzeni stanów.

# Algorytm przeszukiwania lokalnego

- Jeśli ścieżka do celu nie ma znaczenia, możemy rozważyć inną klasę algorytmów.
- Lokalne algorytmy przeszukiwania działają przy użyciu pojedynczego bieżącego węzła i zazwyczaj przenoszą się tylko do sąsiadów tego węzła. Zazwyczaj ścieżki, po których następuje przeszukiwanie, nie są zapamiętywane.
- Lokalny algorytmy przeszukiwania nie są systematyczne. Ale mają dwie kluczowe zalety:
  (1) używają bardzo mało pamięci - zwykle stały rozmiar pamięci, oraz
  (2) często potrafią znaleźć rozsądne rozwiązania w dużych lub nieskończnych przestrzeniach stanów, dla których algorytmy systematycznego przerszukiwania nie są odpowiednie.
- Oprócz poszukiwania celów, lokalne algorytmy przeszukiwania są przydatne do rozwiązywania problemów optymalizacji (optimization problems), w których celem jest znalezienie najlepszego rozwiązania zgodnie z celem działania (objective function).

# Przestrzeń stanów vs. krajobraz przestrzeni stanów

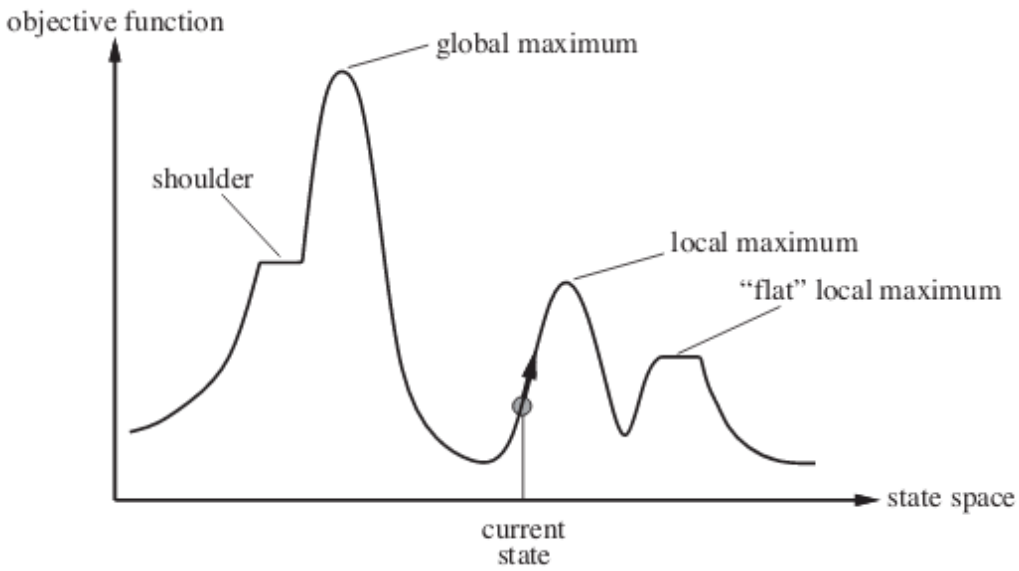
- Krajobraz ma zarówno lokalizację (localization) (zdefiniowaną przez stany), jak i wysokość (elevation) (zdefiniowaną przez wartość heurystycznej funkcji kosztu lub funkcji celu).
- Jeśli wysokość odpowiada kosztowi, wtedy celem jest znalezienie najniższej doliny - globalnego minimum; jeśli wysokość odpowiada do funkcji celu, wówczas celem jest znalezienie najwyższego szczytu - globalnego maksimum.

# Hill cimbing search

- To jest po prostu pętla, w której nieustannie posuwamy się w kierunku rosnącej wartości - to znaczy, pod górę.
- Ten proces kończy się, gdy osiągnięty zostanie szczyt, w którym żaden sąsiad nie ma większej wartości.
- Algorytm nie wymaga pamiętania drzewa przeszukiwania, więc struktura danych dla bieżącego węzła musi tylko rejestrować stan i wartość funkcji celu.
- Wspinaczka na wzgórze nie eksploruje poza bezpośrednich sąsiadów obecnego stanu.
- Wspinaczka jest czasem nazywana zachłannym lokalnym przeszukiwaniem (greedy local search), ponieważ wybiera dobrego sąsiada bez zastanowiania się, dokąd pójść.

# Pseudocode for hill climbing search

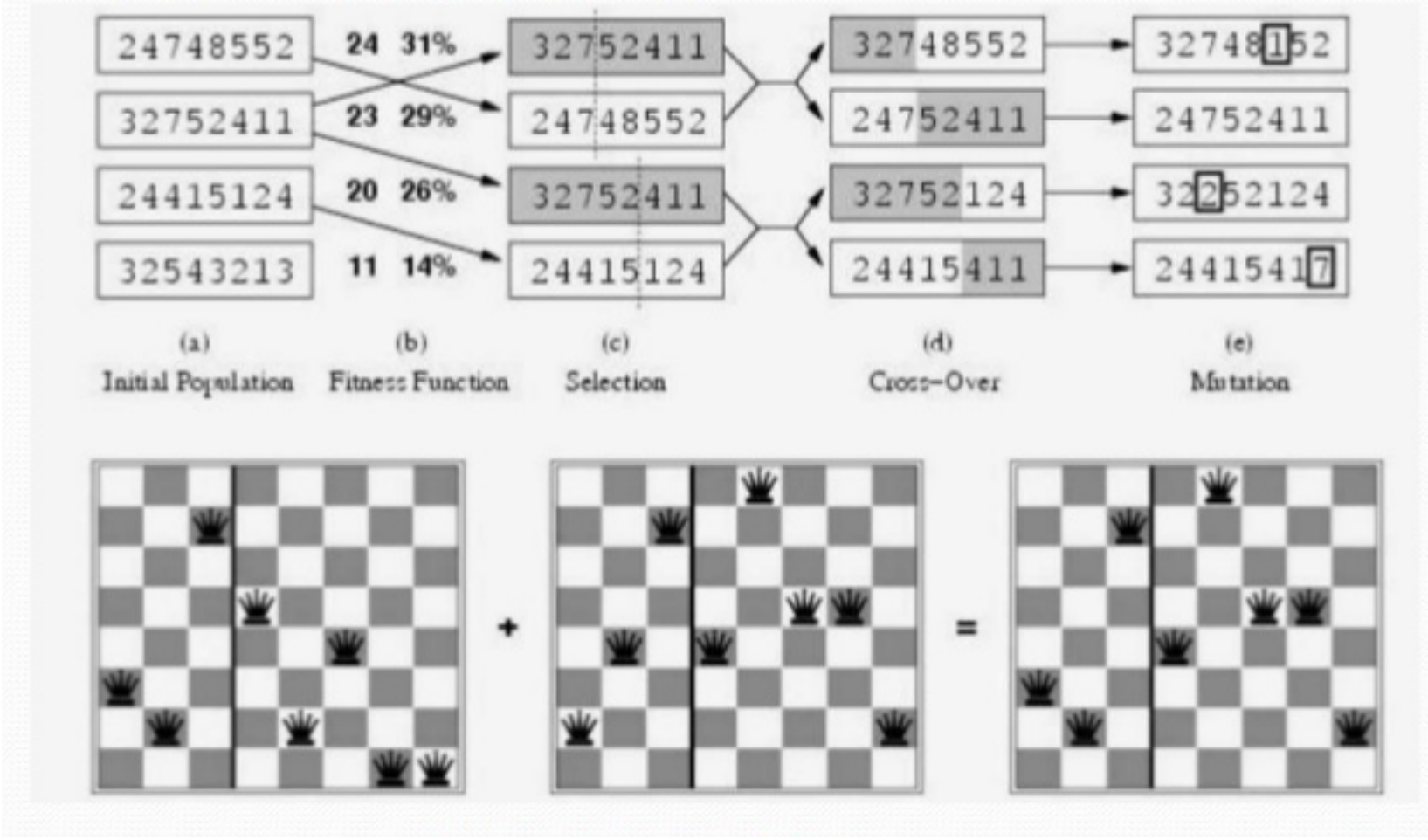
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
   current ← MAKE-NODE(problem.INITIAL-STATE)
   loop do
      neighbor ← a highest-valued successor of current
      if neighbor.VALUE ≤ current.VALUE then return current.STATE
      current ← neighbour
```



# Genetic algorithm

- W algorytmie genetycznym stany następne są generowane z dwóch stanów nadrzędnych.
- Zaczyna się od zestawu k losowo wygenerowanych stanów, zwanych populacją (population). Każdy stan jest reprezentowany jako ciąg znaków (chromosom) nad skończonym alfabetem - w większości przypadków jako ciąg zer i jedynek.
- Każdy stan jest oceniany przez funkcję celu lub (w terminologii GA) funkcję dopasowania (fitness function).
- Dwie pary wybiera się losowo do reprodukcji, zgodnie z wartością wygenerowaną na podstawie funkcji dopasowania.
- crossover point Dla każdej pary, która ma być krzyżowana, wybierany jest losowo pozycja w chromosomach, od której następuje krzyżowanie.
- Potomstwo jest tworzone przez skrzyżowanie wybranych ciągów (chromosomów) w zgłedem punktu podziału (crossover point).

# 8 Queens problem: genetic search algorithm



# Środowisko wieloagentowe

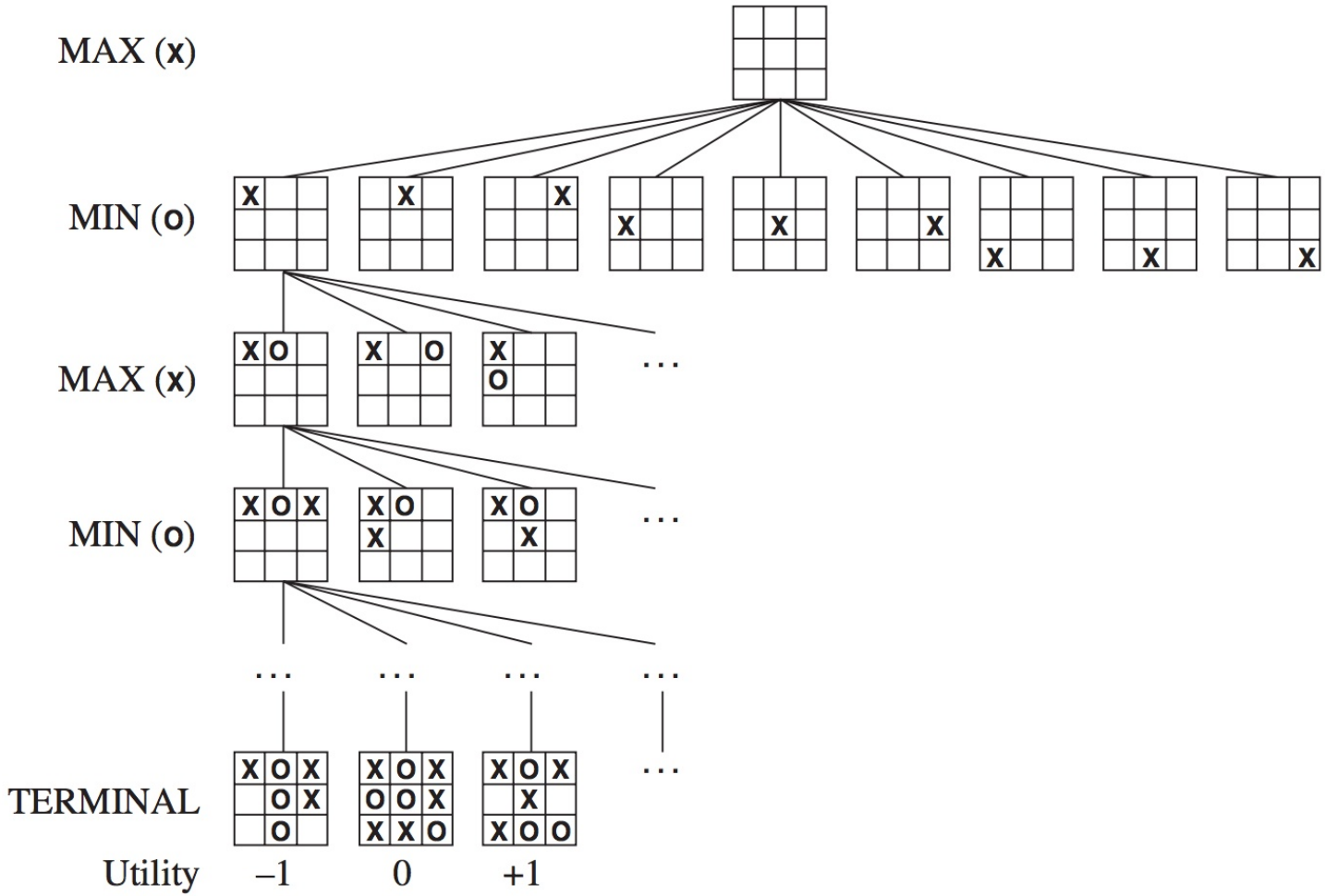
- W sytuacjach wieloagentowych agent często musi brać pod uwagę działania innych agentów i ich wpływ na jego cele.
- W środowisku konkurencyjnym agenci mają sprzeczne cele.
- Powoduje to problemy z przeszukiwaniem przeciwnym (adversial search problems) - często znane jako gry (games).
- Teoria gier matematycznych jest gałęzią ekonomii i często rozważa się ją w środowiskiem wieloagentowym. W sztucznej inteligencji najbardziej popularne są gry specjalne, zwane deterministycznymi (deterministic), dwuosobowymi (two-player games), gry z zerowymi sumami doskonałych informacji (zero-sum games of perfect information) (np. Szachy)

# Gra dwóch graczy jako problem z przeszukiwaniem

- Nazwijmy dwóch graczy MAX i MIN.
- MAX najpierw wykonuje ruch, a potem grają naprzemiennie do końca gry.
- Na koniec punkty są przyznawane wygrywającemu graczowi, a kary dla przegranego.
- Na przykład: Tic-Tac-Toe (kółko i krzyżyk):
  - Każdy gracz powinien umieścić X lub O na jednym polu z 9 dostępnych pól, tworząc kwadrat 3 na 3.
  - Gra naprzemiennie pomiędzy MAX umieszczeniem X i MIN umieszczeniem O, dopóki gra się nie skończy.
  - Gra kończy się, gdy jeden gracz otrzyma trzy X (lub O) z rzędu lub wszystkie pola zostaną wypełnione.
- Stan początkowy, funkcja ACTION i funkcja RESULT określają drzewo gry - drzewo, w którym węzły to stany gry, a krawędź to ruchy.

# Drzewo gry (Tic-Tac-Toe)

- Liczby w węzłach liści wskazują wartość użyteczną stanu końcowego z punktu widzenia MAX.



# Drzewo przeszukiwania

- W przypadku kółko i krzyżyk drzewo gry jest stosunkowo małe - mniej niż 9! = 362,880 węzłów końcowych.
- Jednak realizacja pełnego drzewa gry jest praktycznie trudna.
- Zamiast pełnego drzewa gry używamy pojęcia zwanego drzewem przeszukiwania
- Drzewo przeszukiwania zawiera wystarczającą liczbę węzłów do zbadania ruchów gracza.

# Gra o stałej sumie

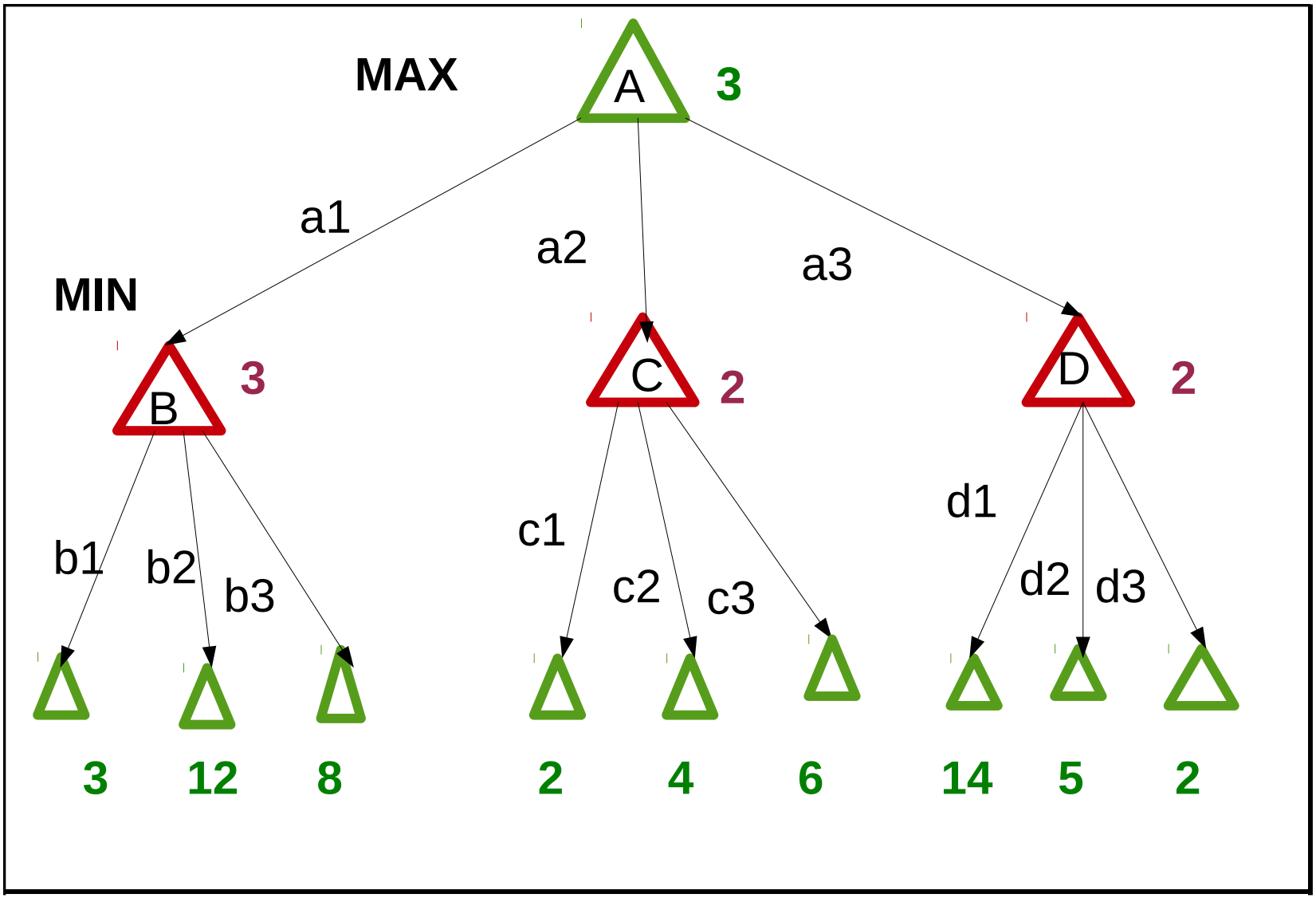
- Gra o stałej sumie to taka, w której łączna wypłata wszystkich graczy jest taka sama.
- Szachy to gra o sumie stałej, ponieważ każde jej wystąpienie ma całkowitą wypłatę 0+1 lub 1+0 lub $\frac{1}{2} + \frac{1}{2}$.
- Gry o sumie zerowej to te, w których łączna wypłata wszystkich graczy wynosi 0.

# Optymalne rozwiązanie

- W przypadku gry modelowanej za pomocą wielu agentów pojęcie optymalnej strategii dla danego agenta zależy od wyborów przez innych agentów.
- W normalnym problemie z przeszukiwaniem (optymalnym) rozwiązaniem jest sekwencja działań zaczynająca się od węzła początkowego do węzła końcowego. W przypadku przeszukiwania przeciwnego, dla każdej akcji MAX, MIN ma ruch. Tak więc MAX musi mieć strategię warunkową, która określa ruchy MAX w stanie początkowym i we wszystkich możliwych stanach wynikających z ruchów MIN.

# Na przykład: Trywialna gra

- Zarówno MAX, jak i MIN mają tylko jedną turę



- Aby sformułować gry jako problem przeszukiwania, potrzebujemy następujących elementów:
  - Initial State (Stan początkowy): określa początkową konfigurację gry
  - PLAYER(s): Oznacza, który gracz ma ruch w stanie s
  - ACTION(s): Zwraca zbiór czynności poprawnych (ruchów) w stanie s
  - RESULT(s, a): Relacja przejścia, która określa wynik działania *a* lub ruchu w stanie s
  - TERMINAL-TEST(s): Test zwraca wartość prawda, gdy gra się skończy, a fałsz w przeciwnym razie. Stany, w których gra się kończy, nazywane są stanami końcowymi (terminal states).
  - UTILITY(s, p): Funkcja użyteczności (funkcja celu lub funkcja wypłaty) (objective function or payoff function) określa ostateczną wartość liczbową dla gry, która kończy się w stanie terminalnym s dla gracza p. (np. w szachach wyniki to wygrana (1), strata (0) lub remis ($\frac{1}{2}$).

# Utility function

- Funkcja użyteczności to odwzorowanie stanów świata na liczby rzeczywiste. Te liczby są interpretowane jako miary poziomu zadowolenia agenta w danym stanie. Kiedy agent nie jest pewien, z jakim stanem świata ma do czynienia, jego użyteczność jest definiowana jako oczekiwana wartość jego funkcji użyteczności w odniesieniu do odpowiedniego rozkładu prawdopodobieństwa stanów.
- Krotka $(N, A, u)$ to gra o normalnej formie (skończona, $n$-osobowa) gdzie:
  - $N$ jest skończonym zbiórem graczy $n$, indeksowanych według $i$;
  - $A = A_1 \times A_2 \times \ldots \times A_n$, gdzie $A_i$ to skończony zbiór działań dostępnych dla gracza $i$. Każdy wektor $a = (a_1, a_2, \ldots, a_n) \in A$ jest nazywany profilem działania.
  - $u = (u_1, u_2, \ldots, u_n)$ gdzie każdy $u_i : A \mapsto R$ to funkcja użyteczności (lub funkcja wypłaty) dla każdego gracza $i$.
- W przypadku gry dwuosobowej $N = 2$.

# Strategia

- Biorąc pod uwagę zbiór dostępnych działań agenta MAX (MIN), jedną ze strategii może być polegać na wybieraniu deterministycznie jednej akcji. Taka strategia nazywa się czystą strategią (pure strategy). Wybór czystej strategii nazywamy dla każdego agenta profilem czystej strategii (pure strategy profile).
- Agenci mogą również przypisywać prawdopodobieństwa (na podstawie swoich preferencji) akcji ze zbioru dostępnych akcji. Tak więc każda strategia agenta jest przypisaniem z pewnym prawdopodobieństwem. Taka strategia nazywa się mieszaną strategią (mixed strategy).
- W przypadku czystych strategii mówiliśmy o funkcji użyteczności (utility function), a w kontekście mieszanych strategii mówimy o oczekiwanej użyteczności (expected utility function).

Istnieję różne sposoby znalezienia optymalnych strategii dla agentów.

- Pareto optimal
- Best response (Nash equilibrium)
- Minimax

# Minimax algorytm

- Wartość minmax węzła n, oznaczona przez $MINMAX(n)$, jest użytecznością (dla MAX) bycia w $n$.
- Ponieważ zawsze reprezentuje użyteczność dla MAX, MAX preferuje stan o maksymalnej wartości, a MIN próbuje obniżyć użyteczność MAX i dlatego wybiera stan o minimalnej wartości.

$$
\begin{aligned}
\text{MINIMAX(s)} &= \text{UTILITY(s)} && \text{jeśli TERMINAL-TEST(s),} \\
&= max_{a \in ACTION(s)} MINIMAX(RESULT(s, a)) && \text{jeśli PLAYER(s) = MAX,} \\
&= min_{a \in ACTION(s)} MINIMAX(RESULT(s, a)) && \text{jeśli PLAYER(s) = MIN,}
\end{aligned}
$$

# Pseudocode: MINIMAX

```
function MINIMAX-DECISION(state) returns an action
  returns arg max_{a∈ACTIONS(s)} MINVALUE(RESULT(state, a))

function MAXVALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MINVALUE(RESULT(s, a)))
  return v

function MINVALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAXVALUE(RESULT(s, a)))
  return v
```

Dziękuję za uwagę