Sztuczna inteligencja. Przeszukiwanie

Paweł Rychlikowski

Instytut Informatyki UWr

2 marca 2023

Czym jest problem?

Definicja

Problem ma następujące pięć komponentów

1. Stan początkowy (i zbiór stanów, ale być może dany implicite)
2. Zbiór akcji (co **agent** może robić)
3. Model przejścia (stan + akcja = nowy_stan)
4. Test określający, czy stan jest końcowy (i znaleźliśmy rozwiązanie)
5. Sposób obliczania kosztu ścieżki (najczęściej podawany jako koszt akcji w stanie)

Agent – program, który odbiera wrażenia o świecie, podejmuje decyzje, wykonuje akcje, maksymalizując jakąś funkcję celu.

Piętnastka. Przypomnienie

- Stan początkowy: jakieś ułożenie, na przykład powyższe
- Stan końcowy: liczby po kolei
- Koszt: jednostkowy
- Model: dowolna zamiana pustego kwadracika z sąsiadem

Akcje i model przejścia

Uwaga

Akcje oraz model przejścia są ze sobą powiązane. Jest kilka wariantów:

1. Zbiór akcji wspólny, funkcja przejścia pilnuje, żeby niemożliwe akcje nie zmieniały stanu
   Przykład: ludzik idzie w labiryncie na ścianę
2. Akcje to funkcja, która dla stanu zwraca zbiór czynności, które agent może zrobić w stanie
3. Akcja i model przejścia to jedna funkcja, która dla stanu zwraca **listę** par postaci:
   (akcja, nowy_stan)

Poziomy abstrakcji.

Zadania z rzeczywistego świata można modelować na wiele różnych sposobów. Dla podróżowania MPK mamy następujące akcje/sekwencje akcji

- Przejedź z przystanku A do przystanku B
- Wsiądź do tramwaju na przystanku A, skasuj/zakup bilet, zajmij wygodne miejsce, obserwuj tablicę, podejdź do drzwi, gdy...
- Wykonuj naprzemienne ruchy lewą i prawą nogą, aż ...
- Napnij głowę większą mięśnia dwugłowego lewego uda, ...

Uwaga

Akcje muszą być zrozumiałe dla agenta, im wyższy poziom, tym łatwiejsze zadanie przeszukiwania.

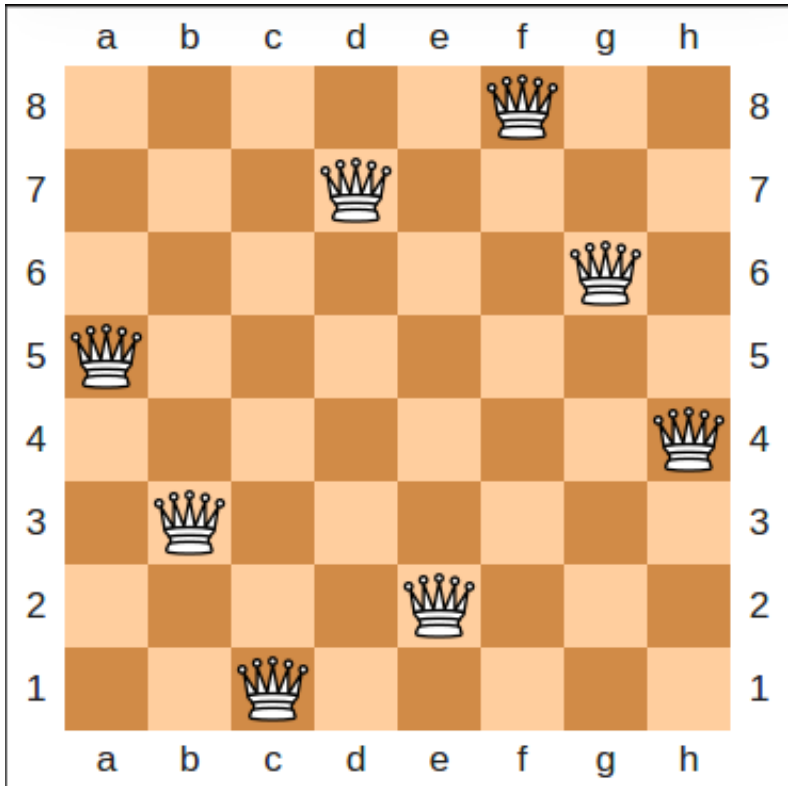
Przykłady

- Przeanalizujemy dla kilku przykładów jak można pewne zadania przedstawiać jako problemy przeszukiwania

Zadanie szachowe z pierwszej listy

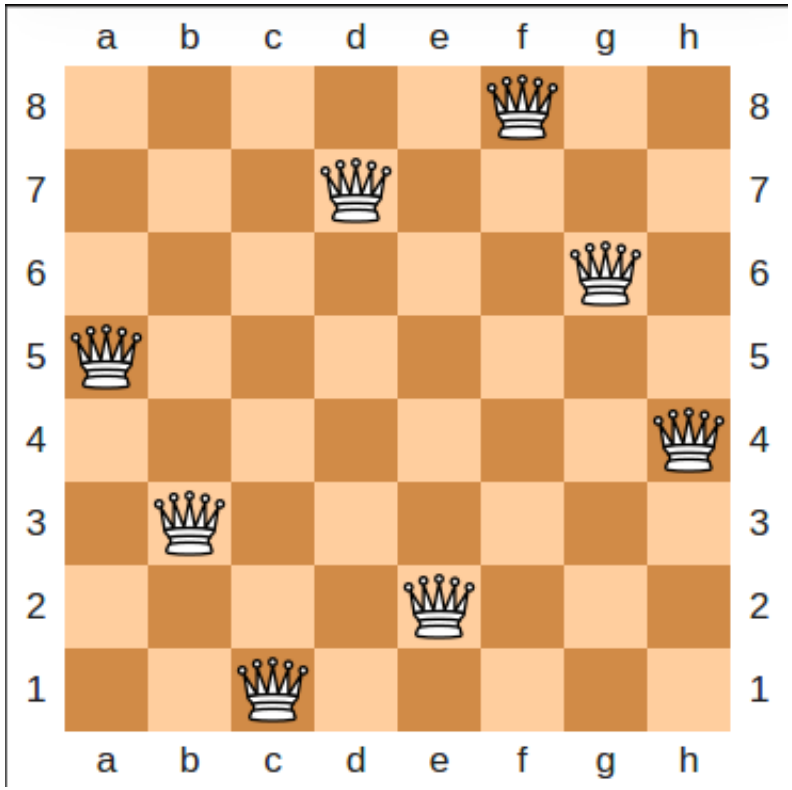
- Stan początkowy: jakieś ułożenie dwóch wież i króli, informacja o tym, kto się rusza
- Stan końcowy: mat
- Koszt: jednostkowy
- Model: ruch szachowy + zmiana gracza aktywnego

Problem ośmiu hetmanów



- W przeciwieństwie do poprzedniego przykładu istnieje tu wiele sformułowań.
- Jakie są dwie najważniejsze opcje?

Problem ośmiu hetmanów



Dwa sposoby opisywania zadania jako problemu przeszukiwania:

1. Stan kompletny (rozważamy sytuacje z 8 hetmanami na planszy, ruch to przestawienie hetmana)
2. Stan niepełny – zaczynamy od pustej planszy, ruchem jest postawienie hetmana.

Wielkość przestrzeni stanów

Uwaga

Rozmieszczamy hetmany wg określonego schematu, rozpoczynając od pustej planszy. Liczymy liczbę stanów.

- $64 \times 63 \times \cdots \times 57 = 178462987637760$ – rozmieszczamy po kolei hetmany, na jednym polu jest 1 hetman.
- $8^8 = 16777216$ – ustalona kolejność, w każdej kolumnie 1 hetman
- $8! = 40320$ – ustalona kolejność, w każdej kolumnie 1 hetman, nie szachują się w wierszach

Problem zabawkowy: hipoteza Knutha

Hipoteza

Zaczynając od 4 możemy dojść do dowolnej liczby wykonując operacje: silni, pierwiastka i podłogi (części całkowitej, `int`).

Przykładowo:

$$\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}} \rfloor = 5$$

Sformułowanie dość oczywiste (stany to liczby)

Uwaga

Przestrzeń jest nieograniczona (nie znamy żadnego twierdzenia, które ograniczałoby przestrzeń prostą funkcją).

Problem poważny: dowodzenie twierdzeń

- Stany to zbiory **lematów** (uprzednio dowiedzionych faktów)
- Stan początkowy: zbiór aksjomatów
- Model: Zbiór reguł wyprowadzania nowych twierdzeń na podstawie twierdzeń już dowiedzionych
- Stan końcowy:: zbiór twierdzeń zawierających twierdzenie Ψ, które właśnie dowodzimy

Modyfikacje zadania znajdowania marszruty

Wariant 1

W co K-tej miejscowości na trasie znajduje się dobra knajpa.

Wariant 2

Powinniśmy zaliczyć co najmniej K dodatkowych atrakcji turystycznych.

Zastanówmy się, co jest stanem w powyższych zadaniach?

Modyfikacje zadania znajdowania marszruty

Wariant 1

W co K-tej miejscowości na trasie znajduje się dobra knajpa.
Stan: (miejscowość, licznik-modulo-K)

Wariant 2

Powinniśmy zaliczyć co najmniej K dodatkowych atrakcji turystycznych.
Stan (1): (miejscowość, liczba-atrakcji-do-zaliczenia) źle!
Stan (2): (miejscowość, zbiór-zaliczonych-atrakcji) lepiej!
Stan (3): (jaki?)

Rzeczywisty świat

Przykładowe zadania z rzeczywistego świata, będące zadaniami przeszukiwania:

- Różne zagadnienia logistyczne
- Projektowanie układów scalonych
- Nawigacja robotów
- Organizacja procesu produkcji
- Tworzenie projektów o określonych właściwościach

Komentarze do listy pracowniowej

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!

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

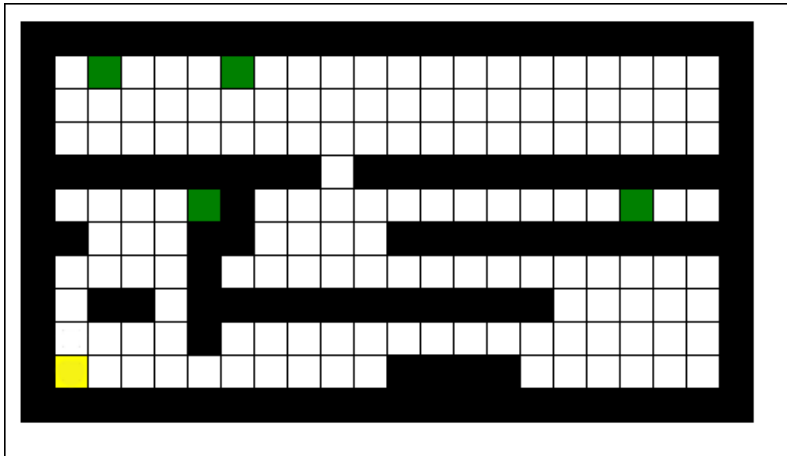
Pytanie

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?)

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

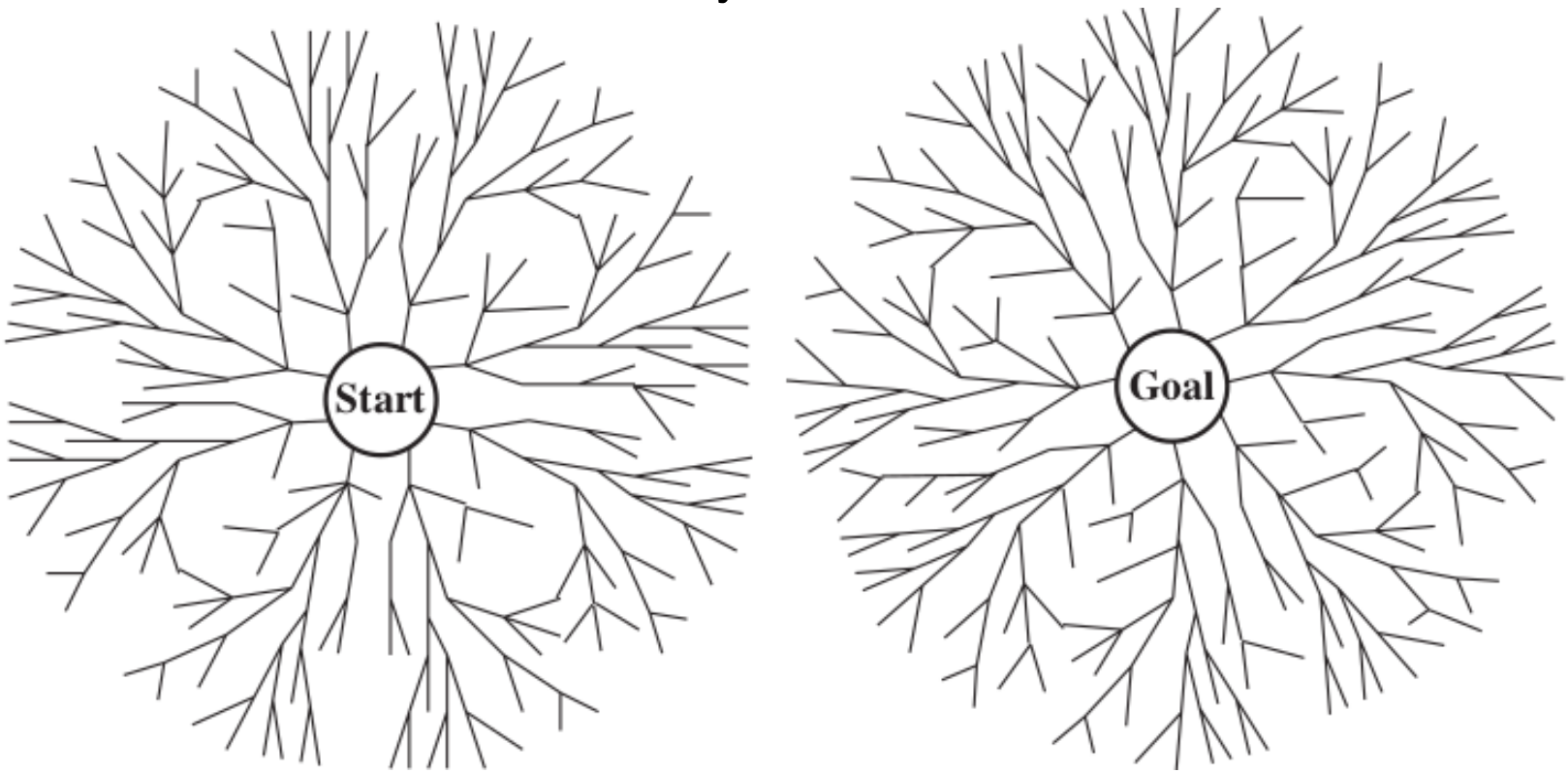
- UCS jest na sztucznej inteligencji, Dijkstra na algorytmach (to oczywiście nie jest poważna różnica).
- UCS jest przedstawiany najczęściej jako instancja algorytmu typu Best First Search
- Graf który przeszukujemy może być duży, nieznany w całości, nieskończony, itd.

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.