Obraz zawierający tekst, Czcionka, logo, Grafika

Opis wygenerowany automatycznie

Programowanie efektownych algorytmów

Problem komiwojażera (TSP)

Data oddania: 22.11.2024

Autor: Krzysztof Zalewa 273032

Spis treści

[1. Problem komiwojażera (TSP) 3](#_Toc183123079)

[2. Specyfikacja sprzętu użytego do badań 3](#_Toc183123080)

[3.Procedura badawcza 3](#_Toc183123081)

[4. Badania 3](#_Toc183123082)

[4.1. Algorytm losowy (ang. Random ) 3](#_Toc183123083)

[4.1.1. Opis algorytmu 3](#_Toc183123084)

[4.1.2. Lista kroków 3](#_Toc183123085)

[4.1.3. Założenia badawcze 4](#_Toc183123086)

[4.1.4. Wyniki 4](#_Toc183123087)

[4.1.4 Wnioski 4](#_Toc183123088)

[4.2. Algorytm najbliższego sąsiada (ang. Nearest neighbour ) 4](#_Toc183123089)

[4.2.1 Opis algorytmu 4](#_Toc183123090)

[4.2.2. Lista kroków 4](#_Toc183123091)

[4.2.3. Założenia badawcze 4](#_Toc183123092)

[4.2.4. Wyniki 4](#_Toc183123093)

[4.3. Algorytm siłowy (ang. Brute-force ) 6](#_Toc183123094)

[4.3.1 Opis algorytmu 6](#_Toc183123095)

[4.3.2. Lista kroków 6](#_Toc183123096)

[4.3.3. Założenia badawcze 6](#_Toc183123097)

[4.3.4. Wyniki 6](#_Toc183123098)

[4.4. Metoda podziału i ograniczeń (ang. Branch and bound ) 8](#_Toc183123099)

[4.5. Przeszukiwanie w głąb (ang. Depth first search ) 8](#_Toc183123100)

[4.5.1 Opis algorytmu 8](#_Toc183123101)

[4.5.2. Lista kroków 8](#_Toc183123102)

[4.5.3. Założenia badawcze 9](#_Toc183123103)

[4.5.4. Wyniki 9](#_Toc183123104)

[4.6. Przeszukiwanie w szerz (ang. Breadth first search ) 10](#_Toc183123105)

[4.6.1 Opis algorytmu 10](#_Toc183123106)

[4.6.2. Lista kroków 10](#_Toc183123107)

[4.6.3. Założenia badawcze 11](#_Toc183123108)

[4.6.4. Wyniki 11](#_Toc183123109)

[4.7 Przeszukiwanie przy minimum kosztów (ang. Least cost ) 12](#_Toc183123110)

[4.7.1 Opis algorytmu 12](#_Toc183123111)

[4.7.2. Lista kroków 12](#_Toc183123112)

[4.7.3. Założenia badawcze 13](#_Toc183123113)

[4.7.4. Wyniki 13](#_Toc183123114)

[4. Źródła 14](#_Toc183123115)

# 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

Dla każdego z poniżej zaprezentowanych algorytmów wykonano badania dla:

* Macierzy symetrycznych i niesymetrycznych
* Trzech gęstości macierzy 30%, 60% i 100%
* Macierzy o ilości wierzchołków od 8 do 12 włącznie
* W algorytmie losowym ograniczeń czasowych 10 000 000, 100 000 000, 1 000 000 000, 10 000 000 000 i 100 000 000 000 ns (0.01 , 0.1 , 1 , 10 i 100 sekund)

Dla każde z 30 konfiguracji (2\*3\*5) wykonano 10 powtórzeń by uzyskać wiarygodne wyniki.

# 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. Lista kroków

1. Tworzona jest pierwsza ścieżka 0,1,2,…n-1
2. Ścieżka jest mieszana
3. Zliczana jest wartość ścieżki
4. Jeżeli obliczona wartość jest mniejsza od najlepszej do tej pory otrzymanej należy je zamienić
5. Algorytm powtarza się tak długo aż nie przekroczy wyznaczonego czasu

### 4.1.3. Założenia badawcze

### 4.1.4. Wyniki

### 4.1.4 Wnioski

## 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.

### 4.2.2. Lista kroków

1. Pierwszy zostanie wybrany wierzchołek 0
2. Dla wybranego wierzchołka sprawdzany jest koszt przejścia do nie odwiedzonych wierzchołków
3. Wybierany jest wierzchołek o najniższym koszcie przejścia
4. Kroki 2 -3 powtarzają się tak długo aż nie zostaną odwiedzone wszystkie wierzchołki grafu
5. Otrzymany wynik porównywany jest z najlepszym do tej pory otrzymanym
6. Jako wierzchołek startowy wybierany jest kolejny wierzchołek 1,2….n-1 jeszcze raz wywoływane są kroki 2 – 6

### 4.2.3. Założenia badawcze

Jako algorytm zachłanny prowadzi on do „jakiegoś” rozwiązania które nie koniecznie będzie optymalne. Dla grafów nie pełnych rozwiązanie może nie być poprawne.

### 4.2.4. 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 |

Tabela . Czasy wykonania funkcji nearest-neighbour [s] (Na czerwono zaznaczono instancje w których nie udało się znaleźć żadnego wyniku)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Symetryczne** | | | **Asymetryczne** | | |
| **Rozmiar Instancji /Gęstość** | **30%** | **60%** | **100%** | **30%** | **60%** | **100%** |
| **8** | 0,00% | N/A | 18,93% | 33,25% | N/A | 15,69% |
| **9** | N/A | 19,81% | 32,09% | N/A | 17,25% | 0,00% |
| **10** | N/A | 1,26% | 12,98% | N/A | N/A | 29,74% |
| **11** | N/A | 5,16% | 25,32% | N/A | 1,93% | 26,28% |
| **12** | N/A | N/A | 4,08% | N/A | 0,01% | 31,38% |

Tabela . Błąd względny funkcji nearest-neighbour (N/A oznacza że nie znaleziono ścieżki)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Symetryczne** | | | **Asymetryczne** | | |
| **Rozmiar Instancji /Gęstość** | **30%** | **60%** | **100%** | **30%** | **60%** | **100%** |
| **8** | 30867 | N/A | 15230 | 38267 | N/A | 24012 |
| **9** | N/A | 34161 | 9417 | N/A | 17918 | 17589 |
| **10** | N/A | 39705 | 18569 | N/A |  | 19255 |
| **11** | N/A | 33758 | 19573 | N/A | 19994 | 17246 |
| **12** | N/A | N/A | 17710 | N/A | 15781 | 16216 |

Tabela . Wyniki funkcji nearest-neighbour (N/A oznacza że nie znaleziono ścieżki)

Rysunek . Czasy wykonania algorytmu dla macierzy symetrycznych [s]

Rysunek . Czasy wykonania algorytmu dla macierzy asymetrycznych [s]

## 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. Lista kroków

1. Tworzona jest pierwsza iteracja 0,1,2,…n-1
2. Rekurencyjnie wywoływany jest algorytm heapa dla n-1
3. Algorytm heapa zatrzymuje się gdy n=1
4. Należy zliczyć wynik otrzymanej iteracji i porównać z najlepszym do tej pory otrzymanym wynikiem. I zamienić jeżeli jest mniejszy.
5. Algorytm cofa się do poprzedniej iteracji algorytmu heapa i zamienia i-ty element z elementem n-1 (gdzie i 0,1,2,…n-1)
6. Jeszcze raz rekurencyjnie wywoływany jest algorytm heapa dla n-1
7. Algorytm heapa zakończy się gdy wytworzy wszystkie możliwe iteracje

### 4.3.3. Założenia badawcze

Brute force na pewno da optymalny wynik. Ale będzie się wykonywał dłużej niż pozostałe algorytmy.

### 4.3.4. Wyniki

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Symetryczne** | | | **Asymetryczne** | | |
| **Rozmiar Instancji /Gęstość** | **30%** | **60%** | **100%** | **30%** | **60%** | **100%** |
| **8** | 0,000584 | 0,001455 | 0,002964 | 0,000594 | 0,00593 | 0,006764 |
| **9** | 0,022613 | 0,012452 | 0,000671 | 0,007114 | 0,047743 | 0,002322 |
| **10** | 0,125065 | 0,106813 | 0,049782 | 0,38329 | 0,47834 | 0,565129 |
| **11** | 0,658866 | 1,088163 | 2,050599 | 4,200745 | 5,931383 | 0,191078 |
| **12** | 2,315414 | 5,455512 | 4,550668 | 50,45979 | 63,29362 | 25,10583 |

Tabela . Czasy wykonania funkcji brute-force [s]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Symetryczne** | | | **Asymetryczne** | | |
| **Rozmiar Instancji /Gęstość** | **30%** | **60%** | **100%** | **30%** | **60%** | **100%** |
| **8** | 30867 | 22264 | 12806 | 28719 | 16035 | 18843 |
| **9** | 23923 | 28513 | 7129 | 26942 | 13728 | 17589 |
| **10** | 24562 | 39212 | 16436 | 26726 | 14207 | 14841 |
| **11** | 40077 | 32101 | 15619 | 35236 | 19117 | 13657 |
| **12** | 39712 | 26906 | 17016 | 31919 | 13727 | 12343 |

Tabela . Wyniki funkcji brute-force

Rysunek . Czasy wykonania algorytmu dla macierzy symetrycznych [s]

Rysunek . Czasy wykonania algorytmu dla macierzy asymetrycznych [s]

## 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. Lista kroków

1. Na stos wrzucany jest wierzchołek 0
2. Wyliczamy ograniczenie dolne oraz ograniczenie górne ( za pomocą nearest neighbour)
3. Tak długo jak stos nie jest pusty
4. Zdejmowany jest wierzchołek na szczycie
5. Jeżeli jego granica dolna jest większa lub równa ograniczeniu górnemu pomijamy ten wierzchołek
6. Jeżeli ścieżka przypisana do wierzchołka jest równa n to zliczana jest jego wartość.
7. Wartość porównywana jest z najlepszym znalezionym rozwiązaniem. I zamieniana jeżeli jest mniejsza.
8. Jeżeli ścieżka jest krótsza od n należy znaleźć nowy wierzchołek i który nie był odwiedzony (i = 0,1,2…n-1)
9. Dla tego wierzchołka należy wyliczyć ścieżka która do niego prowadzi i jego granicę
10. Jeżeli granica jest mniejsza od najlepszego znalezionego rozwiązania wierzchołek dodawany jest na stos.

### 4.5.3. Założenia badawcze

Depth first search będzie szybszy od brute forca da dobry wynik który na pewno będzie poprawny ale może nie być optymalny.

### 4.5.4. Wyniki

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Symetryczne** | | | **Asymetryczne** | | |
| **Rozmiar Instancji /Gęstość** | **30%** | **60%** | **100%** | **30%** | **60%** | **100%** |
| **8** | 0,000186 | 0,000265 | 0,000635 | 0,000166 | 0,000288 | 0,000634 |
| **9** | 0,000363 | 0,000598 | 0,00083 | 0,000751 | 0,00088 | 0,000655 |
| **10** | 0,000106 | 0,002801 | 0,004232 | 0,000523 | 0,000546 | 0,00486 |
| **11** | 0,000469 | 0,007211 | 0,020066 | 0,012145 | 0,003791 | 0,024452 |
| **12** | 0,000246 | 0,007022 | 0,02514 | 0,034566 | 0,015293 | 0,020256 |

Tabela . Czasy wykonania funkcji depth first search [s]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Symetryczne** | | | **Asymetryczne** | | |
| **Rozmiar Instancji /Gęstość** | **30%** | **60%** | **100%** | **30%** | **60%** | **100%** |
| **8** | 3,71% | 0,00% | 0,00% | 0,00% | 11,47% | 0,00% |
| **9** | 0,00% | 0,00% | 0,00% | 0,00% | 0,00% | 0,00% |
| **10** | 0,00% | 0,00% | 0,00% | 8,46% | 0,00% | 0,00% |
| **11** | 0,00% | 0,00% | 0,00% | 10,61% | 0,00% | 0,00% |
| **12** | 0,00% | 0,00% | 0,00% | 0,00% | 0,00% | 4,57% |

Tabela . Błąd względny funkcji depth first search

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Symetryczne** | | | **Asymetryczne** | | |
| **Rozmiar Instancji /Gęstość** | **30%** | **60%** | **100%** | **30%** | **60%** | **100%** |
| **8** | 29723 | 22264 | 12806 | 28719 | 14466 | 20755 |
| **9** | 23923 | 28513 | 7129 | 26942 | 15282 | 17589 |
| **10** | 24562 | 39212 | 16436 | 26830 | 16789 | 14841 |
| **11** | 40077 | 32101 | 15619 | 35395 | 20388 | 13657 |
| **12** | 39712 | 26906 | 17016 | 32837 | 15782 | 11779 |

Tabela . Wyniki funkcji depth first search

Rysunek . Czasy wykonania algorytmu dla macierzy symetrycznych [s]

Rysunek . Czasy wykonania algorytmu dla macierzy asymetrycznych [s]

## 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. Lista kroków

1. Na kolejkę wrzucany jest wierzchołek 0
2. Wyliczamy ograniczenie dolne oraz ograniczenie górne ( za pomocą nearest neighbour)
3. Tak długo jak kolejka nie jest pusta
4. Zdejmowany jest wierzchołek na szczycie
5. Jeżeli jego granica dolna jest większa lub równa ograniczeniu górnemu pomijamy ten wierzchołek
6. Jeżeli ścieżka przypisana do wierzchołka jest równa n to zliczana jest jego wartość.
7. Wartość porównywana jest z najlepszym znalezionym rozwiązaniem. I zamieniana jeżeli jest mniejsza.
8. Jeżeli ścieżka jest krótsza od n należy znaleźć nowy wierzchołek i który nie był odwiedzony (i = 0,1,2…n-1)
9. Dla tego wierzchołka należy wyliczyć ścieżka która do niego prowadzi i jego granicę
10. Jeżeli granica jest mniejsza od najlepszego znalezionego rozwiązania wierzchołek dodawany jest na kolejka.

### 4.6.3. Założenia badawcze

Breadth first search będzie wykonywał się dłużej od brute forca. Wynik będzie poprawny i na pewno optymalny.

### 4.6.4. Wyniki

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Symetryczne** | | | **Asymetryczne** | | |
| **Rozmiar Instancji /Gęstość** | **30%** | **60%** | **100%** | **30%** | **60%** | **100%** |
| **8** | 0,000309 | 0,000893 | 0,021742 | 0,000316 | 0,011754 | 0,027044 |
| **9** | 0,001921 | 0,00582 | 0,183279 | 0,002084 | 0,065804 | 0,216008 |
| **10** | 0,006027 | 0,03631 | 1,813179 | 0,006508 | 0,469833 | 1,932505 |
| **11** | 0,000742 | 0,2272 | 19,78348 | 0,042772 | 7,037906 | 20,38759 |
| **12** | 0,00375 | 1,682285 | 239,4432 | 0,295299 | 25,02463 | 239,5641 |

Tabela . Czasy wykonania funkcji breadth first search [s]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Symetryczne** | | | **Asymetryczne** | | |
| **Rozmiar Instancji /Gęstość** | **30%** | **60%** | **100%** | **30%** | **60%** | **100%** |
| **8** | 30867 | 22264 | 12806 | 28719 | 16340 | 18843 |
| **9** | 23923 | 28513 | 7129 | 26942 | 13728 | 17589 |
| **10** | 24562 | 39212 | 16436 | 26726 | 14207 | 14841 |
| **11** | 40077 | 32101 | 15619 | 39594 | 19117 | 13061 |
| **12** | 39712 | 26906 | 17016 | 32837 | 13727 | 12343 |

Tabela . Wyniki funkcji breadth first search

Rysunek . Czasy wykonania algorytmu dla macierzy symetrycznych [s]

Rysunek . Czasy wykonania algorytmu dla macierzy asymetrycznych [s]

## 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. Lista kroków

1. Na kolejkę priorytetową wrzucany jest wierzchołek 0
2. Wyliczamy ograniczenie dolne oraz ograniczenie górne ( za pomocą nearest neighbour)
3. Tak długo jak kolejka priorytetowa nie jest pusta
4. Zdejmowany jest wierzchołek na szczycie
5. Jeżeli jego granica dolna jest większa lub równa ograniczeniu górnemu pomijamy ten wierzchołek
6. Jeżeli ścieżka przypisana do wierzchołka jest równa n to zliczana jest jego wartość.
7. Wartość porównywana jest z najlepszym znalezionym rozwiązaniem. I zamieniana jeżeli jest mniejsza.
8. Jeżeli ścieżka jest krótsza od n należy znaleźć nowy wierzchołek i który nie był odwiedzony (i = 0,1,2…n-1)
9. Dla tego wierzchołka należy wyliczyć ścieżka która do niego prowadzi i jego granicę
10. Jeżeli granica jest mniejsza od najlepszego znalezionego rozwiązania wierzchołek dodawany jest na kolejkę priorytetową.

### 4.7.3. Założenia badawcze

Least cost powinien być najszybszym zaimplementowany algorytmem i powinien zwracać poprawne i optymalne wyniki.

### 4.7.4. Wyniki

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Symetryczne** | | | **Asymetryczne** | | |
| **Rozmiar Instancji /Gęstość** | **30%** | **60%** | **100%** | **30%** | **60%** | **100%** |
| **8** | 0,000182 | 0,00013 | 0,000262 | 0,000268 | 0,00008125 | 0,000387 |
| **9** | 0,000557 | 0,000336 | 0,000447 | 0,000148 | 0,0008032 | 0,001481 |
| **10** | 0,000469 | 0,001513 | 0,003302 | 0,001321 | 0,00155468 | 0,002506 |
| **11** | 0,000641 | 0,009767 | 0,011304 | 0,009657 | 0,00279568 | 0,008128 |
| **12** | 0,003151 | 0,010295 | 0,016628 | 0,009071 | 0,00313619 | 0,002091 |

Tabela . Czasy wykonania funkcji least cost [s]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Symetryczne** | | | **Asymetryczne** | | |
| **Rozmiar Instancji /Gęstość** | **30%** | **60%** | **100%** | **30%** | **60%** | **100%** |
| **8** | 30867 | 22264 | 12806 | 28719 | 16340 | 20755 |
| **9** | 23923 | 28513 | 7129 | 26942 | 15282 | 17589 |
| **10** | 24562 | 39212 | 16436 | 29310 | 16789 | 14841 |
| **11** | 40077 | 32101 | 15619 | 39594 | 20388 | 13657 |
| **12** | 39712 | 26906 | 17016 | 32837 | 15782 | 12343 |

Tabela . Wyniki funkcji least cost

Rysunek . Czasy wykonania algorytmu dla macierzy symetrycznych [s]

Rysunek . Czasy wykonania algorytmu dla macierzy asymetrycznych [s]

# 4. 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>