Sztuczna inteligencja. O przeszukiwaniu

Paweł Rychlikowski

Instytut Informatyki UWr

3 marca 2021

Pierwsza lista pracowniowa

Zadanie o Panu Tadeuszu

a) **Przeszukujemy** możliwe podziały (sposoby legalnego wstawiania spacji)

b) Można myśleć o tym jako o **modelu** języka: język składa się raczej z długich, niż z krótkich słów

c) Uwaga: konieczny jest algorytm dynamiczny!

Zadanie o Pokerze

a) Najpierw zagadka: Jaki jest związek tego zadania z pasjansem i bombą atomową?

Rozwiązanie zagadki

- Stanisław Ulam pracował w projekcie Manhattan
- Kiedyś, w czasie rekonwalescjencji po operacji układał pasjanse. A te wiadomo, czasem wychodzą, czasem nie.
- Chciał obliczyć prawdopodobieństwo (że pasjans wyjdzie), ale mu nie wychodziło

Wymyślił, że można je obliczyć symulując wiele rozdań na komputerze! (w roku 1947)

Tak powstały metody Monte Carlo (do wykorzystania w wielu miejscach, między innymi w zadaniu o pokerze).

Jeszcze o zadaniu pokerowym

Fragment treści zadania

Sprawdź za pomocą tego programu (wykonując kilka eksperymentów), jak zmienia się prawdopodobieństwo sukcesu, jeżeli pozwolimy Blotkarzowi wyrzucić pewną liczbę wybranych kart przed losowaniem (inaczej mówiąc, pozwalamy Blotkarzowi na skomponowanie własnej talii, oczywiście złożonej z blotek).
Czy potrafisz skomponować zwycięską talię dla Blotkarza (mającą możliwie dużo kart)?

- **ograniczenie:** $p_{\text{Blotkarz wygrywa}} > \frac{1}{2}$,
  cel optymalizacji: maksymalna liczba kart
- Inna możliwość: wspólne kryterium liczbowe (wiele możliwości)
- Optymalizacja wielokryterialna (Optimum w sensie Pareta, będzie zadanie na liście C1)

Oczekiwania wobec rozwiązania

(wracamy do ogólniejszych zagadnień)

- Zupełność: czy program znajdzie drogę do rozwiązania, jeżeli takowa istnieje?
  Czy to jest konieczny warunek użyteczności algorytmu?
- Optymalność: czy będzie ona najkrótsza
- Złożoność czasowa: jak długo będzie trwało szukanie
- Złożoność pamięciowa: ile zużyjemy pamięci

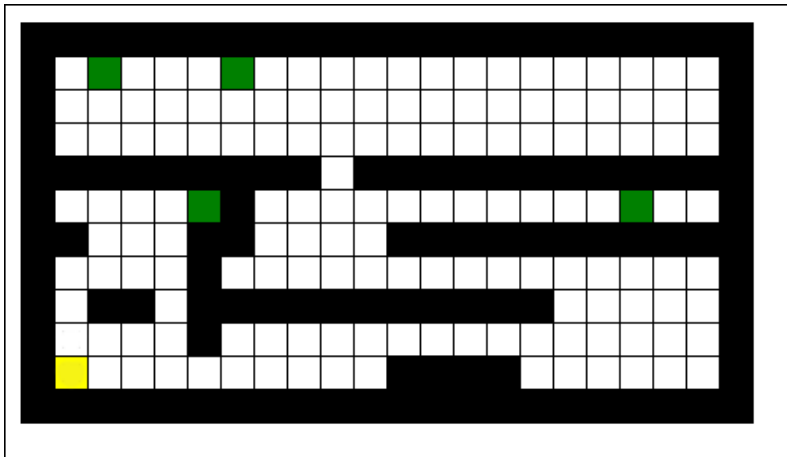
Pytanie

Dlaczego tak ważna jest złożoność pamięciowa?

Drzewa i grafy

- Agent bez pamięci z konieczności operuje drzewem przeszukiwań (bo nie potrafi stwierdzić, że w jakimś stanie już był)
- Najczęściej lepiej modelować świat za pomocą grafu

Labirynt



- Będziemy teraz rozważać różne labirynty, na kwadratowej siatce.
- Labirynt jako problem wyszukiwania:
  - stan – współrzędne pola na którym można stanąć (nie ściany)
  - start – ustalona pozycja w labiryncie (żółta)
  - cel – ustalone pozycje w labiryncie (zielona)
  - model – 4-sąsiedztwo (modulo ściany), akcje to N, W, E, S.
  - koszt – jednostkowy

Labirynt 2. Możliwe modyfikacje

Można wzbogacić przestrzeń stanów w labiryncie

- Dodać drzwi i klucze (czym stanie się stan)?
- Dodać poruszających się (deterministycznie) wrogów (stan?)
- Dodać skrzynie z bronią, apteczki i punkty życia (stan?)

Koniec nagrania I

BFS

BFS = Breadth First Search

Opis

- Mamy 3 grupy stanów: do-zbadania, zbadane i pozostałe.
- Na początku mamy 1 stan do-zbadania: stan startowy
- Stany do zbadania przechowujemy w kolejce FIFO (first-in first out)
- Badanie stanu:
  - Sprawdzenie, czy jest stanem docelowym (jak tak, to **koniec!**)
  - Ustalenie, jakie akcje możemy zrobić w tym stanie, znalezienie nowych stanów do-zbadania

Skrócony opis

Pobieraj stan z kolejki, przetwarzaj, jak kolejka się skończy (nic do-zbadania) to zakończ działanie, (możesz też zakończyć, jak znajdziesz stan docelowy).

DFS

DFS = Depth First Search

Opis

- Stany przetwarzamy w innej kolejności: dzieci aktualnie rozwijanego mają priorytet
- Czyli zamiast FIFO używamy LIFO (List in First out), czyli po prostu stosu.
- Oprócz tego algorytm się nie zmienia.

DLS

DLS = Depth Limited Search

Opis

- Określamy maksymalną głębokość poszukiwania.
- Przeszukujemy w głąb, ale nie rozwijamy węzłów na głębokości większej niż L.
- Wygodnie implementuje się rekurencyjnie (proste ćwiczenie)

Analiza czasowo pamięciowa

- W algorytmach na grafach używa się takich parametrów jak $|V|$ oraz $|E|$ (liczba stanów, liczba krawędzi)
- Dobra złożoność to może być $O(|V| + |E|)$
- W sztucznej inteligencji, gdzie często nie znamy grafu (lub jest on zbyt duży, żeby traktować go jako daną do zadania), używamy innych parametrów

Analiza czasowo pamięciowa (2)

Parametry zadania wyszukiwania

- b – maksymalne rozgałęzienie (branching factor)
- d – głębokość najpłytszego węzła docelowego
- m – maksymalna długość ścieżki w przestrzeni poszukiwań

Uwaga

Mówiąc o czasie (pamięci) często używamy jako jednostki liczby węzłów (przetworzonych/pamiętanych).

Czas i pamięć dla BFS i DFS

BFS

Czas $= (O(b + b^2 + b^3 + \cdots + b^d) = O(b^d))$
Pamięć = Czas
Uwaga: Może być też $O(b^{d+1})$ jak testujemy warunek sukcesu dopiero podczas rozwijania.

DFS

Czas $= O(b^m)$ – niedobrze
Pamięć $= O(bm)$ – dobrze

Uwaga

W tych rozważaniach zakładamy, że przestrzeń jest tak wielka, że nie spamiętujemy odwiedzonych stanów (względnie wiemy, że stany się nie powtarzają)
Oczywiście w skończonych grafach nasze rozważania są nazbyt pesymistyczne!

Iterative Deepening

Uwaga

Iteracyjne pogłębianie to po prostu wywoływanie DLS na coraz to większej głębokości (bez zapamiętywania żadnych pośrednich wyników)

Może wydawać się to stratą czasu, ale:

- działamy w pamięci $O(bd)$,
- na czas wpływa ostatnia wartstwa, czyli $O(b^d)$

UCS. Właściwości

- UCS = Uniform Costs Search
- Zamiast kolejki FIFO mamy kolejkę priorytetową, z priorytetem równym kosztowi dotarcia do węzła.

Uwaga

Oczywiście umożliwia to różnicowanie kosztów dotarcia z węzła do węzła.

Uwaga 2

UCS rozwiązuje ten sam problem co algorytm Dijkstry (i w bardzo podobny sposób). Ale jest różnica powiedzmy filozoficzna

Uniform Cost Search a Dijkstra

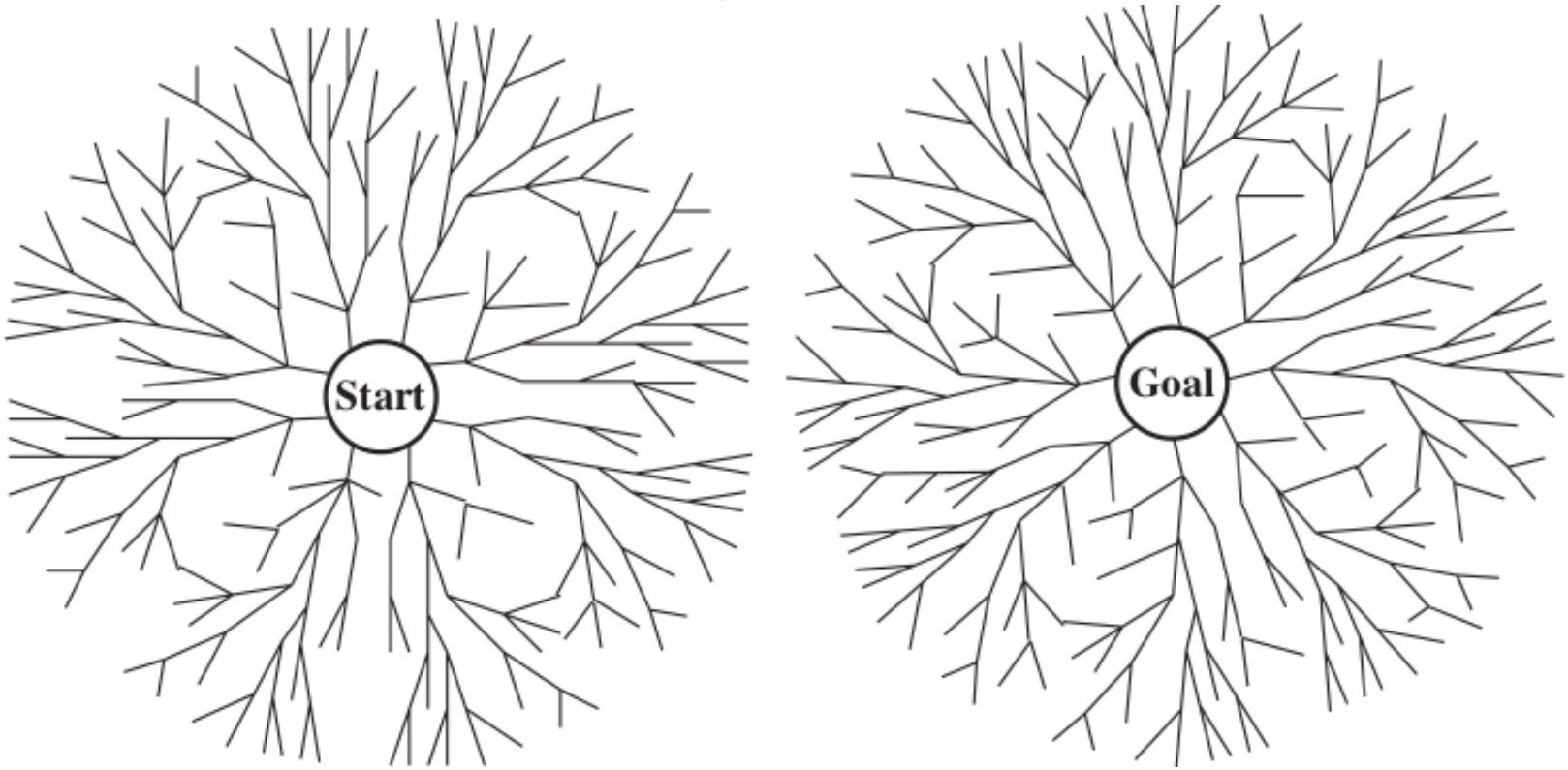
- UCS jest na sztucznej inteligencji, Dijkstra na algorytmach (to oczywiście nie jest poważna różnica).
- UCS jest przedstawiany najczęściej jako instancja algorytmu typu Best First Search
- Graf który przeszukujemy może być duży, nieznany w całości, nieskończony, itd.

Przeszukiwanie dwukierunkowe

Pomysł

Prowadźmy poszukiwania jednocześnie od przodu i od tyłu

Rysunek:



Przeszukiwanie dwukierunkowe. Problemy i korzyści

Problemy

Nie zawsze jest możliwe do zastosowania:

1. Musimy znać stan końcowy (vide hetmany czy obrazki logiczne)
2. Najlepiej jak jest jeden (albo niewiele i umiemy je wszystkie wymienić)
3. Musimy umieć odwrócić funkcję następnika (vide problem Knutha i funkcja `int ( )`)
4. Musimy pamiętać odwiedzone stany (przynajmniej z jednej strony)
   BFS + IDS (lub BFS + BFS) zamiast IDS+IDS

Korzyści

Podstawowa korzyść to czas działania. Dlaczego?
Odpowiedź: Zamiast jednego przeszukania na głębokości d mamy dwa przeszukania na głębokości $d/2$.

Przeszukiwanie bez wiedzy. Podsumowanie

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

Koniec nagrania II

Problemy bezczujnikowe (sensorless)

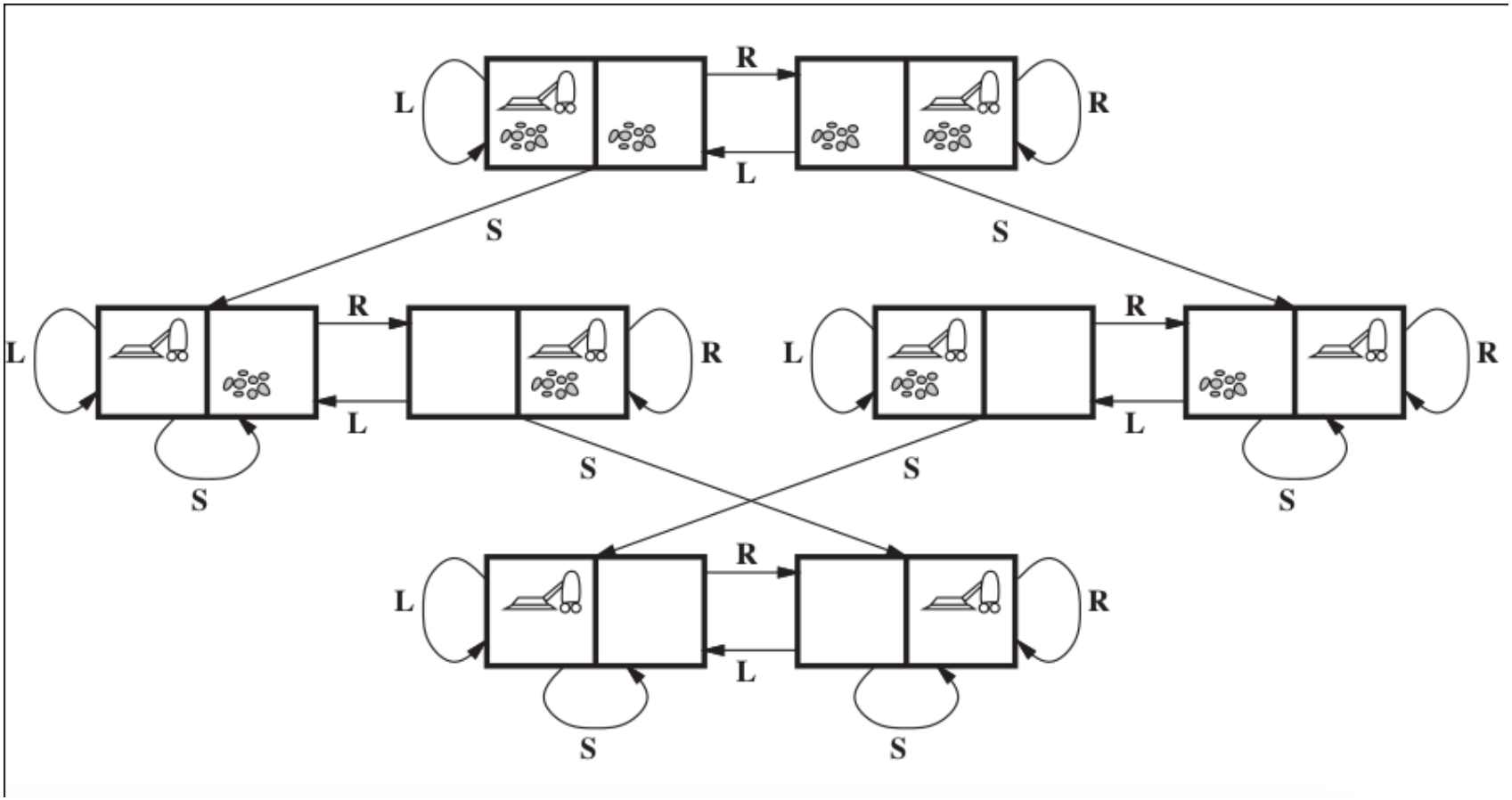
- Czujniki są drogie. Czasem wolimy na przykład znaleźć sekwencje akcji, która doprowadzi do celu niezależnie od stanu.
- Przykład 1 Szeroko działający antybiotyk
- Przykład 2 Robot w linii produkcyjnej, który składa jakieś części wykonując akcje niezależne od tego, jak te części się ułożyły.

Uwaga

Oczywiście rozwiązanie problemu bezczujkowego nie jest optymalne w środowisku z dostępem do sensorów. Zakładamy na przykład, że pewne akcje będą „puste".

Problemy bezczujnikowe (przykładowy odkurzacz)

Wszyscy wiemy o inteligentych odkurzaczach. Ten będzie trochę prostszy (rysunek z przestrzenią stanów):



Przestrzeń przekonań

Definicja

Stanem przekonań jest zbiór stanów oryginalnego problemu, w których agent (być może) się znajduje.

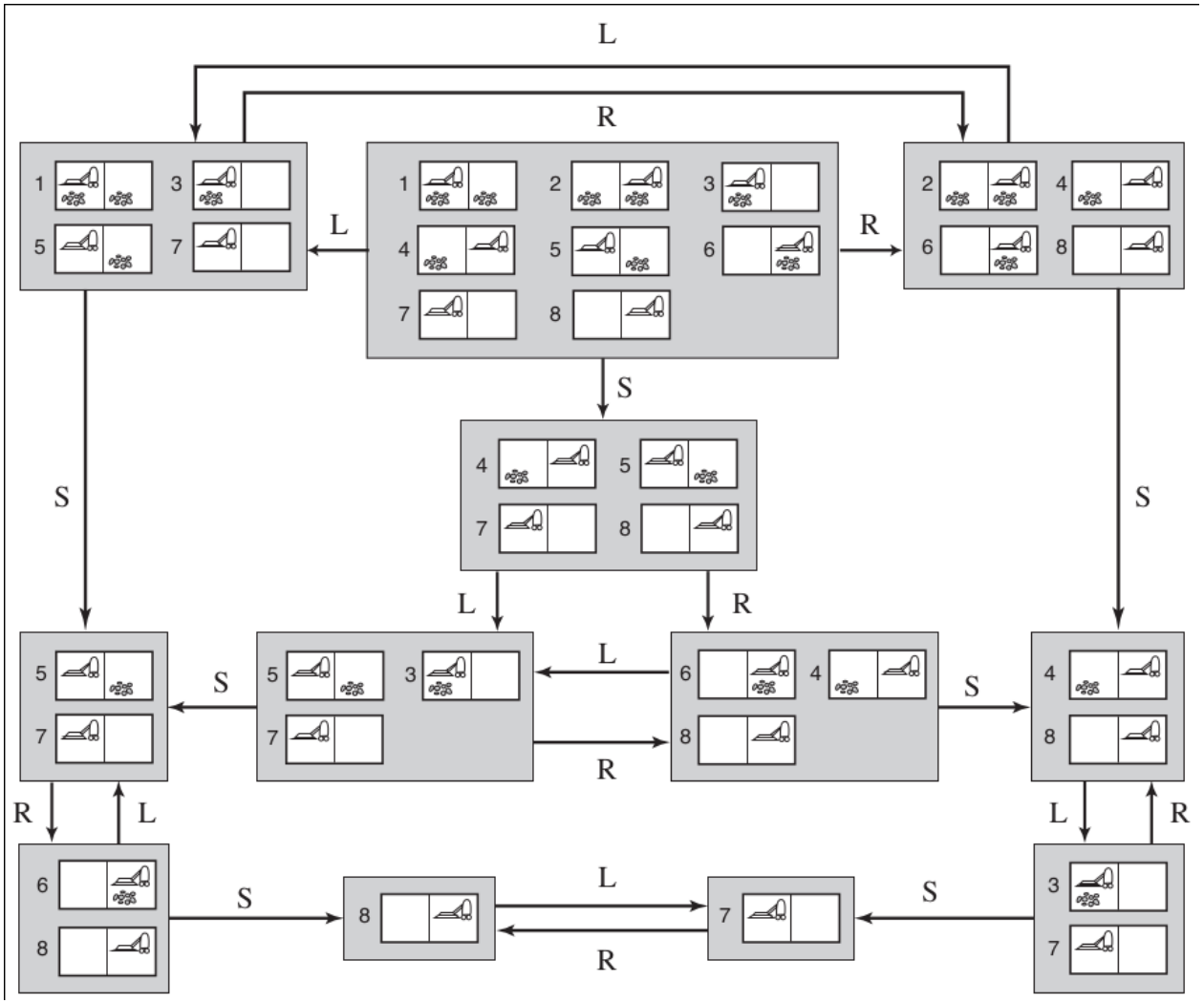
Pytanie 1

Jak się poruszać w takiej przestrzeni?

Pytanie 2

Jaka sekwencja akcji jest rozwiązaniem problemu bezczujnikowego (napiszmy ją na tablicy).

Przestrzeń przekonań odkurzacza. Przykład



(pętle dla wszystkich stanów usunięte ze względu na czytelność.)

Graf przestrzeni przekonań

1. Przejścia w przestrzeni przekonań powstają przez zaaplikowanie funkcji przejścia do stanu (obliczenia obrazu funkcji)
2. Stan jest końcowy jeżeli wszystkie stany w nim zawarte są końcowe.
3. Koszt jednostkowy (spory problem w innym przypadku)
4. Stan startowy: zbiór wszytskich stanów.

Komandos z mapą. Mniej trywialny przykład

- Rozważmy zadanie, w którym do labiryntu wrzucony zostaje komandos z mapą...
- ale zrzut jest w nocy i nie wiadomo, gdzie trafił.
- Problem:

  znajdź sekwencję akcji, która ***na pewno*** *doprowadzi do jednego z celów (akcje niedozwolone nie przesuwają komandosa).*

Komandos. Jak go rozwiązać

- Zadanie z komandosem będzie na liście P2.
- Zbadajmy, jak działa taka przestrzeń przekonań.

Zmniejszanie niepewności

Zobaczmy, jakie są możliwości zmniejszania niepewności w tym zadaniu (program `commando_z_wykladu.py`).

Dodatkowa wiedza o problemie

- Opłaca się iść w kierunku rozwiązania.
- Co to oznacza?

Zakładamy, że umiemy szacować odległość od rozwiązania.

Przykłady

1. Odległość w linii prostej w zadaniu szukania drogi.
2. Odległość taksówkowa (Manhattan distance) w labiryncie.

Przeszukiwanie zachłanne

- Rozwijamy ten węzeł, który wydaje się najbliższu rozwiązania.
- Proste, intuicyjne, ale są problemy. Jakie?

Można ten algorytm „oszukiwać”, w skrajnym przypadku sprawić, żeby rozwiązanie w ogóle nie zostało znalezione (w wersji bez zapamiętywania stanów, w których byliśmy).

Plansza nieprzyjazna dla algorytmu zachłannego