Podstawy sztucznej inteligencji

wykład 2
Strategie przeszukiwania - ślepe i heurystyczne
Algorytmy ewolucyjne

Joanna Kołodziejczyk

03 kwietnia 2011

Plan wykładu

Jak wykonać przeszukiwanie

- Przeszukiwanie przestrzeni stanów wymaga znalezienia ścieżki od stanu początkowego do końcowego.
- Aby tego dokonać buduje się graf/drzewo rozpoczynający się od stanu początkowego (czasami stanu końcowego).
- Węzeł jest **rozwijany** przez zastosowanie operatorów tak, by wygenerować wszystkich potomków stanu(węzła) bieżącego.
- Potomkowie znajdują się o poziom niżej w drzewie niż węzeł rozwijany.
- Strategia przeszukiwania decyduje, który węzeł jest rozwijany w kolejnym kroku.
- Różne strategie dają różne wyniki.

Cel główny

Znaleźć rozwiązanie najmniejszym nakładem obliczeniowym jednocześnie przechowując w pamięci jak najmniej węzłów.

Strategie bez informacji

Nazywane również ślepymi, gdyż rozwijając węzły sprawdzają tylko czy węzeł jest rozwiązaniem (oczekiwanym stanem końcowym). Kończą działanie, gdy trafią na stan końcowy.

Do takich strategii między innymi należą:

- BFS — szukanie wszerz
- DFS — szukanie w głąb
- IDS — iteracyjnie w głąb

Stosowane kryteria oceny algorytmów

Zupełność

Czy strategia zawsze znajdzie rozwiązanie, jeżeli ono istnieje?

Złożoność czasowa

Jaka liczba węzłów jest generowana/rozwijana?

Obszar zajmowanej pamięci

Jaka jest maksymalna liczba węzłów w pamięci?

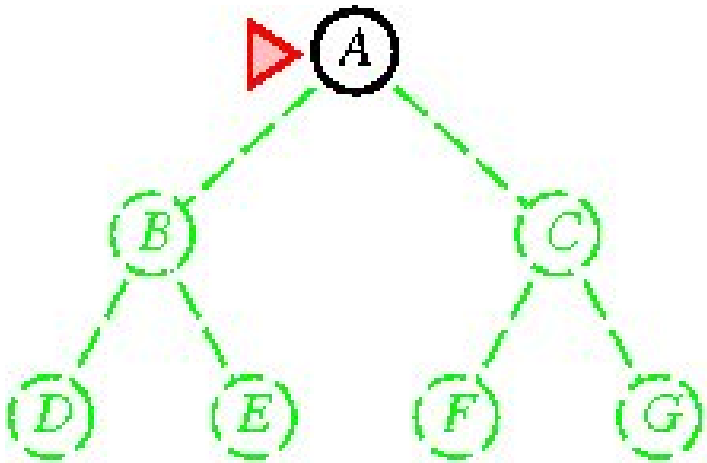
Optymalność

Czy zawsze znajdzie najmniej kosztowne rozwiązanie?

BFS — szukanie wszerz

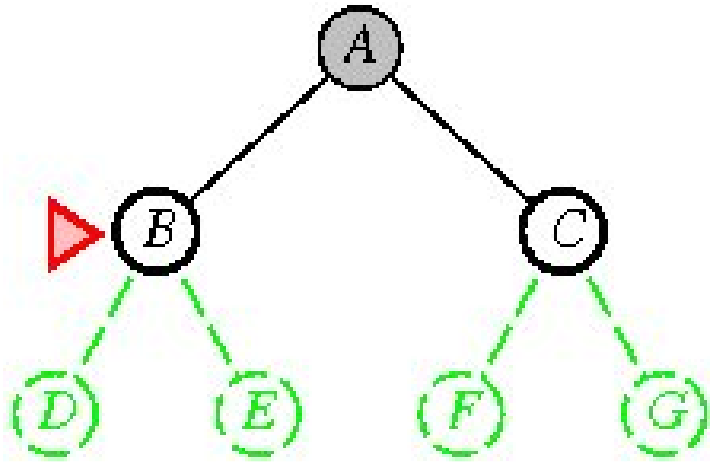
- Pierwszy rozwijany jest węzeł w korzeniu drzewa/grafu (stan początkowy).
- Następnie rozwija wszystkich potomków stanu początkowego, a potem ich potomków itd.
- Czyli BFS rozwija wszystkie węzły drzewa/grafu na poziomie d.
- BFS może być zaimplementowany jako kolejka FIFO tj. nowy węzeł potomny z rozwinięcia węzła bieżącego jest dopisywany na końcu kolejki węzłów czekających do rozwinięcia (LISTA OPEN).

BFS — szukanie wszerz



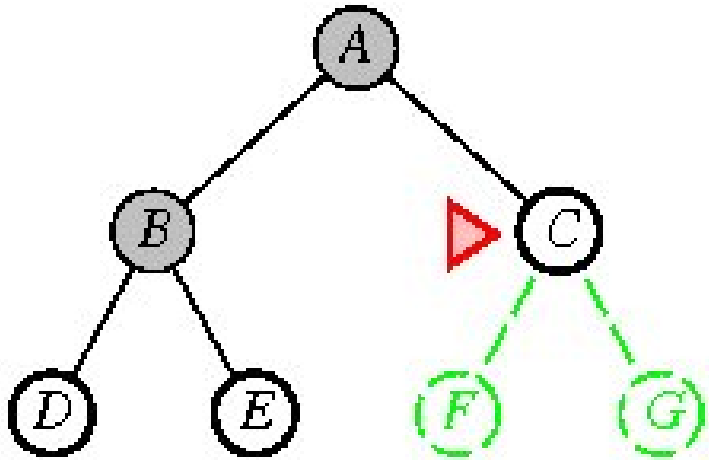
$$LISTA\_OPEN = \{A\}$$
$$LISTA\_CLOSED = \{\}$$

BFS — szukanie wszerz



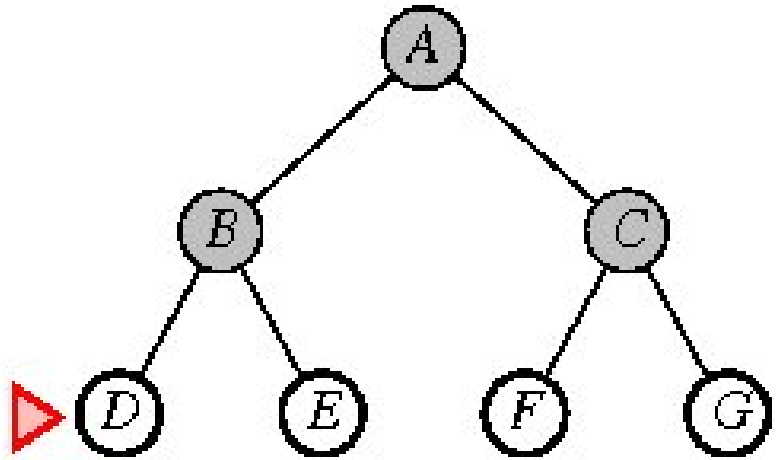
$$LISTA\_OPEN = \{B, C\}$$
$$LISTA\_CLOSED = \{A\}$$

BFS — szukanie wszerz



$$LISTA\_OPEN = \{C, D, E\}$$
$$LISTA\_CLOSED = \{A, B\}$$

BFS — szukanie wszerz



$$LISTA\_OPEN = \{D, E, F, G\}$$
$$LISTA\_CLOSED = \{A, B, C\}$$

Ocena algorytmu BFS

Zupełność

Tak, jeżeli b (liczba węzłów potomnych) jest skończone.

Złożoność czasowa

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. Co daje:

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

Ocena algorytmu BFS

Obszar zajmowanej pamięci

$O(b^{d+1})$, bo przechowuje każdy węzeł w pamięci (albo węzły są liśćmi (LISTA OPEN) i czekają na rozwinięcie, lub są już rozwinięte i przechowuje się je w pamięci, by wielokrotnie nie sprawdzać stanów już odwiedzonych (LISTA CLOSED)).

Optymalność

Tak, jeżeli w każdym kroku koszt ścieżki jest stały.

Wnioski

1. Wymagania pamięciowe algorytmu są trudniejsze do spełnienia niż czasowe.
2. Problemy, gdzie rozwiązanie jest na dużej głębokości i liczba potomków jest duża nie są rozwiązywalne strategią BFS.

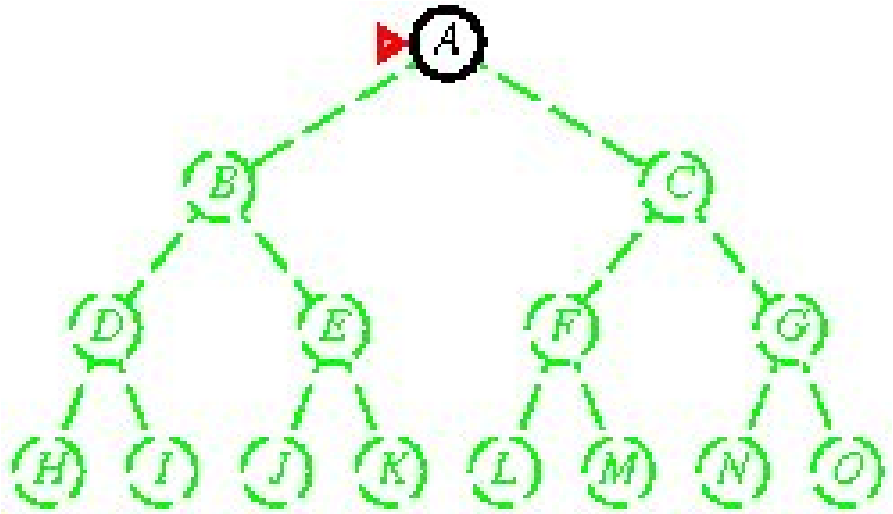
Głębokość	Węzły	Czas	Pamięć
2	1100	0.11 seconds	1 MB
4	111100	11 seconds	106 MB
6	10^7	19 minutes	10 gigabytes
8	10^9	31 hours	1 terabyte
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3523 years	1 exabyte

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

DFS — szukanie w głąb

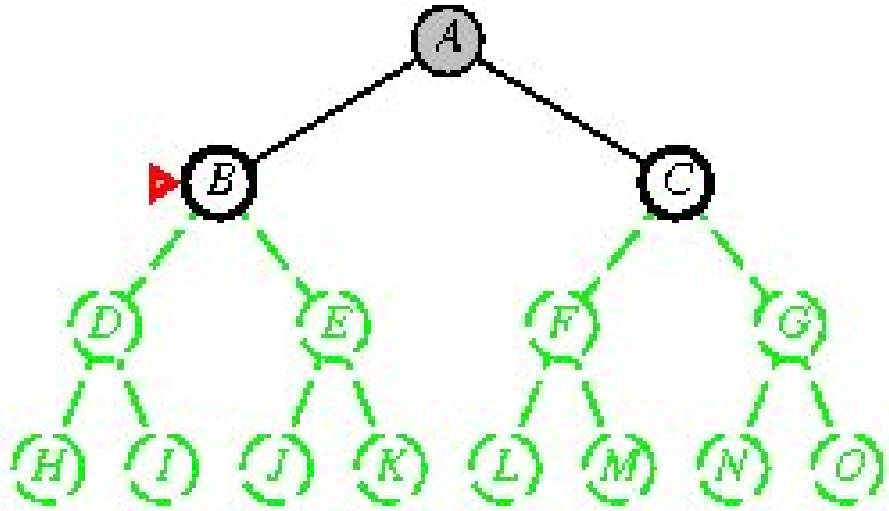
- Zaczyna od korzenia czyli stanu początkowego.
- Rozwija jedną gałąź aż do osiągnięcia maksymalnej głębokości w drzewie/grafie (m).
- Jeżeli liść nie jest rozwiązaniem, to wraca do najbliższego, płytszego, jeszcze nie rozwiniętego węzła i rozwija go.
- Implementacją tej strategii może być kolejka LIFO (stos). tj. nowy węzeł potomny z rozwinięcia węzła bieżącego jest dopisywany na początku kolejki węzłów czekających do rozwinięcia (LISTA OPEN).

DFS — szukanie w głąb



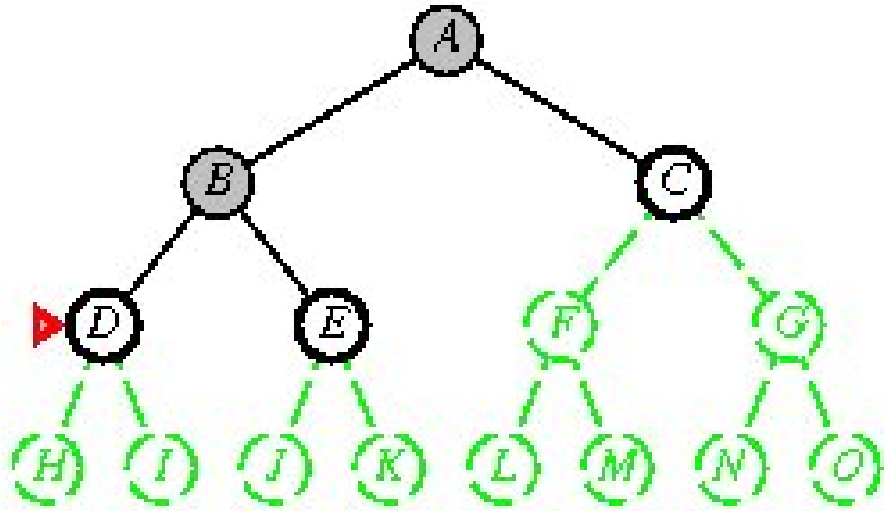
$$LISTA\_OPEN = \{A\}$$
$$LISTA\_CLOSED = \{\}$$

DFS — szukanie w głąb



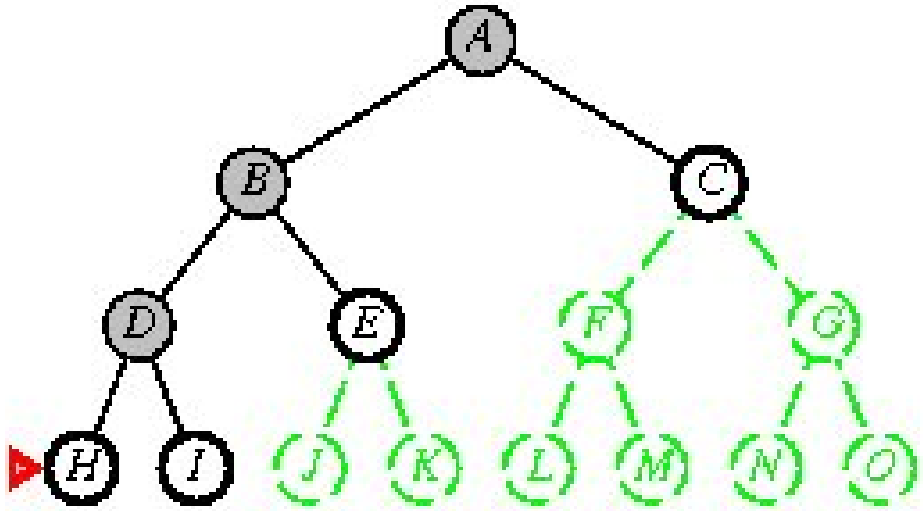
$$LISTA\_OPEN = \{B, C\}$$
$$LISTA\_CLOSED = \{A\}$$

DFS — szukanie w głąb



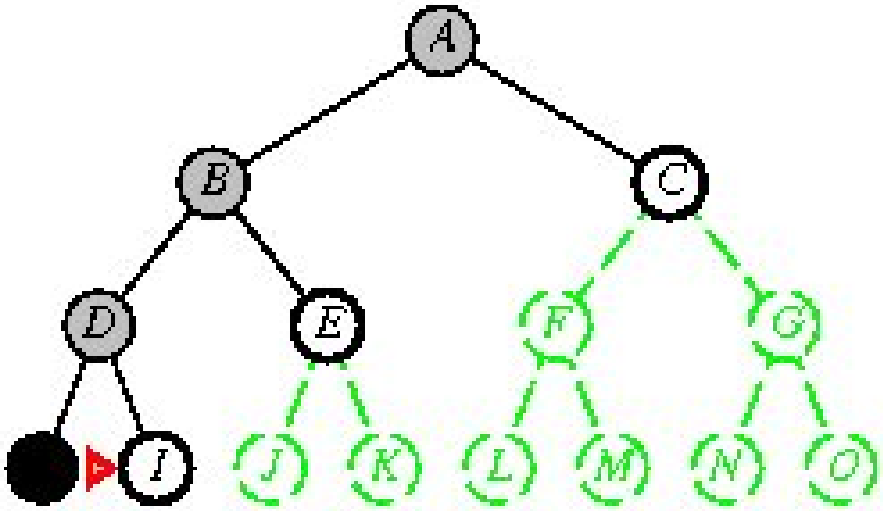
$$LISTA\_OPEN = \{D, E, C\}$$
$$LISTA\_CLOSED = \{A, B\}$$

DFS — szukanie w głąb



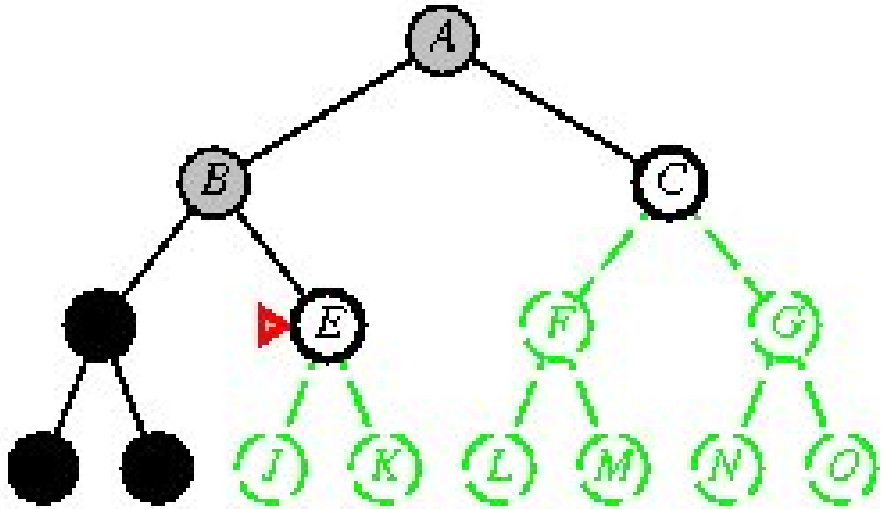
$$LISTA\_OPEN = \{H, I, E, C\}$$
$$LISTA\_CLOSED = \{A, B, D\}$$

DFS — szukanie w głąb



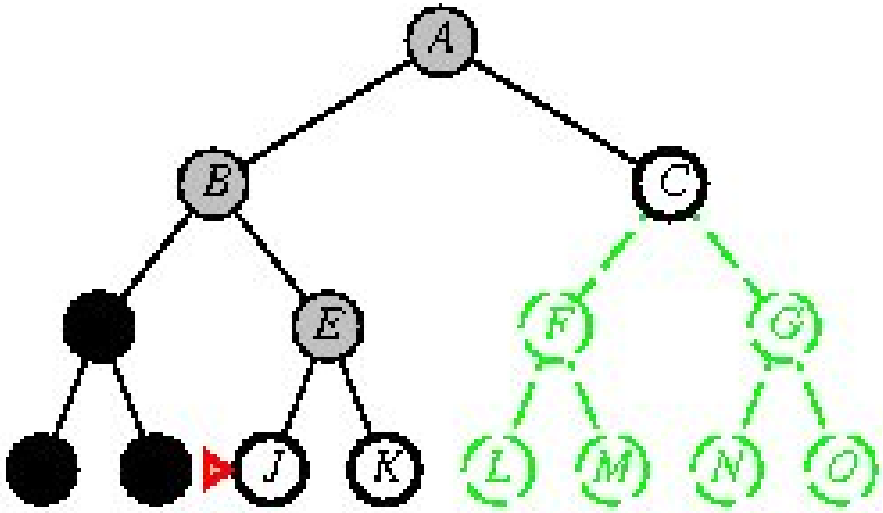
$$LISTA\_OPEN = \{I, E, C\}$$
$$LISTA\_CLOSED = \{A, B, D, H\}$$

DFS — szukanie w głąb



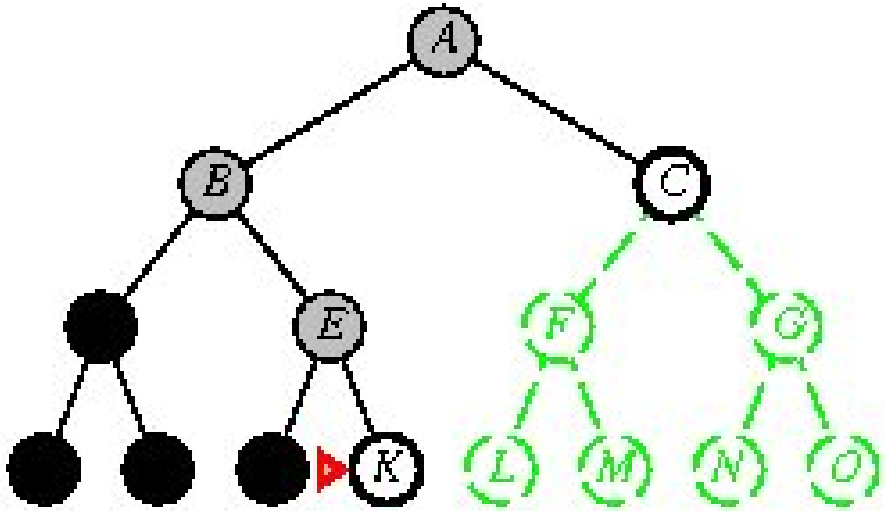
$$LISTA\_OPEN = \{E, C\}$$
$$LISTA\_CLOSED = \{A, B, D, H, I\}$$

DFS — szukanie w głąb



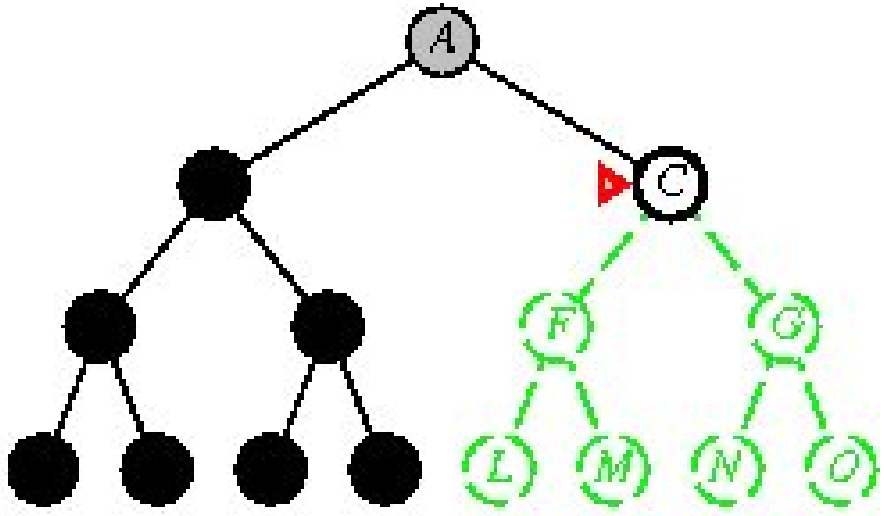
$$LISTA\_OPEN = \{J, K, C\}$$
$$LISTA\_CLOSED = \{A, B, D, H, I, E\}$$

DFS — szukanie w głąb



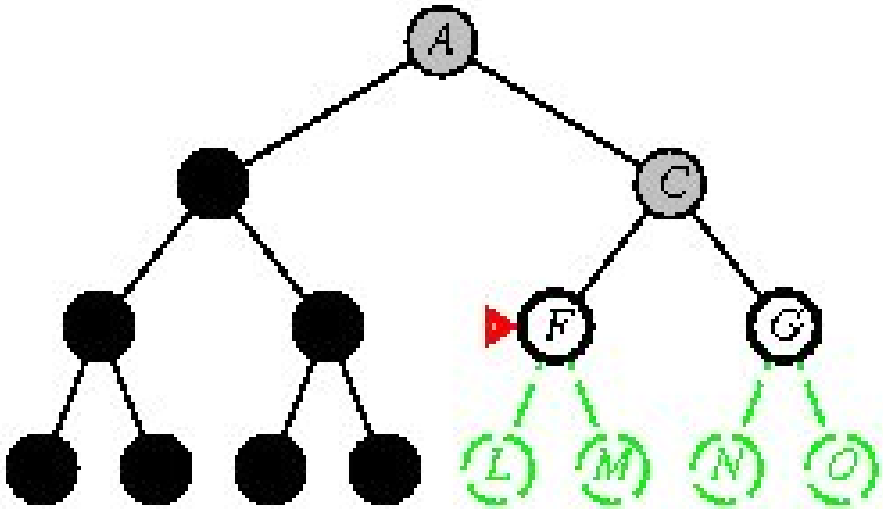
$$LISTA\_OPEN = \{K, C\}$$
$$LISTA\_CLOSED = \{A, B, D, H, I, E, J\}$$

DFS — szukanie w głąb



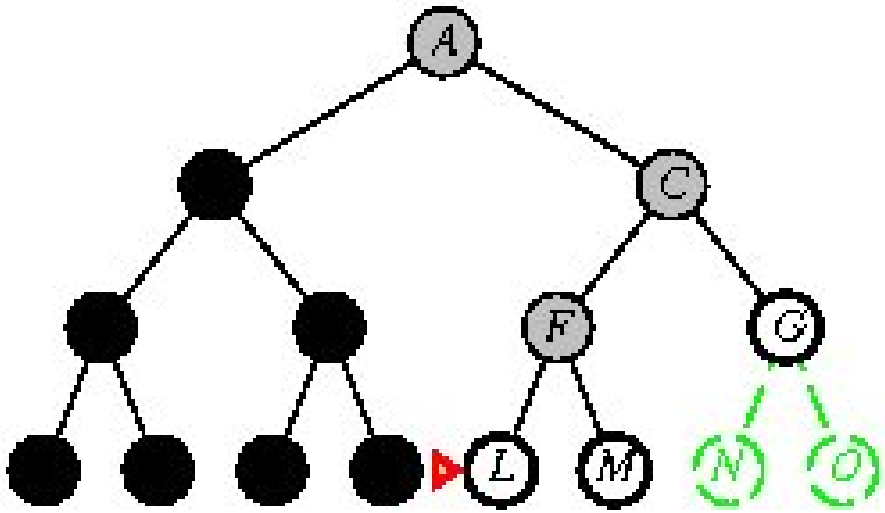
$$LISTA\_OPEN = \{C\}$$
$$LISTA\_CLOSED = \{A, B, D, H, I, E, J, K\}$$

DFS — szukanie w głąb



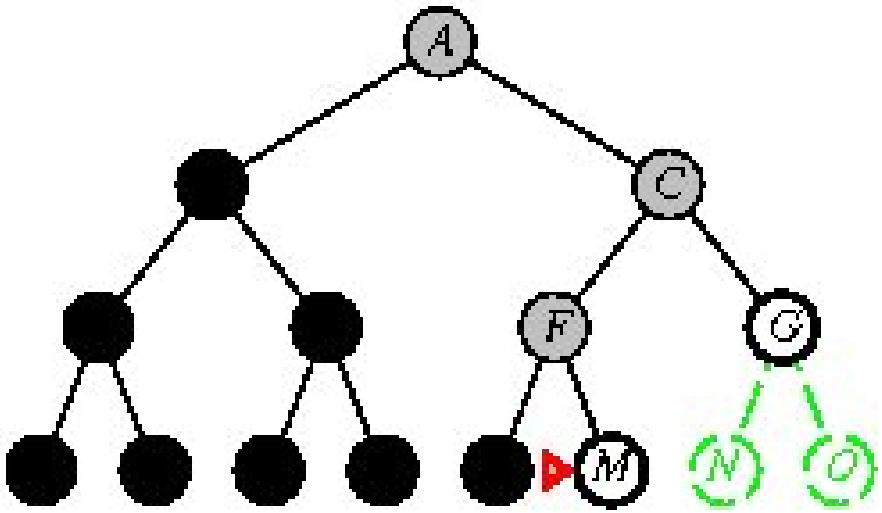
$$LISTA\_OPEN = \{F, G\}$$
$$LISTA\_CLOSED = \{A, B, D, H, I, E, J, K, C\}$$

DFS — szukanie w głąb



$$LISTA\_OPEN = \{L, M, G\}$$
$$LISTA\_CLOSED = \{A, B, D, H, I, E, J, K, C, F\}$$

DFS — szukanie w głąb



$$LISTA\_OPEN = \{M, G\}$$
$$LISTA\_CLOSED = \{A, B, D, H, I, E, J, K, C, F, L\}$$

Ocena algorytmu DFS

Zupełność

Tak, jeżeli stosowana jest LISTA CLOSED (graf).
Nie, jeżeli nie pamięta się już odwiedzonych węzłów (drzewo).

Złożoność czasowa

$O(b^m)$: gdy m — maksymalna głębokość drzewa. Złożoność wzrasta, gdy m jest dużo większe od d. W praktyce bardzo często istnieje w drzewie wiele rozwiązań, zatem DFS znajdzie nieoptymalne i ciągle może być szybszy niż BFS.

Ocena algorytmu DFS

Obszar zajmowanej pamięci

Przechowuje pojedynczą ścieżkę w pamięci od korzenia do liścia i braci tych węzłów na każdym poziomie (LISTA OPEN).
Zatem ma niewielkie wymagania pamięci - złożoność liniowa: $O(bm)$.

Optymalność

Nie, gdyż może znaleźć rozwiązanie na poziomie większym niż d, jeżeli jest ono w gałęzi przeszukiwanej wcześniej.

IDS — iteracyjnie w głąb

Iterative deepening depth-first search jest modyfikacją DFS gdzie ogranicza się głębokość przeszukiwania. Dzięki temu łączy w sobie zalety DFS i BFS.

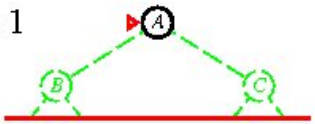
- Algorytm poszukuje najlepszej wartości ograniczenia poziomu przeszukiwań (zmienna *limit*).
- Jest to realizowane przez stopniowe zwiększanie *limit* od 0 do d co 1, dopóki nie znajdzie rozwiązania.
- Rozwiązanie zostanie znalezione na poziomie d (najbliższe rozwiązanie).
- Algorytm jest zupełny i optymalny, gdy koszt ścieżki jest niemalejącą funkcją głębokości drzewa.
- Sprawdza się w zadaniach, w których nie jest znany poziom rozwiązania.

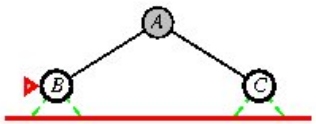
IDS — przykład

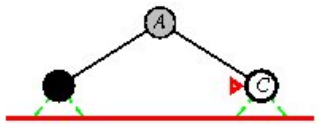
Limit = 0

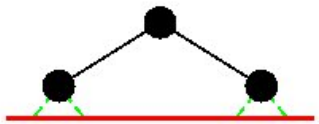
IDS — przykład

Limit = 1

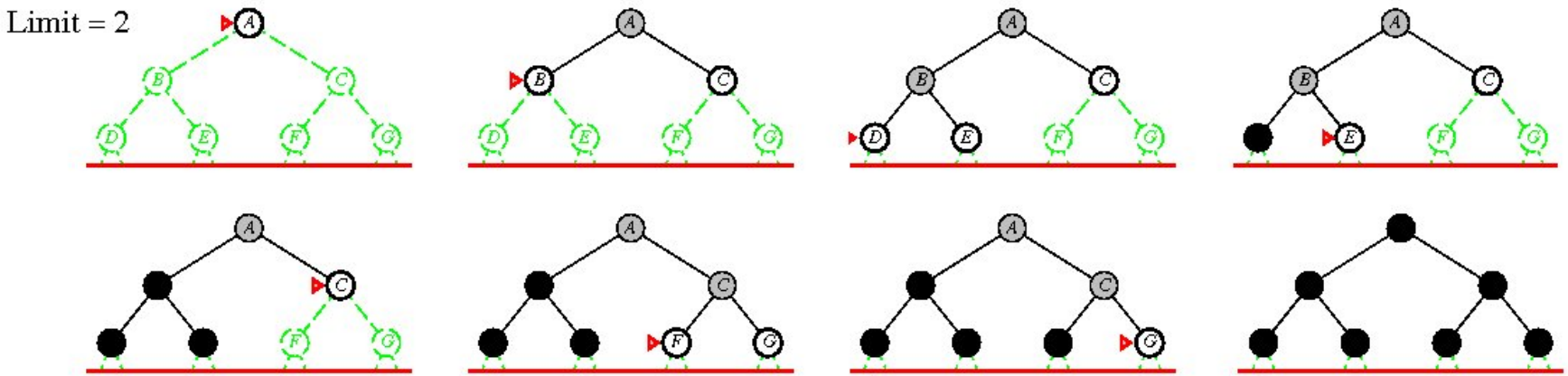




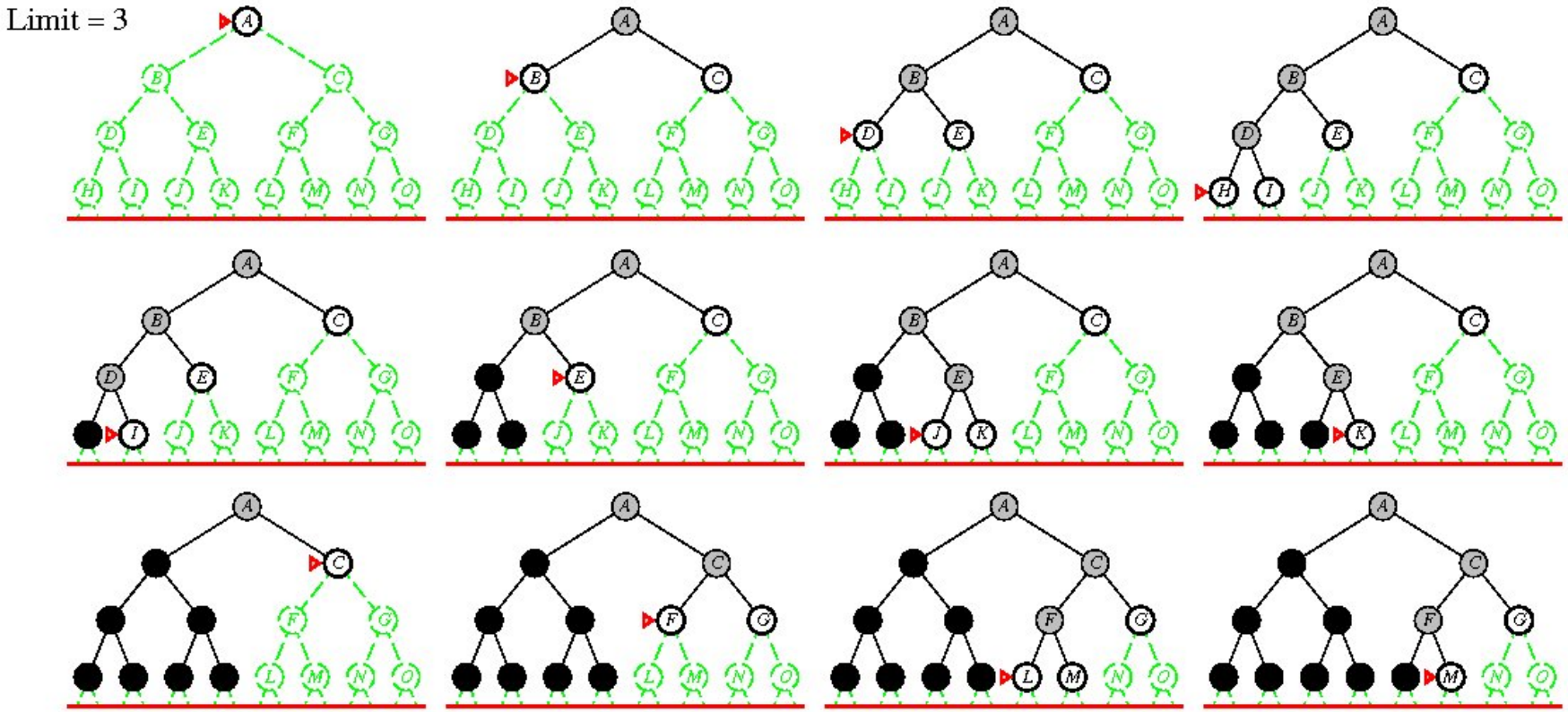




IDS — przykład



IDS — przykład



Ocena algorytmu IDS

Zupełność

Tak, zawsze znajdzie istniejące rozwiązanie.

Złożoność czasowa

Algorytm wydaje się być kosztowny ze względu na wielokrotne powtarzanie szukania na niektórych poziomach.
Acz... Węzły na poziomie d będą przeszukiwane 1 raz, na poziomie $d-1$ dwa razy itd. aż do korzenia, który będzie rozwinięty $d+1$ razy. Stąd całkowita liczba węzłów to:
$N(IDS) = (d+1)1 + db + (d-1)b^2 + \cdots + 3b^{d-2} + 2b^{d-1} + b^d$
dla BFS to: $N(BFS) = b + b^2 + \cdots + b^d + (b^{d+1} - b)$
Porównanie: $b = 10$ i $d = 5$:
$N(IDS) = 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456$,
gdy $N(BFS) = 10 + 100 + 1,000 + 10,000 + 100,000 + 999,990 = 1,111,100$

Ocena algorytmu IDS

Obszar zajmowanej pamięci

Przechowuje w pamięci pojedynczą ścieżkę od korzenia do liścia i wszystkich braci.
Złożoność algorytmu DFS: $O(bm)$

Optymalność

Tak, jeżeli koszt ścieżki jest stały lub jest niemalejącą funkcją głębokości w drzewie.

Strategie z informacją — informed search

Strategie heurystyczne

Wykorzystują dodatkową informację, specyficzną wiedzę o rozwiązywanym problemie, co pozwala 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

Traveling Salesman Person (problem komiwojażera)

Założenia:

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

Dla N miast istnieje $1 \cdot 2 \cdot 3 \cdot \cdots \cdot (N-1)$ możliwych kombinacji (tras komiwojażera). Czas jest proporcjonalny do N, a całkowity do $N!$, zatem
dla 10 miast jest $10! = \mathbf{3\ 628\ 800}$ tras
a dla 11 miast aż **39 916 800** tras.

Best-First search

Best-first search (najpierw najlepszy) wykorzystuje funkcję oceny ($f(n)$).

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

Warianty best-first search:

- *Szukanie zachłanne* (greedy best-first search)
- A^*

Funkcja heurystyczna

Funkcja odwzorowująca stany we współczynnik ich użyteczności.

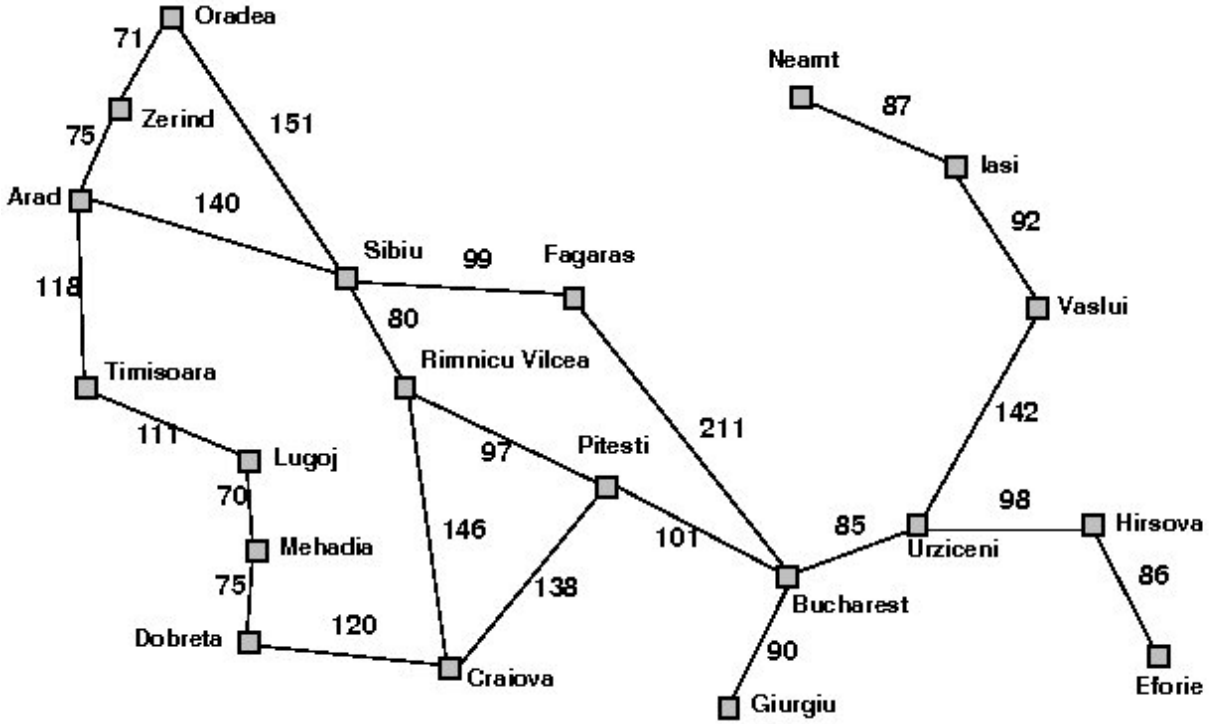
- Najczęściej jej przeciwdziedzina to $\mathcal{R}^+$.
- Zazwyczaj określa „odległość" od rozwiązania.
- Przyjmuje się, że $h(n) = 0$, gdy $n = cel$ (rozwiązanie).

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

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

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

Przykład — podróż po Rumunii (założenia)



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h_{SLD} — straight-line distance heuristic. Odległość pomiędzy dwoma miastami wyznaczona jako długość odcinka pomiędzy nimi.

- $h_{SLD}(Arad) = 366$
- h_{SLD} nie jest bezpośrednią daną wejściową problemu.

Przeszukiwanie zachłanne

Greedy best-first search

Rozwija węzły, które są najbliższe rozwiązania wg $f(n)$, zakładając, że prawdopodobnie prowadzi to do najszybszego rozwiązania.

Oszacowuje koszt węzła stosując tylko funkcję heurystyczną

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

Minimalizowanie $h(n)$ jest podatne na niewłaściwy punkt startowy (prowadzi do suboptymalnego rozwiązania).

Greedy best-first search przypomina DFS, bo wybiera w pewnym sensie jedną ścieżkę.

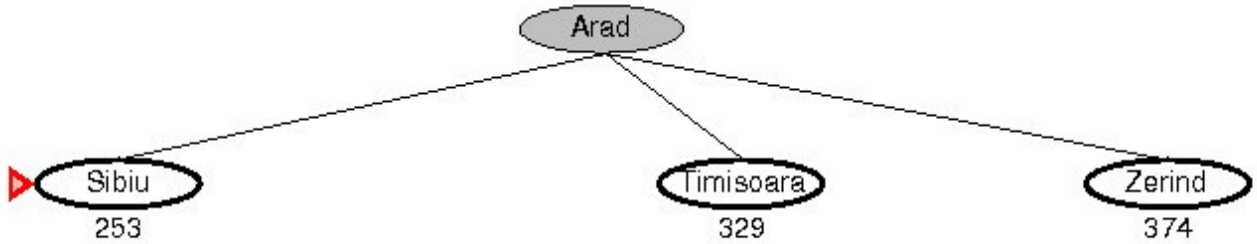
Przeszukiwanie zachłanne — przykład

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

Stan początkowy: Arad

$LISTA\_OPEN = \{Arad.h(Arad) = 366\}$

Przeszukiwanie zachłanne — przykład

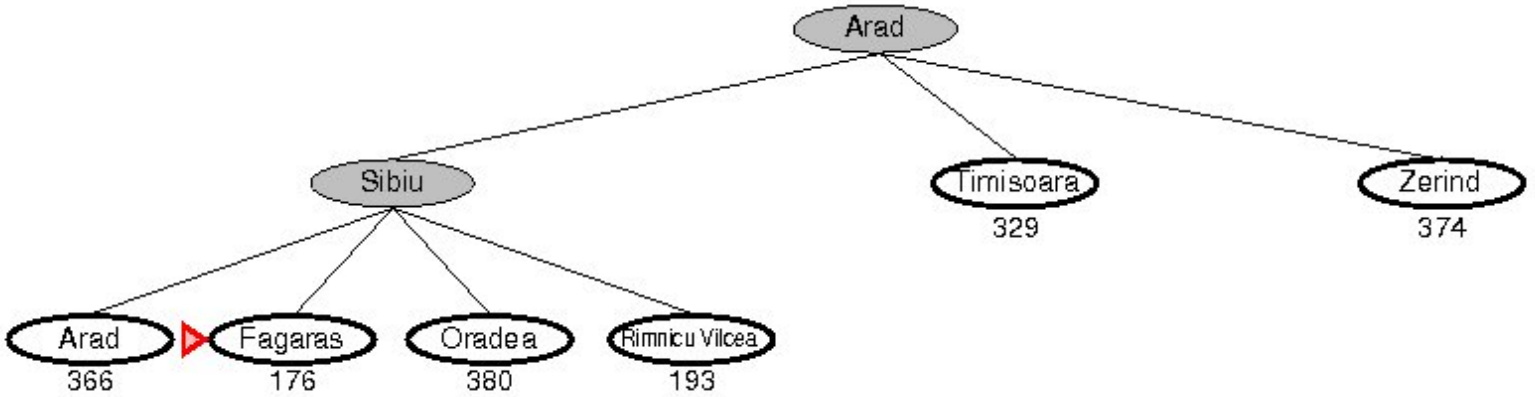


Pierwszy krok rozwija węzeł Arad i generuje potomków: Sibiu, Timisoara and Zerind

$LISTA\_OPEN = \{Sibiu.h(Sibiu) = 253, Timisoara.h(Timisoara) = 329, Zerind.h(Zerind) = 374\}$

Greedy best-first wybierze Sibiu.

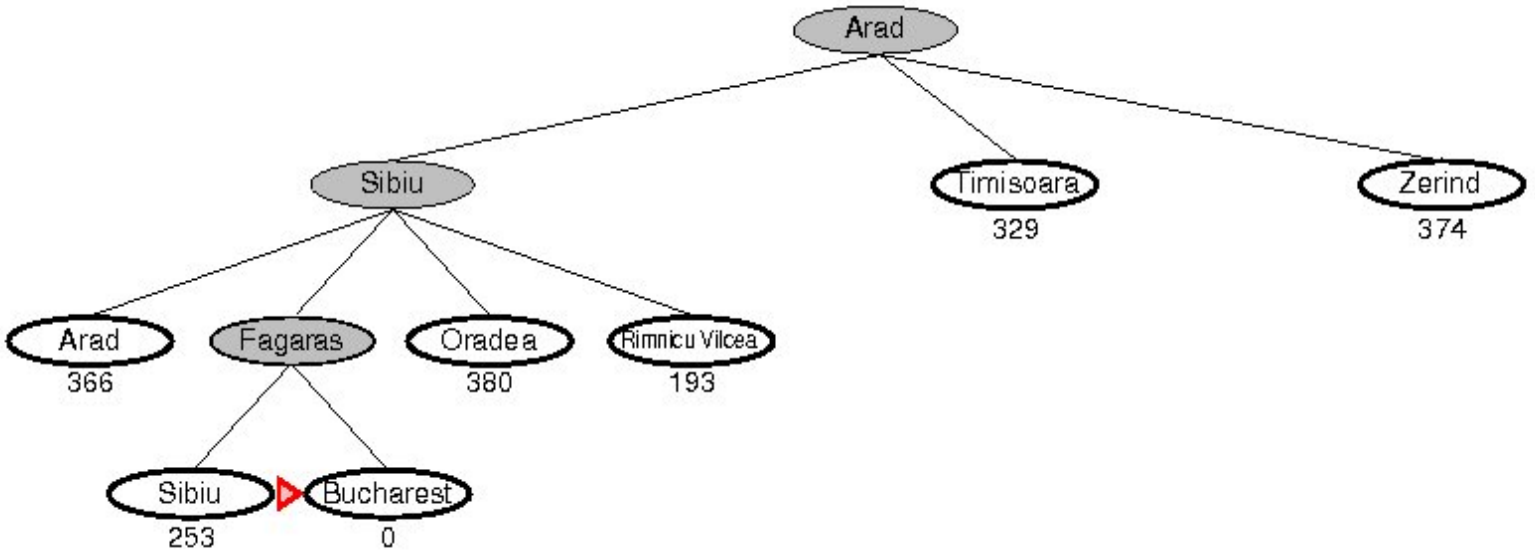
Przeszukiwanie zachłanne — przykład



Po rozwinięciu Sibiu otrzymamy :
$LISTA\_OPEN = \{Fagaras.h(Fagaras) = 176, RimnicuVilcea.h(RimnicuVilcea) = 193, Timisoara.h(Timisoara) = 329, Arad.h(Arad) = 366, Zerind.h(Zerind) = 374, Oradea.h(Oradea) = 380\}$

Greedy best-first wybierze Fagaras.

Przeszukiwanie zachłanne — przykład



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

Ocena algorytmu Greedy Best First Search

Zupełność

Tak, jeżeli nie ma pętli nieskończonych, co można zapewnić zastosowaniem LISTY CLOSED i dobrej heurystyki.

Złożoność czasowa

$O(b^m)$ — choć dobra heurystyka może dać znaczne zmniejszenie złożoności czasowej.

Obszar zajmowanej pamięci

Przechowuje w pamięci wszystkie węzły.
Złożoność algorytmu: $O(b^m)$

Optymalność

Nie.

A*

Główne cechy algorytmu:

Funkcja oceny dla A*

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

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

A*

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

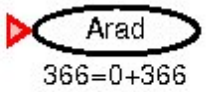
Twierdzenie

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

- Nigdy nie przeceni kosztu dotarcia do rozwiązania: $h(n) \leq h^*(n)$, gdzie $h^*(n)$ jest rzeczywistym kosztem osiągnięcia rozwiązania z n. Innymi słowy jest z natury optymistyczna, gdyż 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 może być większa od rzeczywistego dystansu.

A* : przykład

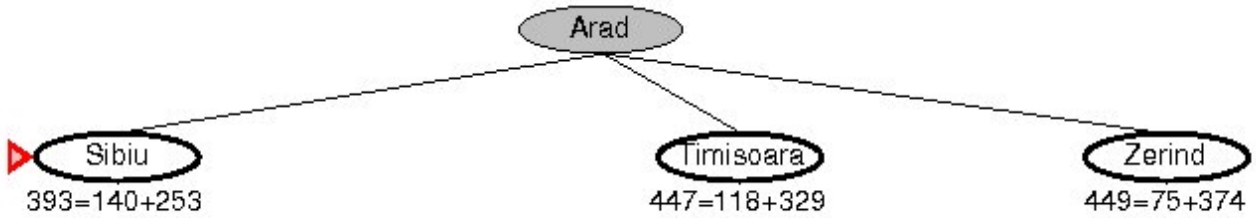


Szukamy najkrótszej drogi z Aradu do Bukaresztu.

Stan początkowy: Arad

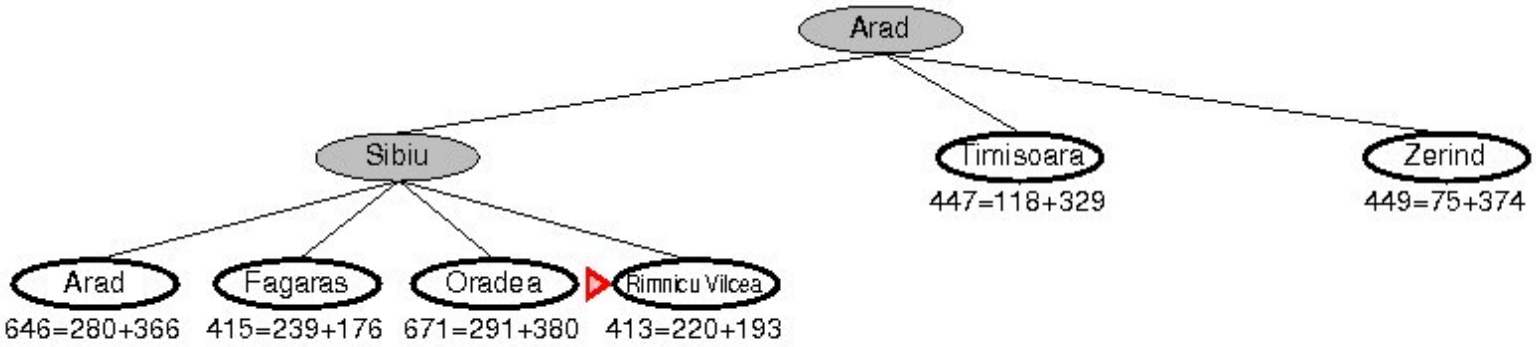
$f(Arad) = g(Arad) + h(Arad) = 0 + 366 = 366$

A* : przykład



Pierwszy krok rozwija węzeł Arad i generuje potomków:
$f(Sibiu) = g(Arad, Sibiu) + h(Sibiu) = 140 + 253 = 393$
$f(Timisoara) = g(Arad, Timisoara) + h(Timisoara) = 118 + 329 = 447$
$f(Zerind) = g(Arad, Zerind) + h(Zerind) = 75 + 374 = 449$
A^* wybierze Sibiu.

A* : przykład



Jeżeli rozwiniemy Sibiu to otrzymamy potomków, których dodajemy do LISTY OPEN:

$f(Arad) = g(Sibiu, Arad) + h(Arad) = 280 + 366 = 646$

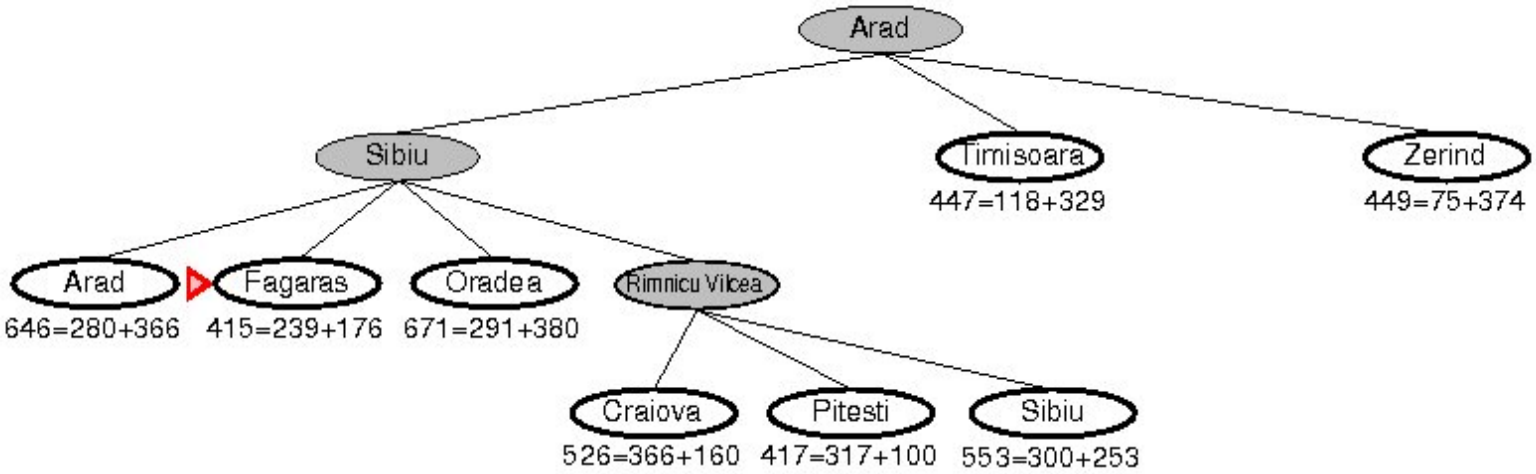
$f(Fagaras) = g(Sibiu, Fagaras) + h(Fagaras) = 239 + 179 = 415$

$f(Oradea) = g(Sibiu, Oradea) + h(Oradea) = 291 + 380 = 671$

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

A^* wybierze Rimnicu Vilcea.

A* : przykład



Jeżeli rozwiniemy Rimnicu Vilcea to otrzymamy potomków, których dodajemy do LISTY OPEN:

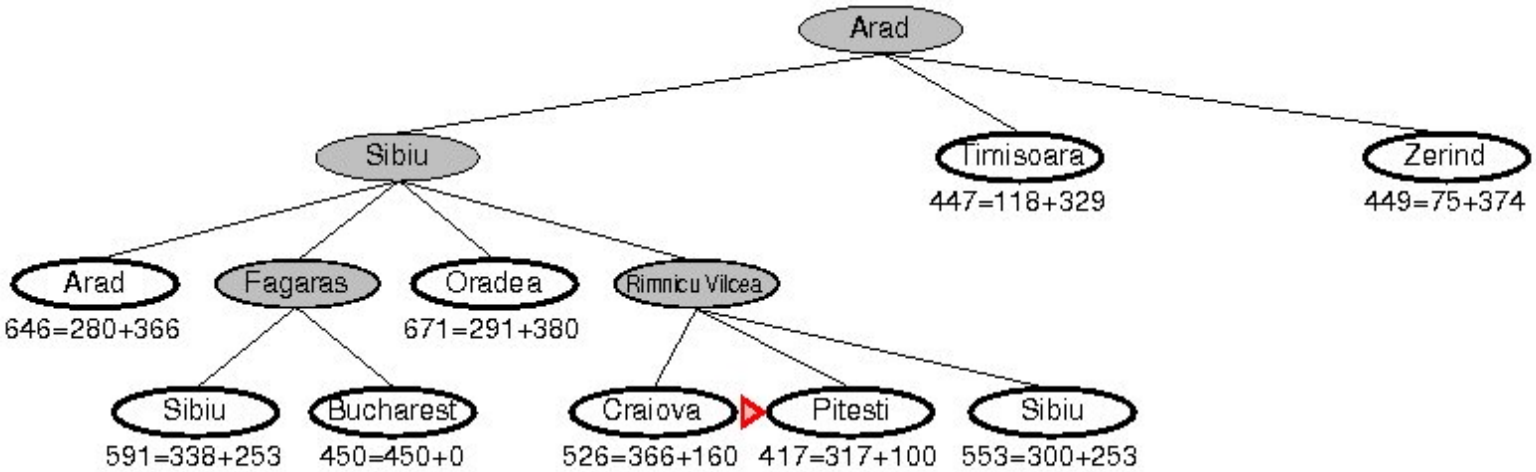
$f(Craiova) = g(RimnicuVilcea, Craiova) + h(Craiova) = 360 + 160 = 526$

$f(Pitesti) = g(RimnicuVilcea, Pitesti) + h(Pitesti) = 317 + 100 = 417$

$f(Sibiu) = g(RimnicuVilcea, Sibiu) + h(Sibiu) = 300 + 253 = 553$

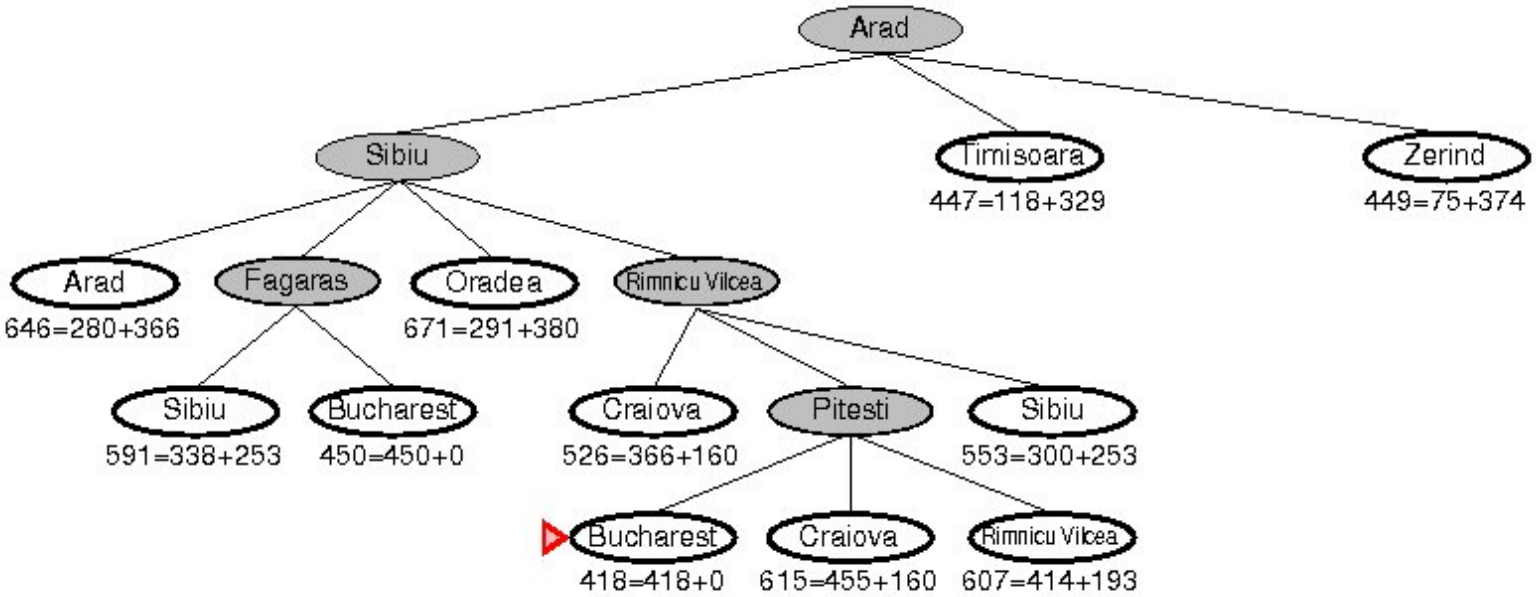
A^* wybierze Fagaras - węzeł pamiętany z poprzednich rozwinięć.

A* : przykład



Jeżeli rozwiniemy Fagaras to najmniejszą wartość $f(Fagaras)$ ma Pitesti (417km), które rozwijamy:
$f(Sibiu) = g(Fagaras, Sibiu) + h(Sibiu) = 338 + 253 = 591$
$f(Bucharest) = g(Fagaras, Bucharest) + h(Bucharest) = 450 + 0 = 450$

A* : przykład



Wybieramy do rozwinięcia Pitesti, bo $f(Pitesti)$ ma najmniejszą wartość i dotrzemy najkrótszą drogą do Bukaresztu (418km):
$f(Bucharest) = g(Pitesti, Bucharest) + h(Bucharest) = 418 + 0 = 418$

Ocena algorytmu A*

Zupełność

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

Złożoność czasowa

Niestety wykładnicza

Obszar zajmowanej pamięci

Przechowuje w pamięci wszystkie węzły (wykładnicza).

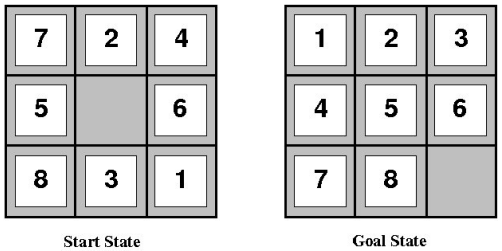
Optymalność

Tak, jeżeli $h(n)$ dopuszczalne.

Jak dobierać heurystykę: przykład

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

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



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

Jak dobierać heurystykę: przykład

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.

Plan wykładu

1. Strategie — czyli jak znaleźć rozwiązanie problemu
2. Globalne metody przeszukiwania (optymalizacji)
   - Zadanie optymalizacji
   - Algorytmy ewolucyjne

Jakie problemy wynikają z metod przeszukiwania

Duże wymagania pamięciowe

Metody dotychczas omawiane systematycznie eksplorowały przestrzeń zachowując w pamięci alternatywne ścieżki do momentu odnalezienia rozwiązania.

Przeformułowanie celu

W wielu problemach takich jak: projektowanie układów cyfrowych, harmonogramowanie, wyznaczanie drogi pojazdu, optymalizacja sieci telekomunikacyjnych itd. nie znane jest rozwiązanie oczekiwane (stan końcowy). Natomiast ważne jest znalezienie **rozwiązania (stanu, ścieżki)** o jak najlepszych walorach.

Globalne metody przeszukiwania

Globalne metody przeszukiwania

Operują na pojedynczym stanie z pewnej przestrzeni i generują krok do stanu sąsiedniego. Algorytmy nie są systematyczne, ale mają następujące zalety:

- Małe wymagania pamięciowe: zazwyczaj stała wielkość.
- Często znajdują akceptowalne rozwiązania w dużej i nieskończonej przestrzeni rozwiązań, gdy metody systematyczne zawodzą.

Metody te są dedykowane do zadań optymalizacji, w których celem jest znalezienie najlepszego stanu (spośród wielu kandydatów) według funkcji celu, która jest funkcją heurystyczną.

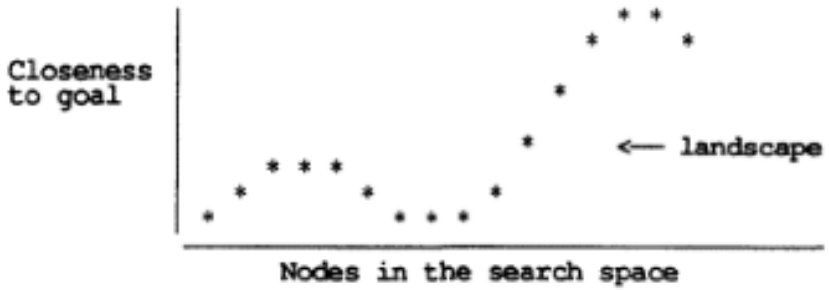
Problemy, w których wykorzystuje się globalne metody przeszukiwania

- pokrycie wierzchołków w grafie (odnaleźć rozwiązanie z najmniejszą liczbą węzłów)
- problem komiwojażera (cykl w grafie o najkrótszej długości cyklu)
- problem spełnialności w rachunku zdań (znalezienie wszystkich klauzul spełniających pewne zadane przypisanie)
- Nurse scheduling problem (NSP) (plan zmian z uwzględnieniem zbioru ograniczeń)
- problemy klasteryzacji (znajdowanie najlepszego dopasowania)

Krajobraz przestrzeni stanów

Krajobraz przestrzeni stanów (landscape) posiada:

- położenie (oś X) — definiowane przez stan
- wzniesienia (oś Y)— zdefiniowane prze wartość funkcji celu (funkcji heurystycznej) oceniającej „closeness to goal".



Każdy stan jest bardzo często równy „pełnym konfiguracjom", które można traktować jako potencjalne rozwiązanie problemu, a zadanie dla strategii polega na poszukiwaniu optymalnej konfiguracji/rozwiązania.

Globalna optymalizacja

Zadanie optymalizacji

Najczęstszą formą jest optymalizacji wyrażania zadania optymalizacji jest minimalizacja funkcji f o zmiennych rzeczywistych w przestrzeni parametrów $\overrightarrow{x} \in P$. Zazwyczaj dany jest zbiór ograniczeń na wektor rozwiązań $\overrightarrow{x}_{min}$.

W problemach rzeczywistych funkcje wielu zmiennych mają dużą liczbą minimów i maksimów lokalnych. Znajdowanie dowolnego lokalnego optimum jest w miarę łatwo osiągalne przez metody lokalne. Znajdowanie globalnego optimum jest znacząco bardziej wymagające i dla wielu problemów do dnia dzisiejszego niewykonalne.

Ograniczenia na dziedzinę (paramatery) lub przeciwdziedzinę(wartości funkcji)

Rozważmy zadanie minimalizacji funkcji Rosenbrocka

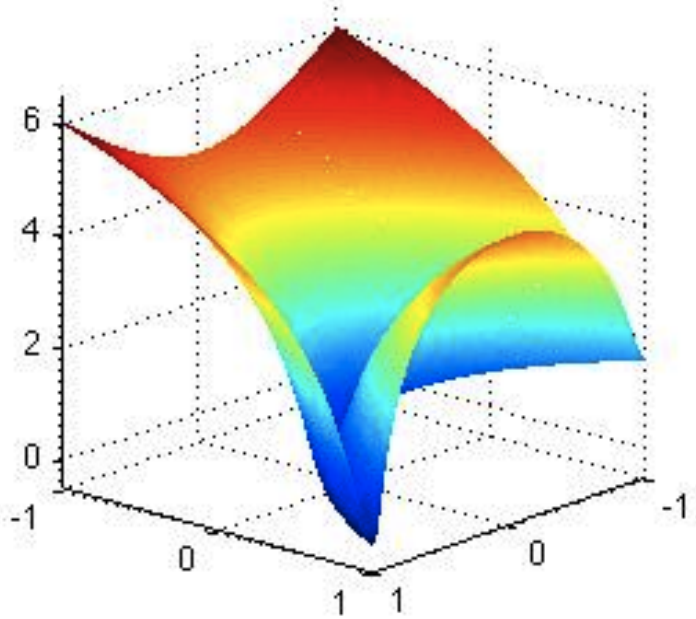
$$f(x) = 100(x_2 + x_1^2)^2 + (1 + x_1)^2$$

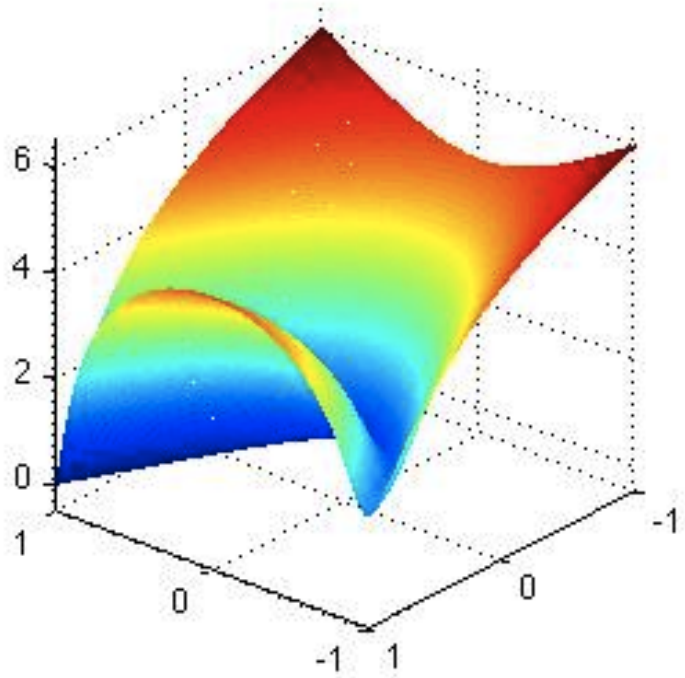
w przestrzeni ograniczonej przez koło o promieniu 1 ze środkiem w początku układu współrzędnych.
Czyli należy znaleźć minimum funkcji $f(x)$, gdy parametry funkcji spełniają nierówność:

$$x_1^2 + x_2^2 \leq 1$$

Zatem jest to problem optymalizacji funkcji nieliniowej z nieliniowym ograniczeniem.

Funkcja Rosenbrocka





Przykład zadania optymalizacji z wieloma funkcjami celu

Optymalizacja wielokryterialna to często znalezienie kompromisu pomiędzy sprzecznymi celami. Np. patrząc na pewien zakład produkcyjny możemy określić następujące kryteria optymalizacji:

1. minimalizacja czasu pomiędzy przyjęciem zlecenia a dostawą
2. maksymalizacja zysków
3. minimalizacja kosztów reklamy, zatrudnienia personelu, zużycia materiałów itp.
4. maksymalizacja jakości produktu
5. minimalizacja negatywnego wpływu na środowisko naturalne

Pewne zależności pomiędzy celami są od razu widoczne, ale wiele jest ukrytych i dopiero wnikliwa analiza danych może wskazać pewne ukryte zależności.

Taksonomia metod optymalizacji http://www.it-weise.de/projects/book.pdf

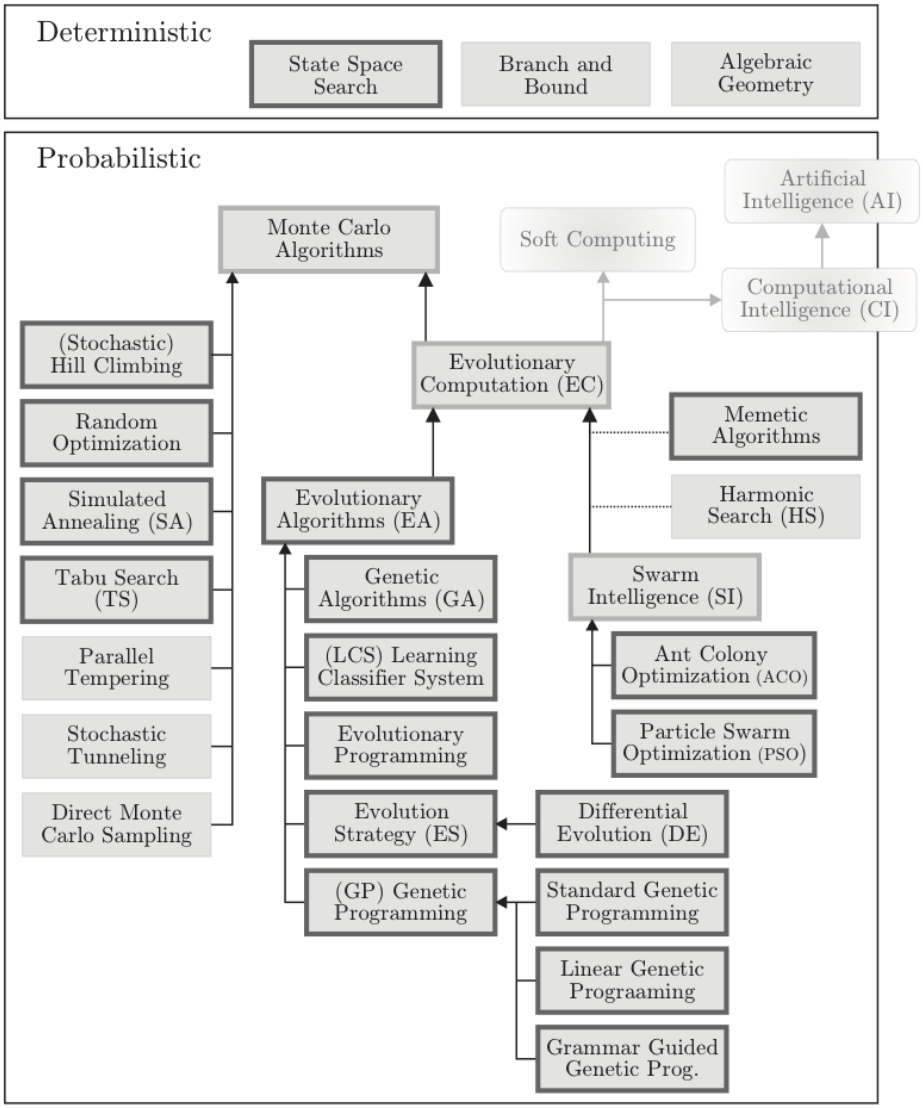
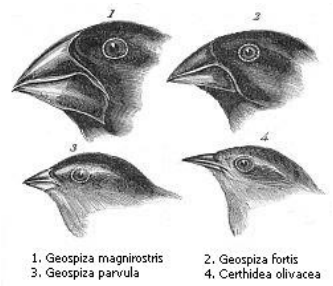


Figure 1.1: The taxonomy of global optimization algorithms.

Podstawy — dobór naturalny

Dobór naturalny

Podstawowy mechanizm ewolucji. Proces, który prowadzi do ukierunkowanych zmian w populacji. Dzięki temu zwiększa się średnie przystosowanie populacji do środowiska.



Finches from Galapagos Archipelago

Ewolucja

Organizm rozwija się w środowisku i w populacji osobników tego samego gatunku. Osobniki różnią się między sobą. Środowisko ocenia osobniki na postawie ich cech zewnętrznych (fenotypu), które to zależą od zestawu genów osobnika.

Selekcja, krzyżowanie

Selekcja

jest nieustająca, powolna, powszechna. Darwin zauważył, że na świat przychodzi więcej potomstwa, niż może pomieścić środowisko zatem większość musi zginąć. Przeżywają tylko nieliczni, ale za to najlepiej przystosowani do środowiska.

Krzyżowanie

W procesie rozrodu osobniki przekazują potomstwu swoje własne cechy, a cechy najsilniejszych osobników mieszają się, dając kombinacje dotąd nie występujące.

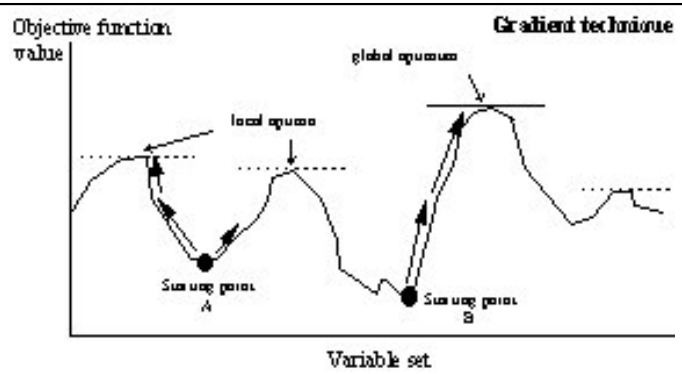
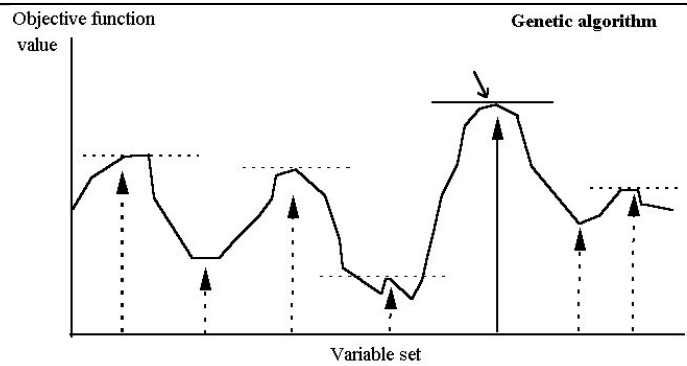
Algorytm ewolucyjny

Jest to metoda rozwiązywanie problemów optymalizacji bez/z ograniczeniami jedno/wielokryterialnych w oparciu o selekcję naturalną.

Cechy algorytmu

- Wielokrotnie zmienia populację osobników (reprezentacja rozwiązania problemu).
- Losowo wybiera z populacji osobniki, które staną się rodzicami tworzącymi nowe potomne osobniki do następnej generacji osobników.
- Z każdą następną generacją populacja ewoluuje w kierunku rozwiązania optymalnego.
- Można rozwiązywać problemy nierozwiązywalne innymi metodami: nieciągła, nieróżniczkowalna, stochastyczna lub silnie nieliniowa funkcja celu.

Porównanie z innymi metodami optymalizacji

Algorytm klasyczny	Algorytm ewolucyjny
	
W każdym kroku tworzy pojedynczy punkt przestrzeni. Sekwencja punktów prowadzi do rozwiązania (sub)optymalnego.	W każdej iteracji tworzy populację punktów. Najlepszy punkt z populacji prowadzi do rozwiązania (sub)optymalnego
Wybiera kolejny punkt w ciągu deterministycznie.	Losowo wybiera osobniki do następnej populacji.
Funkcja generowanie potomków wynika z problemu	Funkcja generowania potomków jest niezależna od problemu (operatory geneteyczne)

Terminologia algorytmów ewolucyjnych

Funkcja przystosowania (celu)

Jest to funkcja, która jest optymalizowana np.
$f(x_1, x_2, x_3) = (2x_1 + 1)^2 + (3x_2 + 4)^2 + (x_3 - 2)^2$
Dla klasycznych algorytmów optymalizacji jest to funkcja celu.
Funkcja ta mierzy jak dobry (użyteczny) jest osobnik jako rozwiązanie danego problemu.

Osobnik/chromosom/gen

Osobnik jest to punkt w przestrzeni (potencjalne rozwiązanie problemu). Osobnik ma postać zakodowanego ciągu o określonej długości zwanego chromosomem. Poszczególne elementy chromosomu to geny.
Na przykład wektor $(2, -3, 1)$ jest osobnikiem z genami 2, -3, 1.
Wartość funkcji przystosowania dla tego osobnika to $f(2, -3, 1) = 51$.

Terminologia algorytmów ewolucyjnych cd.

Populacja

Populacja to tablica osobników, np. jeżeli liczność populacji jest 100, a długość osobnika 3, to populacja jest reprezentowana przez tablicę 100x3.
Ten sam osobnik może pojawić się w populacji więcej niż raz.

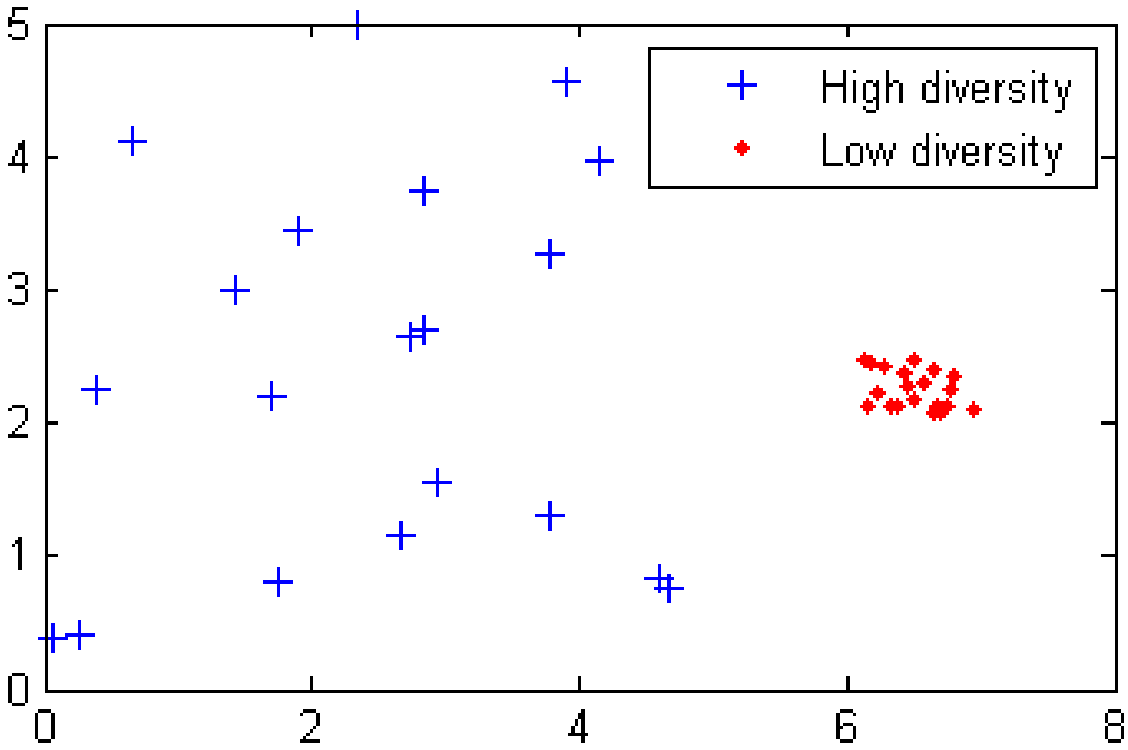
Generacja

W każdej iteracji algorytm wykonuje serię operacji, obliczeń na osobnikach i populacji. W wyniku tego zmieniają się osobniki i powstaje **nowa populacja**. Każda kolejna populacja nazywana jest nową generacją.

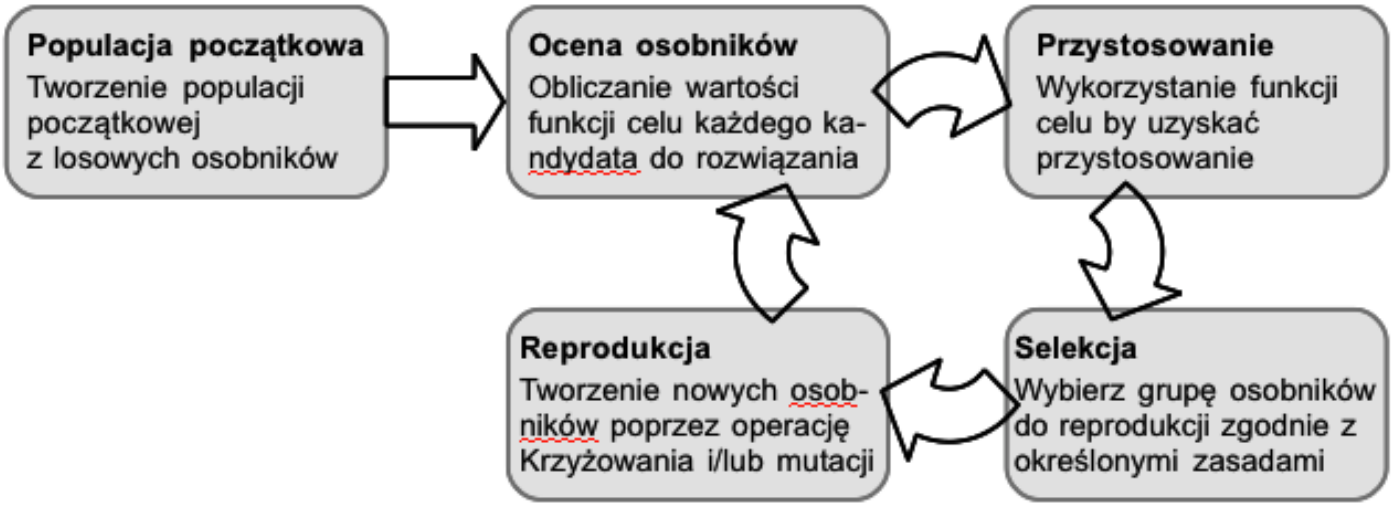
Terminologia algorytmów ewolucyjnych cd.

Zróżnicowanie populacji

Jest to parametr określający średnią odległość pomiędzy osobnikami w populacji. Duża różnorodność, to duża wartość średniej. Mała różnorodność, to niewielka odległość pomiędzy osobnikami.



Algorytm ewolucyjny — schemat działania



Rodzice/dzieci

Rodzice to osobniki wybrane do reprodukcji służą do tworzenia nowych osobników nazywanych dziećmi. Algorytm ma tendencje do wybierania jako rodziców te osobniki, które mają wyższe przystosowanie.

Dopasowanie algorytmu ewolucyjnego do rozwiązywanego problemu

Zanim przystąpi się do rozwiązywania problemu z użyciem AE należy określić niezbędne do uruchomienia algorytmu parametry.

1. Ustalenie kodowania: ustalenie chromosomu jako reprezentanta wyniku.
2. Ustalenie funkcji przystosowania: Co jest celem optymalizacji? Jak oceniać osobniki?
3. Wybranie operatorów genetycznych: Jakie stosować krzyżowanie, mutacje i selekcje?
4. Ustalenie wartości pewnych parametrów: np. liczność populacji, prawdopodobieństwo krzyżowania, prawdopodobieństwo mutacji, itp.
5. Warunek zatrzymania algorytmu. Kiedy zatrzymać cykl generowania nowych populacji?

Rodzaje chromosomów stosowane w AE

Kodowanie — reprezentacja danych — zbiór stanów przestrzeni zadania przedstawiony w postaci skończonego alfabetu znaków.

- Klasyfikacja ze względu na przyjmowane wartości kodowe:
  - binarne
  - całkowitoliczbowe
  - zmiennoprzecinkowe
  - ciągi znaków
- Klasyfikacja ze względu na strukturę:
  - standardowe
  - permutacyjne
  - drzewiaste

Kodowanie heterogeniczne

Geny heterogeniczne

Geny na różnych pozycjach w chromosomie przechowują informacje różnego typu. Każda kodowana cecha fenotypu ma ściśle przypisaną na stałe pozycję w chromosomie. W wyniku krzyżowania i mutacji geny nie przemieszczają się, tylko zmieniają wartości

- **Przykład problemu:** Problem plecakowy
  Kodowanie: Każdy gen ma wartość 0 lub 1 i określa, czy element jest, czy go nie ma w plecaku. Długość chromosomu zależy od liczby elementów w zbiorze.
- **Przykład problemu:** Optymalizacja funkcji n zmiennych danej wzorem
  Kodwanie: Każdy gen jest liczbą rzeczywistą i jest jedną z n zmiennych. Zazwyczaj kolejność jest zachowana $x_1, x_2, \ldots, x_n$. Wartości x-ów zależą od dziedziny
- **Przykład problemu:** n-hetmanów
  Kodwanie: Gen jest liczbą całkowitą od 1 do n i wskazuje wiersz położenia hetmana. Kolejne geny, to kolejne kolumny.

Kodowanie permutacyjne

Geny są homogeniczne

Geny przechowują podobne informacje, są wymienialne. W wyniku krzyżowania i mutacji geny nie zmieniają wartości, natomiast zmieniają pozycje.

Przykład problemu: Problem komiwojażera
Kodowanie: Chromosom podaje numery miast (kolejność) w jakiej komiwojażer przejechał trasę. W przykładzie poniżej ChromosomA opisuje trasę przejazdu z miasta 1 do 5 potem do 3 itd.

Chromosom	A	1	5	3	2	6	4	7	9	8
Chromosom	B	8	5	6	7	2	3	1	4	9

Poprawność kodowania

Kodowanie jest bardzo istotnym etapem projektowania algorytmu. Sposób kodowania wydatnie wpływa na szybkość i jakość znajdywanych wyników, na sposób w jaki przeszukiwana jest przestrzeń rozwiązań. Złe kodowanie może spowodować, że nigdy nie zostanie przeszukany fragment przestrzeni, w którym znajdują się najlepsze rozwiązania.

D - zbiór fenotypów
G - zbiór genotypów
$c : D \longrightarrow G$ - funkcja kodująca

Kodowanie powinno zapewnić:

$$\forall_{d \in D} \exists_{g \in G} c(d) = g$$

Każde rozwiązanie zadania można przedstawić jako genotyp i kodowanie nie wprowadza dodatkowych ekstremów lokalnych.

Zdefiniowanie funkcja przystosowania

Funkcja przystosowania (ang. FF — fitness function)

FF przypisuje każdemu chromosomowi pewną wartość liczbową, która mówi jak daleko od rozwiązania jest badany osobnik lub jak użyteczny jest dany osobnik. Jest to funkcja heurystyczna.

FF jest zawsze zależna od rozwiązywanego problemu!

Wyższa (gdy zadaniem jest maksymalizacja) wartość funkcji oznacza lepsze cechy i osobnik ma większe szanse przejścia do następnego pokolenia. (Darwinowska przeżywalność najlepiej przystosowanych).

Przykłady funkcji przystosowania

$f(ch_i)$ — funkcja przystoswania dla i-tego chromosomu.

Problem komiwojażera (minimalizacja drogi)

$$f(ch_i) = \sum_{j=1}^{n-1} dist(ch_{ij}, ch_{ij+1})$$

Problem plecakowy (maksymalizacja zysku przy ograniczeniu)

$$f(ch_i) = \begin{cases} \sum_{j=1}^{n} value(ch_{ij}) & \text{if } f(ch_i) < Capasity; \\ f(ch_i) = 0 & \text{otherwise.} \end{cases}$$

Funkcja zadana wzorem

$f(ch_i) = funkcja$ np.

$f(x_1, x_2, x_3) = (2x_1 + 1)^2 + (3x_2 + 4)^2 + (x_3 - 2)^2$

Metody selekcji

Selekcja

Jest to metoda, która wybiera osobniki do reprodukcji. Powinna zapewnić zbieżność do optimum. Wybieranie zawsze i tylko najlepszych nazywa się dużą presją selekcyjną i takie metody nie zawsze dają dobre rozwiązania (utykanie w lokalnych optimach).

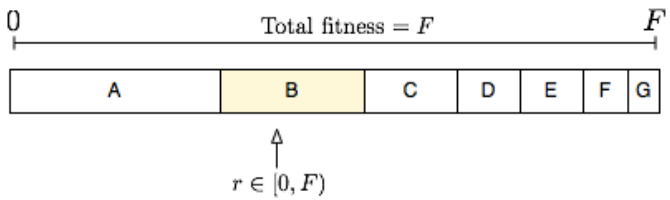
- Selekcja metodą ruletki
- Metoda rankingowa
- Selekcja turniejowa
- Elitaryzm: stosuje się jako dodatkowy element do metod selekcji. Chromosom(y) o największej wartości funkcji przystosowania jest zawsze kopiowany do następnej generacji, przez co nie można go utracić.

Selekcja metodą ruletki

Algorytm

Powtarzaj wielkość populacji razy:

1. Oblicz sumę ($S = \sum_{i=1}^{N} f(x_i)$) przystosowania wszystkich osobników w populacji, gdzie N - liczność populacji.
2. Wygeneruj losowa liczbę (R) z przedziału 0 do S
3. Sumuj kolejno po $S_c = \sum_j f(x_j)$ kolejnych chromosomów. Gdy $S_c > R$, to chromosom j przepisz do nowej populacji.



Cechy/wady

Może się zdarzyć, że jeden chromosom zajmie dużą przestrzeń z koła, zdominuje następną generację. Nadaje się dla FF maksymalizowanej.

Selekcja rankingowa

Algorytm

1. Nadaj rangi chromosomom w populacji na podstawie ich przystosowania. Najgorszy dostaje rangę 1, przedostatni 2 a najlepszy N, gdzie N jest licznością populacji.
2. Traktuj ranking jako wartości oszacowania użyteczności chromosomów i urucham metodę ruletki na tych wartościach.

Cechy/wady

Rozwiązuje problem ruletki, czyli daje szansę chromosomom o małej FF na wylosowanie. Ma jednak słabą zbieżność, bo najlepszy chromosom różni się od innych chromosomów nieznacznie.

Selekcja turniejowa

Algorytm

Powtarzaj wielkość populacji razy:

1. wybierz losowo k (rozmiar turnieju) osobników z populacji
2. wybierz najlepszego osobnika z grupy turniejowej z prawdopodobieństwem p
3. wybierz drugiego osobnika z grupy turniejowej z prawdopodobieństwem $p(1-p)$,trzeciego osobnika z grupy turniejowej z prawdopodobieństwem $p(1-p)^2$ itd.

Turniej deterministyczny, gdy wybiera się najlepszego osobnika z $p = 1$. Jeżeli $k = 1$, to wybór nowej populacji jest losowy.

Cechy/wady

Sprawdza się w implementacjach równoległych, łatwo może manipulować presją selekcyjną. FF może być min/maksymalizowana.

Metody krzyżowania dla chromosomów heterogenicznych

W wyniku krzyżowania powstają dwa osobniki, które zazwyczaj podmieniają swoich rodziców w populacji.

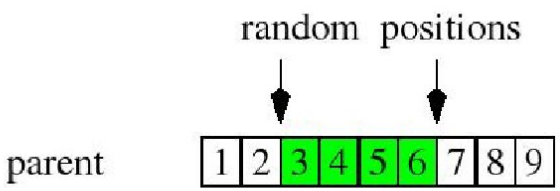
- Jednopunktowe: Losowo wybierz z populacji 2 rodziców. Wylosuj punkt krzyżowania. Rozmnóż osobniki poprzez zamianę ciągów od punktu krzyżowania.
- Dwupunktowe: Wylosuj dwóch rodziców z populacji. Wylosuj dwa punkty krzyżowania. Rozmnóż osobniki poprzez zamianę ciągów pomiędzy punktami krzyżowania.
- Jednorodne: Wylosuj dwóch rodziców do krzyżowania. Dla każdego genu losowo określ czy nastąpi zamiana genów w chromosomach czy też nie.
- Krzyżowanie uśredniające: (dla chromosomów o wartościach rzeczywistych). Dla rodziców x_1 oraz x_2, z rozkładem jednostajnym losujemy liczbę α. Wartości potomków: $y_1 = x_1 + \alpha(x_2 - x_1)$ i $y_2 = x_2 + (x_1 - y_1)$.

Metody mutacji dla chromosomów heterogenicznych

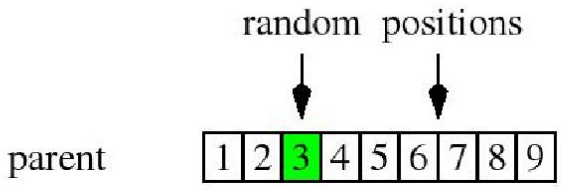
W wyniku mutacji powstaje nowy chromosom, który najczęściej podmienia w populacji chromosom z którego powstaje.

- Jednopunktowa: Wylosuj osobnika do mutacji. Wylosuj punkt mutacji. Zmień wartość genu na dozwoloną/ wylosowaną wartość z alfabetu.
- Na określonej długości: Wylosuj osobnika do mutacji. Wylosuj punkt mutacji i długość zmutowanego fragmentu. Zmień wartości genów na dozwolone/losowe wartości z alfabetu.
- Dla chromosomów o wartościach rzeczywistych. Losujemy liczbę α. Wartość po mutacji: $y = x_1 + \alpha x_1$.

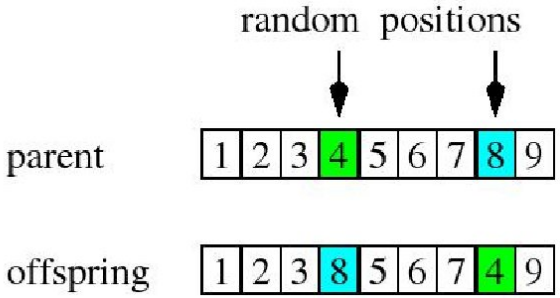
Metody mutacji dla chromosomów homogenicznych



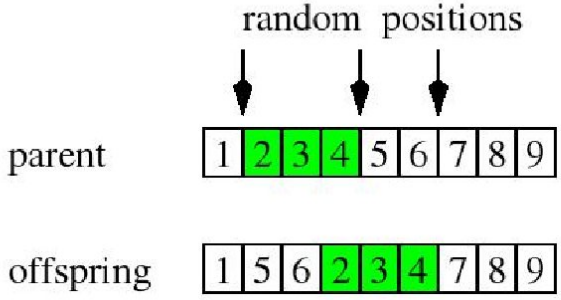
by inversion
(Lin-2-Opt)



by insertion
(Or-Opt)



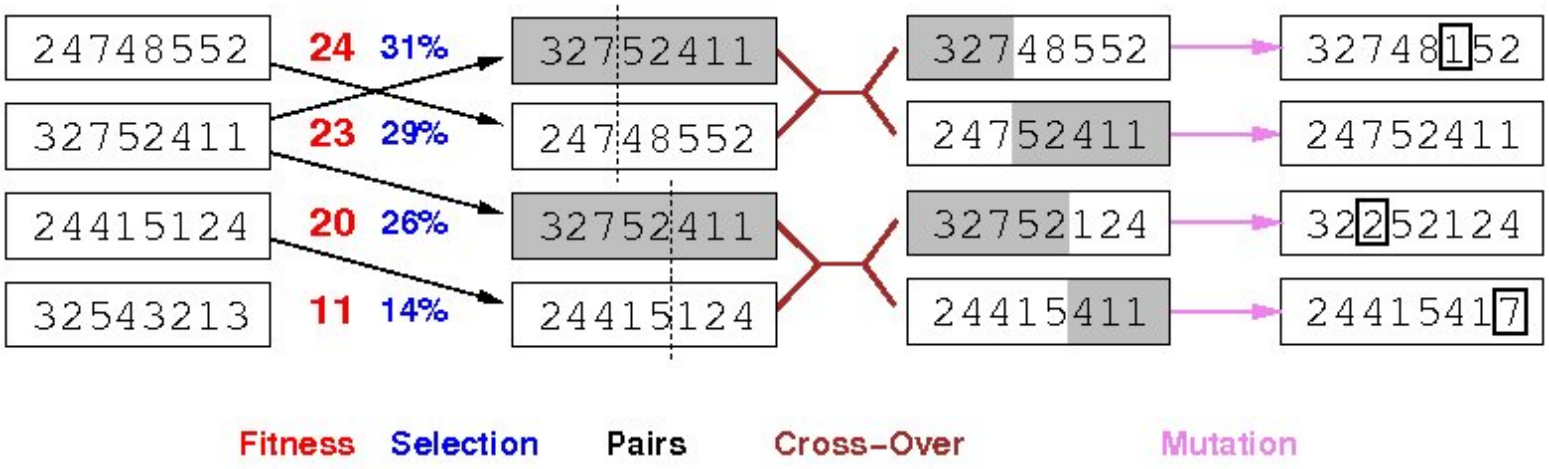
by 2-exchange

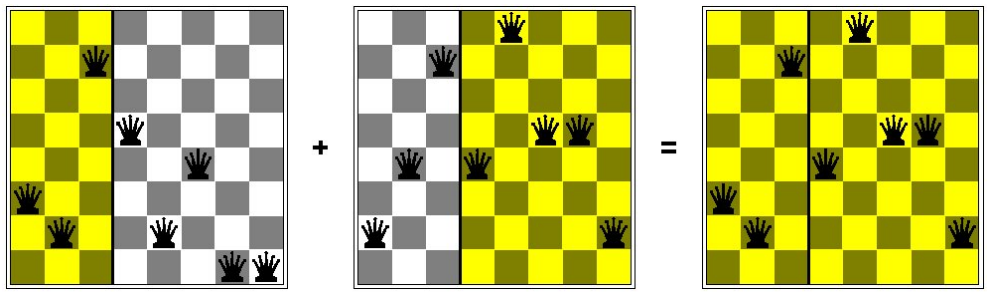


by shifting

Przykład działania algorytmu ewolucyjnego dla n-hetmanów

FF to liczba nie atakujących się par. Najlepsze rozwiązanie FF=28.
Selekcja metodą ruletki. Krzyżowanie, mutacja jednopunktowe.





Warunki zatrzymania AE

Generowanie nowych populacji odbywa się w cyklu, który może się zakończyć, gdy spełniony zostanie warunek zakończenia. Warunek ten może być pojedynczy lub złożony. Do najczęściej stosowanych należą:

- Znaleziono rozwiązanie optymalne lub suboptymalne.
- Osiągnięto założoną liczbę generacji.
- Przekroczono założony budżet (czas obliczeniowy/zysk).
- W kolejnych iteracjach nie uzyskuje się polepszenia najlepszego osobnika lub średniego przystosowania populacji.
- Kontrola manualna.

Parametry, od których zależy efektywność AE

Pewne wartości sterujące zachowaniem metod selekcji, krzyżowania i mutacji są określane przed uruchomieniem AE i zazwyczaj pozostają niezmienne. Do nich zaliczyć można np.:

- liczność populacji
- prawdopodobieństwo krzyżowania
- prawdopodobieństwo mutacji
- liczność grupy turniejowej.

Wielkości wymienionych parametrów są najczęściej dobierane empirycznie. Ponadto rezultaty działania AE zależą od stosowanych selekcji i operatorów reprodukcji.

Ocena algorytmów ewolucyjnych

Zalety

- Szerokie zastosowanie w rozwiązywaniu problemów, dla których nie zawsze znane są dobre inne techniki.
- Może być uruchamiany interaktywnie (zmiana parametrów).
- Niewielki (pozornie) koszt adaptacji algorytmu dla nowego problemu czy nowej przestrzeni stanów.
- Dobre wyniki w poszukiwaniu optimów dzięki operowaniu na genotypie, a nie fenotypie.

Trudności

- Znalezienie odpowiedniej reprezentacji rozwiązania.
- Odpowiedni dobór parametrów i operatorów (liczność populacji, prawdopodobieństwa krzyżowania i mutacji).
- Czas uzyskania satysfakcjonującego rozwiązania.