

Sztuczna Inteligencja

Soma Dutta

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

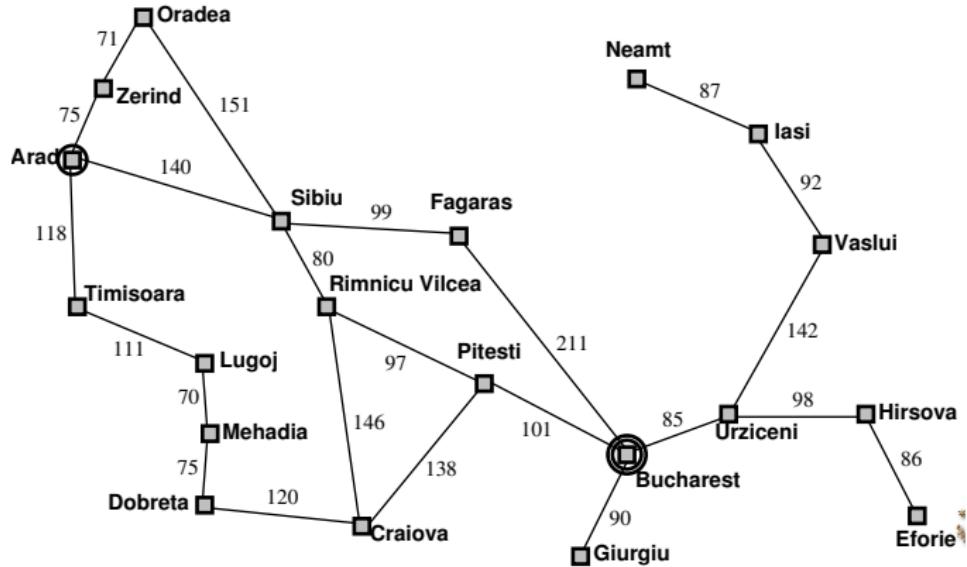
Wykład - 3: Heurystyczne algorytmy przeszukiwania i gry

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.



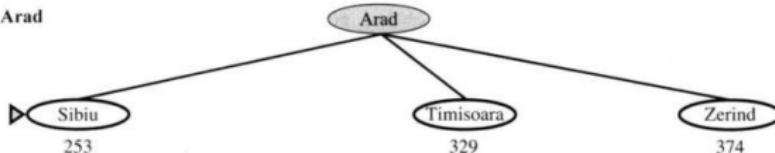
Greedy best-first search

- ▶ 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 założkiem, algorytm nigdy nie osiągnie celu.
- ▶ W najgorszym przypadku złożoność czasowa i pamięciowa dla wersji drzewa jest $O(b^m)$.

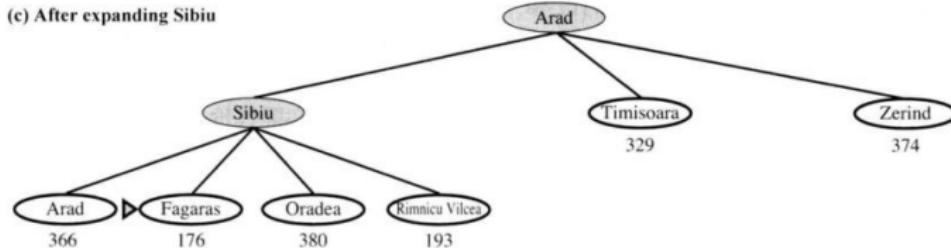
(a) The initial state



(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Fagaras

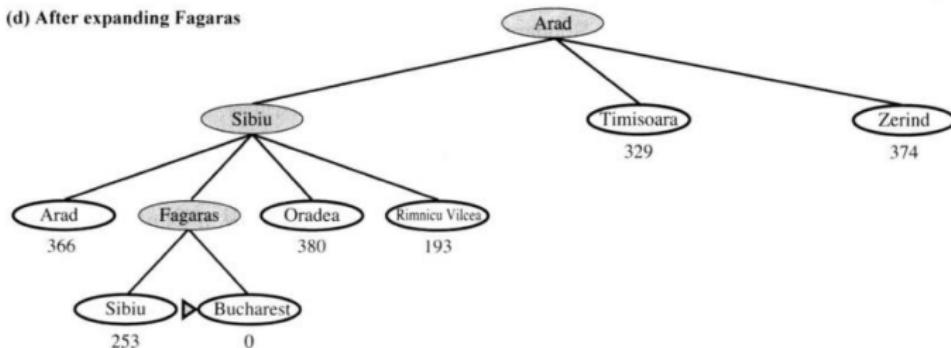


图 3.23 使用直线距离启发式 h_{SLD} 的贪婪最佳优先树搜索。结点上都标明了该结点的 h 值

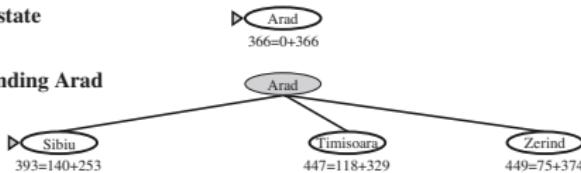
Przeszukiwanie A^*

- ▶ 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.

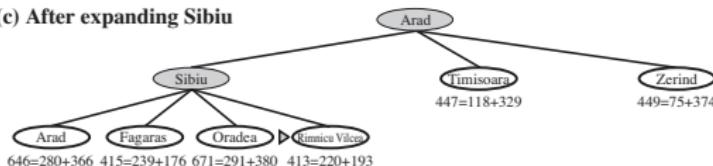
(a) The initial state



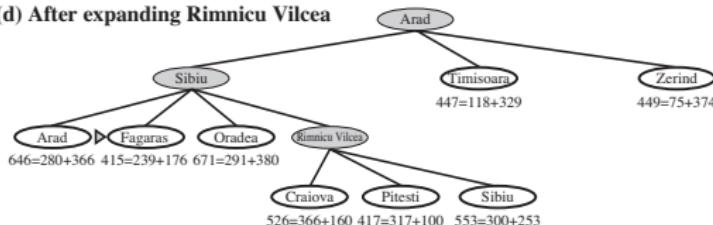
(b) After expanding Arad



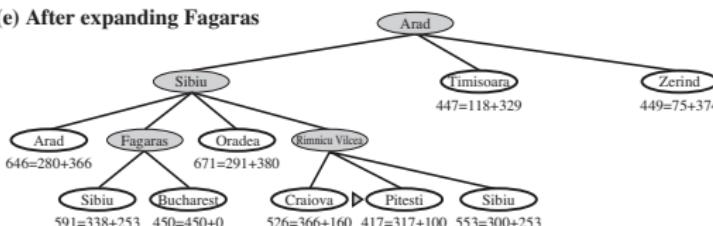
(c) After expanding Sibiu



(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 ograniczoną pamięcią

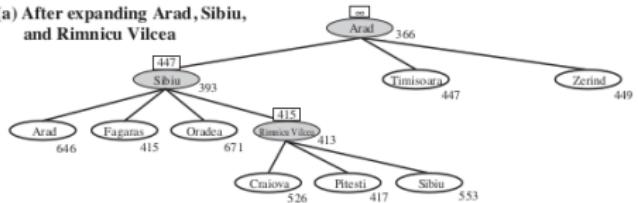
Memory-bounded heuristic search

- ▶ 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 znaczących kosztów ogólnych związanych z utrzymywaniem posortowanej kolejki węzłów.

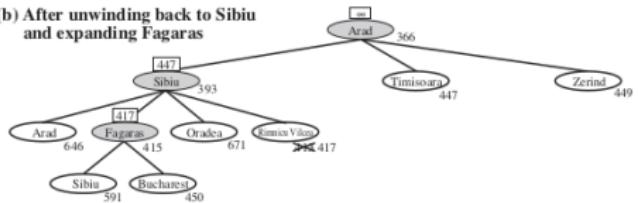
Recursive best-first search (RBFS):

- ▶ 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 przekroczone jeśli zostanie to ograniczenie (limit), to rekurencja rozwija się ponownie dla alternatywnej ścieżki. Gdy rekurencja się rozwija, RBFS zastępuje f-wartość każdego węzła na ścieżce przez najlepszą wartość f swoich dzieci. Tak więc pamiętaana jest f-wartość najlepszego liścia w zapamiętanym poddrzewie i może w razie potrzeby zdecydować o ponownym rozwinięciu.

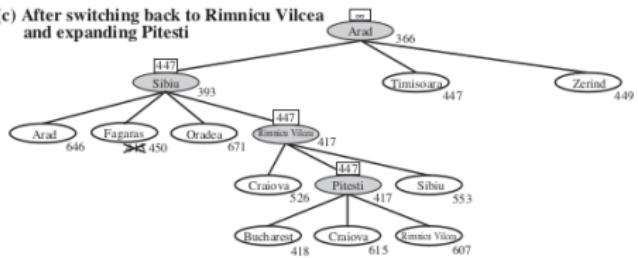
(a) After expanding Arad, Sibiu, and Rimnicu Vilcea



(b) After unwinding back to Sibiu and expanding Fagaras

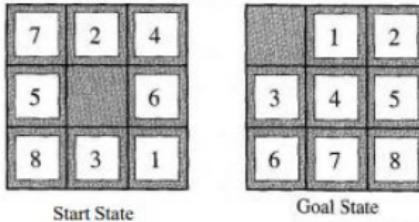


(c) After switching back to Rimnicu Vilcea and expanding Pitesti



- ▶ 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 płytaków na niewłaściwym miejscu
(Na rysunku wszystkie osiem płytaków jest na niewłaściwym pozycji, więc stan początkowy miałby $h_1 = 8$.)
- ▶ h_2 = suma odległości płytaków od ich pozycji celowych. Ponieważ płytaki nie mogą poruszać się po przekątnych, odległość, którą policzymy, jest sumą odległości poziomie i pionowe. 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 obserwowlane, deterministyczne i gdzie możemy zorganizować systematyczne przeszukiwanie, które w rezultacie zwraca sekwencję działań, tzn. rozwiązanie.
- ▶ Co dzieje się, gdy problem nie jest w pełni obserwowlany 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 nieskończonych przestrzeniach stanów, dla których algorytmy systematycznego przeszukiwania 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

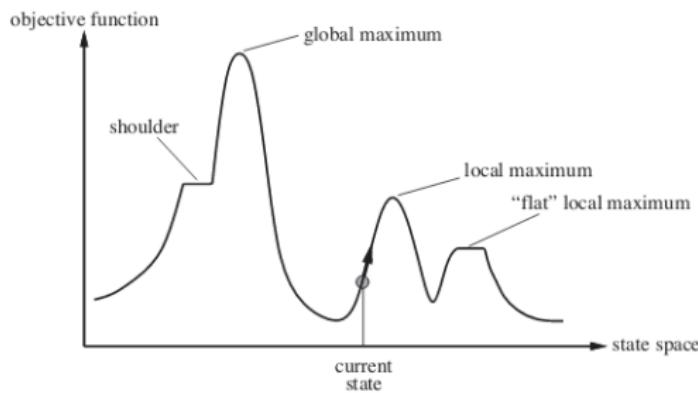
- ▶ Krajobraz ma zarówno lokalizację (localization) (zdefiniowaną przez stany), jak i wysokość (elevation) (zdefiniowaną przez wartość heurystycznej funkcji kosztu lub funkcji celu).
- ▶ Jeśli wysokość odpowiada kosztowi, wtedy celem jest znalezienie najniższej doliny - globalnego minimum; jeśli wysokość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

Hill climbing 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 wzgórzu 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

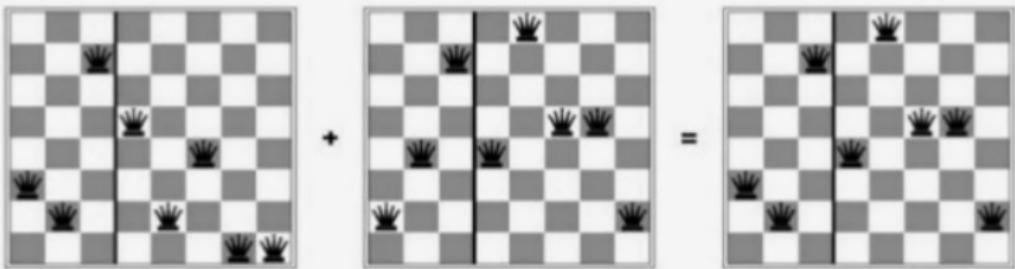
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    current ← MAKE-NODE(problem.INITIAL-STATE)
    loop do
        neighbor ← a highest-valued successor of current
        if neighbor.VALUE ≤ current.VALUE then return current.STATE
        current ← neighbour
```



Genetic algorithm

- ▶ W algorytmie genetycznym stany następne są generowane z dwóch stanów nadzędnych.
- ▶ 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 względem punktu podziału (crossover point).

8 Queens problem: genetic search algorithm



Środowisko wieloagentowe

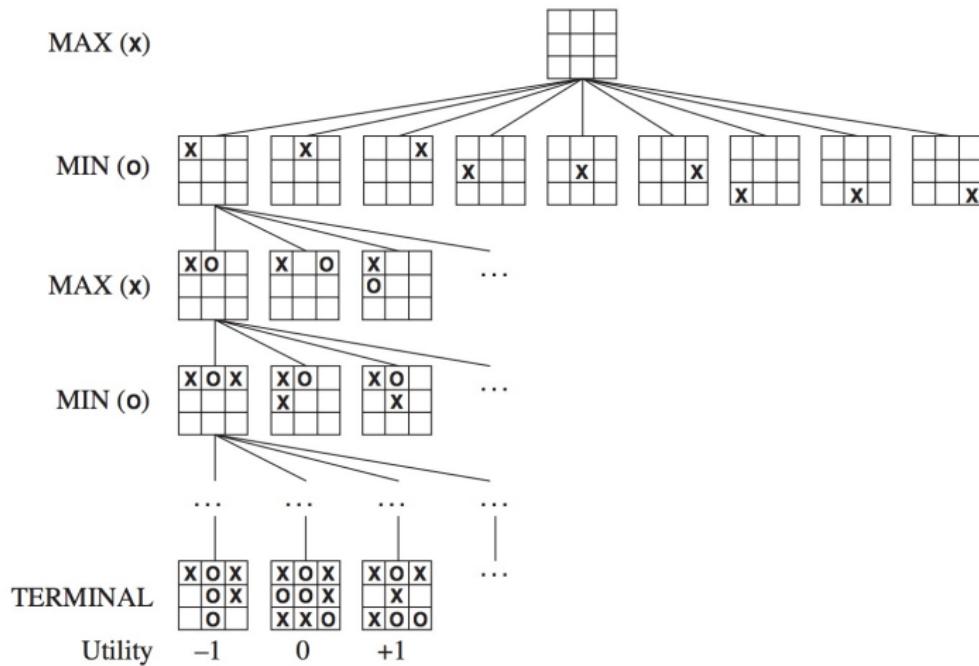
- ▶ W sytuacjach wieloagentowych agent często musi brać pod uwagę działania innych agentów i ich wpływ na jego cele.
- ▶ W środowisku konkurencyjnym agenci mają sprzeczne cele.
- ▶ Powoduje to problemy z przeszukiwaniem przeciwnym ([adversarial search problems](#)) - często znane jako [gry \(games\)](#).
- ▶ Teoria gier matematycznych jest gałęzią ekonomii i często rozważa się ją w środowisku 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

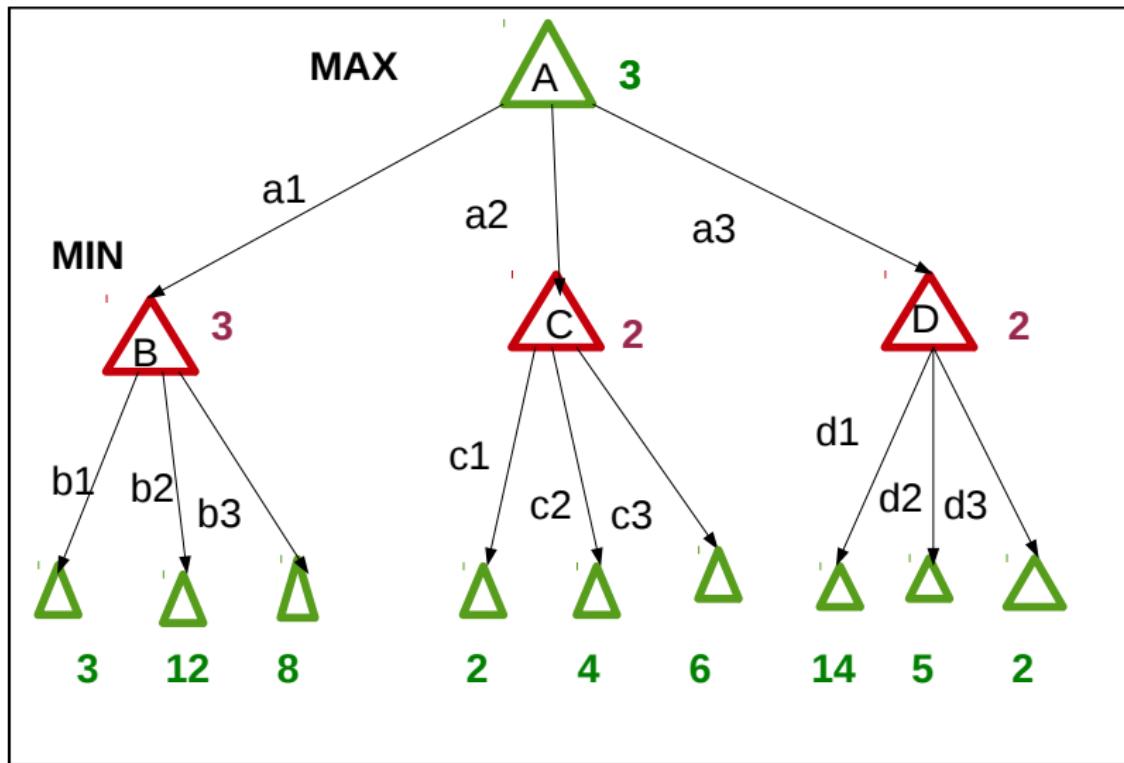
- ▶ Gra o stałej sumie to taka, w której łączna wypłata wszystkich graczy jest taka sama.
- ▶ Szachy to gra o sumie stałej, ponieważ każde jej wystąpienie ma całkowitą wypłatę $0+1$ lub $1+0$ lub $\frac{1}{2} + \frac{1}{2}$.
- ▶ Gry o sumie zerowej to te, w których łączna wypłata wszystkich graczy wynosi 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 zbiorem graczy n , indeksowanych według i ;
 - ▶ $A = A_1 \times A_2 \times \dots \times A_n$, gdzie A_i to skończony zbiór działań dostępnych dla gracza i . Każdy wektor $a = (a_1, a_2, \dots, a_n) \in A$ jest nazywany profilem działania.
 - ▶ $u = (u_1, u_2, \dots, u_n)$ 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 **mieszana 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).

Istnieją różne sposoby znalezienia optymalnych strategii dla agentów.

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

Minimax algorytm

- ▶ Wartość minmax węzła n , oznaczona przez $\text{MINMAX}(n)$, 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.

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

Pseudocode: MINIMAX

```
function MINIMAX-DECISION(state) returns an action
    returns arg  $\max_{a \in ACTIONS(s)} MINVALUE(RESULT(state, a))$ 
function MAXVALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
     $v \leftarrow -\infty$ 
    for each  $a$  in ACTIONS(state) do
         $v \leftarrow \text{MAX}(v, MINVALUE(RESULT(s, a)))$ 
    return  $v$ 
function MINVALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
     $v \leftarrow \infty$ 
    for each  $a$  in ACTIONS(state) do
         $v \leftarrow \text{MIN}(v, MAXVALUE(RESULT(s, a)))$ 
    return  $v$ 
```

Dziękuję za uwagę

Sztuczna Inteligencja

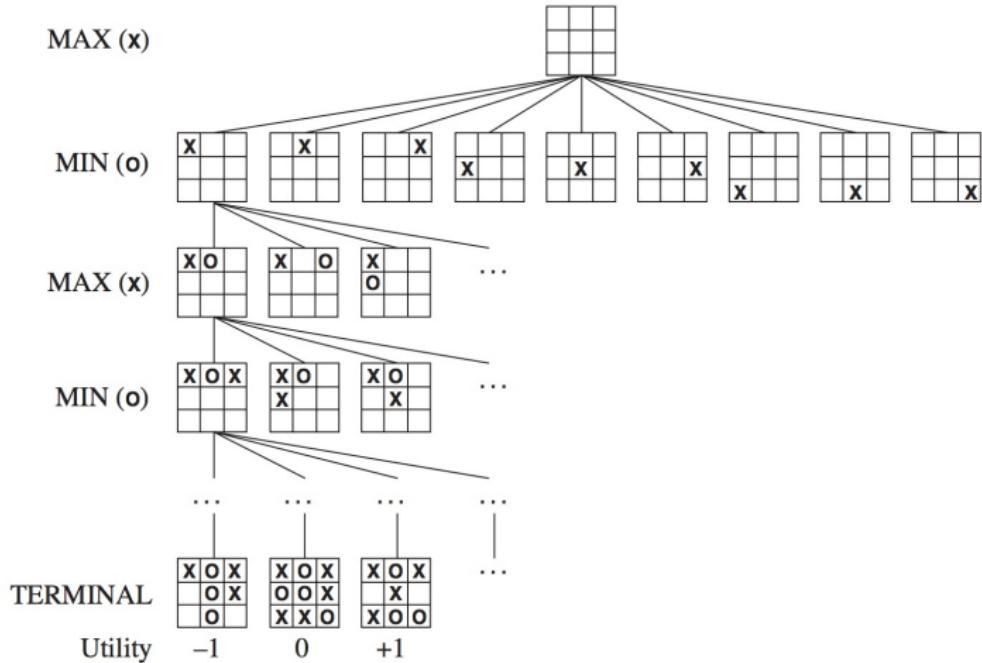
Soma Dutta

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

Wykład - 4: Gry i Algorytmy spełniające więzy

Semestr letni 2022

- ▶ Aby sformułować grę jako problem przeszukiwania, potrzebujemy następujące pojęcia:
 - ▶ **Initial State (Stan początkowy)**: określa początkową konfigurację gry
 - ▶ **PLAYER(s)**: określa, 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 postaci normalne (skończona, n -osobowa) gdzie:
 - ▶ N jest skończonym zbiorem n graczy, indeksowanych według i ;
 - ▶ $A = A_1 \times A_2 \times \dots \times A_n$, gdzie A_i to skończony zbiór działań dostępnych dla gracza i . Każdy wektor $a = (a_1, a_2, \dots, a_n) \in A$ jest nazywany profilem działania.
 - ▶ $u = (u_1, u_2, \dots, u_n)$ gdzie każdy $u_i : A \mapsto R$ to funkcja użyteczności (lub funkcja wypłaty) dla każdego gracza i .
- ▶ W przypadku gry dwuosobowej $N = 2$.

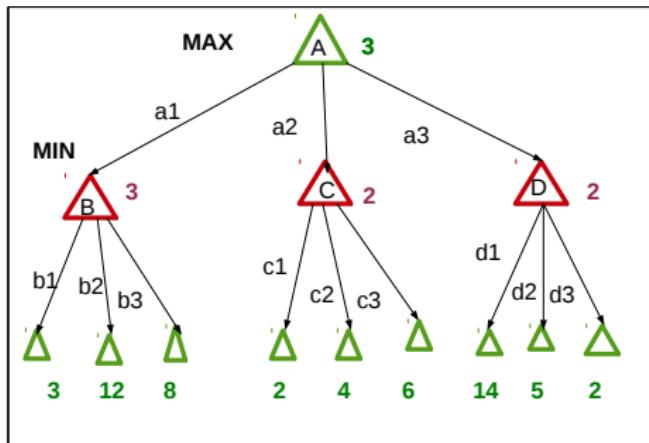
Strategia

- ▶ 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 **mieszana 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).

Przykład i MINIMAX z ostatniego wykładu

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

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

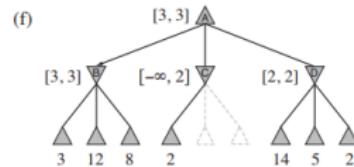
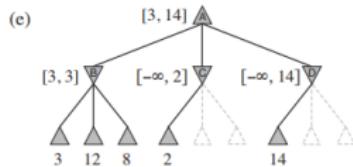
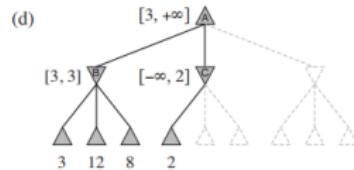
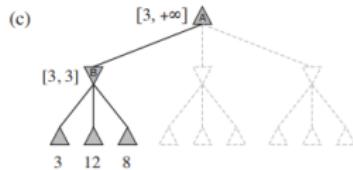
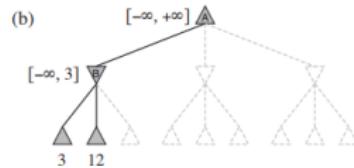
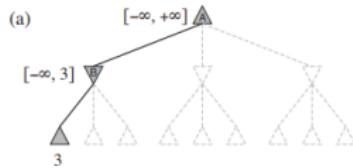


MINIMAX: Pomiar wydajności

- ▶ Wykorzystuje proste rekurencyjne obliczenie wartości minimax każdego następnika (potomnika) węzła. Rekurencja sięga aż do liści.
- ▶ Na przykład, w pokazanym drzewie gry najpierw pojawia się w lewym dolnym węźle z wartościami użytecznymi odpowiednio 3, 12 i 8. Następnie ten algorytm wybiera minimalną wartość 3, a wartość ta jest zapisana dla MAX jako najgorsza możliwa wartość użyteczna po wybraniu akcji a_1 (zobacz B). Po podobnym procesie algorytm przechowuje 2 jako wartości użyteczności dla obu akcji a_2, a_3 (zobacz C, D). Na koniec algorytm wybiera 3 jako maksimum najgorszych możliwych wartości użyteczności branych pod uwagę we wszystkich działaniach a_1, a_2, a_3 .
- ▶ Algorytm wykonuje przeszukiwanie w głąb ([depth-first search](#)) drzewa gry. Tak więc, jeśli maksymalna głębokość wynosi m , i jest najwyżej b dopuszczalnych ruchów w każdym węźle, wówczas złożoność czasowa wynosi $O(b^m)$, a złożoność pamięciowa wynosi $O(bm)$.

Alpha-Beta Pruning

- Liczba węzłów, które należy sprawdzić w MINIMAX, jest wciąż wykładnicza względem głębokości drzewa gry. 'Alpha-beta pruning' wprowadza przeszukiwanie tylko efektywnych węzłów i odcina część drzewa.



- (a) Na początku zakres wartości w węzle głównym A jest przechowywany jako $[-\infty, \infty]$. Aby sprawdzić wartość $\text{MINIMAX}(A)$, przeszukiwanie rozpoczyna się od lewej skrajnej gałęzi dla MAX.
- (b, c) Następnie od lewej do prawej sprawdzane są wszystkie możliwe wartości dla działań b_1, b_2, b_3 . Najgorsza możliwa wartość użyteczna dla MAX, po tym, jak MIN wybierze własne działanie, jest przechowywana w węźle B jako 3. Na tym etapie optymalny zakres wartości dla węzła A jest przechowywany jako $[3, \infty]$ a dla B to $[3, 3]$.
- (d) Następnie rozpoczyna się przeszukiwanie od następnego węzła C , sprawdzanie od lewej skrajnej gałęzi c_1 . Ponieważ wartość użyteczności 2 jest już mniejsza niż optymalna wartość użyteczności (3), dla MAX w węźle B , algorytm nie sprawdza pozostałych gałęzi C . Zatem w węźle C zakres optymalnych wartości użyteczności zostanie ustawiony jako $[-\infty, 2]$.
- (e, f) Teraz przeszukiwanie przechodzi do D i rozpoczyna sprawdzanie od lewej do prawej wszystkich możliwych działań d_1, d_2, d_3 . Tak jak poprzednio, wybiera najgorszą możliwą wartość, która wynosi 2, dla MAX. Dlatego optymalny zakres wartości użyteczności dla MAX przy D jest ustalony jako $[2, 2]$.

- ▶ Wreszcie po sprawdzeniu wszystkich gałęzi a_1, a_2, a_3 w A maksymalny optymalny zakres wartości użyteczności dla MAX jest ustalony jako $[3, 3]$. Więc $\text{MINIMAX}(A) = 3$.
- ▶ **główny pomysł:** Jeśli gracz ma lepszy wybór w którymkolwiek z nadzędnych węzłów x lub wyższych, gałąź sięgająca x może zostać przycięta.
- ▶ α = najlepsza wartość (tj. najwyższa wartość) znaleziona do tej pory na ścieżce MAX
 β = najlepsza wartość (tj. najmniejsza wartość) znaleziona do tej pory na ścieżce MIN.
- ▶ Przeszukiwanie aktualizuje wartość $[\alpha, \beta]$ w każdym węźle, gdy porusza się wzdłuż ścieżek drzewa, i przycina pozostałe gałęzie w węźle, gdy tylko wiadomo, że wartość bieżącego węzła jest gorsza niż bieżące wartości α lub β .

Gry z niedoskonałą decyzją w czasie rzeczywistym: funkcja heurystyczna

- ▶ ‘Alpha-beta pruning’ wciąż musi sprawdzać węzły liści, a czasami ta głębokość stwarza problem, gdy ruchy muszą być wykonywane bardzo szybko.
- ▶ Claude Shannon (1950): ‘Programming a computer for playing chess’

W tym artykule Shannon zasugerował, że zamiast szukać aż do węzłów końcowych, programy powinny zatrzymać się na wcześniejszym poziomie i zastosować funkcję oceny heurystycznej do stanów, aby sprawdzić skuteczność działania.

Aby nieco zmienić MINIMAX lub 'Alapha-Beta prunning'

- ▶ Zastąpienie funkcji Utility heurystyczną funkcją oceny EVAL, która szacuje użyteczność stanu.
- ▶ Zastąpienie TERMINAL-TEST testem odcięcia (CUT-OFF TEST), który decyduje, kiedy zastosować EVAL
- ▶ To daje nam następującą heurystyczną funkcję minimax dla stanu s i maksymalnej głębokości d .
- ▶ Za pomocą EVAL oszacujemy oczekiwany użyteczność gry z danego stanu. Pomyśl jest podobny do użycia funkcji heurystycznej do oszacowania odległości węzła docelowego w algorytmach przeszukiwania.

H-MINIMAX(s, d)

$$\begin{aligned} &= \text{EVAL}(s) && \text{jeśli } \text{CUTOFF-TEST}(s, d), \\ &= \max_{a \in \text{ACTION}(s)} H\text{-MINIMAX}(\text{RESULT}(s, a), d + 1) && \text{jeśli } \text{PLAYER}(s) = \text{MAX}, \\ &= \min_{a \in \text{ACTION}(s)} H\text{-MINIMAX}(\text{RESULT}(s, a), d + 1) && \text{jeśli } \text{PLAYER}(s) = \text{MIN}, \end{aligned}$$

Przykład heurystycznej funkcji oceny dla szachów

- ▶ Ponieważ przeszukiwanie musi zostać przerwane w stanie nieterminalnym, algorytmy muszą zgadywać o wyniku EVAL dla stanu końcowego. EVAL jest zaprojektowany w oparciu o różne funkcje gry.
- ▶ Na przykład, w książkach wprowadzających do gry w szachy mamy pewne wartości konkretne dla każdego rodzaju figur: Takie jak pion jest wart 1, skoczek lub goniec wart jest 3, jednego gońca wart jest 5, a królowa 9. Każdy typ odpowiada jednej kategorii. Zatem EVAL może być postaci:

$$EVAL(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

gdzie f_i to kategoria odpowiadająca każdemu typowi i w_i to liczba sztuk tego typu.

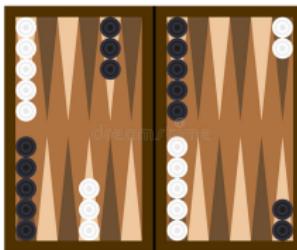
- ▶ Na przykład, jeśli w stanie mamy pięć pionków, jednego skoczka, jedną wieżę, jednego gońca i jedną królową, wtedy
 $EVAL(s) = 5 \times 1 + 1 \times 3 + 1 \times 3 + 1 \times 5 + 1 \times 9 = 25.$

Głębokość na CUTOFF-TEST(s, x)

- ▶ Głębokość (d) odcięcia przeszukiwania można określić na podstawie średniej liczby ruchów, które można wybrać w określonym czasie. Czas ten można z góry ustalić na podstawie tego, ile czasu pozwalamy agentowi oprogramowania na wybór ruchu.
- ▶ Wobec tego CUTOFF-TEST(s, x) zwraca wartość prawda, jeśli głębokość x jest poniżej d , a zatem EVAL(s) należy obliczyć dla s .

Gry stochastyczne

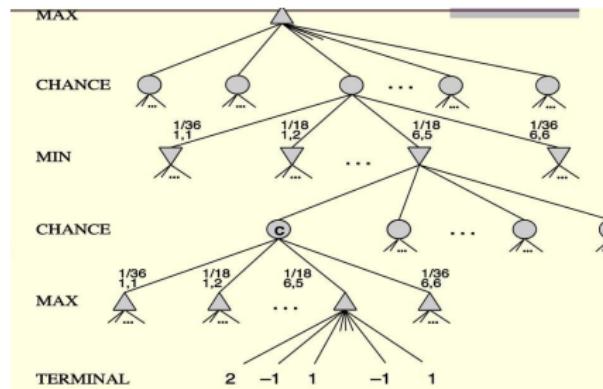
Backgammon to typowa gra, która łączy szansę i umiejętności.



- ▶ W grze gra kolejno dwóch graczy. W każdej turze gracz musi rzucić dwiema kostkami.
- ▶ Każdy gracz ma 15 warcabów (zwykle białe i czarne). Liczby pojawiające się w dwóch kostkach określają liczbę prawidłowych ruchów dla każdego gracza.
- ▶ Założymy, że białe wyrzucają 6-5 i mają cztery ruchy. Ale białe nie wiedzą, jakie liczby wyrzucają czarne, a zatem jakie będą dopuszczalne ruchy czarnych.
- ▶ Dlatego nie możemy zbudować standardowego drzewa gry jak dla gry w kółko i krzyżyk.

Drzewo gry dla blackgammona

- ▶ Drzewo gry dla gry Blackgammon zawiera losowe węzły oprócz węzłów dla MAX i MIN. Gałęzie z każdego węzła losowego oznaczają możliwe rzuty kostkami i są oznaczone rzutami i ich prawdopodobieństwami.
- ▶ Z 36 równie prawdopodobnych rzutów 21 jest różnych, ponieważ zasady dla 6-5 są takie same jak dla 5-6.
- ▶ 6 podwójnych ma prawdopodobieństwo $\frac{1}{36}$, a dla każdego z pozostałych 15 rzutów prawdopodobieństwo wynosi $\frac{2}{36}$ (tzn. $\frac{1}{18}$).



Zamiast MINIMAX mamy EXPECTMINIMAX

- ▶ W przypadku gier stochastycznych zamiast dokładnej wartości minimax wprowadzamy oczekiwana wartość minimax.

EXPECTMINIMAX(s)

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

gdzie r = możliwy rzut kostką (lub inne zdarzenie losowe) i $\text{RESULT}(s, r) = s$

Kilka słów o innych kryteriach optymalizacji

Przypomnijmy definicję gry omówioną w trzecim wykładzie

- ▶ Krotka (N, A, u) to gra w normalnej formie (skończona, n -osobowa) gdzie:
 N jest skończonym zbiorem graczy n , indeksowanych przez i ;
 $A = A_1 \times A_2 \times \dots \times A_n$, gdzie A_i to skończony zbiór działań dostępnych dla gracza i . Każdy wektor $a = (a_1, a_2, \dots, a_n) \in A$ jest nazywany profilem działania.
 $u = (u_1, u_2, \dots, u_n)$ gdzie każdy $u_i : A \mapsto R$ to funkcja użyteczności (lub funkcja wypłaty) dla każdego gracza i .
- ▶ W przypadku gry dwuosobowej $N = 2$.

Rozważmy tę macierz wypłat dla dwóch graczy, którzy mają dwie możliwe akcje C i D. Wiersz odpowiada graczowi 1, a kolumna odpowiada graczowi 2.

	<i>C</i>	<i>D</i>
<i>C</i>	(-1, -1)	(-4, 0)
<i>D</i>	(0, -4)	(-3, -3)

Liczba ujemna $-k$ oznacza stratę k jednostek.

Optymalna strategia Pareto i najlepsza odpowiedź

- (i) **Pareto Optimal:** Kryterium optymalności Pareto stanowi porządek określony na profilach użyteczności agentów. Na przykład, można powiedzieć, że wypłata $(-1, -1)$ jest lepsza niż wypłata $(-3, -3)$ dla obu agentów. Tak więc profil strategii (C, C) dominuje (D, D) w oparciu o kryterium optymalności pareto i **Wtedy optymalna strategia to (C, C) .**
- (ii) **Best response:** Gdy gracz 1 wybiera C , dla gracza 2 akcja D jest optymalna, a gdy gracz 1 wybiera D , akcja D jest ponownie optymalna dla gracza 2. Tak więc dla gracza 2 najlepszą odpowiedzią jest zawsze D . Za pomocą podobnego argumentu możemy sprawdzić, dla gracza 1 najlepszą odpowiedzią jest również D . Tak więc **strategia (D, D) jest optymalna, biorąc pod uwagę najlepsze odpowiedzi obu graczy.**

	C	D
C	$(-1, -1)$	$(-4, 0)$
D	$(0, -4)$	$(-3, -3)$

Problemy spełniania więzów (Constraint Satisfaction Problems (CSP))

- ▶ Podczas przeszukiwania algorytmów i gier rozważaliśmy **atomową reprezentację** każdego stanu.
- ▶ W CSP reprezentujemy każdy stan za pomocą zbioru zmiennych, z których każda ma wartość. Jak omówiliśmy w pierwszym wykładzie, jest to zatem kategoria **rozproszonej reprezentacji** środowiska zadań.
- ▶ **CSP:** CSP składa się z trzech komponentów X , D i C , gdzie
 - ▶ $X = \{X_1, X_2, \dots, X_n\}$ jest zbiorem zmiennych,
 - ▶ $D = \{D_1, D_2, \dots, D_n\}$ jest zbiorem składającym się z dziedzin dla każdej zmiennej, a
 - ▶ C jest zbiorem więzów (ograniczeń), które określają dopuszczalne kombinacje wartości dla zmiennych.
- ▶ Każda domena $D_i = \{v_{i1}, v_{i2}, \dots, v_{ik}\}$ jest zbiorem wartości dla X_i .
- ▶ Każdy z więzów (ograniczenie) C_j może być reprezentowany jako para $\langle \text{zakres}, \text{rel} \rangle$ gdzie zakres jest krotką zmiennych, które dotyczą ograniczenia C_j , a rel jest relacją, która definiuje wartości które te zmienne mogą przyjąć.

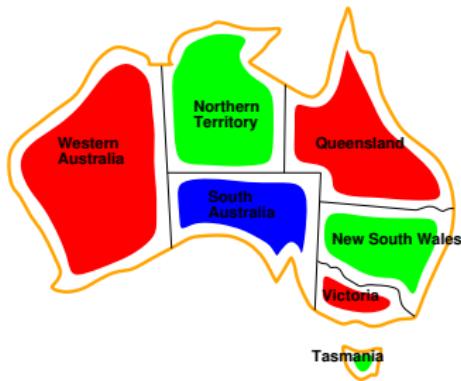
Rozwiązańe w CSP

- ▶ **Przykład-1:** Niech $X = \{X_1, X_2\}$, $D_1 = D_2 = \{A, B\}$, a wybrany z więzów (ograniczenie) to $X_1 \neq X_2$. To ograniczenie można przedstawić jako $\langle(X_1, X_2), \{(A, B), (B, A)\}\rangle$ lub $\langle(X_1, X_2), X_1 \neq X_2\rangle$.
- ▶ Każdy stan w CSP jest definiowany przez **przypisanie (assignment)** wartości do niektórych lub wszystkich zmiennych, np.
 $\{X_i = v_i, X_j = v_j, \dots, X_k = v_k\}$ gdzie $\{X_i, X_j, \dots, X_k\} \subseteq X$.
- ▶ Przypisanie, które nie narusza żadnego ograniczenia, nazywane jest **przypisaniem spójnym (consistent assignment)**.
- ▶ **Pełne przypisanie (complete assignment)** to takie, które przypisuje wartość każdej zmiennej.
- ▶ **Rozwiązańe (solution)** CSP to spójne, pełne przypisanie.
- ▶ Oba (A, B) , (B, A) są rozwiązaniami dla przykładu-1.

Przykład-2: problem kolorowania mapy

- ▶ Rozważmy problem kolorowania mapy Australii. Australia ma następujące stany:
Western Australia (WA), Northern Territory (NT), South Australia (SA), Queensland (Q), New South Wales (NSW), Victoria (V), Tasmania (T)
- ▶ Zadaniem jest pokolorowanie każdego regionu kolorem czerwonym, zielonym albo niebieskim w taki sposób, aby żadne sąsiednie regiony nie miały tego samego koloru.
- ▶ $X = \{WA, NT, SA, Q, NSW, V, T\}$,
 $D_1 = \dots = D_7 = \{\text{red}, \text{green}, \text{blue}\}$ i
 $C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}.$
- ▶ Ograniczenie $SA \neq WA$ można przedstawić jako
 $\{(red, green), (red, blue), (green, red), \dots, (blue, green)\}.$

Jedno z rozwiązań



Rozwiązania są wartościowaniem spełniającym wszystkie więzy, np.

{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green}

Rodzaje więzów

- ▶ Więzy **unarne** dotyczą pojedynczych zmiennych
np. $SA \neq green$
- ▶ Więzy **binarne** dotyczą dwu zmiennych
np. $SA \neq WA$
- ▶ Więzy **wyższego rzędu** dotyczą 3 lub więcej zmiennych,
np. Sudoku (Zwykle używamy więź **Alldiff**, który określa, że wszystkie zmienne związane z ograniczeniem muszą mieć różne wartości.)
- ▶ **Preferencje** (więzy nieostre)
np. red jest lepszy niż green (często reprezentowane przez funkcję kosztu przypisania wartości do zmiennej)
Rozważenie rozwiązań problemu spełniającego ograniczenia, a także preferencje należą do innej gałęzi, zwanej **problemami optymalizacji ograniczeń** (**Constraint Optimization Problems**)

Sudoku

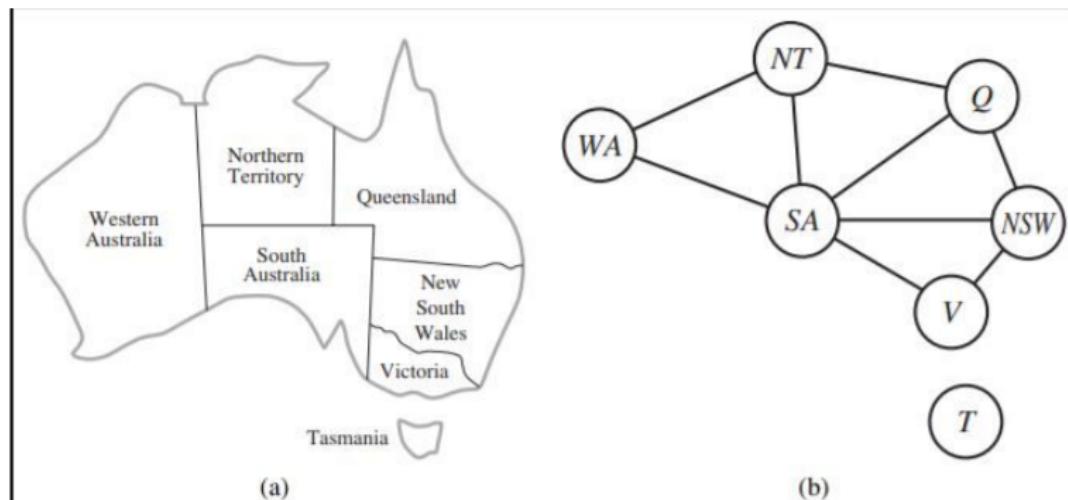
X_{11}	X_{12}	X_{13}		X_{19}
X_{21}	X_{22}	X_{23}		X_{29}
X_{31}	X_{32}	X_{33}		X_{19}
X_{71}	X_{72}	X_{73}		X_{79}
X_{81}	X_{82}	X_{83}		X_{89}
X_{91}	X_{92}	X_{93}		X_{99}

		3		2		6		
9			3		5			1
		1	8		6	4		
		8	1		2	9		
7								8
		6	7		8	2		
		2	6		9	5		
8			2		3			9
		5		1		3		

- ▶ $X = \{X_{11}, X_{12}, \dots, X_{99}\}$ i $D_i = \{1, 2, \dots, 9\}$
- ▶ Więzy:
 - ▶ W wierszu (rzędzie) wszystkie cyfry są różne
 $Alldiff(X_{11}, X_{12}, X_{13}, \dots, X_{19})$
⋮
 - ⋮
 - ▶ W kolumnie wszystkie cyfry są różne
 $Alldiff(X_{11}, X_{21}, X_{31}, \dots, X_{91})$
⋮
 - ⋮
 - ▶ W kwadracie 3×3 wszystkie cyfry są różne
 $Alldiff(X_{11}, X_{12}, X_{13}, X_{21}, X_{22}, X_{23}, X_{31}, X_{32}, X_{33})$
⋮

CSP-Graf: mapa Australii

- ▶ Jak możemy znaleźć rozwiązanie, biorąc pod uwagę CSP przy użyciu agenta oprogramowania?
- ▶ Pierwszym krokiem jest to, że możemy przedstawić CSP jako graf.



- ▶ Węzły grafu odpowiadają zmiennym CSP
- ▶ Krawędź łączy dwie zmienne, jeśli uczestniczą w ograniczeniu

Sprawdzanie lokalnej spójności

- ▶ Aby stworzyć algorytm przeszukiwania rozwiązania dla CSP, oprócz grafu reprezentującego przestrzeń stanu problemu **musimy dodać metodę sprawdzania spójności lokalnej**.
- ▶ W algorytmach CSP występują dwie części;
 - (i) **jedna polega na poszukiwaniu przypisania** z kilku możliwych przypisań dla zmiennych lub
 - (ii) **za pomocą propagacji więzów algorytm ustala**, które wartości zmiennej (połączonej z bieżącą zmienną) należy uznać za dopuszczalne (tj. które wartości należy wykluczyć z rozważania)
- ▶ Tak więc potrzebne jest **sprawdzanie lokalnej spójności**. Rozpoznając CSP jako graf, metoda sprawdzania lokalnej spójności wymusza eliminację niespójnych wartości na rozpatrywanej części grafu.

Dziękuję za uwagę

Sztuczna Inteligencja

Soma Dutta

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

Wykład - 6: Zbiory przybliżone i Zbiory rozmyte

Semestr letni 2022

Podsumowanie ostatniego wykładu

- ▶ Redukty i reguły generowane przy użyciu reduktów
- ▶ Skracanie reguł w taki sposób, aby zawierały minimalną liczbę deskryptorów
- ▶ Klasyfikacja nowego przypadku za pomocą reguł

Plany

- ▶ Klasifikacja zbioru przykładów testowych nie jest tak prosta - czasami trzeba wykonać operację aproksymacji
- ▶ Nauczymy się przybliżonych operacji bazujących na zbiorze przybliżonym
- ▶ Istnieje inne podejście do przybliżania zbioru obiektów, które nie mają jednoznacznego opisu. Jest to znane jako teoria zbiorów rozmytych.

Przykład - 1: pierwsza metoda

	a	b	c	d	dec
o_1	0	2	1	0	0
o_2	1	2	2	1	0
o_3	2	0	2	1	1
o_4	0	2	1	1	2
u	0	0	2	1	?

► **Reguły:**

$$b = 2 \wedge c = 1 \wedge d = 0 \Rightarrow dec = 0$$

$$b = 2 \wedge c = 2 \wedge d = 1 \Rightarrow dec = 0$$

$$b = 0 \wedge c = 2 \wedge d = 1 \Rightarrow dec = 1$$

$$b = 2 \wedge c = 1 \wedge d = 1 \Rightarrow dec = 2$$

► **Po skróceniu:**

$$b = 2 \wedge d = 0 \Rightarrow dec = 0 \quad \text{lub} \quad c = 1 \wedge d = 0 \Rightarrow dec = 0$$

$$b = 2 \wedge c = 2 \Rightarrow dec = 0$$

$$b = 0 \Rightarrow dec = 1$$

$$c = 1 \wedge d = 1 \Rightarrow dec = 2$$

► $Rules(u) = \{b = 0 \Rightarrow dec = 1\}$

► Zatem u jest klasyfikowany z $dec = 1$ z maksymalnym wsparciem 1.

Przykład-2

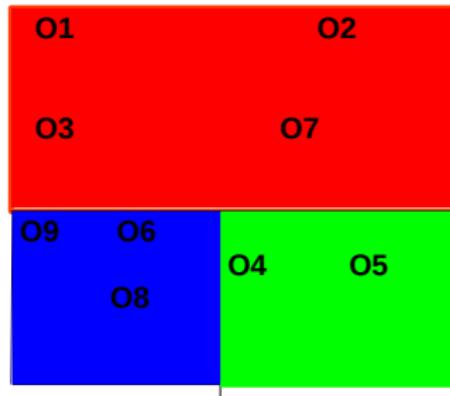
- ▶ Zakładamy, że a_1 = gorączka, a_2 = kontakt z pacjentem, a_3 = kaszel, d = Zainfekowany

	a_1	a_2	a_3	d
o_1	wysoka	bliski	średni	tak
o_2	wysoka	bliski	średni	tak
o_3	wysoka	bliski	średni	tak
o_4	więcej niż średnia	daleki	silny	nie pewne
o_5	więcej niż średnia	daleki	silny	nie
o_6	więcej niż średnia	daleki	lekki	nie
o_7	wysoka	bliski	średni	tak
o_8	więcej niż średnia	daleki	lekki	nie
o_9	więcej niż średnia	daleki	lekki	tak

- ▶ $\{\{o_1, o_2, o_3, o_7\}, \{o_4, o_5\}, \{o_6, o_8, o_9\}\}$
- ▶ $\{o_4, o_5\}, \{o_6, o_8, o_9\}$ mają sprzeczności, ponieważ mają elementy, które mają ten sam opis, ale różne decyzje

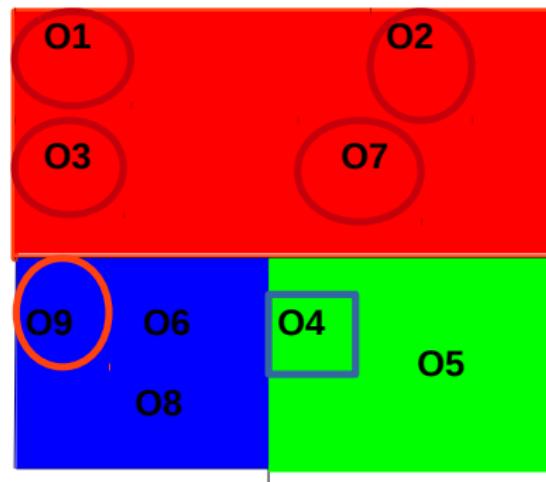
Klasy równoważności w odniesieniu do atrybutów warunkowych

- ▶ $\{\{o_1, o_2, o_3, o_7\}, \{o_4, o_5\}, \{o_6, o_8, o_9\}\}$
- ▶ Jak znaleźć reguły opisujące $\{o_4, o_5\}$ and $\{o_6, o_8, o_9\}$?



Klasy decyzyjne

- ▶ Klasy decyzyjne:
 - ▶ Tak: $X_1 = \{o_1, o_2, o_3, o_7, o_9\}$
 - ▶ Nie pewne: $X_2 = \{o_4\}$
 - ▶ Nie: $X_3 = \{o_5, o_6, o_8\}$
- ▶ Jak opisać klasy decyzyjne X_1, X_2, X_3 ?



Jak możemy opisać klasy decyzyjne

	a_1	a_2	a_3	d
o_1	wysoka	bliski	średni	tak
o_2	wysoka	bliski	średni	tak
o_3	wysoka	bliski	średni	tak
o_4	więcej niż średnia	daleki	siłny	nie pewne
o_5	więcej niż średnia	daleki	siłny	nie
o_6	więcej niż średnia	daleki	lekki	nie
o_7	wysoka	bliski	średni	tak
o_8	więcej niż średnia	daleki	lekki	nie
o_9	więcej niż średnia	daleki	lekki	tak

- ▶ Klasa decyzyjna $X_1 = \{o_1, o_2, o_3, o_7, o_9\}$
- ▶ Lower approximation (dolna aproksymacja): $\underline{X_1}_A = \cup\{[u]_A : [u]_A \subseteq X_1\} = \{o_1, o_2, o_3, o_7\}$ - Na podstawie informacji o elementach zawartej w wartościach atrybutów z A możemy stwierdzić, że elementy z pewnością należą do klasy decyzyjnej
- ▶ Upper approximation (górna aproksymacja): $\overline{X_1}_A = \cup\{[u]_A : [u]_A \cap X_1 \neq \emptyset\} = \{o_1, o_2, o_3, o_7, o_6, o_8, o_9\}$ - Na podstawie informacji o elementach zawartej w wartościach atrybutów z A możemy stwierdzić, że elementy może należą do klasy decyzyjnej

Opis klasy decyzyjnej

- ▶ Zauważmy, że za pomocą A nie możemy zdefiniować jednoznacznie X_1 , ponieważ istnieją pewne wystąpienia w X_1 , które mają różne właściwości.
- ▶ Opis X_1 jest podzielony na trzy części: Pewne przypadki ($\underline{X_1}_A$), Możliwe przypadki ($\overline{X_1}_A$) i Graniczne przypadki ($\overline{X_1}_A \setminus \underline{X_1}_A$).
- ▶ Możemy opisać klasę decyzji X_1 , czyli 'Zainfekowany = Tak', w następujący sposób.
 - ▶ **Pewne przypadki** : gorączka = wysoka \wedge kontakt z pacjentem = bliski \wedge kaszel = słiny \Rightarrow zainfekowany = tak.
 - ▶ **Możliwe przypadki**: gorączka = więcej niż średnia \wedge kontakt z pacjentem = daleki \wedge kaszel = lekki \Rightarrow zainfekowany = tak.

Radzenie sobie ze sprzecznymi danymi

- ▶ Generalized decision function (ugólniona funkcja decyzyjna) :
 $\partial_B(u) = \{dec(u') : u' \in [u]_B\}.$
- ▶ Na przykład, $\partial_A(o_4) = \{nie\;pewne, nie\}$
- ▶ Rough membership function (funkcja przyblizonego należenia):
 $\overrightarrow{\mu}_B(u) = \langle \mu_B^1(u), \mu_B^2(u), \dots, \mu_B^n(u) \rangle$ gdzie $\mu_B^i(u) = \frac{|[u]_B \cap X_i|}{|[u]_B|}$ dla każdej wartości decyzji i .
- ▶ Na przykład, $[o_6]_A = \{o_6, o_8, o_9\}$. Więc $\overrightarrow{\mu}_A(o_6) = \langle \frac{1}{3}, 0, \frac{2}{3} \rangle$.
- ▶ Zakładając $B = \{a_1, a_2\}$, mamy tylko dwie klasy równoważności:
 $\{\{o_1, o_2, o_3, o_7\}, \{o_4, o_5, o_6, o_8, o_9\}\}.$
Więc, $[o_6]_B = \{o_4, o_5, o_6, o_8, o_9\}$ i $\overrightarrow{\mu}_B(o_6) = \langle \frac{1}{5}, \frac{1}{5}, \frac{3}{5} \rangle$.
- ▶ Niech $C = \{a_2, a_3\}$. Należy notować, że C to redukt w poprzednim znaczeniu.
- ▶ Zauważmy, że $[u]_A = [u]_C$ dla każdego elementu $u \in \{o_1, \dots, o_9\}$.
- ▶ Tak więc $\partial_A(u) = \partial_C(u)$ oraz $\overrightarrow{\mu}_A(u) = \overrightarrow{\mu}_C(u)$ dla każdego elementu $u \in \{o_1, \dots, o_9\}$.

Niesprzeczna tabela decyzyjna

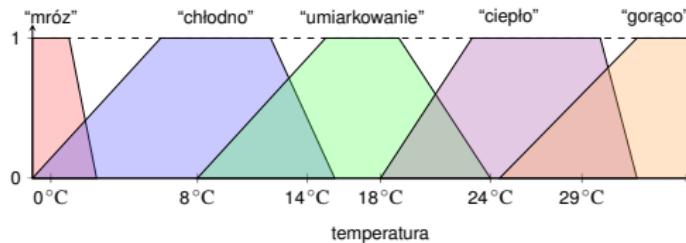
	a_1	a_2	a_3	d	∂_A	$\vec{\mu}_A$
o_1	wysoka	bliski	średni	tak	{tak}	$\langle 1, 0, 0 \rangle$
o_2	wysoka	bliski	średni	tak	{tak}	$\langle 1, 0, 0 \rangle$
o_3	wysoka	bliski	średni	tak	{tak}	$\langle 1, 0, 0 \rangle$
o_4	więcej niż średnia	daleki	silny	nie pewne	{nie pewne, nie}	$\langle 0, \frac{1}{2}, \frac{1}{2} \rangle$
o_5	więcej niż średnia	daleki	silny	nie	{nie pewne, nie}	$\langle 0, \frac{1}{2}, \frac{1}{2} \rangle$
o_6	więcej niż średnia	daleki	lekki	nie	{tak, nie}	$\langle \frac{1}{3}, 0, \frac{2}{3} \rangle$
o_7	wysoka	bliski	średni	tak	{tak}	$\langle 1, 0, 0 \rangle$
o_8	więcej niż średnia	daleki	lekki	nie	{tak, nie}	$\langle \frac{1}{3}, 0, \frac{2}{3} \rangle$
o_9	więcej niż średnia	daleki	lekki	tak	{tak, nie}	$\langle \frac{1}{3}, 0, \frac{2}{3} \rangle$

- ▶ (gorączka = więcej niż średnia) \wedge (kontakt z pacjentem = daleki) \wedge (kaszel = silny) \Rightarrow
 $Zainfekowany | \partial_A = \{nie\;pewne,\;nie\}$
- ▶ (gorączka = więcej niż średnia) \wedge (kontakt z pacjentem = daleki) \wedge (kaszel = silny) \Rightarrow
 $Zainfekowany | \mu_A \langle 0, \frac{1}{2}, \frac{1}{2} \rangle$

Przybliżenie przy użyciu metody zbiorów rozmytych

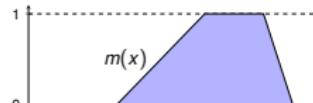
- ▶ To podejście różni się nieco od wcześniejszego. Podejście to uogólnia definicję zbioru w następujący sposób.
- ▶ Zbiór wszystkich parzystych liczb naturalnych: $E = \{2, 4, 6, 8, \dots\} \subseteq \mathbb{N}$
- ▶ Funkcja charakterystyczna: $Ch_E : \mathbb{N} \mapsto \{0, 1\}$ jest taka, że $Ch_E(x) = 1$ jeśli $x \in E$, inaczej $Ch_E(x) = 0$.
- ▶ Jak zapisujemy zbiór wszystkich liczb rzeczywistych **bliskich 2**? To nie jest jednoznaczny opis i może mieć różne interpretacje.
- ▶ Możemy ten zbiór opisać za pomocą funkcji $\mu_2 : \mathbb{R} \mapsto [0, 1]$ takiej że
 - $\mu_2(x) = 1$, jeśli $0 \leq |2 - x| \leq 1$
 - $= .9$, jeśli $1 < |2 - x| \leq 2$
 - \vdots
- ▶ Zamiast dwóch wartości logicznych (prawda i fałsz), dopuszcza się istnienie nieskończoności wielu wartości (odpowiadających liczbom rzeczywistym od 0 do 1)

- ▶ Tak więc dla każdego $x \in \mathbb{R}$, $\mu_2(x)$ reprezentuje stopień przynależności x do pojęcia (bliskich 2) reprezentowanego przez μ_2 .
- ▶ Również można to zapisać następująco $(x, \mu_2(x))$ dla każdego $x \in \mathbb{R}$.
- ▶ **Zbiór rozmyty:** Zbiór rozmyty z dziedziną X jest reprezentowany przez funkcję $f : X \mapsto [0, 1]$. Funkcję f nazywamy zbiorem rozmytym na dziedzinie X .

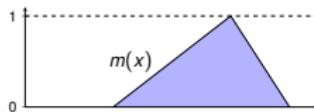


Pojęcia "ciepło" czy "gorąco" są określone w sposób niestry: trudno jednoznacznie określić ich granice, ich zakresy mogą się częściowo pokrywać.

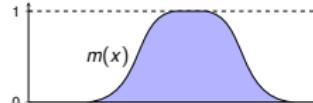
Funkcje mogą mieć kształt trapezu...



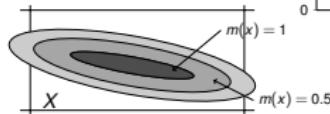
...trójkąta...



...ale też inny (np. sigmoidalny)...



... a zbiór X nie musi być zbiorem liczb rzeczywistych



Reguły w rozmytych zbiorach

- ▶ We wcześniejszym podejściu zaczęliśmy od tabeli zawierającej przypadki i ich opisy pod względem wartości niektórych atrybutów
- ▶ gorączka = wysoka \wedge kontakt z pacjentem = bliski \wedge kaszel = śliny
 \Rightarrow zainfekowany = tak.
- ▶ W aktualnie rozważanym podejściu każde zdanie ma pewną wartość, które mogą być inne niż prawda lub fałsz, można je nazwać stopniem prawdy.
- ▶ Na przykład, reprezentujmy pojęcie wysoka jak o $\mu_{\text{wysoka}} : [35, 42] \mapsto [0, 1]$ gdzie $[35, 42]$ jest dziedziną dla temperatury gorączki. Może ta funkcja być taka, że
$$\begin{aligned}\mu_{\text{wysoka}}(x) &= 1 \text{ jeśli } x \geq 40 \\ &= .9 \text{ jeśli } 38 \leq x < 40 \\ &= .7 \text{ jeśli } 37 \leq x < 38 \\ &= .5 \text{ jeśli } 36 \leq x < 37 \\ &= .1 \text{ jeśli } 35 \leq x < 36\end{aligned}$$
- ▶ Gorączka może więc być traktowana jako zmienna, który może mieć wartości z $[35, 42]$. Jeśli gorączka ma wartość 38, stopień prawdziwości zdania 'gorączka = wysoka' jest $\mu_{\text{wysoka}}(38) = .9$

Obliczanie wartości prawdy w zdaniach prostych i złożonych

- ▶ ‘gorączka = wysoka’ jest przykładem prostego zdania.
- ▶ Zauważliśmy, że każde proste zdanie jest reprezentowane przez zbiór rozmyty, czyli funkcja z odpowiedniej domeny do przedziału $[0, 1]$. Wartość prawdziwości zdania zależy od funkcji odpowiadającej temu rozmytemu zbiorowi.
- ▶ Np. zbiór rozmyty ‘wysoka’ jest reprezentowany przez funkcję μ_{wysoka} z domeny temperatury do $[0, 1]$. Tutaj ‘wysoka’ dotyczy atrybutu ‘gorączka’, więc jako domenę mamy przedział temperatur.
- ▶ Podobnie ‘silny’ jest stosowany do atrybutu ‘kaszela’, więc możemy przyjąć domenę reprezentującą różną intensywność kaszlu, a pojęcie ‘silny’ można przedstawić jako zbiór rozmyty, czyli funkcja μ_{silny} , która zwraca wartości w $[0, 1]$.
- ▶ Przypuszczać mamy:
 - ▶ gorączka = wysoka ma stopień prawdziwości .9,
 - ▶ kaszel = sliny ma stopień prawdziwości .7
- ▶ Jak obliczyć wartość prawdy zdania złożonego:
 $(\text{gorączka} = \text{wysoka}) \wedge (\text{kaszela} = \text{sliny})$

Spójniki logiczne: klasyczny scenariusz z dwiema wartościami

Spójniki logiczne: \wedge (oraz), \vee (lub), \Rightarrow (jeśli ... to), \neg (nie)

- ▶ Niech p i q będą dwoma zdaniami.

\wedge	0	1	\vee	0	1	\Rightarrow	0	1	\neg
0	0	0	0	0	1	0	1	1	0
1	0	1	1	1	1	1	0	1	0

- ▶ $\neg : \{0, 1\} \mapsto \{0, 1\}$, i dla pozostałych spójników mamy $\{0, 1\} \times \{0, 1\} \mapsto \{0, 1\}$.
- ▶ Spójniki logiczne w kontekście zbiorów rozmytych są generalizowane z kontekstu dwuwartościowego

Logika rozmyta: Koniunkcja

- ▶ Niech α, β będą zbiorami rozmytymi na dziedzinie X . Tzn.,
 $\mu_\alpha : X \mapsto [0, 1]$ i $\mu_\beta : X \mapsto [0, 1]$
- ▶ Koniunkcję α i β definiujemy jako zbiór rozmyty $\alpha \wedge \beta$ o funkcji przynależności $\mu_{\alpha \wedge \beta} : X \mapsto [0, 1]$ określony wzorem
$$\mu_{\alpha \wedge \beta}(x) = T(\mu_\alpha(x), \mu_\beta(x)) \quad \forall x \in X$$
- ▶ Funkcja $T : [0, 1] \times [0, 1] \mapsto [0, 1]$ jest **T-normą**.

Własności T-normy

- ▶ Warunki brzegowe:

$$T(0, x) = 0 \text{ oraz } T(1, x) = x.$$

- ▶ Monotoniczność:

Jeśli $x \leq y$, $T(x, z) \leq T(y, z)$.

- ▶ Symetria:

$$T(x, y) = T(y, x)$$

- ▶ Łączność:

$$T(x, T(y, z)) = T(T(x, y), z)$$

Przykładowe T-normy

- ▶ T-norma Zadeha:

$$T(x, y) = \min(x, y) \quad \forall x, y \in [0, 1]$$

- ▶ T-norma Mengara:

$$T(x, y) = x \cdot y \quad \forall x, y \in [0, 1]$$

- ▶ T-norma Łukasiewicza:

$$T(x, y) = \max(0, x + y - 1) \quad \forall x, y \in [0, 1]$$

Przykład

Przymajemy $X = \{x_1, x_2, x_3, x_4\}$

- ▶ Dla α równego $\{(x_1, 0.4), (x_2, 0), (x_3, 0.5), (x_4, 1)\}$ oraz
 β równego $\{(x_1, 0.6), (x_2, 0.5), (x_3, 0), (x_4, 1)\}$
otrzymujemy następujące wyniki dla różnych koniunkcji :
- ▶ Zadeh: $\alpha \wedge \beta$ równa się zbiorowi rozmytemu
 $\{(x_1, 0.4), (x_2, 0), (x_3, 0), (x_4, 1)\}$
- ▶ Mengar: $\alpha \wedge \beta$ równa się zbiorowi rozmytemu
 $\{(x_1, 0.24), (x_2, 0), (x_3, 0), (x_4, 1)\}$
- ▶ Łukasiewicz: $\alpha \wedge \beta$ równa się zbiorowi rozmytemu
 $\{(x_1, 0), (x_2, 0), (x_3, 0), (x_4, 1)\}$

Altarnetywa jako Ko-norma

- ▶ Warunki brzegowe:

$$C(0, x) = x \text{ oraz } C(1, x) = 1.$$

- ▶ Monotoniczność:

Jeśli $x \leq y$, $C(x, z) \leq C(y, z)$.

- ▶ Symetria:

$$C(x, y) = C(y, x)$$

- ▶ Łączność:

$$C(x, C(y, z)) = C(C(x, y), z)$$

Przykładowe Ko-normy

- ▶ Ko-norma Zadeha:

$$C(x, y) = \max(x, y) \quad \forall x, y \in [0, 1]$$

- ▶ K-norma Mengara:

$$C(x, y) = x + y - xy \quad \forall x, y \in [0, 1]$$

- ▶ T-norma Łukasiewicza:

$$C(x, y) = \min(x + y, 1) \quad \forall x, y \in [0, 1]$$

Negacja: zbiór rozmyty

- ▶ Negację zbioru α definiujemy jako zbiór $\neg\alpha$ o funkcji przynależności $\mu_{\neg\alpha} : X \mapsto [0, 1]$ określonej wzorem
$$\mu_{\neg\alpha}(x) = 1 - \mu_\alpha(x).$$
- ▶ Dla α równego $\{(x_1, 0.4), (x_2, 0), (x_3, 0.5), (x_4, 1)\}$, $\neg\alpha$ równa się:
$$\{(x_1, 0.6), (x_2, 1), (x_3, 0.5), (x_4, 0)\}$$

Dualność: T-norm i ko-norm

- ▶ $\alpha \vee \beta = \neg(\neg\alpha \wedge \neg\beta)$
- ▶ Na przykład: (*min, max, 1 - x*)
- ▶ (*T-Łukasiewicz, C-Łukasiewicz, 1-x*)

Implikacja dla zbiorów rozmytych

- ▶ Jak w kontekście klasycznym, możemy zdefinować:
 $\alpha \Rightarrow \beta = \neg\alpha \vee \beta$.
- ▶ Zadeh: $\mu_{\alpha \Rightarrow \beta}(x) = \max(1 - \mu_\alpha(x), \mu_\beta(x))$
- ▶ Łukasiewicz: $\mu_{\alpha \Rightarrow \beta}(x) = \min(1, 1 - \mu_\alpha(x) + \mu_\beta(x))$

Dziękuję za uwagę

Sztuczna Inteligencja

Soma Dutta

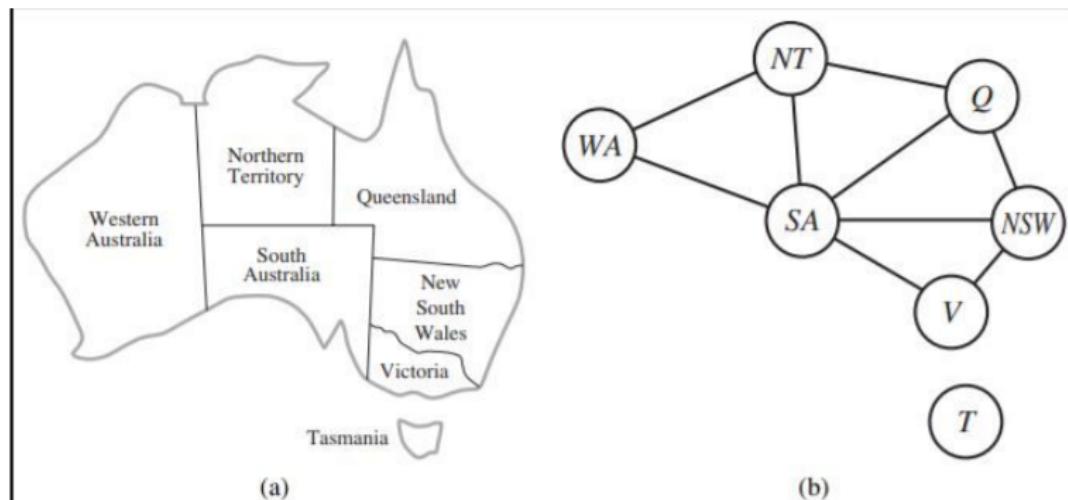
Wydział Matematyki i Informatyki, UWM w Olsztynie
soma.dutta@matman.uwm.edu.pl

Wykład - 5: Algorytmy poszukujące rozwiązań spełniających więzy
i Zbiory przybliżone

Semestr letni 2022

CSP-Graf: mapa Australii

- ▶ Jak możemy znaleźć rozwiązanie, biorąc pod uwagę CSP przy użyciu agenta oprogramowania?
- ▶ Pierwszym krokiem jest to, że możemy przedstawić CSP jako graf.



- ▶ Węzły grafu odpowiadają zmiennym CSP
- ▶ Krawędź łączy dwie zmienne, jeśli uczestniczą w ograniczeniu

Sprawdzanie lokalnej spójności

- ▶ Aby stworzyć algorytm przeszukiwania rozwiązania dla CSP, oprócz grafu reprezentującego przestrzeń stanu problemu **musimy dodać metodę sprawdzania spójności lokalnej**.
- ▶ W algorytmach CSP występują dwie części;
 - (i) **jedna polega na poszukiwaniu przypisania** z kilku możliwych przypisań dla zmiennych lub
 - (ii) **za pomocą propagacji więzów algorytm ustala**, które wartości zmiennej (połączonej z bieżącą zmienną) należy uznać za dopuszczalne (tj. które wartości należy wykluczyć z rozważania)
- ▶ Tak więc potrzebne jest **sprawdzanie lokalnej spójności**. Rozpoznając CSP jako graf, metoda sprawdzania lokalnej spójności wymusza eliminację niespójnych wartości na rozpatrywanej części grafu.

Plany

- ▶ Algorytmy i techniki dla problemów CSP
- ▶ Wyznaczanie reguł i ograniczeń z tablic danych z zastosowaniem zbiorów przybliżonych

Etapy w poszukiwaniu rozwiązania CSP

- ▶ Tworzenie grafu ograniczenia dla danego CSP
- ▶ Opracowanie algorytmu przeszukiwania, który przypisuje wartości kolejnym zmiennym, poprzez sprawdzenie jego lokalnej spójności
- ▶ Istnieją różne kryteria sprawdzania spójności, np.
 - ▶ Spójność węzła ([Node consistency](#))
 - ▶ Spójność łuku ([Arc consistency](#))
 - ▶ Spójność ścieżki ([Path consistency](#))

Spójność węzła

- ▶ Zwykle ma to zastosowanie w przypadku, gdy ograniczenie jest unarne: na przykład w przypadku problemu z kolorowaniem mapy $SA \neq \text{green}$.
- ▶ Ponieważ dziedziną dla każdej zmiennej jest $\{\text{red}, \text{green}, \text{blue}\}$, algorytm może zacząć się od niej, ale po zastosowaniu spójności węzłów zmniejszy dziedzinę do $\{\text{red}, \text{blue}\}$.
- ▶ **Definicja:** Pojedyncza zmienna (odpowiadająca węzłowi na grafie CSP) jest spójna węzłowo, jeśli wszystkie wartości w dziedzinie zmiennej spełniają unarne ograniczenia zmiennej.

Spójność łuku

- ▶ **Definicja:** Węzeł (zmienna) X_i jest zgodny (arc consistent) z innym węzłem X_j , jeśli dla każdej wartości w dziedzinie D_i istnieje wartość w dziedzinie D_j , która spełnia binarne ograniczenie (X_i, X_j) .
- ▶ Na przykład, rozważmy ograniczenie $Y = X^2$ gdzie $X, Y \in \{0, 1, \dots, 9\}$. Możemy przedstawić ograniczenie jako $\langle (X, Y), \{(0, 0), (1, 1), (2, 4), (3, 9)\} \rangle$. Jeśli zastosujemy spójność łuku w algorytmie przeszukiwania, wówczas metoda zredukuje dziedzinę X do $\{0, 1, 2, 3\}$, a dziedzinę Y do $\{0, 1, 4, 9\}$.
- ▶ Jednak spójność łuku nie jest użyteczna w przypadku problemu kolorowania mapy Australii. Wiemy że, $SA \neq WA$ co można przedstawić jako
$$\{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\}.$$
Ponieważ dla każdej wartości SA istnieje wartość WA, spójność łuku nie zmniejszy dziedziny żadnej ze zmiennych.

Pseudokod dla spójności łuku: AC-3



```
function AC-3(csp) returns false if an inconsistency is found and true otherwise
  inputs: csp, a binary CSP with components ( $X, D, C$ )
  local variables: queue, a queue of arcs, initially all the arcs in csp
  while queue is not empty do
     $(X_i, X_j) \leftarrow \text{REMOVE-FIRST(queue)}$ 
    if REVISE(csp,  $X_i, X_j$ ) then
      if size of  $D_i = 0$  then return false
      for each  $X_k$  in  $X_i.\text{NEIGHBORS} - \{X_j\}$  do
        add  $(X_k, X_i)$  to queue
  return true
```

```
function REVISE(csp,  $X_i, X_j$ ) returns true iff we revise the domain of  $X_i$ 
  revised  $\leftarrow$  false
  for each  $x$  in  $D_i$  do
    if no value  $y$  in  $D_j$  allows  $(x, y)$  to satisfy the constraint between  $X_i$  and  $X_j$  then
      delete  $x$  from  $D_i$ 
      revised  $\leftarrow$  true
  return revised
```

Spójność ścieżki

- ▶ Rozważmy problem kolorowania mapy Australii dwoma kolorami: czerwonym i niebieskim. Jeśli zastosujemy tutaj spójność łuku, to okaże się, że każda zmienna jest spójna względem łuku, ponieważ dla każdej wartości przyjmowanej przez jedną zmienną inną wartość może być przypisana drugiej zmiennej
- ▶ Na pewno problem nie ma rozwiązania, jeśli weźmiemy pod uwagę dwa kolory. To że problem nie ma rozwiązania, można wykazać za pomocą spójności ścieżki.



- ▶ **Definicja:** Zbiór 2 zmiennych $\{X_i, X_j\}$ jest ścieżką spójną względem do innej zmiennej X_m , jeśli dla każdego spójnego przypisania $\{X_i = a, X_j = b\}$ istnieje przypisanie dla X_m , które spełnia ograniczenie $\{X_i, X_m\}$ i $\{X_m, X_j\}$.

Sprawdzanie lokalnej spójności

- ▶ W algorytmach CSP występują dwie części;
 - (i) pierwsza polega na poszukiwaniu przypisania z kilku możliwych przypisań zmiennym
 - (ii) druga - za pomocą propagacji więzów algorytm ustala, które wartości zmiennej (połączonej z bieżącą zmienną) należy uznać za dopuszczalne (tj. które wartości należy wykluczyć z rozważań)
- ▶ Sudoku można rozwiązać posługując się tylko propagacją więzów.
- ▶ Istnieje wiele problemów CSP, których nie można rozwiązać jedynie poprzez propagację ograniczeń. W takich przypadkach musimy zastosować algorytmy przeszukiwania.

Backtracking search (poszukiwanie z powrotami)

- ▶ Jest to przeszukiwanie ‘depth-first’ które wybiera wartości dla jednej zmiennej w jednym kroku i wycofuje się z tego, gdy zmienna nie ma żadnej dopuszczalnej wartości do przypisania.
- ▶ Algorytm wielokrotnie wybiera zmienną, której nie została przypisana wartość, a następnie kolejno wypróbowuje wszystkie wartości z jej dziedziny, próbując znaleźć rozwiązanie.
- ▶ Jeśli zostanie wykryta niespójność, zwraca błąd, wymuszając w poprzednim przypisaniu wybranie innej wartości.

Pseudokod: backtracking search

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
    return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← INFERENCE(csp, var, value)
            if inferences ≠ failure then
                add inferences to assignment
                result ← BACKTRACK(assignment, csp)
                if result ≠ failure then
                    return result
            remove {var = value} and inferences from assignment
    return failure
```

- ▶ Algorytm modelowany jest na podstawie ‘depth-first search’.
- ▶ Funkcja SELECT-UNASSIGNED-VARIABLE() decyduje, którą zmienną wybrać jako następną, a funkcja ORDER-DOMAIN-VALUES() porządkuje wartości zmiennej, która ma być wybrana do sprawdzenia - mogą być różne definicje tych funkcji
- ▶ Funkcja INFERENCE() zasadniczo sprawdza lokalną spójność przypisywania wartości do zmiennych. Ta funkcja może być zdefiniowana wykorzystując spójność łuku lub spójność ścieżki itp.



Metody wyboru nieprzypisanej zmiennej

- ▶ **Minimum Remaining Values (MRV)**: Wymusza wybranie zmiennej o najmniejszej liczbie dopuszczalnych wartości, nazywanej zmienną z minimalną liczbą pozostałą wartością. Jeśli jakaś zmienna X nie ma żadnej dopuszczalnej wartości, MRV wybierze X i natychmiast wykryje błąd.
- ▶ Został również nazwany **metodą najbardziej ograniczonej zmiennej (most constrained variable)** lub **heurystyką pierwszą do porażki (fail-first heuristic)**, ta druga, ponieważ wybiera zmienną, która najprawdopodobniej spowoduje błąd wkrótce, tym samym przycinając drzewo przeszukiwania.

Wybór zmiennej w kolejnym kroku przeszukiwania:

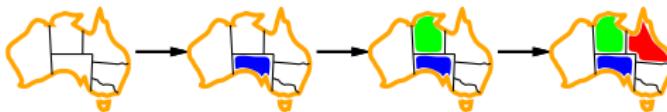
- **najbardziej ograniczona zmienna**, tzn. zmienna z najmniejszą liczbą dopuszczalnych wartości



- ▶ **Degree heuristic:** W przypadku, gdy na początku wszystkie zmienne mają taką samą liczbę dopuszczalnych wartości, stopień heurystyczny staje się bardziej użyteczny.
- ▶ Wybiera zmienną, która jest powiązana z największą liczbą więzów z innymi nieprzypisanymi zmiennymi.

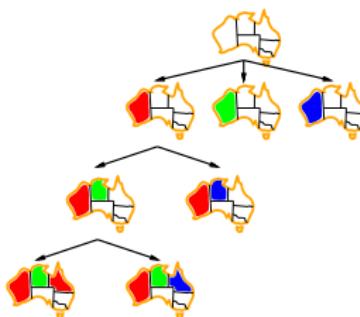
Wybór zmiennej w kolejnym kroku przeszukiwania:

- najbardziej ograniczająca zmienna, tzn. zmienna z największą liczbą więzów z pozostałymi zmiennymi



Tworzenia uporządkowania wartości dziedzin

- ▶ **Least constraining value:** Po wybraniu zmiennej lepiej jest wybrać wartość, która eliminuje najmniejszą liczbę opcji dla sąsiednich zmiennych.
- ▶ Ogólnie usiłuje pozostawić maksymalną elastyczność dla kolejnych przypisań zmiennych.
- ▶ Na przykład, jeśli wybierzemy 'WA = red', a 'NT = green', a następnie, jeśli naszym następnym wyborem będzie Q, to niebieski będzie złym wyborem, ponieważ eliminuje ostatnią dopuszczalną wartość pozostawioną sąsiadowi Q, SA. Ten sposób preferuje zatem bardziej kolor czerwony od niebieskiego.



Metoda projektowania ‘INFERENCE’

- ▶ Sprawdzanie w przód jest najprostszą formą wnioskowania. Ilekroć zmienna X jest przypisana przy sprawdzaniu w przód, zadaniem metody jest zapewnienie spójności łuku między X a każdą zmienną Y , która jest połączona z X . Oznacza to, że usunie z dziedziny Y każdą wartość, która jest niezgodna z wartością wybraną dla X .

	WA	NT	Q	NSW	V	SA	T
Initial domain	R, G, B						
After WA = red	R	G, B	R, G, B	R, G, B	R, G, B	G, B	R, G, B
After Q = green	R	B	G	R, B	R, G, B	B	R, G, B
After V = blue	R	B	G	R	B		R, G, B

- ▶ Poprzednio widzieliśmy, że na podstawie opisu środowiska zadań możemy stworzyć agenta bazującego na oprogramowaniu do poszukiwania optymalnego rozwiązania ([w algorytmach przeszukiwania](#)), optymalnych strategii ([w przypadku środowiska wieloagentowego, takiego jak gry](#)) i wnioskowania prowadzącego do rozwiązania za pomocą reprezentacji ograniczeń ([w przypadku CSP](#))
- ▶ Jeśli przed uruchomieniem algorytmu przeszukiwania mamy już opis problemu dostępny w postaci ograniczeń, zadanie staje się łatwiejsze. Ale często musimy nawet najpierw sformułować ograniczenia opisujące problem.
- ▶ Kolejnym trudnym zadaniem jest [wyrażenie ograniczeń lub reguł z tablic danych, która zawiera opisy zmiennych i ich możliwych wartości](#).

Teoria Zbiorów przybliżonych

- ▶ Zbiory przybliżone (Pawlak, 1981)
- ▶ Redukty i reguły generowane z reduktów (Skowron, Rauszer, 1992)
- ▶ $A = \{a_1, a_2, \dots, a_n\}$: zbiór cech (atrybutów lub zmiennych) opisujących przykłady
- ▶ U_{trn} : zbiór przykładów opisanych wektorami wartości cech $\langle v_{11}, v_{21}, \dots, v_{n1} \rangle$ gdzie $v_{11} \in D_1, v_{21} \in D_2, \dots, v_{n1} \in D_n$, a D_i to domena wartości dla a_i .
- ▶ Przykład: $U_{trn} = \{o_1, o_2, o_3, o_4\}$, $A = \{a, b, c, d\}$ - atrybuty warunkowe, (U_{trn}, A) : System informacyjny
- ▶ dec : atrybut decyzji, (U_{trn}, A, dec) : system decyzyjny

System Informacyjny

	a	b	c	d	
o_1	0	2	1	0	
o_2	1	2	2	1	
o_3	2	0	2	1	
o_4	0	2	1	1	

System decyzyjny

	a	b	c	d	dec
o_1	0	2	1	0	0
o_2	1	2	2	1	0
o_3	2	0	2	1	1
o_4	0	2	1	1	2

Redukty decyzyjny

- ▶ **Definicja:** Zbiór atrybutów $R \subseteq A$ jest reduktem decyzyjnym dla zbioru przykładów U_{trn} , jeśli
 - ▶ dla każdej pary przykładów $o_i, o_j \in U_{trn}$ o różnych decyzjach $dec(o_i) \neq dec(o_j)$ o ile istnieje $a \in A$ rozróżniający tę parę przykładów, to istnieje $a_k \in R$ rozróżniający tę parę przykładów: $v_{ik} \neq v_{jk}$ gdzie $v_{ik} = a_k(o_i)$, $v_{jk} = a_k(o_j)$.
 - ▶ R jest minimalnym zbiorem mającym powyższą własność, tzn., dla dowolnego $R' \subseteq R$, istnieje para przykładów w U_{trn} o różnych decyzjach i takich samych wartościach na wszystkich atrybutach $a_i \in R'$.

	a	b	c	d	dec
o_1	0	2	1	0	0
o_2	1	2	2	1	0
o_3	2	0	2	1	1
o_4	0	2	1	1	2

- ▶ **Redukty:** $\{a, d\}$, $\{b, c, d\}$

Minimalny redukt

- ▶ **Definicja:** Redukt R jest minimalny, jeśli zawiera najmniejszą możliwą liczbę atrybutów, tzn. dla każdego reduktu R' , $|R| \leq |R'|$.
- ▶ **Fakt:** Problem znalezienia minimalnego reduktu jest NP-trudny.
- ▶ W powyższym przykładzie minimalny redukt to $\{a, d\}$

Generowanie reguł z reduktu

- ▶ $Rules(R) = \{\bigwedge_{a_i \in R} a_i(o) = v_i \Rightarrow dec = dec(o) : o \in U_{trn}\}$
- ▶ Znajdźmy $Rules(\{b, c, d\})$:

	a	b	c	d	dec
o_1	0	2	1	0	0
o_2	1	2	2	1	0
o_3	2	0	2	1	1
o_4	0	2	1	1	2

- ▶ Reguły:
 - $b = 2 \wedge c = 1 \wedge d = 0 \Rightarrow dec = 0$
 - $b = 2 \wedge c = 2 \wedge d = 1 \Rightarrow dec = 0$
 - $b = 0 \wedge c = 2 \wedge d = 1 \Rightarrow dec = 1$
 - $b = 2 \wedge c = 1 \wedge d = 1 \Rightarrow dec = 2$

Skracanie reguł z reduktu

- ▶ Skracanie reguły polega na odrzuceniu niektórych atrybutów z warunku reguły.
- ▶ Reguła $\alpha \wedge s \Rightarrow dec = \beta$ może zostać zastąpiona przez $\alpha \Rightarrow dec = \beta$, jeśli $\alpha \Rightarrow dec = \beta$ pozostaje spójna ze zbiorem treningowym.
- ▶ **Fakt:** Może się zdarzyć, że różne reguły z tą samą decyzją zostaną skrócone do tej samej postaci. To oznacza, zbiór reguł po skróceniu może być mniejszy niż oryginalny.

Przykład

	a	b	c	d	dec
o_1	0	2	1	0	0
o_2	1	2	2	1	0
o_3	2	0	2	1	1
o_4	0	2	1	1	2

► **Reguły:**

$$b = 2 \wedge c = 1 \wedge d = 0 \Rightarrow dec = 0$$

$$b = 2 \wedge c = 2 \wedge d = 1 \Rightarrow dec = 0$$

$$b = 0 \wedge c = 2 \wedge d = 1 \Rightarrow dec = 1$$

$$b = 2 \wedge c = 1 \wedge d = 1 \Rightarrow dec = 2$$

► **Po skróceniu:**

$$b = 2 \wedge d = 0 \Rightarrow dec = 0 \quad \text{lub} \quad c = 1 \wedge d = 0 \Rightarrow dec = 0$$

$$b = 2 \wedge c = 2 \Rightarrow dec = 0$$

$$b = 0 \Rightarrow dec = 1$$

$$c = 1 \wedge d = 1 \Rightarrow dec = 2$$

Klasyfikacja oparta na wsparciu

- ▶ $Rules$ = zbiór reguł z jednoznaczną decyzją
- ▶ U_{trn} = zbiór przykładów treningowych
- ▶ u : obiekt do klasyfikacji
- ▶ Klasyfikacja przez maksymalizację wsparcia

$$Rules(u) = \{\alpha \Rightarrow dec = \beta : u \text{ spełnia } \alpha\}$$
$$\max \arg \beta | \{y \in U_{trn} : \exists \alpha \Rightarrow dec = \beta \in Rules(x) (y \text{ spełnia } \alpha \wedge dec(y) = \beta)\}|$$

Przykład

	a	b	c	d	dec
x_1	0	2	1	0	0
x_2	1	2	2	1	0
x_3	2	0	2	1	1
x_4	0	2	1	1	2
u	0	0	2	1	?

► **Reguły:**

$$b = 2 \wedge c = 1 \wedge d = 0 \Rightarrow dec = 0$$

$$b = 2 \wedge c = 2 \wedge d = 1 \Rightarrow dec = 0$$

$$b = 0 \wedge c = 2 \wedge d = 1 \Rightarrow dec = 1$$

$$b = 2 \wedge c = 1 \wedge d = 1 \Rightarrow dec = 2$$

► **Po skróceniu:**

$$b = 2 \wedge d = 0 \Rightarrow dec = 0 \quad \text{lub} \quad c = 1 \wedge d = 0 \Rightarrow dec = 0$$

$$b = 2 \wedge c = 2 \Rightarrow dec = 0$$

$$b = 0 \Rightarrow dec = 1$$

$$c = 1 \wedge d = 1 \Rightarrow dec = 2$$

► $Rules(u) = \{b = 0 \Rightarrow dec = 1\}$

► Zatem u jest klasyfikowany z $dec = 1$ z maksymalnym wsparciem 1.

Dziękuję za uwagę

Sztuczna Inteligencja

Soma Dutta

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

Wykład - 1: Wprowadzenie

Semestr letni 2022

Sztuczna inteligencja, początkowe podejście Alana Turinga

Alan Turing (1950): Computing machinery and intelligence

- ▶ Czy maszyny mogą myśleć? Czy maszyny mogą zachowywać się inteligentnie?
- ▶ **Turing test** (https://mfiles.pl/pl/index.php/Test_Turinga):
Turing zdefiniował, że maszyna jest inteligentna o ile jest w stanie przejść następujący test:
 - ▶ Osoba A siedzi w zamkniętym pokoju z dwoma terminalami komputerowymi.
 - ▶ Jeden terminal jest podłączony do maszyny, a drugi do osoby B. Osoba A nie wie, który jej terminal jest podłączony do maszyny, a który do osoby B.
 - ▶ Osoba A może zadawać pytania na obu terminalach. Zadanie osoby A polega na podjęciu decyzji, który terminal należy do maszyny.
- ▶ Maszyna jest inteligentna jeśli osoba A nie jest w stanie odróżnić (w określonym czasie) maszyny od osoby B.

- ▶ Obecnie istnieje wiele chatterbotów pracujących online, które zwykle nie są w stanie ukryć swojej sztuczności co okazuje się już po krótkim czasie.
- ▶ (Zobacz: <https://www.spidersweb.pl/2018/06/atom2vec-sztuczna-inteligencja.html>)

Co to oznacza sztuczna inteligencja

Obejmuje różne aspekty: wiedza, wnioskowanie, język, rozumienie, uczenie - to główne wśród nich.

- ▶ Zrozumienie różnych aspektów inteligencji
- ▶ Próby przedstawiania tego w jakiś matematyczny sposób
- ▶ Budowanie systemów, np. modelowanych w postaci agentów, które mogą zachowywać się intelligentnie jak ludzie.

Z raportu Unii Europejskiej



Brussels, 19.2.2020
COM(2020) 65 final

WHITE PAPER

On Artificial Intelligence - A European approach to excellence and trust

- ▶ Sztuczna inteligencja odnosi się do systemów, które wykazują inteligentne zachowanie poprzez analizowanie ich środowiska i podejmowanie akcji, do pewnego stopnia w sposób autonomiczny, w celu uzyskania specyficznych celów.

Z jednego artykułu napisanego przez znanych autorów

- ▶ The Mathematics of Learning: Dealing with Data: Tomaso Poggio and Steve Smale
- ▶ Systemy sztucznej inteligencji to systemy oprogramowania (mogą to być również systemy sprzętowe) zaprojektowane przez ludzi, które mając do zrealizowania złożony cel, działają w wymiarze fizycznym i cyfrowym przez postrzeganie swego środowiska, poprzez pozyskiwanie danych, interpretowanie strukturalnych i niestrukturalnych danych, wnioskowanie o wiedzy lub przetwarzanie informacji wywodzonych z danych i decydowanie o wyborze najlepszych akcji do wykonania w celu osiągnięcia celu. Systemy sztucznej inteligencji mogą używać reguł symbolicznych lub wyuczać się modeli numerycznych, mogą również adaptować swoje zachowanie przez analizę wyniku ich poprzedniego oddziaływania na środowisko.

Trochę historii: Rozwój różnych aspektów sztucznej inteligencji

Okres	Rozwój
1930-1950	(1930) Kurt Gödel: Udowodniono pewne twierdzenia dotyczące wnioskowania logicznego, znane jako 'twierdzenie o完备性 of logiki pierwszego rzędu', (1936) Alan Turing: 'problemu stopu programu' (1950) Newell, Simon: (Na podstawie twierdzenia Goedla) pierwszy automatyczny system dowodzenia twierdzeń (1940) McCulloh, Pitts: Przedstawiono pierwszy model matematyczny dla sieci neuronowych (1950) Artykuł Turinga: Computing Machinery and Intelligence
1956	Spotkanie w Dartmouth: powstaje termin 'Sztuczna Inteligencja' McCarthy: Wprowadzono LISP, język, który może przetwarzanie struktury symboliczne
1960 - 1990	(1965) Lotfi A. Zadeh: Zbiór rozmyte (1970) PROLOG, Język programowania został wprowadzony Wykazano, że sieci neuronowe są w stanie uczyć się na przykładach treningowych Nettalk, system, który był w stanie nauczyć się mowy z przykładowych tekstów (w oparciu o rozpoznanie wzorców) System hybrydowy łączący sieć neuronową i logikę rozmytą (1982) Teoria zbiorów przybliżonych, Z. Pawlak
1990 - dotychczas	Eksploracja danych, Data science - ma na celu pozyskiwanie informacji z dużych zbiorów danych i bazy-wiedzy Rozproszone autonomiczne agenty, których celem jest rozwiązywanie problemów we współpracy wielu agentów

Co rozumiemy przez agenta

- ▶ Agent to obiekt, który może **postrzegać swoje środowisko za pomocą sensorów** i **oddziaływać na to środowisko za pomocą aktuatorów**.
- ▶ Na przykład:
 - ▶ Agent ludzki (Human agent) ma oczy, uszy i inne narządy zmysłów jako **sensory** i dlonie, nogi, drogi głosowe jako **aktuatory**.
 - ▶ Agent robota (Robotic agent) ma kamerę, dalmierze na podczerwień jako **sensory** i różne silniki jako **aktuatory**.
 - ▶ Agent oprogramowania (Software agent) za pomocą **naciśnięć klawiszy, zawartości plików, pakietów sieciowych** realizuje zadania sensorów i przez **wyświetlanie na ekranie, tworzenie zapisów plików, wysyłanie pakietów sieciowych** realizuje zadania aktuatorów.

Postrzeganie środowiska przez agenta

Duża dziedzina: Nauki kognitywne (Cognitive Science)

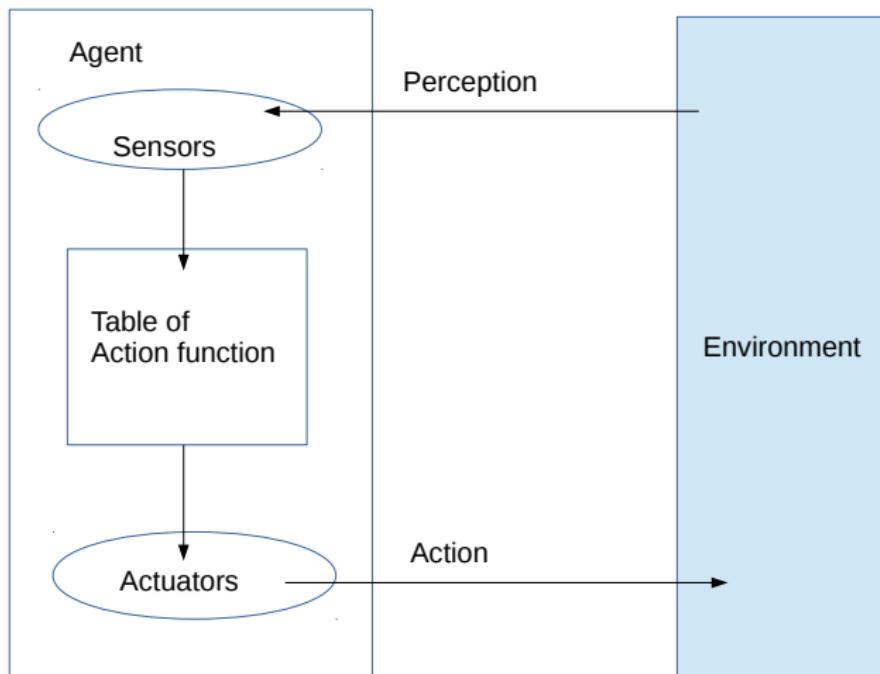
- ▶ **Percept:** Oznosi się to do danych percepcyjnych agenta w danym momencie. Jest to wejście, które agent postrzega w danym momencie.
- ▶ **Sekwencja percepacji:** Oznacza pełną historię postrzeganą przez agenta.

Działanie agenta

- ▶ Wybór działania przez agenta zależy od całej dotychczas obserwowanej sekwencji percepacji.
- ▶ Matematycznie opisuje to funkcja agenta, która przypisuje akcję do zadanej sekwencji percepacji.
- ▶ Funkcję tę można również przedstawić jako tabelę ze wszystkimi możliwymi sekwencjami percepacji (o skończonej długości) i odpowiednimi działaniami podejmowanymi przez agenta.

Pierwszy krok do zaprojektowania inteligentnego agenta

- ▶ **Task environment** (Środowisko zadań): Opis środowiska zadań według PEAS (Performance (Wydajność), Environment (Środowisko), Actuators (Aktuator), Sensors (Sensory)).



Przykład: inteligentny agent

Świat odkurzacza (Vaccum cleaner world):

- ▶ Założmy, że ma dwie lokalizacje: kwadrat A i kwadrat B.
- ▶ Odkurzacz rozpoznaje, w którym kwadracie znajduje się i czy na tym miejscu jest brud.
- ▶ Może wybrać ruch w lewo, ruch w prawo, wyssać brud lub nie robić nic.
- ▶ Prosta funkcja agenta: jeśli bieżący kwadrat jest brudny, to ssać; w przeciwnym razie przejdź na drugi kwadrat.

Odkurzacz można modelować jak poniżej



Sekwencja percepcji	Akcja
[A, Czysty]	W prawo
[A, Brudny]	Ssać
[B, Czysty]	W Lewo
[B, Brudny]	Ssać
[A, Czysty] [A, Czysty]	W prawo
[A, Czysty] [A, Brudny]	Ssać
:	:

Czego więcej potrzebujemy do modelowania inteligentnego agenta

- ▶ **Racjonalność agenta:** Dla każdej możliwej sekwencji percepacji racjonalny agent powinien wybrać działanie, które maksymalizuje miarę jego wydajności, biorąc pod uwagę argumenty dostarczone przez sekwencję percepacji i bazę wiedzy agenta.
- ▶ Więc racjonalność zależy od:
 - (i) miary wydajności
 - (ii) wcześniejszej wiedzy agenta na temat środowiska
 - (iii) działań, które agent może wykonać
 - (iv) dotychczasowej sekwencji percepji agenta

Różne rodzaje intelligentnych agentów

- ▶ Task environment: Podział z uwagi na działanie na środowisko zadań
 - ▶ W pełni obserwowlany (Fully observable): Jeśli sensory agenta dają pełny stan środowiska (np. krzyżówka)
 - ▶ Częściowo obserwowlane (Partially observable): Jeśli brakuje niektórych warunków środowiska z powodu szumu lub niedokładności sensora (np. prowadzenie taksówki)
 - ▶ Nieobserwowlany (Unobservable): Agent może nie mieć sensora
- ▶ Podział z uwagi na działanie na podstawie wiedzy o kolejnym stanie środowiska:
 - ▶ Deterministyczny (Deterministic): Jeśli następny stan środowiska jest całkowicie określony przez aktualny stan i można zaplanować działanie agenta (np. krzyżówka)
 - ▶ Niedeterministyczny (Non-deterministic): Gdy niepewność środowiska można opisać jedynie możliwymi wynikami (np. kółko i krzyżyk)
 - ▶ Stochastyczny (Stochastic): Gdy niepewność środowiska jest opisana poprzez przypisanie prawdopodobieństwa do możliwych wyników (np. diagnoza medyczna)

W		S		
				O

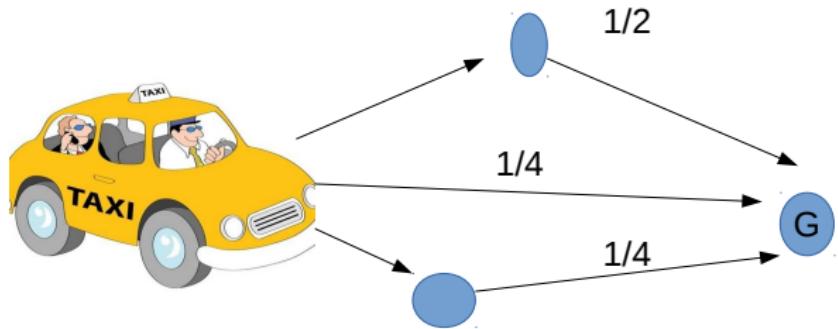
WARSAW O L E

DETERMINISTIC

O		
	X	

NON-DETERMINISTIC

STOCHASTIC



Różne rodzaje inteligentnych agentów

- ▶ Single agent vs. multiagent: Klasyfikacja na podstawie działania innych agentów
 - ▶ Model konkurencji (Competitive): Gdy jeden agent próbuje zmaksymalizować swoją miarę wydajności w oparciu o zachowanie innego agenta (np. gry w szachy)
 - ▶ Model współpracy (Cooperative): Gdy współpraca między agentami poprawia wydajność każdego z nich (np. Interaktywny nauczyciel języka angielskiego)
- ▶ Klasyfikacja agentów na podstawie warunków akcji
 - ▶ Epizodyczny (Episodic): Gdy doświadczenie agenta jest podzielone na epizody atomowe (odcinki), a działanie w bieżącym odcinku nie zależy od działań w poprzednich odcinkach. (np. analiza obrazu)
 - ▶ Sekwencyjny (Sequential): Gdy działanie w bieżącym odcinku zależy od działań w poprzednim odcinku (np. gry w szachy)
 - ▶ Statyczny vs. dynamiczny (Static vs. dynamic): Jeśli środowisko może się zmienić, gdy agent rozważa/wykonuje akcję, to agent jest dynamiczny (np. jazda taksówka); w przeciwnym razie jest statyczny (np. krzyżówka)

W		S	
			O

R1: city

C5: animal

WARSAW

O
L
E

DETERMINISTIC SINGLE-AGENT

A 3x3 grid with the following values:

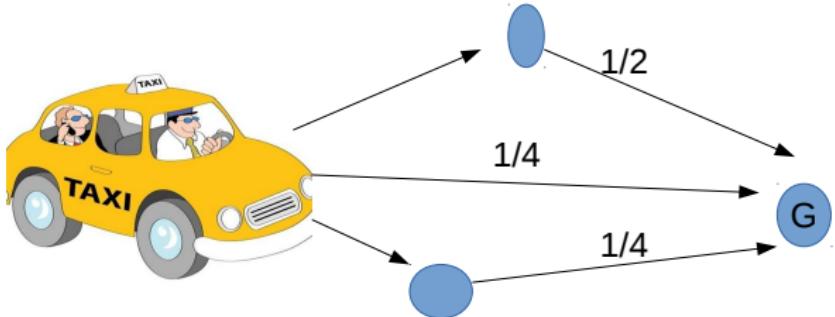
O		
		X

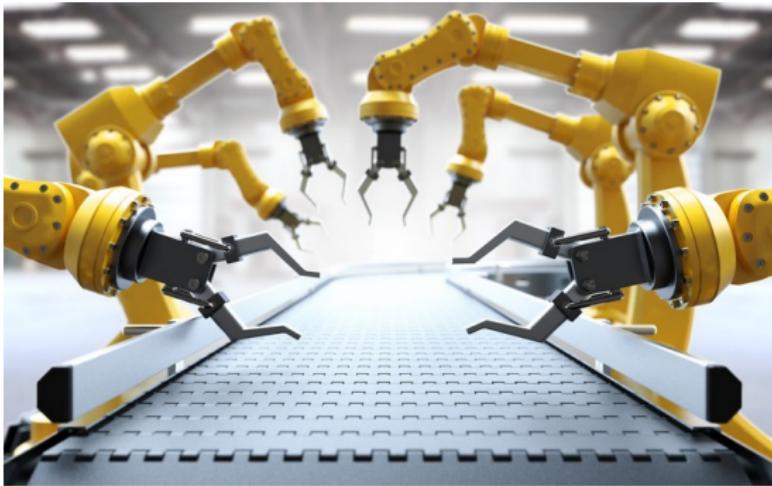
NON-DETERMINISTIC

MULTIAGENT-COMPETITIVE

STOCHASTIC

MULTIAGENT-COOPERATIVE





Part-picking robot

FULLY OBSERVABLE

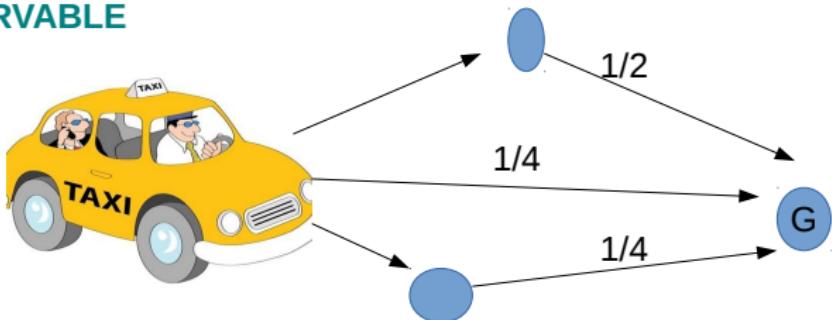
EPISODIC

STATIC

PARTIALLY OBSERVABLE

SEQUENTIAL

DYNAMIC



Przykłady środowisk zadań

Task environment	Observable	Deterministic	Agent	Episodic	Static
Crossword puzzle (Krzyżówka)	Fully	Deterministic	Single	Sequential	Static
Chess with a clock (Szachy)	Fully	Deterministic	Multi	Sequential	Semi
Poker	Partially	Deterministic	Multi	Sequential	Static
Backgammon (Trik-trak)	Fully	Deterministic	Multi	Sequential	Static
Taxi driving (Jazda taksówką)	Partially	Stochastic	Multi	Sequential	Dynamic
Medical diagnosis (Diagnoza medyczna)	Partially	Stochastic	Single	Sequential	Dynamic
Image analysis (Analiza obrazu)	Fully	Deterministic	Single	Episodic	Semi
Part-picking robot (Robot do pobierania części)	Partially	Stochastic	Single	Episodic	Dynamic
Refinery controller (Kontroler rafinerii)	Partially	Stochastic	Single	Sequential	Dynamic
Interactive English tutor (Interaktywny nauczyciel języka angielskiego)	Partially	Stochastic	Multi	Sequential	Dynamic

Funkcja agenta kontra program agenta

- ▶ Funkcja agenta opisana jest regułami w tabeli i Program agenta implementuje tę funkcję agenta
- ▶ Program agenta jest więc związany z fazą budowy inteligentnego agenta
- ▶ Program agenta możemy przedstawić w następujący sposób

```
function TABLE-DRIVEN-AGENT(percept) returns an action
  persistent: percepts, a sequence, initially empty
            table, a table of actions, indexed by percept sequences, initially fully specified
  append percept to the end of percepts
  action → LOOKUP(percept, table)
  return action
```

Składniki programu agenta

Składniki umożliwiają odpowiedzi na następujące pytania:

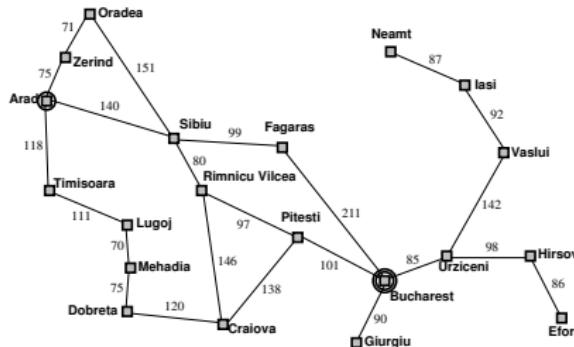
- ▶ Jak teraz wygląda świat? ([Obecne środowisko](#))
- ▶ Jakie działanie należy wybrać? ([Wybór akcji](#))
- ▶ Co będzie wynikiem działania? ([Wpływ działania](#))

W oparciu o te składniki program agenta może być trzech typów:

- (i) Reprezentacja atomowa (Atomic representation)
- (ii) Reprezentacja rozproszona (Factored representation)
- (iii) Reprezentacja ustrukturyzowana (Structured representation)

Reprezentacja atomowa

- ▶ Każdy stan świata jest niepodzielny i nie ma wewnętrznej struktury
- ▶ Na przykład, znalezienie trasy dojazdu z jednego miejsca do drugiego
- ▶ W szczególności potrzebujemy
 - ▶ algorytmów wyszukiwania (Searching algorithms),
 - ▶ strategii gier (Game strategies)



Sformułowanie problemu:

- stan początkowy: początkowy stan przed rozwiązyaniem problemu
- cel: stan docelowy lub formuła oceniająca, czy dany stan spełnia cel
- rozwiązanie: ciąg akcji prowadzący od stanu początkowego do celu
- koszt rozwiązania: funkcja oceny kosztu rozwiązania równa sumie kosztów poszczególnych akcji występujących w rozwiążaniu

Rozwiązania o niższym koszcie są lepsze niż rozwiązania o wyższym koszcie.

Reprezentacja rozproszona

- ▶ W tym kontekście, poza samą lokalizacją miasta, skupimy się również na czynnikach, takich jak koszt dotarcia tam - z uwzględnieniem pieniędzy, odległości, czasu.
- ▶ Tak więc każdy stan zawiera zestaw zmiennych lub atrybutów, z których każdy może mieć określoną wartość
- ▶ Tutaj warto wymienić i poznać:
 - ▶ [Algorytmy spełniające więzy](#) (Constrained satisfaction algorithms)
 - ▶ [Rachunek zdań](#) (Propositional logic)
 - ▶ [Planowanie](#) (Planning)
 - ▶ [Sieci bayesowskie](#) (Bayesian network)
 - ▶ [Algorytmy uczenia maszynowego](#) (Machine learning algorithms)

Reprezentacja ustrukturyzowana

- ▶ W tym kontekście mamy do czynienia z sytuacjami, w których oprócz indywidualnych informacji o różnych stanach, mamy różne relacje między stanami
- ▶ Tutaj warto wymienić i poznać:
 - ▶ Rachunek pierwszego rzędu (First order logic)
 - ▶ Uczenie się oparte na wiedzy (Knowledge-based learning)
 - ▶ Przetwarzanie języka naturalnego (Processing of natural language)

Łącząc różne aspekty, skoncentrujemy się na następujących tematach

1. Algorytmy wyszukiwania (w tym heurystyczne)
2. Strategie w grach
3. Wnioskowanie w logice
4. Sieci neuronowe
5. Zbiory rozmyte
6. Systemy decyzyjne i uczące się: Zbiory przybliżone, reguły decyzyjne
7. Sieci bayesowskie

Bibliografia

- ▶ Artificial Intelligence: A Modern Approach - Stuart Russell and Peter Norvig
- ▶ Machine Learning - Tom M. Mitchell
- ▶ Wstęp do Sztucznej Inteligencji - Mariusz Flasiński

Metoda oceny

- ▶ W ćwiczeniach będzie pięć zestawów zadań. Za każdy zestaw można uzyskać 2 punkty.
- ▶ Rozwiązania każdego zestawu zadań będzie można złożyć w ciągu trzech tygodni. Dwie osoby mogą utworzyć grupę, aby pracować nad zestawami zadań i przedstawiać swoje rozwiązania jako grupa. Za opóźnienia w przesłaniu rozwiązań zadań i kopiowanie ich z innych prac będą odejmowane punkty.
- ▶ Na niektórych ćwiczeniach będą przyprowadzane krótkie testy. Szczegóły dotyczące testów zostaną podane na tydzień przed testem.
- ▶ Ocena za ćwiczenia będzie wystawiona na podstawie średniej z wszystkich uzyskanych punktów.
- ▶ Ta uzyskana ocena za ćwiczenia będzie proponowana również jako ocena za wykład.
- ▶ Do zaliczenia przedmiotu wymagane jest uzyskanie pozytywnej oceny. Jeśli ktoś nie zaliczy przedmiotu na podstawie powyższych kryteriów lub zechce poprawiać ocenę, to może zdawać egzamin pisemny.

Dziękuję za uwagę

Sztuczna Inteligencja

Soma Dutta

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

Wykład - 2: Algorytmy Przeszukiwania

Semestr letni 2022

Podsumowanie ostatniego wykładu

- ▶ Komponenty inteligentnego agenta
 - ▶ Opis środowiska zadań agenta (PEAS)
 - ▶ Funkcja agenta i program agenta
- ▶ Różne klasyfikacje programu agenta
 - (i) Reprezentacja atomowa (Atomic representation)
 - (ii) Reprezentacja wielo-składnikowa/ rozposzona (Factored representation)
 - (iii) Reprezentacja ustrukturyzowana (Structured representation)

Algorytmy Przeszukiwania

- ▶ Algorytmy przeszukiwania w najprostszym przypadku związane z kategorią **reprezentacji atomowej** - np. znalezienie **najkrótszej trasy** między dwoma miastami
- ▶ Znalezienie **możliwie najkrótszej trasy** z jednego miasta do drugiego może zawierać połączenia, czas, koszty itp - Tutaj potrzebujemy algorytmów przeszukiwania z **reprezentacją wielo-składnikową** : Często algorytmy dla tego typu problemów nazywamy **Algorytmami spełniającymi wieży** (Constrained satisfaction algorithms)
- ▶ Dodajmy do poprzedniej sytuacji, że w jednym mieście na możliwie najkrótszej trasie nagle zdarzył się wypadek i wszystkie połączenia są opóźnione - znalezienia rozwiązania tej sytuacji nie da się opisać zestawem wcześniej zdefiniowanych atrybutów, tak jak w przypadku reprezentacji rozproszonych : Potrzebujemy tutaj relacyjnego opisu takich sytuacji, który jest związany z **ustrukturyzowanymi reprezentacjami**

Prosty agent odruchowy (Simple reflex agent)



Sekwencja percepcji	Akcja
[A, Czysty]	W prawo
[A, Brudny]	Ssuć
[B, Czysty]	W Lewo
[B, Brudny]	Ssuć
[A, Czysty] [A, Czysty]	W prawo
[A, Czysty] [A, Brudny]	Ssuć
:	:

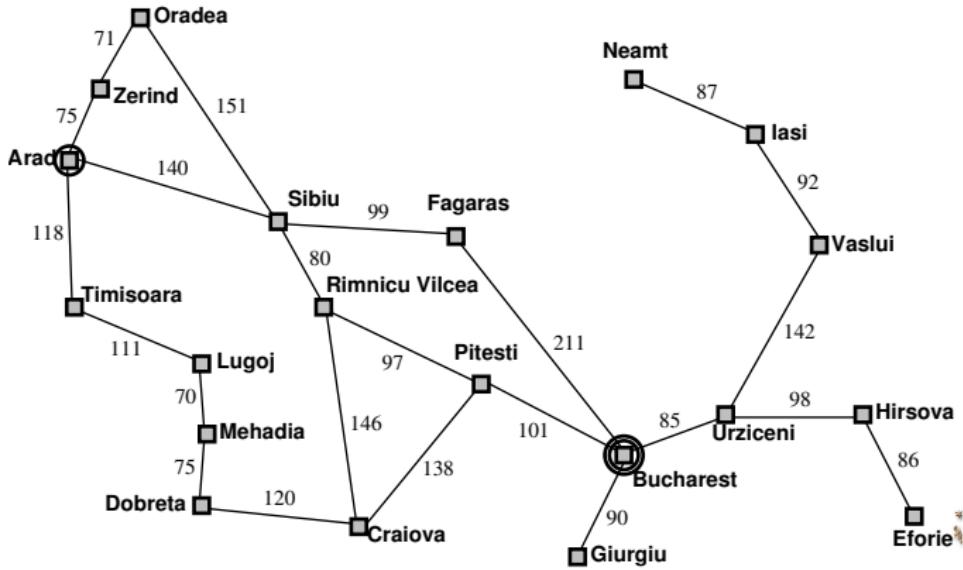
Algorytmy przeszukiwania w prostym przypadku

- ▶ Agent oparty na celach (Goal-based agent) (agent bazujący na celu)
- nazywamy agentem do rozwiązywania problemów (problem solving agent)
- ▶ Budowanie agenta opartego na celach
 - ▶ Sformułuj problem (cel): Zestaw stanów spełniających cel
 - ▶ Przeszukaj Ścieżkę od stanu początkowego problemu do jednego ze stanów spełniających cel
 - ▶ Wykonaj algorytm przeszukiwania

Sformułowanie problemu

- ▶ Stan początkowy (Initial state): $In(s)$
- ▶ Opis możliwych działań dostępnych dla agenta: $ACTION(s)$
- ▶ Relacja przejścia opisujący wynik każdej akcji z określonego stanu :
 $RESULT(s, a)$ - zwraca stan, do którego można dotrzeć za pomocą jednej akcji - ten stan nazywany jest **następnikiem poprzedniego stanu** (successor of the previous state)
- ▶ (**Goal test**) Test celu w celu ustalenia, czy stan jest stanem celu
- ▶ (**Path cost**) Funkcja kosztu ścieżki przypisująca wartość liczbową każdej ścieżce. Koszt ścieżki może być sumą kosztów każdego pojedynczego działania ($c(s, a, s')$) tzn. kroku na ścieżce.

(**State space of the problem**) Przestrzeń stanu problemu jest zbiorem wszystkich stanów, które można osiągnąć ze stanu początkowego za pomocą sekwencji działań

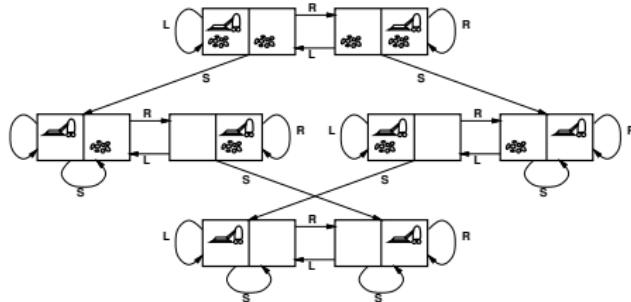


- ▶ Stan początkowy: $In(Arad)$
- ▶ $Action(Arad) = \{\rightarrow^Z, \rightarrow^S, \rightarrow^T\}$
- ▶ Stany następne Aradu: $\{Zerind, Sibiu, Timisoara\}$
- ▶ $Result(Arad, \rightarrow^T) = Timisoara$
- ▶ $c(Arad, \rightarrow^T, Timisoara) = 118$
- ▶ Stan docelowy: Bucharest

Rozwiązywanie problemu przeszukiwania

- ▶ (Solution) Rozwiązaniem jest sekwencja działań, która poczynając od stanu początkowego prowadzi do stanu docelowego (stanu spełniającego cel)
- ▶ (Optimal solution) Optymalne rozwiązanie to rozwiązanie, które ma najmniejszy koszt pośród wszystkich kosztów rozwiązań
- ▶ Algorytm przeszukiwania został zaprojektowany w celu znalezienia możliwej sekwencji działań, od stanu początkowego do stanu docelowego

Przykład: Odkurzacz



- stany: stan pomieszczeń (czysto/brudno) i lokalizacja robota
- akcje: *Lewo, Prawo, Odkurzaj, NicNieRob*
- cel: czysto
- koszt rozwiązania: 1 dla każdej akcji (0 dla *nicNieRob*)

Liczba stanów: $2 \times 2^2 = 8$

W większym środowisku z n lokalizacjami: $n \times 2^n$ stany

Przeszukiwanie drzewa

- ▶ Przeszukiwanie drzewa (Tree search):
 - ▶ Możliwe sekwencje akcji tworzą drzewo, którego korzeniem jest stan początkowy, a gałęzie odpowiadają różnym akcjom
 - ▶ Algorytm przeszukiwania rozpoczyna się od sprawdzenia, czy węzeł główny jest węzłem celu
 - ▶ Następnie proces realizowany przez algorytm rozwija kolejny zestaw węzłów, do których można dotrzeć z poprzedniego węzła - Zestaw węzłów dostępnych do rozszerzenia nazywa się granicą (frontier)
 - ▶ Proces rozszerzania granicy trwa aż do znalezienia rozwiązania lub do momentu gdy nie ma już stanów do rozszerzenia
 - ▶ Mogą istnieć różne algorytmy przeszukiwania drzewa w zależności od strategii wyboru węzła do rozwinięcia

Pseudokod dla przeszukiwania drzewa

```
function TREE-SEARCH(problem) returns a solution, or failure
```

```
initialize the frontier with initial state of the problem
```

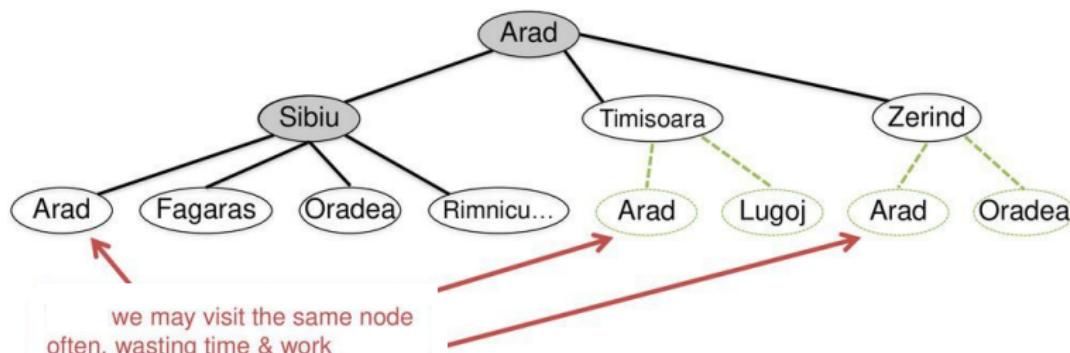
```
Loop do
```

```
    if the frontier is empty then return failure
```

```
    choose a leaf-node and remove it from frontier
```

```
    if the node contains a goal state return the corresponding solution
```

```
    expand the chosen node, adding the resulting node to the frontier
```



Przeszukiwanie grafów

- ▶ Przeszukiwanie grafu pozwala uniknąć powtarzania odwiedzania stanów

```
function GRAPH-SEARCH(problem) returns a solution, or failure
```

```
    initialize the frontier with initial state of the problem
```

```
    initialize the explored set to be empty
```

```
    Loop do
```

```
        if the frontier is empty then return failure
```

```
        choose a leaf-node and remove it from frontier
```

```
        if the node contains a goal state return the corresponding solution
```

```
        add the node to the explored set
```

```
        expand the chosen node, adding the resulting node to the frontier
```

```
            only if not in the frontier or explored set
```



A sequence of search trees generated by a graph search

At each stage, we have extended each path by one step. Notice that at the third stage, the northernmost city (Oradea) has become a dead end: both of its successors are already explored via other paths.

Struktura danych potrzebna do algorytmów przeszukiwania

- ▶ Algorytmy przeszukiwania korzystają z odpowiednich struktur danych, aby móc śledzić budowane drzewo lub graf
- ▶ Dla każdego węzła n wykorzystywane są struktury danych, które przechowujące następujące informacje
 - ▶ $n.\text{STATE}$: Stan w przestrzeni stanów, któremu odpowiada węzeł n
 - ▶ $n.\text{PARENT}$: Węzeł w drzewie wyszukiwania, z którego generowane jest n
 - ▶ $n.\text{ACTION}$: Działanie (akcja) zastosowane w węźle nadziednym w celu osiągnięcia węzła n
 - ▶ $n.\text{PATH-COST}$: $g(n)$ - Koszt ścieżki ze stanu początkowego do węzła n
- ▶ Dlatego węzeł zawiera informacje o stanie węzła, takie jak węzeł nadziedny, dla którego generowane jest działanie, oraz koszt osiągnięcia tego stanu

Wykorzystanie kolejki (queue) w algorytmach przeszukiwania

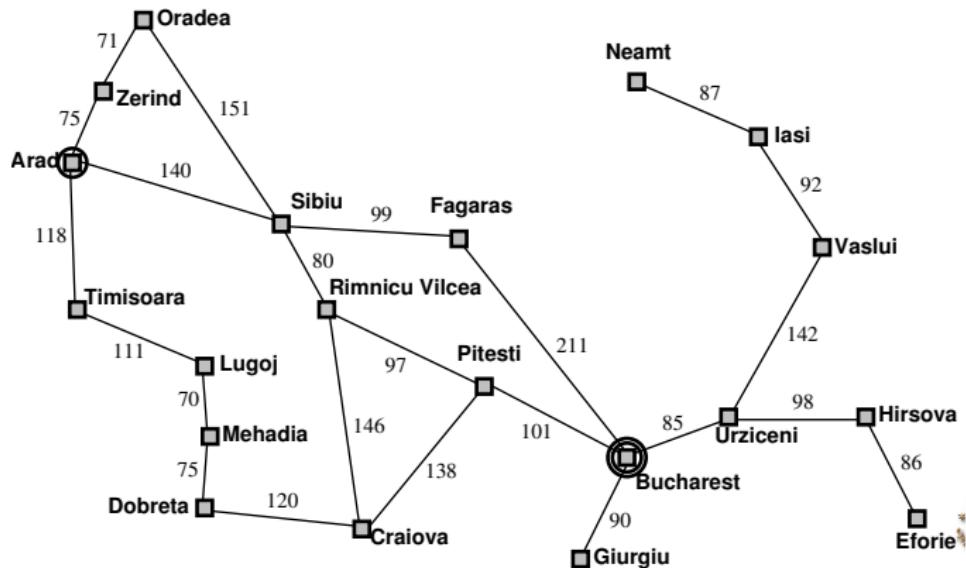
- ▶ Założymy, że musimy przechowywać 'frontier' w taki sposób, aby algorytmy przeszukiwania mogły łatwo wybrać następny węzeł do rozszerzenia zgodnie z preferowaną strategią
- ▶ Różne warianty kolejki są używane bazując na preferowanej strategii
 - ▶ **FIFO** (First In First Out): zwraca najstarszy element w kolejce
 - ▶ **LIFO** (Last In First Out): zwraca najnowszy element w kolejce
 - ▶ **PRIORITY QUEUE**: element o najwyższym priorytecie pojawia się w określonej kolejności przy tej strategii
- ▶ Możemy korzystać ze standardowych funkcji z dowolnej kolejki. Np.
 - ▶ `Empty(queue)?`: zwraca prawdę tylko wtedy, gdy w kolejce nie ma więcej elementów
 - ▶ `Pop(queue)`: usuwa pierwszy element kolejki i zwraca go
 - ▶ `Insert(element, queue)`: wstawia element i zwraca nową kolejkę

Aby śledzić eksplorowany zbiór

- ▶ Eksplorowany zbiór może być zaimplementowany przez '[hash table](#)', która może skutecznie sprawdzać powtarzalność stanów

Różne aspekty pomiaru wydajności

- ▶ Wydajność (jakość) algorytmu można mierzyć na cztery sposoby:
 - ▶ Kompletność (Completeness): Czy algorytm gwarantuje znalezienie rozwiązania, o ile istnieje rozwiązanie?
 - ▶ Optymalność (Optimality): Czy algorytm znajduje optymalne rozwiązanie?
 - ▶ Złożoność czasowa (Time complexity): Ile czasu zajmuje znalezienie rozwiązania?
 - ▶ Złożoność pamięciowa (Space complexity): Ile pamięci potrzeba przy przeszukiwaniu?
- ▶ Złożoności czasowa i pamięciowa zależą od następujących czynników:
 - ▶ Czynnik rozgałęziający (Branching factor): b - maksymalna liczba następników dowolnego węzła
 - ▶ Głębokość (Depth): d - głębokość najbardziej płytkiego węzła docelowego
 - ▶ Maksymalna długość (Maximum length): m - maksymalna długość dowolnej ścieżki w przestrzeni stanów



- ▶ $b = 4$
- ▶ $d = 3$
- ▶ $m = 11 !$

Koszt przeszukiwania

- ▶ Aby ocenić jakość algorytmu przeszukiwania, możemy wziąć pod uwagę koszt przeszukiwania.
- ▶ Koszt przeszukiwania może obejmować złożoność czasową i pamięciową.
- ▶ Można również rozważyć całkowity koszt, który obejmuje koszt przeszukiwania i koszt ścieżki rozwiązania.

Podział strategii przeszukiwania

- ▶ Ślepe (Brutalne) strategie przeszukiwania (Uninformed/blind search strategies):
 - ▶ W tych strategiach nie są dostępne żadne dodatkowe informacje oprócz definicji problemu
 - ▶ Mogą generować następcy i odróżniać stan docelowy od innych stanów
- ▶ Heurystyczne strategie przeszukiwania (Informed/heuristic search strategies):
 - ▶ W tych strategiach poza definicją problemu wiadomo, czy jeden stan celu jest bardziej obiecujący niż drugi stan celu
 - ▶ Ponieważ heurystyki te mogą mieć zakodowaną wiedzę na temat konkretnego problemu, mogą skuteczniej znajdować rozwiązania

Brutalne (Ślepe) strategie przeszukiwania

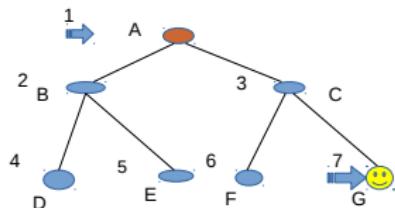
- ▶ **Breadth-first search** (Przeszukiwanie wrzec) przeszukiwania grafowego
 - ▶ najpierw rozwijany jest węzeł główny
 - ▶ następnie rozwijane są wszystkie następcy węzła głównego
 - ▶ ogólnie wszystkie węzły na danej głębokości są rozszerzane przed dowolnym węzłem na następnym poziomie
 - ▶ wykorzystywana jest kolejka FIFO
 - ▶ zawsze wybiera najkrótszą ścieżkę do każdego węzła należącego do zbioru wierzchołków bezpośrednio osiągalnych z danego wierzchołka granicy (frontier)
- ▶ **Pomiar wydajności** (Miary jakości)
 - ▶ **Kompletny:** Jeśli najbliższy węzeł celu znajduje się na skończonej głębokości d , algorytm w końcu go znajdzie
 - ▶ **Nieoptimalny:** Najbliższy (do danego wierzchołka) węzeł docelowy niekoniecznie musi być optymalny. Jest optymalny, gdy koszt ścieżki nie jest funkcją malejącą (np. wszystkie działania mają taki sam koszt)

- ▶ Złożoność czasowa: $b + b^2 + \dots + b^d = O(b^d)$
 - ▶ Złożoność pamięciowa: $O(b^{d-1})$ ponieważ każdy węzeł wygenerowany dla ostatniego poziomu jest przechowywany w pamięci jako zbiór eksplorowany

```

function BREADTH-FIRST-SEARCH(problem) returns solution or failure
node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
frontier ← a FIFO queue with node as the only element
explored ← an empty set
loop do
  if EMPTY?(frontier) then return failure
  node ← POP(frontier) /* chooses the shallowest node */
  add node.STATE to explored
  for each action in problem.ACTIONS(node.STATE) do
    child ← CHILD-NODE(problem, node, action)
    if child.STATE is not in frontier or explored then
      if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
      frontier ← INSERT(child, frontier)

```



Brutalne strategie przeszukiwania

► Uniform cost search:

- ▶ Zamiast rozszerzać ścieżkę do najpłytszego węzła, rozszerza węzeł n o ścieżkę o najniższym koszcie $g(n)$
- ▶ Odbywa się to poprzez przechowywanie 'frontiers' jako kolejki priorytetowej (PRIORITY QUEUE) uporządkowanej według g
- ▶ Test celu jest dodawany do węzła, gdy jest on wybierany do rozwinięcia, a nie kiedy jest generowany po raz pierwszy
- ▶ Dodawany jest test, aby sprawdzić, czy istnieje lepsza ścieżka do węzła znajdującego się obecnie na 'frontier'

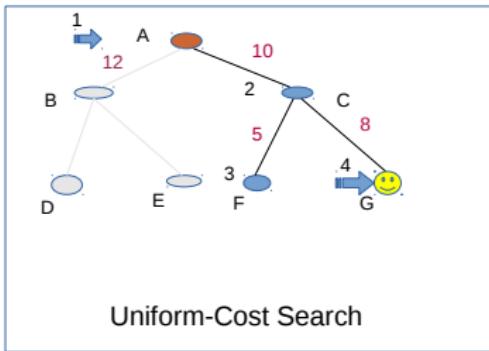
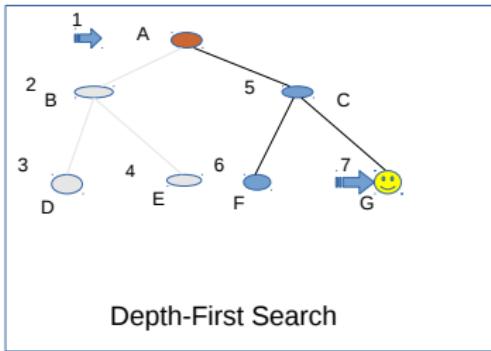
► Pomiar wydajności

- ▶ Optymalny
- ▶ Niekompletny: Ponieważ nie ma znaczenia liczba kroków na ścieżce, jeśli istnieje nieskończona pętla z działaniami o zerowym koszcie, program może utknąć
- ▶ Złożoność czasowa i pamięciowa: Jeśli C jest kosztem optymalnego rozwiązania, a ϵ najmniejszym kosztem dla każdego działania, to złożoność czasowa i pamięciowa jest $O(b^{1+\lfloor \frac{C}{\epsilon} \rfloor})$.

Brutalne strategie przeszukiwania

► Depth-first search (Przeszukiwanie w głąb)

- ▶ Rozszerza najgłębszy węzeł na bieżącej 'frontier'.
- ▶ Natychmiast przechodzi do najgłębszego poziomu, w którym węzły nie mają następców, a potem przechodzi do następnego najgłębszego węzła, który wciąż nie został odwiedzony.
- ▶ Używa kolejkę LIFO.



Pomiar wydajności

- ▶ **Kompletny:** Kompletny gdy jest używany do przeszukiwania grafów w skończonej przestrzeni, i niekompletny gdy używana jest wersja przeszukiwania drzewa. W nieskończonej przestrzeni stanów, jeśli napotkamy nieskończoną ścieżkę nieposiadającą węzła celu, wówczas algorytm przeszukiwania nie byłby kompletny.
- ▶ **Nieoptymalny**
- ▶ **Złożoność czasowa:** Może wygenerować wszystkie węzły $O(b^m)$, co może być znacznie większe niż rozmiar przestrzeni stanów. Również m może być znacznie większe niż d .
- ▶ **Złożoność pamięciowa:** Przechowuje tylko jedną ścieżkę od korzenia do liścia i innego nieodwiedzonego 'syna'. Wymaga tylko przechowywania $O(bm)$ węzłów.

Brutalne strategie przeszukiwania

- ▶ **Depth-limited search:** Dodano wstępnie zdefiniowane ograniczenia na głębokość l
 - ▶ **Niekompletny:** Gdy $l < d$
 - ▶ **Nieoptymalny:** Gdy $l > d$
 - ▶ **Złożoność czasowa:** $O(b^l)$.
 - ▶ **Złożoność pamięciowa:** $O(bl)$.
- ▶ **Iterative deepening search:** Określa ograniczenie l poprzez iteracyjne poszukiwanie najlepszego ograniczenia, zaczynając od $l = 0$, aż do osiągnięcia celu gdy $l = d$.
- ▶ **Kompletny:** Gdy b jest skończone.
- ▶ **Nieoptymalny:** Gdy koszt ścieżki jest niemalejącą funkcją węzła głębokości.
- ▶ **Złożoność czasowa:** $O(b^d)$.
- ▶ **Złożoność pamięciowa:** $O(bd)$.

Brutalne strategie przeszukiwania

- ▶ Bidirectional search (Przeszukiwanie dwukierunkowe)
 - ▶ Wprowadza dwa jednocześnie przeszukiwania - jedno w przód od stanu początkowego, a drugie w tył z jednego z węzłów docelowych.
 - ▶ Kiedy 'frontiers' dwóch przeszukiwań przecinają się, osiągany jest węzeł celu.
 - ▶ Złożoność czasowa jest znacznie mniejsza - $2b^{\frac{d}{2}}$.

Dziękuję za uwagę