Politechnika Wrocławska

Programowanie efektownych algorytmów

Problem komiwojażera (TSP)

Data oddania: 22.11.2024

Autor: Krzysztof Zalewa 273032

Spis treści

1. Problem komiwojażera (TSP)

Problem komiwojażera (ang Travelling salesman problem) jest problemem obliczeniowym o złożoności NP. W problemie komiwojażera dany jest zbiór miast (Wierzchołków w grafie) i

kosztów podróży między każdym z nich. Celem jest znalezienie najtańszej drogi zaczynającej i kończącej się w tym samym mieście (wierzchołku). Obecnie nie istnieje rozwiązanie problemu komiwojażera które wykonało by się w czasie wielomianowym. Mimo tego że optymalne rozwiązanie nie istnieje jest wiele innych dobrych rozwiązań.

2. Specyfikacja sprzętu użytego do badań

Badania zostały wykonanie na komputerze stacjonarnym o specyfikacji:

Procesor: Intel(R) Core(TM) i7-9700K CPU

Zegar: 3.60GHz

Wielkość pamięci RAM: 32,0 GB (dostępne: 31,8 GB)

3.Procedura badawcza

4. Badania

4.1. Algorytm losowy (ang. Random)

4.1.1. Opis algorytmu

Algorytm losowy to najprostsze możliwe rozwiązanie problemu komiwojażera. Polega on na wylosowaniu ścieżki i porównaniu jej kosztu z najlepszym znalezionym do tej pory. Jeżeli nowy koszt jest mniejszy to należy ustawić obecny koszt jako najlepszy i kontynuować.

4.1.2. Założenia

4.1.3. Wyniki

	Symetryczne			Asymetryczne		
Rozmiar Instancji /Gęstość	30%	60%	100%	30%	60%	100%
8	0,000102	0,000119	0,00003131	0,00009777	0,00007901	0,0000317
9	0,000114	0,00017	0,00004568	0,00017193	0,00014056	0,00004516
10	0,000339	0,000239	0,00006631	0,00034104	0,0003382	0,00006679
11	0,00039	0,000386	0,00009162	0,00053767	0,00032088	0,00011535
12	0,000822	0,000707	0,00012198	0,0008222	0,00070706	0,00012296

4.2. Algorytm najbliższego sąsiada (ang. Nearest neighbour)

4.2.1 Opis algorytmu

Algorytm najbliższego sąsiada jest algorytmem zachłannym. Zaczynając z wierzchołka wybiera się jeden wierzchołek do którego droga jest najkrótsza. Ten proces jest powtarzany tak długo aż

przejdziemy przez wszystkie miasta. Początkowy wierzchołek wybierany jest w pętli, więc NN wykona się dla każdego z wierzchołków. Jako algorytm zachłanny prowadzi on do „jakiegoś" rozwiązania które nie koniecznie będzie najlepsze ale na pewno będzie poprawne.

4.2.2. Założenia

4.2.3. Wyniki

4.3. Algorytm siłowy (ang. Brute-force)

4.3.1 Opis algorytmu

Algorytm siłowy polega na wygenerowaniu każdej możliwej iteracji rozwiązania. Koszt każdej iteracji jest zliczany i porównywany z najlepszym do tej pory otrzymanym. Wygenerowanie każdej iteracji w mojej implementacji jest osiągane przy użyciu algorytmu heapa. Algorytm heapa generuje iteracje poprzez zmianę i tego elementu z n-1. Następnie jest on wywoływany rekurencyjnie dla zmienionej ścieżki.

4.3.2. Założenia

4.3.3. Wyniki

4.4. Metoda podziału i ograniczeń (ang. Branch and bound)

Metoda branch and bound polega na systematycznym przeszukiwaniu przestrzeni rozwiązań (Punkty 4.5 – 4.7). Najważniejszym elementem tej metody jest ograniczanie liczby przypadków do sprawdzenia poprzez zastosowanie podziału (branch) i ograniczeń (bound) .

4.5. Przeszukiwanie w głąb (ang. Depth first search)

4.5.1 Opis algorytmu

Przeszukiwanie w głąb polega na przejściu przez wszystkie wierzchołki w odpowiedniej kolejności. Zaczynając od wierzchołka 0 wybieramy jedno z jego dzieci. Następnie dla wybranego dziecka wybieramy jego dziecko (wnuk wierzchołka 0). Proces ten powtarzamy tak długo aż nie trafimy na liścia. Jeżeli wybrany wierzchołek jest liściem należy zliczyć koszt wytworzonej trasy i porównać z najlepszym otrzymanym do tej pory. Jeżeli obecny koszt jest lepszy należy go zamienić. Następnie cofamy się do poprzednio odwiedzonego wierzchołka i wybieramy inne dziecko niż to z którego przszliśmy. DFS zwykle wykonywany jest przy pomocy stosu.

4.5.2. Założenia

4.5.3. Wyniki

4.6. Przeszukiwanie w szerz (ang. Breadth first search)

4.6.1 Opis algorytmu

Przeszukiwanie w szerz jest podobne do przeszukiwania w głąb. Zaczynamy w wierzchołku 0 i po kolei wybieramy wszystkie jego dzieci. Następnie dla tych dzieci wybieramy wszystkie ich dzieci. Tą procedurę powtarzamy aż do momentu gdy dotrzemy do liści. Następnie zliczamy koszt wytworzonej trasy i porównać z najlepszym otrzymanym do tej pory. Jeżeli obecny koszt jest lepszy należy go zamienić. BFS wykonywany jest przy pomocy kolejki.

4.6.2. Założenia

4.6.3. Wyniki

4.7 Przeszukiwanie przy minimum kosztów (ang. Least cost)

4.7.1 Opis algorytmu

Przeszukiwanie przy minimum kosztów zaczyna się podobnie do DFS i BFS. Zaczynając w wierzchołku 0 dla każdego dziecka wyliczamy granicę według pewnej funkcji. Wybieramy dziecko o najniższym koszcie. Proces ten powtarzany jest aż do momentu gdy dotrzemy do liści. LC wykonywany jest przy pomocy kolejki priorytetowej.

4.7.2. Założenia

4.7.3. Wyniki

4. Plik konfiguracyjny

5. Źródła

[1]. https://www.ceas3.uc.edu/ret/archive/2016/ret/docs/abstract/reading33.pdf
[2]. http://seor.vse.gmu.edu/~khoffman/TSP_Hoffman_Padberg_Rinaldi.pdf
[3]. https://eduinf.waw.pl/inf/alg/001_search/0109.php
[4]. https://eduinf.waw.pl/inf/alg/001_search/0110.php
[5].