

Sztuczna inteligencja. Przeszukiwanie z wiedzą o problemie

Paweł Rychlikowski

Instytut Informatyki UWr

8 marca 2023

Problemy bezczujnikowe (sensorless)

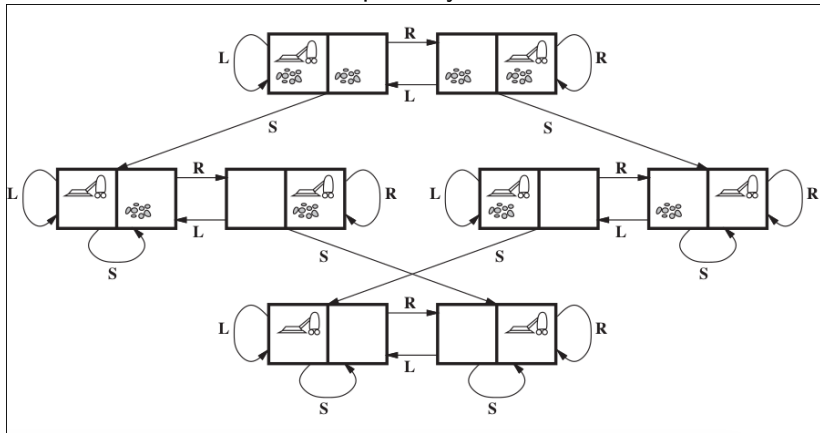
- Czujniki są drogie. Czasem wolimy na przykład znaleźć sekwencje akcji, która doprowadzi do celu niezależnie od stanu.
- **Przykład 1** Szeroko działający antybiotyk
- **Przykład 2** Robot w linii produkcyjnej, który składa jakieś części wykonując akcje niezależne od tego, jak te części się ułożyły.

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 **inteligentnych odkurzaczach**. Ten będzie trochę prostszy:



Przestrzeń przekonań

Definicja

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

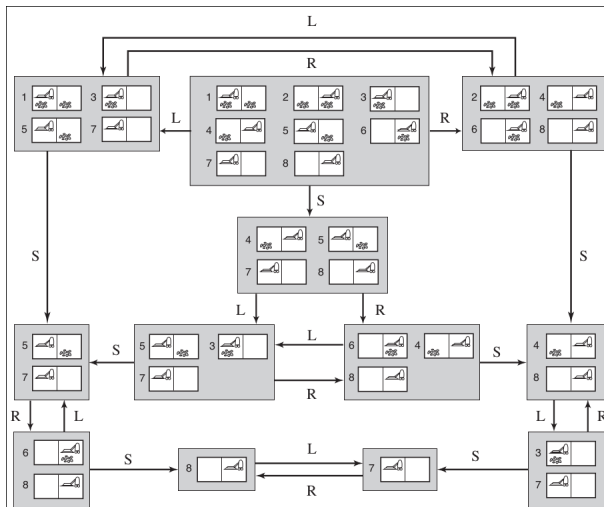
Pytanie 1

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 (jeżeli oryginalny był jednostkowy)
4. **Stan startowy**: zbiór wszystkich **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_wykładu.py`).

Rozpoczynamy kolejny mini-dział, czyli
przeszukiwanie z wiedzą 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.

- 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

Definicje

- $g(n)$ – koszt dotarcia do węzła n
- $h(n)$ – szacowany koszt dotarcia od n do (najbliższego) punktu docelowego ($h(s) \geq 0$)
- $f(n) = g(n) + h(n)$

Algorytm

Przeprowadź przeszukiwanie, wykorzystując $f(n)$ jako priorytet węzła (czyli rozwijamy węzły od tego, który ma najmniejszy f).

- Zwróćmy uwagę, że algorytm przypomina BFS (w którym, jak pamiętamy, używamy kolejki FIFO) oraz algorytm UCS (uniform cost search, Dijkstry).
- Jedyną różnicą między A* i UCS jest użycie funkcji f , a nie funkcji g jako priorytetu w kolejce priorytetowej.

Wymagania dla heurystyki

Oczywiście od wyboru funkcji h (nazywanej **heurystyką**) zależą właściwości algorytmu A^*

Wymienimy najważniejsze właściwości funkcji h .

1. **Nieujemna**: $h(s) \geq 0$, dla każdego s
2. **Rozsądna**: $h(s_{\text{end}}) = 0$
3. **Dopuszczalna** (admissible):
 $h(s) \leq$ prawdziwy koszt dotarcia ze stanu s do stanu końcowego
Inaczej: **optymistyczna**
4. **Spójna** (consistent), s_1, s_2 to sąsiednie stany:

$$\text{cost}(s_1, s_2) + h(s_2) \geq h(s_1)$$

Ostatnia własność przypomina **własność trójkąta** w definicji metryki (na tablicy)

Pojęcie **optymistyczna** wydaje się dość intuicyjne w kontekście heurystyki.

Zwróćmy uwagę, że:

- Dla zadania: dojechać z punktu A do B po drogach publicznych, heurystyka szacująca koszt dotarcia do B jako odległość euklidesową z punktu X , w którym jesteśmy, do celu jest optymistyczna:
zakładamy bowiem optymistycznie, że istnieje prosta, pozbawiona zakrętów droga prościutko do B
- Jeżeli podróżujemy po Manhattanie (czyli w miejscu, gdzie wszystkie drogi przecinają się pod kątem prostym), poprzednia heurystyka nadal będzie optymistyczna. Ale bardziej realistyczne będzie liczenie tzw. odległości taksówkowej, która jest po prostu sumą różnic na współrzędnych x oraz y .

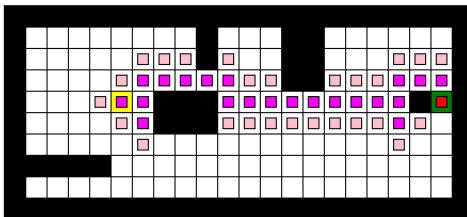
Jak wkrótce zobaczymy, wybór bardziej realistycznej (ale ciągle optymistycznej) heurystyki zaowocuje lepszym działaniem algorytmu A^* .

- Ponieważ h ma być **szacowaną odległością** od celu, umówimy się, że własność **nieujemności** i **rozsądnosci** funkcji h są konieczne, żeby mówić o algorytmie A^*
- Pozostałe dwie własności z poprzedniego slajdu (optymistyczna i spójna) są „mile widziane”, ale niekonieczne.

Kilka prostych faktów

- 1. UCS to A^* z super-optimistyczną heurystyką ($h(s) = 0$)
- 2. Spójna heurystyka jest optymistyczna
Dowód: Indukcja po węzłach (na ćwiczeniach, okolice C2)

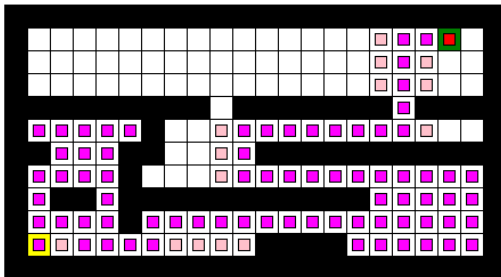
A* w labiryncie (1)



Używamy odległości taksówkowej między bieżącą kratką a celem jako heurystyki (czyli **optymistycznie** zakładamy, że nie spotkamy po drodze żadnej ściany).

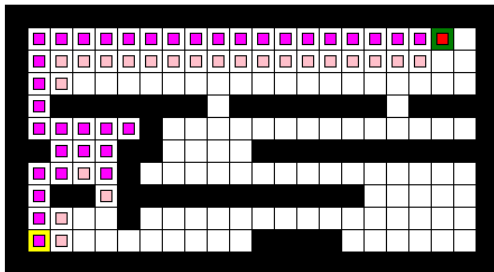
Kolor **rózowy**: węzły w kolejce, kolor **purpurowy** – węzły rozwinięte. Jedynie dwa rozwinięte węzły są poza optymalną ścieżką. Na tym rysunku (i kolejnych) widzimy stan algorytmu w momencie osiągnięcia węzła końcowego.

A^* w labiryncie (2)



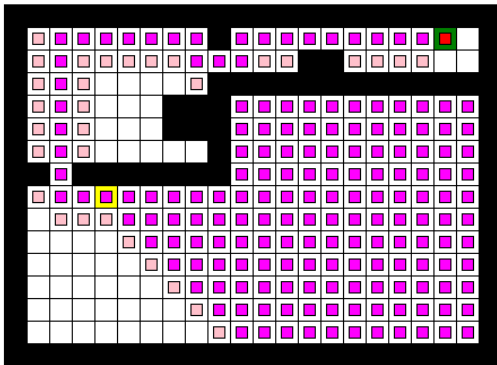
W dolnej części labiryntu heurystyka trochę prowadzi na manowce

A^* w labiryncie (3)



... ale jeżeli w poprzednim labiryncie przebić drzwi, to wówczas znowu jest prawie idealnie.

A^* w labiryncie (4)



Heurystyka mocno „oszukana” przebiegiem labiryntu.

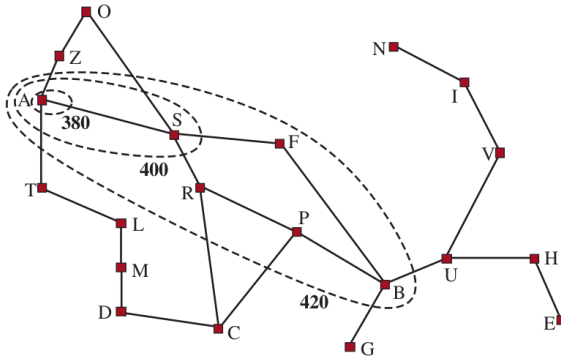


Figure 3.20 Map of Romania showing contours at $f = 380$, $f = 400$, and $f = 420$, with Arad as the start state. Nodes inside a given contour have $f = g + h$ costs less than or equal to the contour value.

Twierdzenie 1

A^* jest zupełny (warunki jak dla UCS).

Twierdzenie 2

Jeżeli h jest spójna, to A^* zwraca optymalną ścieżkę (wersja grafowa)

Twierdzenie 3

Jeżeli h jest optymistyczna, to A^* w drzewie zwraca optymalną ścieżkę.

Dowody: będą, ale najpierw jeszcze trochę praktyki.

A^* oczywiście nie daje gwarancji znalezienia rozwiązania dla grafów nieskończonych, w których istnieją nieskończone ścieżki o skończonej sumie wag krawędzi

Heurystyki dla ósemki

Uwaga

Pewne aspekty tworzenia heurystyk można dość dobrze prześledzić na przykładzie **ósemki**

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

Pytanie

Jak (optymistycznie) oszacować odległość tych dwóch stanów?

- Heurystyka musi być prosta do policzenia.
- Przy projektowaniu heurystyki kluczowe jest pilnowanie **optymizmu** (czyli że niedoszacujemy odległości).
- Choć teoretycznie wymagany jest silniejszy warunek (spójności), ale w praktyce naturalne optymistyczne heurystyki są spójne...
- ... a o optymizm łatwiej zadbać (i łatwiej przypomnieć sobie definicję)

Heurystyki dla ósemki (2)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

Pomysł 1

Jak coś jest nie na swoim miejscu, to musi się ruszyć o co najmniej 1 krok. Zliczajmy zatem, ile kafelków jest poza punktem docelowym ($h_1(s) = 8$)

Pomysł 2

Jak coś jest nie na swoim miejscu, to musi pokonać cały dystans do punktu docelowego. Zliczajmy zatem, ile kroków od celu jest każdy z kafelków ($h_2(s) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$)

Pytanie

Która intuicyjnie jest lepsza?

- Intuicja mówi, że jeżeli coś **dokładniej** szacujemy, to algorytm bazujący na tych dokładniejszych szacunkach będzie działał lepiej
- Z dwóch optymistycznych heurystyk ta, która daje większe wartości, jest dokładniejsza.

- Dla h_2 efektywność A^* jest 50000 razy większa niż IDS.
- Istnieją heurystyki dające jeszcze 10000 krotne przyspieszenie dla 15-ki, a milionowe dla 24-ki (wobec h_2)

Kiedy możliwy jest ruch w łamigłówce ósemka? Docelowe pole jest: (koniunkcja warunków):

- a) sąsiadujące
- b) wolne

Możemy rezygnować z części (lub wszystkich) warunków, otrzymując **łatwiejsze** łamigłówki.

Uwaga

Liczba ruchów w łatwiejszym zadaniu od startu do punktu docelowego jest często sensowną heurystyką w zadaniu oryginalnym.

Heurystyka h_1

Ruch możliwy jest **zawsze**.

Heurystyka h_2

Ruch możliwy jest **gdy pole jest obok** (niekoniecznie puste).

Heurystyka h_3

Ruch możliwy jest **gdy pole jest puste** (niekoniecznie obok).

Relaksacja w zadaniu poszukiwania w labiryntach lub przy podróży samochodem drogami:

Relaksacja w zadaniu poszukiwania w labiryntach lub przy podróży samochodem drogami:

1. W labiryncie: pomijanie ścian, czyli odległość taksówkową

Relaksacja w zadaniu poszukiwania w labiryntach lub przy podróży samochodem drogami:

1. W labiryncie: pomijanie ścian, czyli odległość taksówkową
2. W atlasie drogowym: pomijanie dróg, czyli odległość euklidesową (helikopterem)

Operacja maksimum dla heurystyk

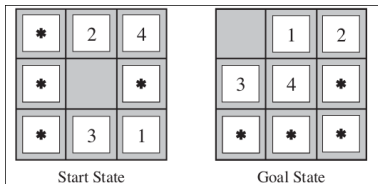
- Jak mamy dwie heurystyki h_1 i h_2 , obie optymistyczne
- to możemy zdefiniować $h_3(x) = \max(h_1(x), h_2(x))$

Uwaga

h_3 też będzie optymistyczna!

Heurystyki możemy budować korzystając z **baz wzorców**,
zapamiętujących koszty rozwiązań **podproblemów** danego zadania.

Przykład:



- Znajdujemy wszystkie podproblemy dla danego stanu (które mamy w bazie)
- A następnie bierzemy maksimum kosztów jako wartość heurystyki
- Możemy do tego maksimum dołożyć jakieś proste heurystyki (typu h_2).

Pytanie

A czy nie moglibyśmy użyć sumowania, zamiast maksimum?

Działanie bazy wzorców (2)

- Niestety suma daje **niedopuszczalne** heurystyki (bo pewne ruchy liczymy wielokrotnie, gwiazdki w jednym wzorcu są istotnymi kafelkami w innym)
- **Pytanie:** Jak temu zapobiec?
- **Odpowiedź:** stosując „rozłączne” wzorce (nic się nie powtarza) i w każdym wzorcu liczyć tylko ruchy kafelków z liczbami.

To to są te najefektywniejsze heurystyki dla 8-ki

Uwaga

Niemniej warto wiedzieć, że czasem rezygnuje się z optymalności i stosuje niedopuszczalne heurystyki (które czasem przeszacowują odległość), ze względu na szybkość działania.

Wyszukiwanie dróg (Google Maps, etc)

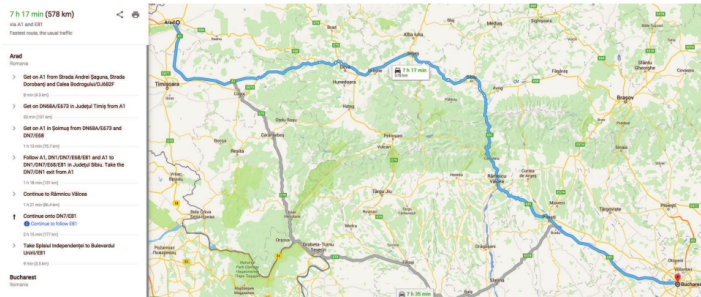


Figure 3.28 A Web service providing driving directions, computed by a search algorithm.

Charakterystyka:

- Duży graf, dany explicite
- Wynik potrzebny od zaraz
- Możliwy preprocessing (bo zapytania mogą być podobne do siebie)

Landmarks

Zakładamy istnienie pewnej liczby **punktów orientacyjnych**, będziemy się do nich odwoływać przy liczeniu heurystycznej odległości między węzłami

Zakładamy, że policzyliśmy odległości między **każdym p.o.** a **każdym** węzłem.

Heurystyka

$$h(n) = \min_{L \in \text{Landmarks}} (C^*(n, L) + C^*(L, \text{goal}))$$

Ogólnie nie jest optymistyczna, chyba że ścieżka przechodzi przez *jakiś* punkt orientacyjny!

Differential heuristic

$$h(n) = \max_{L \in \text{Landmarks}} (C^*(n, L) - C^*(L, \text{goal}))$$

(jak wyjdzie ujemna, to dajemy 0)

- Myślimy, że punkt orientacyjny jest **za celem** (jadąc do L mijamy cel po drodze)
- Jak cel jest trochę z boku drogi, to tracimy dokładność, ale nie optymizm.