Programowanie gier

Wykład 4
Pathfinder

Joanna Kołodziejczyk

styczeń 2017

Plan wykładu

1. Pathfinding
2. Przeszukiwanie wszerz
3. Przeszukiwanie w głąb
4. Strategie heurystyczne
5. Greedy Best First Search
6. A^*

Zadanie

Pathfinding

Jest to zadanie przejścia z punktu A do punktu B.

Popularne zadanie w różnego typu grach od zwykłego labiryntu do gier RPG. Przechodzenie z punktu do punktu często wymaga większej inteligencji niż tylko poruszanie się po prostej łączącej punkty początkowy i końcowy trasy. Postaci nie mogą przechodzić przez przeszkody, tylko muszą znaleźć drogę dookoła.

Pathfinding to zadanie szukania

Cechy zadania:

- Nie jest znana a priori sekwencja kroków (stanów) prowadząca do rozwiązania.
- Kroki owe muszą być wyznaczone przez analizę bardzo licznych alternatyw.
- Szukanie to proces polegający na testowaniu różnych kierunków poszukiwań i odrzucaniu błędnych.
- Zaletą jest łatwość formułowania problemu: zbioru stanów, zbioru operatorów, stan początkowy i cel.

Różne algorytmy

Niektóre strategie ślepe (*ang. blind search*) inaczej bez wiedzy (*ang. uninformed search*) lub zachłanne.

- Przeszukiwanie wszerz — BFS (Breadth-first search).
- Przeszukiwanie o stałym koszcie — UCS (Uniform-cost search).
- Przeszukiwanie w głąb — DFS (Depth-first search).
- Przeszukiwanie w głąb z ograniczeniem (Depth-limited search).
- Iteracyjne zagłębianie (Iterative deepening search).
- Algorytm Dijkstry

Niektóre strategie heurystyczne (szukanie z wiedzą (*ang. informed search*))

- Greedy Best-first search
- A^* search
- Heurystyki

Plan wykładu

1 Pathfinding

2 Przeszukiwanie wszerz

3 Przeszukiwanie w głąb

4 Strategie heurystyczne

5 Greedy Best First Search

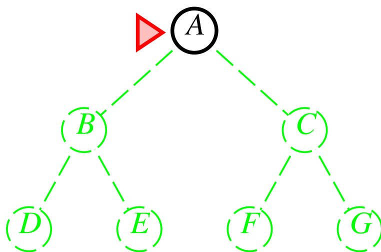
6 A^*

Pathfinder z przeszukiwaniem wszerz

Breadth First Search (BFS)

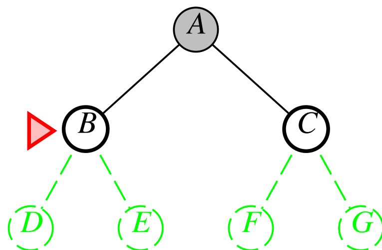
- Pierwszy rozwijany jest węzeł w korzeniu drzewa — jest to stan początkowy wyznaczony ze specyfikacji rozwiązywanego zadania.
- Następnie rozwija wszystkich potomków wyznaczonych w kroku poprzednim, a potem ich potomków itd.
- Uogólniając: BFS rozwija wszystkie węzły drzewa na poziomie d.
- BFS może być zaimplementowany jako kolejka FIFO tj. nowy węzeł potomny z rozwinięcia węzła bieżącego jest dopisywany na końcu kolejki węzłów czekających do rozwinięcia.

BFS — działanie



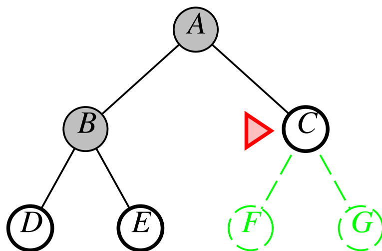
$LISTA = \{A\}$

BFS — działanie



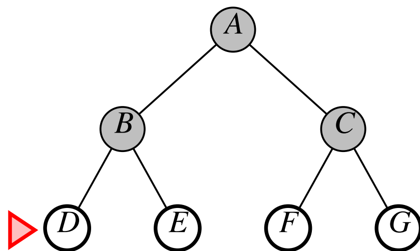
$$LISTA = \{B, C\}$$

BFS — działanie



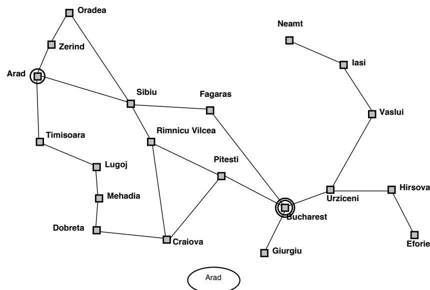
$LISTA = \{C, D, E\}$

BFS — działanie



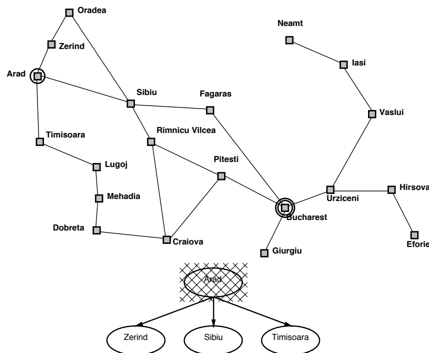
$$LISTA = \{D, E, F, G\}$$

Przeszukiwanie wszerz — przykład



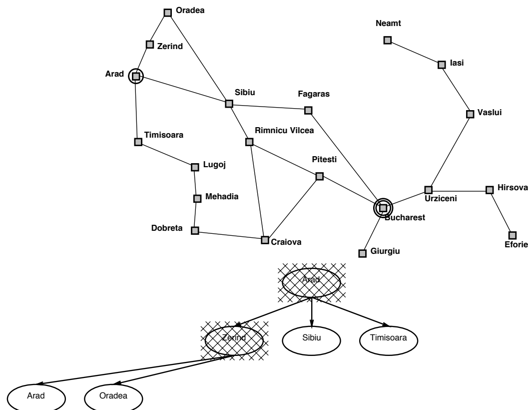
$$LISTA = \{Arad\}$$

Przeszukiwanie wszerz — przykład



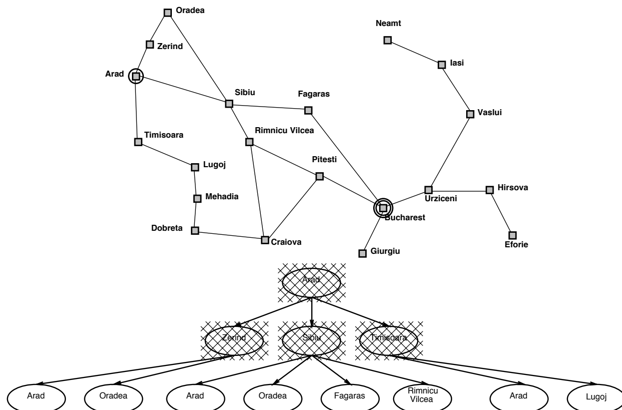
$LISTA = \{Zerind, Sibiu, Tisimora\}$

Przeszukiwanie wszerz — przykład



$$LISTA = \{Sibiu, Tisimora, Arad, Oradea\}$$

Przeszukiwanie wszerz — przykład



$LISTA =$

Ocena algrytmu BFS

Zupełność: czy zawsze znajdzie rozwiązanie jeżeli ono istnieje?

Tak, jeżeli b jest skończone.

Złożoność czasowa: liczba węzłów generowana/rozwijana.

Zakłada się, że węzeł w korzeniu może mieć b potomków. Każdy jego potomek też może mieć b potomków. Rozwiązanie jest na poziomie drzewa d. W najgorszym przypadku trzeba rozwinąć wszystkie węzły z poziomu d oprócz ostatniego węzła.

$$1 + b + b^2 + b^3 + \ldots + b^d + (b^{d+1} - b) = O(b^{d+1})$$

Ocena algrytmu BFS c.d.

Obszar zajmowanej pamięci: maksymalna liczba węzłów w pamięci.

$O(b^{d+1})$
Przechowuje każdy węzeł w pamięci, bo albo węzły są liśćmi i czekają na rozwinięcie, lub są przodkami, których trzeba pamiętać.

Optymalność: czy zawsze znajdzie najmniej kosztowne rozwiązanie?

Tak, jeżeli w każdym kroku koszt ścieżki jest stały. Nie musi być optymalny (najbliższe rozwiązanie nie musi być optimum).

Wymagania algorytmu wszerz

Wnioski

1. Wymagania pamięciowe algorytmu są dużo większe niż czasowe.
2. Problemy o złożoności wykładniczej nie mogą być rozwiązane metodą BFS. Żadna z metod ślepych nie jest efektywna dla dużego n.

Depth	Nodes	Time	Memory
2	1100	0.11 seconds	1 MB
4	111100	11 seconds	106 MB
6	10^7	19 minutes	10 gigabytes
8	10^9	31 hours	1 terabyte
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3523 years	1 exabyte

Wymagania czasowe i pamięciowe dla BFS. Założono, że $b = 10$, 10000 *węzłów/sek*, 1000 *bytów/węzeł.*

Plan wykładu

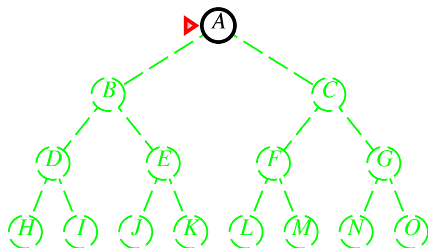
1 Pathfinding
2 Przeszukiwanie wszerz
3 Przeszukiwanie w głąb
4 Strategie heurystyczne
5 Greedy Best First Search
6 A^*

Szukanie w głąb — DFS

Depth-first search: zawsze rozwija najgłębsze jeszcze nierozwinięte węzły.

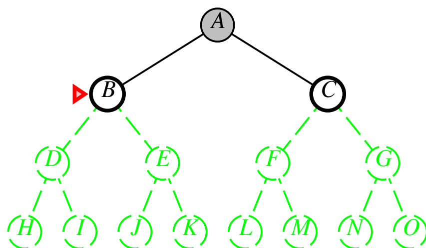
- Zaczyna od korzenia w drzewie.
- Rozwija jedną gałąź aż do osiągnięcia maksymalnej głębokości w drzewie.
- Jeżeli liść nie jest rozwiązaniem, to wraca się do najbliższego, płytszego, jeszcze nie rozwiniętego węzła i rozwija się go.
- Implementacją tej strategii może być kolejka LIFO (stos).
- Procedura implementacyjna może być wykonana jako funkcja rekurencyjna.

Przeszukiwanie w głąb — działanie



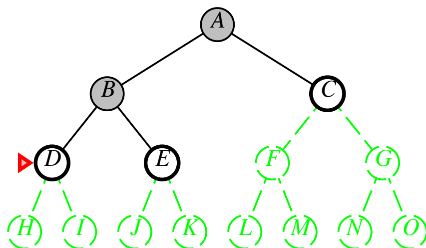
$$LISTA = \{A\}$$

Przeszukiwanie w głąb — działanie



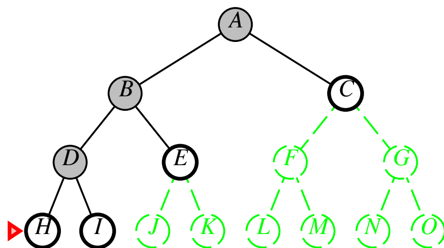
$LISTA = \{B, C\}$

Przeszukiwanie w głąb — działanie



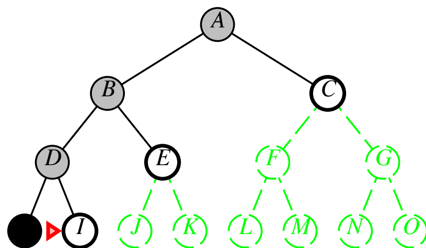
$$LISTA = \{D, E, C\}$$

Przeszukiwanie w głąb — działanie



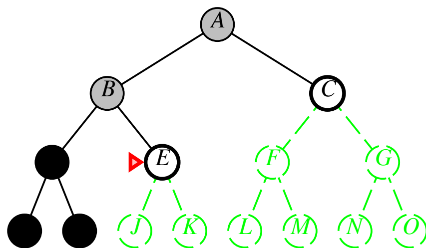
$$LISTA = \{H, I, E, C\}$$

Przeszukiwanie w głąb — działanie



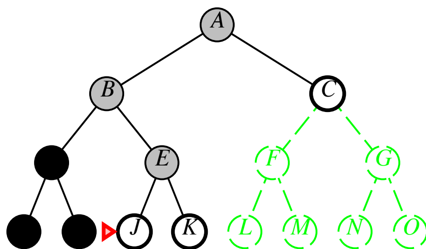
$$LISTA = \{I, E, C\}$$

Przeszukiwanie w głąb — działanie



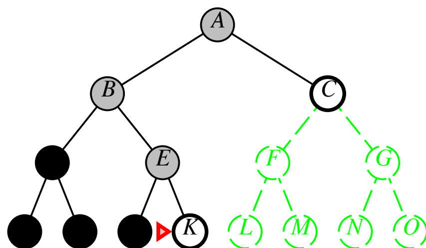
$$LISTA = \{E, C\}$$

Przeszukiwanie w głąb — działanie



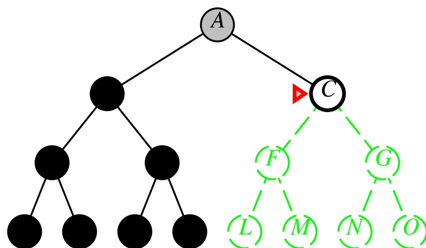
$$LISTA = \{J, K, C\}$$

Przeszukiwanie w głąb — działanie



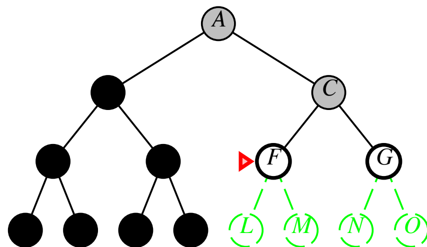
$$LISTA = \{K, C\}$$

Przeszukiwanie w głąb — działanie



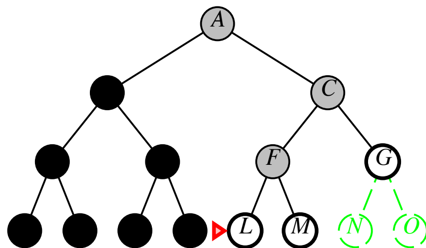
$LISTA = \{C\}$

Przeszukiwanie w głąb — działanie



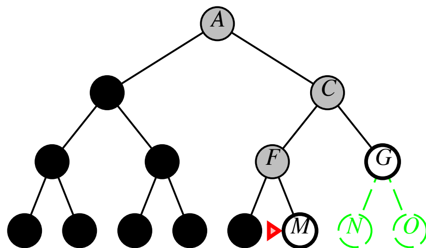
$$LISTA = \{F, G\}$$

Przeszukiwanie w głąb — działanie



$$LISTA = \{L, M, G\}$$

Przeszukiwanie w głąb — działanie



$$LISTA = \{M, G\}$$

Ocena algrytmu DFS

Zupełność: czy zawsze znajdzie rozwiązanie jeżeli ono istnieje?

Nie, chyba że przeszukiwana przestrzeń jest skończona (nie pojawiają się pętle)

Złożoność czasowa: liczba węzłów generowana/rozwijana.

Zakłada się, że m to maksymalna głębokość drzewa. Złożoność jest olbrzymia, gdy m jest dużo większe od d. Natomiast, gdy jest wiele rozwiązań może być szybszy niż BFS.

$$O(b^m)$$

Ocena algrytmu DFS c.d.

Obszar zajmowanej pamięci: maksymalna liczba węzłów w pamięci.

Przechowuje pojedynczą ścieżkę w pamięci od korzenia do liścia i braci węzłów na każdym poziomie. Rozwinięte węzły mogą zostać usunięte z pamięci, bo ścieżka do nich nie zawiera rozwiązania.
Niewielkie wymagania pamięci - złożoność liniowa: $O(bm)$

Optymalność: czy zawsze znajdzie najmniej kosztowne rozwiązanie?

Nie jest, tych samych powodów co zupełność

Powtarzające się stany

- W dotychczas omawianych algorytmach zignorowano problem odwiedzania tych samych stanów.
- W niektórych problemach, jak np. wyznaczanie drogi z punktu do punktu (routing problem) powtarzanie jest naturalne. Zazwyczaj dotyczy to zadań z przestrzenią odwracalną (odpowiednie dla dwukierunkowego szukania). Dla takich zadań drzewo przeszukiwań jest **nieskończone**.
- Brak rozpoznawania powtarzających się stanów może prowadzić do wzrostu złożoności problemu z liniowego do wykładniczego!
- Jedynym sposobem na uniknięcie tego problemu jest trzymanie rozwiniętych już węzłów w pamięci. Dodanie takiej cechy do algorytmu prowadzi do przeszukiwania przestrzeni stanów na **grafie**.
- Algorytm, który nie pamięta swojej historii jest skazany na powtórzenia.

Jak graf wpływa na algorytm

- Zbiór CLOSED powinien być zaimplementowany jako tablica hashująca, co zapewni efektywne sprawdzanie powtarzających się węzłów.
- Algorytm BFS są również optymalne do przeszukiwania grafu.
- DFS i wszystkie jego odmiany nie mają już liniowych wymagań pamięciowych, bo należy pamiętać wszystkie węzły.

Plan wykładu

1. Pathfinding
2. Przeszukiwanie wszerz
3. Przeszukiwanie w głąb
4. Strategie heurystyczne
5. Greedy Best First Search
6. A^*

Strategie pathfinding z informacją — informed search

Strategie heurystyczne

wykorzystują dodatkową informację, specyficzną wiedzę o rozwiązywanym problemie, co pozwoli na zwiększenie wydajności algorytmów przeszukiwania w stosunku do metod ślepych.

Eksplozja kombinatoryczna

to gwałtowny wzrost obliczeń przy niewielkim wzroście rozmiaru danych wejściowych (cecha zadań z klasy NP-trudnych).

Przykład eksplozji kombinatorycznej dla problemu TSP

Przykład wzrostu obliczeń na problemie TSP (*Traveling Salesman Person*) (komiwojażera). Założenia:

- dana jest mapa
- dane są dystanse pomiędzy parami miast
- każde miasto odwiedzane jest tylko raz
- cel: znaleźć najkrótsza drogę drogę, jeżeli podróż zaczyna się i kończy w jednym z miast

Dla N miast istnieje $1 \cdot 2 \cdot 3 \cdot \cdots \cdot (N-1)$ możliwych kombinacji. Czas jest proporcjonalny do N, a całkowity do $N!$, zatem dla 10 miast jest $10! = 3,628,800$ stanów do sprawdzenia przez metody zachłanne badające całą przestrzeń, a dla 11 miast aż $39,916,800$.

Ogólna strategia heurystyczna

Strategie z informacją (heurystyczne)

wykorzystują dla każdego węzła funkcję oceny $f(n)$.

- Dla węzła n funkcja $f(n)$ oszacowuje jego „użyteczność”.
- Funkcja oceny mierzy odległość do rozwiązania.
- Rozwija się najbardziej użyteczne (takie, które wydają się być najlepsze w bieżącej chwili) jeszcze nie rozwinięte węzły.
- Implementacja strategii to lista węzłów posortowanych według malejącej wartości $f(n)$ (przy założeniu, że $f(goal) = 0$, a $f(n) > 0$).

Dwie strategie:

- greedy best-first search
- A^*

Funkcja heurystyczna

Funkcja heurystyczna

Funkcja odwzorowująca stany we współczynnik ich użyteczności.

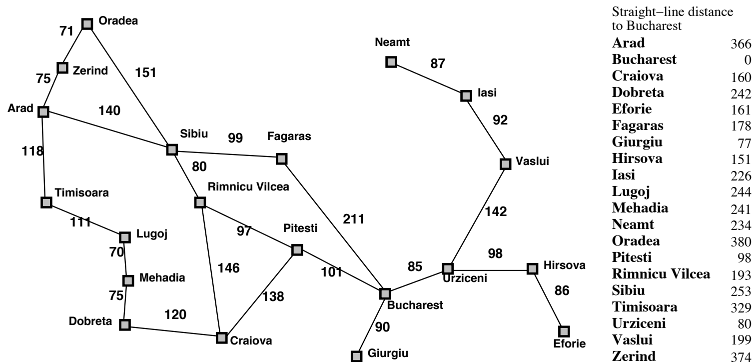
- Najczęściej jej przeciwdziedzina to $\mathcal{R}^+$.
- Zazwyczaj określa „odległość'' od rozwiązania.
- Przyjmuje się, że $h(n) = 0$, gdy $n = cel$ (rozwiązanie).

$$h : \Omega \rightarrow \mathcal{R}$$

gdzie Ω przestrzeń stanów, a $\mathcal{R}$ zbiór liczb rzeczywistych.

$h(n)$ — jest **oszacowaniem!** kosztu przejścia od węzła n do rozwiązania.

Podróż po Rumunii z funkcją heurystyczną



h_{SLD} — straight-line distance heuristic.

- $h_{SLD}(Arad) = 366$
- h_{SLD} nie może być policzona z danych wejściowych problemu

Plan wykładu

1. Pathfinding
2. Przeszukiwanie wszerz
3. Przeszukiwanie w głąb
4. Strategie heurystyczne
5. Greedy Best First Search
6. A^*

Greedy Best First Search

Greedy best-first search

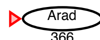
Rozwija węzły, które są najbliższe rozwiązania, zakładając, że prawdopodobnie prowadzą to do najszybszego rozwiązania. Oszacowuje koszt węzła stosując tylko funkcję heurystyczną

$$f(n) = h(n)$$

Minimalizowanie $h(n)$ jest podatne na niewłaściwy punkt startowy.

Greedy best-first search przypomina DFS, bo wybiera raczej jedną ścieżkę.

Greedy Best First Search - przykład

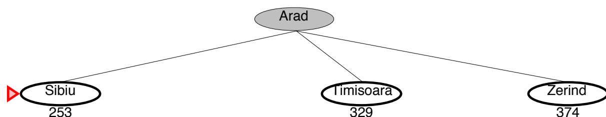


Zakłada się, że za pomocą greedy best first search szukamy najkrótszej drogi z Aradu do Bukaresztu.

Stan początkowy: Arad

$$Lista = \{Arad.h(Arad) = 366\}$$

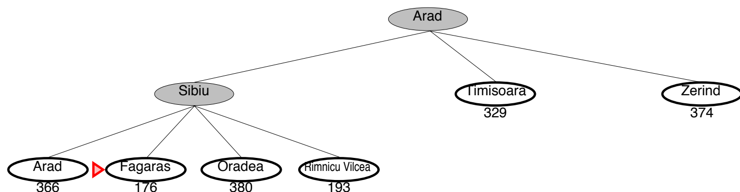
Greedy Best First Search - przykład



Pierwszy krok rozwija węzeł Arad i generuje potomków: Sibiu, Timisoara and Zerind

$Lista = \{Sibiu.h(Sibiu) = \mathbf{253}, Timisoara.h(Timisoara) = \mathbf{329}, Zerind.h(Zerind) = \mathbf{374}\}$ Greedy best-first wybierze Sibiu.

Greedy Best First Search - przykład

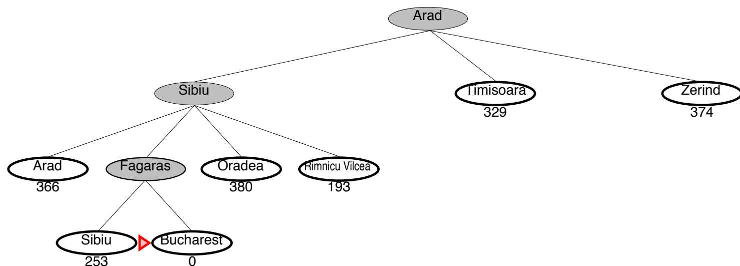


Jeżeli rozwiniemy Sibiu to otrzymamy listę:

$Lista = \{Fagaras.h(Fagaras) = \mathbf{176}, RimnicuVilcea.h(RimnicuVilcea) = \mathbf{193}, Timisoara.h(Timisoara) = \mathbf{329}, Arad.h(Arad) = \mathbf{366}, Zerind.h(Zerind) = \mathbf{374}, Oradea.h(Oradea) = \mathbf{380}\}$

Greedy best-first wybierze Fagaras.

Greedy Best First Search - przykład



Jeżeli Fagaras jest rozwinięte to otrzymamy :Sibiu and Bukareszt.
Cel został osiągnięty $h(Bukareszt) = 0$, acz nie jest optymalny: (zobacz Arad, Sibiu, Rimnicu Vilcea, Pitesti)

Ocena algorytmu Greedy Best First Search

Zupełność: czy zawsze znajdzie rozwiązanie jeżeli ono istnieje?

Nie, może utknąć w pętlach. Zupełny tylko w warunkach kontroli powtórzeń (graf) i skończonej przestrzeni stanów.

Złożoność czasowa: liczba węzłów generowana/rozwijana.

$$O(b^m)$$

choć dobra heurystyka może dać znaczne zmniejszenie złożoności czasowej.

Ocena algrytmu Greedy Best First Search c.d.

Obszar zajmowanej pamięci: maksymalna liczba węzłów w pamięci.

Przechowuje wszystkie węzły

$$O(b^m)$$

Optymalność: czy zawsze znajdzie najmniej kosztowne rozwiązanie?

Nie.

Plan wykładu

1. Pathfinding
2. Przeszukiwanie wszerz
3. Przeszukiwanie w głąb
4. Strategie heurystyczne
5. Greedy Best First Search
6. A^*

Algorytm A^*

Funkcja oceny ma postać:

$$f(n) = g(n) + h(n)$$

gdzie:

- $g(n)$ = koszt osiągnięcia bieżącego węzła n z węzła początkowego
- $h(n)$ = oszacowany koszt przejścia z węzła n do rozwiązania
- $f(n)$ = oszacowany pełny koszt ścieżki przez węzeł n do rozwiązania

Algorytm A^*

Algorytm A^* ma szansę być zupełny i optymalny jeżeli $h(n)$ spełni pewne warunki.

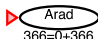
Twierdzenie

Algorytm A^* jest optymalny jeżeli funkcja heurystyczna jest dopuszczalna (admissible), tj. taką że:

- Nigdy nie przeceni kosztu dotarcia do rozwiązania: $h(n) \leq h^*(n)$, gdzie $h^*(n)$ jest rzeczywistym kosztem osiągnięcia rozwiązania z n
- Jest z natury optymistyczna bo zakłada że koszt rozwiązania jest mniejszy od rzeczywistego.
- $h(n) \geq 0$, stąd $h(G) = 0$ dla każdego rozwiązania G.

Przykład: $h_{\mathrm{SLD}}(n)$ - nigdy nie zawyży rzeczywistego dystansu.

Algorytm A^* — przykład

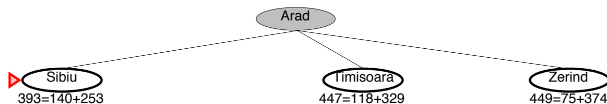


Szukamy najkrótszej drogi z Aradu do Bukaresztu.

Stan początkowy: Arad

$f(Arad) = g(Arad) + h(Arad) = 0 + 366 = 360$

Algorytm A^* — przykład

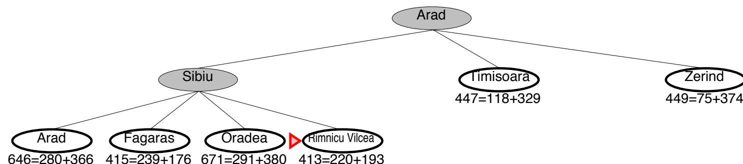


Pierwszy krok rozwija węzeł Arad i generuje potomków:

$$f(Sibiu) = g(Arad, Sibiu) + h(Sibiu) = \mathbf{140} + \mathbf{253} = \mathbf{393}$$
$$f(Timisoara) = g(Arad, Timisoara) + h(Timisoara) = \mathbf{118} + \mathbf{329} = \mathbf{447}$$
$$f(Zerind) = g(Arad, Zerind) + h(Zerind) = \mathbf{75} + \mathbf{374} = \mathbf{449}$$

A^* wybierze Sibiu.

Algorytm A^* — przykład



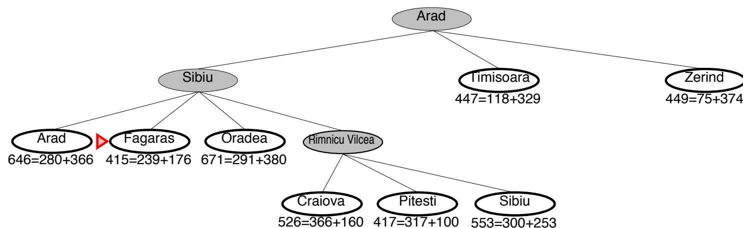
Jeżeli rozwiniemy Sibiu to otrzymamy:

$$f(Arad) = g(Sibiu, Arad) + h(Arad) = 280 + 366 = 646$$
$$f(Fagaras) = g(Sibiu, Fagaras) + h(Fagaras) = 239 + 179 = 415$$
$$f(Oradea) = g(Sibiu, Oradea) + h(Oradea) = 291 + 380 = 671$$

$$f(RimnicuVilcea) = g(Sibiu, RimnicuVilcea) + h(RimnicuVilcea) = 220 + 192 = 413$$

A^* wybierze Rimnicu Vilcea.

Algorytm A^* — przykład

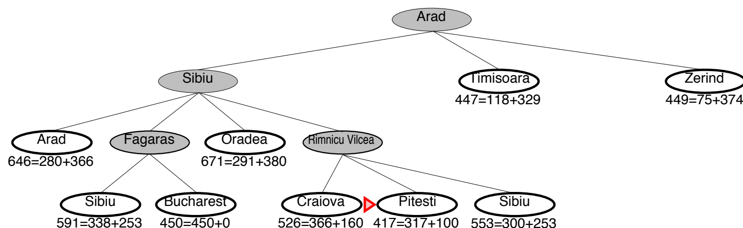


Jeżeli rozwiniemy Rimnicu Vilcea to otrzymamy:

$f(Craiova) = g(RimnicuVilcea, Craiova) + h(Craiova) = 360 + 160.326$
$f(Pitesti) = g(RimnicuVilcea, Pitesti) + h(Pitesti) = 317 + 100 = 417$
$f(Sibiu) = g(RimnicuVilcea, Sibiu) + h(Sibiu) = 300 + 253 = 553$

A^* wybierze Fagaras - węzeł pamiętany z poprzednich rozwinięć.

Algorytm A^* — przykład

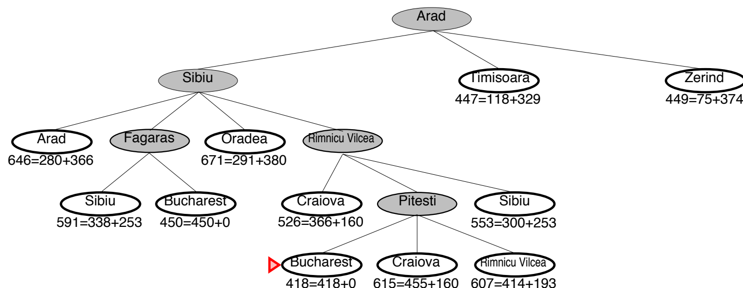


Jeżeli rozwiniemy Fagaras to najmniejszą wartość f ma Pitesi (417km):

$$f(Sibiu) = g(Fagaras, Sibiu) + h(Sibiu) = 338 + 253 = 591$$

$$f(Bucharest) = g(Fagaras, Bucharest) + h(Bucharest) = 450 + 0 = 450$$

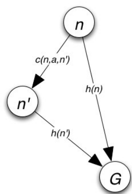
Algorytm A^* — przykład



Wybieramy do rozwinięcia Pitesi, bo f ma najmniejszą wartość i dotrzemy najkrótszą drogą do Bukaresztu (418km):

$$f(Bucharest) = g(Pitesti, Bucharest) + h(Bucharest) = 418 + 0 = 418$$

Monotoniczność heurystyki



Dla grafów należy dodać jeszcze jeden warunek do heurystyki, by zapewnić jej optymalność. Cechę tę nazywa się **monotonicznością** lub **spójnością**.

Dopuszczalna heurystyka h jest spójna (lub spełnia wymóg monotoniczności), jeżeli dla każdego węzła n i każdego jego następcy n' powstałego w wyniku akcji a zachodzi:

$$h(n) \leq c(n, a, n') + h(n')$$

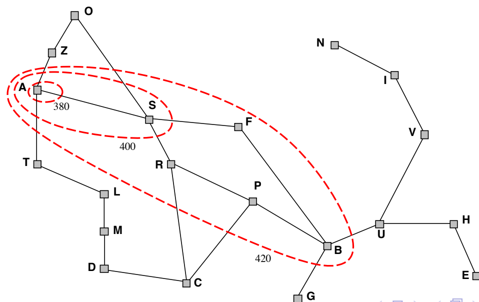
Jeżeli heurystyka h jest spójna, wówczas funkcja f wzdłuż dowolnej ścieżki jest niemalejąca:

$f(n) = g(n) + h(n)$

Monotoniczność A^*

Z monotoniczności wynika z

- A^* rozwija węzły w kolejności rosnącej wartości f.
- Stopniowo wyznacza się f-kontur na węzłach. Kontur i zawiera wszystkie węzły z $f = f_i$, gdzie $f_i < f_{i+1}$.
- Jeżeli $h(n) = 0$ kontur ma kształt kół.
- A^* nigdy nie rozwinie węzłów z $f(n) > f(G)$, czyli ma własności odcinania nieobiecujących stanów.



Ocena algorytmu A^*

Zupełność: czy zawsze znajdzie rozwiązanie jeżeli ono istnieje?

Tak, chyba że jest nieskończenie wiele węzłów z $f \leq f(G)$.

Złożoność czasowa: liczba węzłów generowana/rozwijana.

Niestety wykładnicza

Ocena algrytmu A^* c.d.

Obszar zajmowanej pamięci: maksymalna liczba węzłów w pamięci.

Przechowuje wszystkie węzły

Optymalność: czy zawsze znajdzie najmniej kosztowne rozwiązanie?

Tak, jeżeli $h(n)$ **dopuszczalne,**
A^* rozwija wszystkie węzły o $f(n) < C^*$
A^* rozwija niektóre węzły o $f(n) = C^*$
A^* nie rozwija węzłów o $f(n) > C^*$

Jak dobrać heurystykę dla układanki 8-elementowej?

Liczba węzłów na drzewie do przeszukania to 3^{22} a na grafie 181440.

Proponowane heurystyki:
$h_1(n)$ = liczba niewłaściwie ułożonych elementów
$h_2(n)$ = całkowita odległość Manhattan (tj. liczba pól do lokalizacji końcowej)

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

$h_1(S) = 6$
$h_2(S) = 4 + 0 + 3 + 3 + 1 + 0 + 2 + 1 = 14$

Porównanie różnych heurystyk

Jeżeli $h_2(n) \geq h_1(n)$ dla każdego n (przy czym obie dopuszczalne), to h_2 **zdominuje** h_1 i gwarantuje przeszukanie mniejszej liczby węzłów.

Przykładowe koszty poszukiwania do zadanego poziomu dla układanki 8-elementowej:

$d = 8$	IDS = 6384 węzłów
	A*(h_1) = 39 węzłów
	A*(h_2) = 25 węzłów
$d = 14$	IDS — nieopłacalne
	A*(h_1) = 539 węzłów
	A*(h_2) = 113 węzłów

Dla k heurystyk dopuszczalnych $h_a, h_b, \ldots h_k$, określonych dla zadania wybiera się:

$$h(n) = \max(h_a(n), h_b(n), \ldots, h_k(n))$$

Dla układanki h_2 jest zatem lepsza.

Jak znaleźć funkcję heurystyczną?

Relaxed problem polega na zmniejszeniu liczby reguł ograniczających rozwiązanie i wówczas poszukiwanie heurystyki. Np. dla układanki.

- Można przesuwać klocki gdziekolwiek, zamiast na puste sąsiednie pole, to $h_1(n)$ dałoby najkrótsze rozwiązanie.
- Jeżeli klocki można przesuwać na sąsiednie pola, nawet gdy są zajęte, to $h_2(n)$ dałoby najkrótsze rozwiązanie.

Takie uproszczenia dają ocenę nie wyższą niż koszt dokładny problemu w rzeczywistym ujęciu.

Opracowano program (ABSolver), który znalazł heurystyki dla kostki Rubika.