Projektowanie Efektywnych Algorytmow
Projekt
20/10/2020

248820 Przemysław Rychter

(1) Brute force

spis treści	*strona*

1. Sformułowanie zadania

Zadanie polega na opracowaniu, implementacji i zbadaniu efektywności algorytmu przeglądu zupełnego rozwiązującego problem komiwojażera w wersji optymalizacyjnej. Problem komiwojażera (e*ng. Travelling salesman problem, TSP)* to zagadnienie polegające (w w. optymalizacyjnej) na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym.

- **Graf pełny** to zbiór wierzchołków, przy czym między każdymi dwoma wierchołkami istnieje krawędź je łącząca [1]
- **Cykl Hamiltona** to droga wiodąca przez wszystkie wierzchołki dokładnie raz, z wyjątkiem jednego wybranego, w którym cykl Hamiltona zaczyna się oraz kończy [2]

Problem komiwojażera możemy rozumieć jako zadanie polegające na znalezienu najlepszej drogi dla podróżującego, który chce odwiedzić n miast, i skończyć podróż w miejscu jej początku. Połączenie między każdym miastem ma swój „koszt” określający efektywność jej przebywania. Najlepsza droga to taka, której całkowity koszt (suma kosztów przebycia wszystkich połączeń między miastami w drodze) jest najmiejszy.

Problem dzieli się na symetryczny i asymetryczny. Pierwszy polega na tym, że dla dowolnych miast A i B z danej instancji, koszt połączenia jest taki sam w przypadku przebywania połączenia z A do B jak z B do A, czyli dane połączenie ma po prostu jeden koszt niezależnie od kierunku ruchu. W asymetrycznym problemie komiwojażera koszty te mogą być różne.

2. Metoda

Metoda przeglądu zupełnego, tzw. przeszukiwanie wyczerpujące (*eng.exhaustive search*) bądź metoda siłowa (*eng. brute force*), polega na znalezieniu i sprawdzeniu wszystki rozwiązań dopuszczalnych problemu, wyliczeniu dla nich wartości funkcji celu i wyborze rozwiązania o ekstremalnej wartości funkcji celu – najniższej (problem minimalizacyjny) bądź najwyższej (problem maksymalizacyjny).

Wszystkie możliwe rozwiązania problemu komiwojażera, to wszystkie możliwe cykle Hamiltona dla danej instancji problemu. Algorytm oparty na metodzie przeglądu zupełnego powinien wszystkie takie cykle znaleźć i wybrać jako optymalny ten o najmiejszym koszcie. Ilość różnych cykli pełnym grafie nieskierowanym wynosi $\frac{(n-1)!}{2}$ [3] - tyle różnych cykli o możliwych różnych kosztach istnieje dla instancji problemu symetrycznego. W pełnym grafie skierowanym ilość różnych cykli wynosi $(n-1)!$ [3] - tyle różnych cykli o możliwych różnych kosztach istnieje dla instancji problemu asymetrycznego.

Początkowo można pomyśleć że ilość cykli to ilość permutacji (czyli ilość możliwości ustawienia wszystkich miast w kolejności odwiedzin) czyli $n!$.

Oznaczmy kolejne miasto jako 0,1,2, … ,n-1. Wypiszmy wszystkie permutacje dla $n=4$ których jest $n!=24$

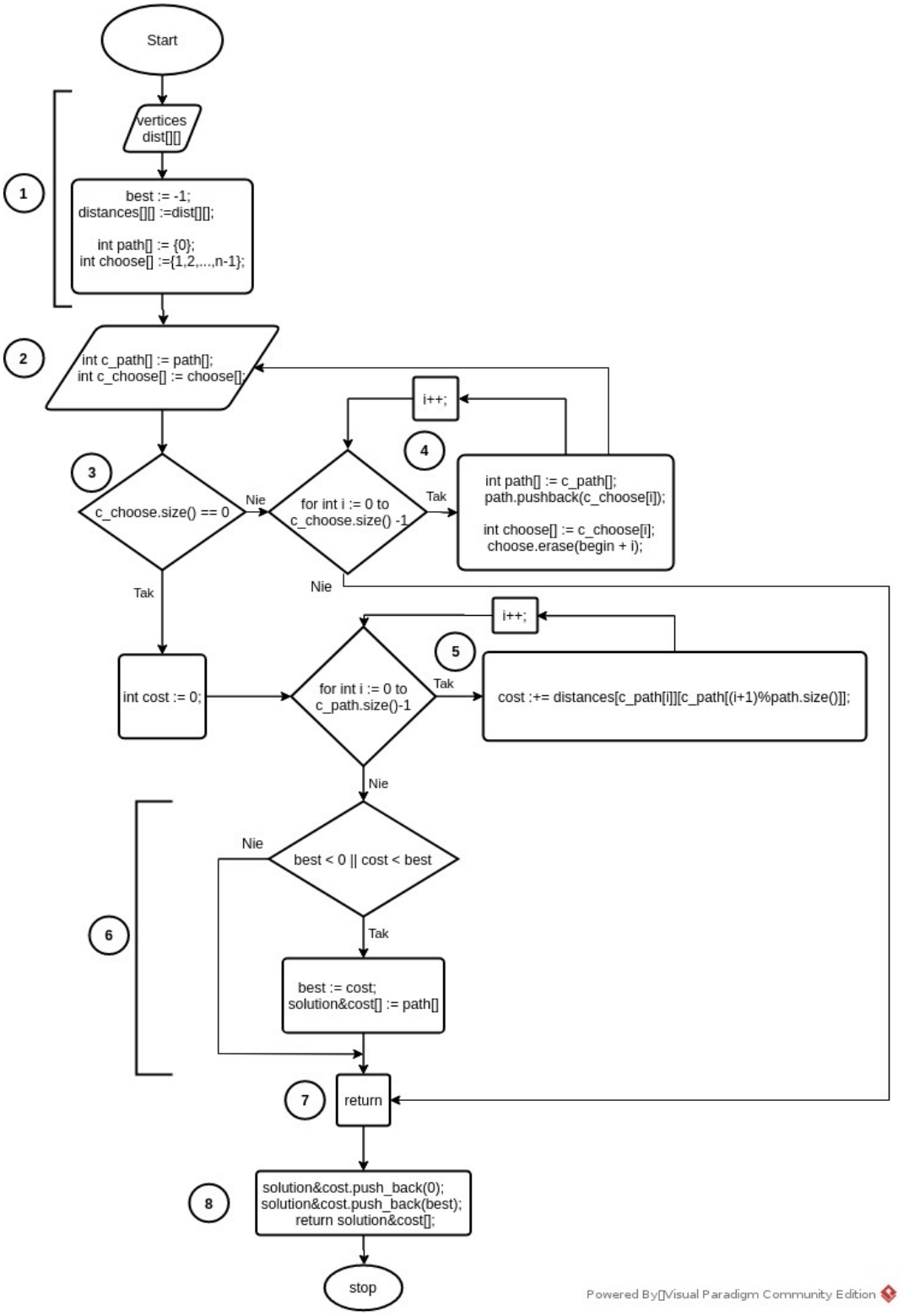
Tabela 1: Permutacje zbioru: {0,1,2,3,4}

0 1 2 3	3 0 1 2	2 3 0 1	1 2 3 0
0 1 3 2	2 0 1 3	3 2 0 1	1 3 2 0
0 2 1 3	3 0 2 1	1 3 0 2	2 1 3 0
0 2 3 1	1 0 2 3	3 1 0 2	2 3 1 0
0 3 1 2	2 0 3 1	1 2 0 3	3 1 2 0
0 3 2 1	1 0 3 2	2 1 0 3	3 2 1 0

W pierwszej kolumnie zostały wypisane wszystkie permutacje zaczynajace się od 0. W następnych kolumnach permutacje zostały utworzone przesuwając miasta w prawo. Przesunięcie **nie zmienia cyklu,** czyli wszystkie permutacje w danym wierszu reprezentują jeden, ten sam cykl. Widzimy, że rzeczywiście ilość cykli dla asymetrycznego problemu komiwojażera wynosi $(n-1)!$ i tworzy się je wypisując wszystkie permutacje $(n-1)$ wierzchołków. Wniosek jest taki, że w algorytmie możemy wybrać dowolny wierchołek zawsze jako początek drogi, znaleźć wszystkie permutację pozostałych wierzchołków - $(n-1)!$ obliczyć koszt każdego cyklu i wybrać ten o najmiejszym koszcie.

3. Algorytm

Algorytm jest skonstruowany z dwóch funkcji pierwsza pobiera jako argumenty ilość miast oraz macierz odległości między każdymi dwoma miastami jej celem jest inicjalizacja wspólnych danych dla wszystkich wywołań drugiej funkcji rekurencyjnej – kosztu najlepszej ścieżki oraz macierzy odległości. Pierwsza funkcja inicjalizuje też dane dla pierwszego wywołania drugiej funkcji(rekurencyjnej) czyli tablice aktualnie tworzonej scieżki – {0} oraza tablice wierchołków możliwych do wybrania podczas tworzenia ścieżki.



Rysunek 1: Schemat blokowy algorytmu opartego na metodzie brute force

1. Wywołanie pierwszej funkcji z argumentami: vertices – ilość wierchołków, dist[][] macierz odległości między miastami. Zmienna best, wspólna dla wszystki wywołań rekurencyjnych jest inicjalizowana wartością 1, będzie ona przechowywać koszt aktualnie najlepszej drogi. Inicjalizacja zmiennej path[] czyli aktualnie tworzona droga, wybieramy wierzchołek 0-owy jako pierwszy. Inicjalizacja zmiennej choose[] jest to tablica z której możemy wybrać następny wierchołek w celu stworzenia drogi.

2. Kopie zmiennych path[] oraz choose[] czyli c_path[] oraz c_choose[] są przekazywane do drugiej funkcji – rekurencyjnej. c_path – current_path, c_choose – current_choose

3. Jesli tablica c_choose[] jest pusta oznacza to że cykl jest kompetny i można przejść do obliczenia kosztu i sprawdzenia czy jest optymalny.

4. Jeśli tablica c_choose[] zawiera wierchołki do stworzenia cyklu, to w pętli dla każdego wierchołka z c_choose jest tworzona kopia aktualnej trasy (path), dodawany jest do niej koleny wierzchołek z c_choose, a następnie funkcja wywołuje samą siebie z nową trasą (path) oraz nową tablicą c_choose (choose) bez wierchołka który został dodany (skoro został dodany już do trasy to nie chcemy żeby się powtarzał)

5. Cykl jest kompletny zostały wykorzystane wszystkie wierzchołki, więc liczony jest koszt cyklu, czyli w pętli są sumowane koszty połączeń między kolejnymi wierchołkami w cyklu. Operator % został zastosowany w celu obliczenia kosztu z ostatniego węzła do 0-owego.

6. Jeśli aktualnie najlepszy koszt jest < 0 (czyli jest to pierwszy znaleziony cykl) lub wyliczony koszt jest mniejszy od aktualnie najlepszego, to obliczony koszt zapisujemy/nadpisujemy jako najlepszy we wspólnej dla wszystkich wywołań rekurencyjnych zmiennej best, a aktualną scieżkę zapisujemy/nadpisujemy (również we wspólnej dla wszystkich wywołań rekurencyjnych) zmiennej solution&cost.

7. Jest to moment w którym druga funkcja rekurencyjna(i wszystkie jej podwywołania) została/zostały zakończone i dalej będzie wykonywana funkcja pierwsza

8. Funkcja pierwsza zwraca znalezioną optymalną ścieżkę w postaci 0, a, b, … , 0 oraz zwraca koszt optymalnej ścieżki na końcu tablicy ze ścieżką.

4. Dane testowe

Do sprawdzenia poprawności działania algorytmu i wykonania badań wybrano następujący zestaw instancji:

tsp_6_1.txt

tsp_6_2.txt

tsp_10.txt

tsp_12.txt

tsp_13.txt

dostępnych na stronie: http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/

5. Procedura badawcza

Należało zbadać zależność czasu rozwiązania problemu od wielkości instancji. W przypadku algorytmu realizującego przegląd zupełny przestrzeni rozwiązań dopuszczalnych nie występowały parametry programu, które mogły mieć wpływ na czas i jakość uzyskanego wyniku. W związku z tym procedura badawcza polegała na uruchomieniu programu sterowanego plikiem inicjującym .ini (format pliku: nazwa_instancji liczba_wykonań rozwiązanie_optymalne [ścieżka optymalna];nazwa_pliku_wyjściowego).

tsp_6_1.txt 100 132 0 1 2 3 4 5 0

tsp_6_2.txt 100 80 0 5 1 2 3 4 0

tsp_10.txt 10 212 0 3 4 2 8 7 6 9 1 5 0

tsp_12.txt 10 264 0 1 8 4 6 2 11 9 7 5 3 10 0

tsp_13.txt 2 269 0 10 3 5 7 9 11 2 6 4 8 1 12 0 ,

tsp_6_13bf.csv

Każda instancji rozwiązywana była zgodnie z liczbą jej wykonań, np. tsp_12.txt wykonana została 10 razy. Do pliku wyjściowego tsp_6_13bf.csv zapisywane były informacje o instancji: jej nazwa, liczba wykonań algorytmu, koszt ścieżki oraz scieżka optymalna z pliku „conf.ini”. Nastepnie zapisywane były czasy wykonań algorytmu dla tej instancji. Plik wyjściowy zapisywany był w formacie csv. Poniżej przedstawiono fragment zawartości pliku wyjściowego.

tsp_10.txt 10 212 0 3 4 2 8 7 6 9 1 5 0
962577
957641
957269
964468
989286
980189
963179
961114
958137
958644
tsp_12.txt 10 264 0 1 8 4 6 2 11 9 7 5 3 10 0
142703330
144366287
150783115
131272567
129479924
118588543
128278460
129604332
131362550
124704060

Pomiary zostały wykonane na platofrmie sprzętowej:
procesor: Intel® Core™ i5-8250U CPU @ 1.60GHz × 8
pamięć operacyjna: 31,2 GiB
system operacyjny: z rodziny Linux - Ubuntu 20.04.1 LTS 64-bit

Pomiary czasu zostały wykonane za pomocą biblioteki std::chrono [4].

Po każdym powtórzeniu wykoniania algorytmu dla danej instancji, w programie głównym „main.cpp” sprawdzana była zgodność znalezionej ścieżki oraz jej kosztu, ze ścieżką oraz kosztem podanym w pliku konfiguracyjnym „conf.ini” W przypadku znalzienia innej scieżki program informuje o znalezieniu innej ścieżki i/lub kosztu. Sytuacja w której opracowany algorytm znalazłby inna ścieżke i/lub koszt niż w pliku konfiguracyjnym nie miała miejsca.

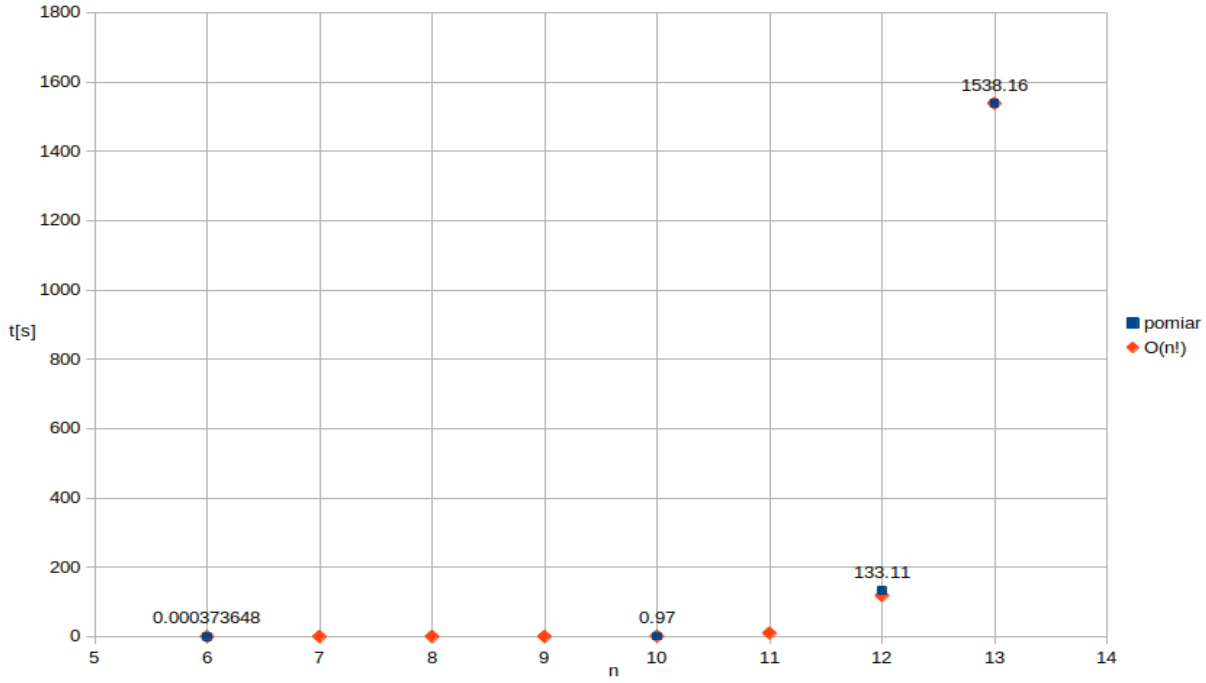
Wyniki zostały opracowane w programie LibreOffice Calc.

6. Wyniki

Wyniki zgromadzone zostały w pliku:

tsp_6_13bf.csv

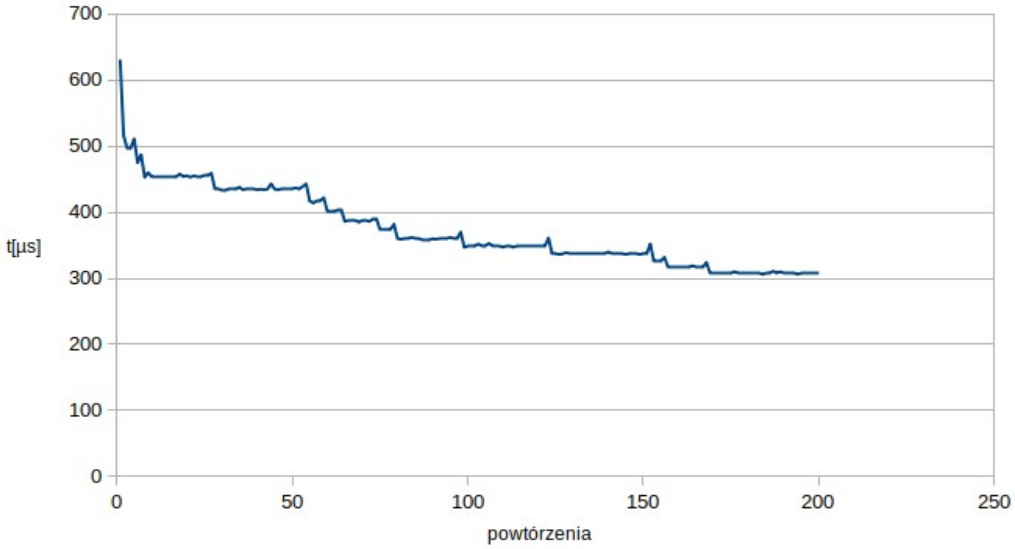
Plik został dołączony do raportu i znajdują się na dysku Google pod adresem https://drive.google.com/drive/folders/1yHS-PS9DVc7rIv4o933_UdkAuBjO8zVU. Na dysku zostały także umieszone: folder z programem w którym znajduja się pliki źródłowe, folder z instancjami, plik konfiguracyjny „conf.ini” oraz instrukcja kompilacji w systemie Linux.



Rysunek 2: Wpływ wielkości instancji n na czas uzyskania rozwiazania problemu komiwojażera metodą brute force

Wyniki przedstawione zostały w postaci wykresu zależności czasu uzyskania rozwiązania problemu od wielkości instancji (rysunek 2).Na wykresie zostały przedstawione uśrednione pomiary czasu dla instancji o danym rozmiarze oraz punkty wykresu O(n!) - został on określony funkcją f(x) = x!*c, gdzie c oznacza stałą, która została obliczona w celu dopasowania punktów wykresu O(n!) do rzeczywistych pomiarów czasu wykonania algorytmu.

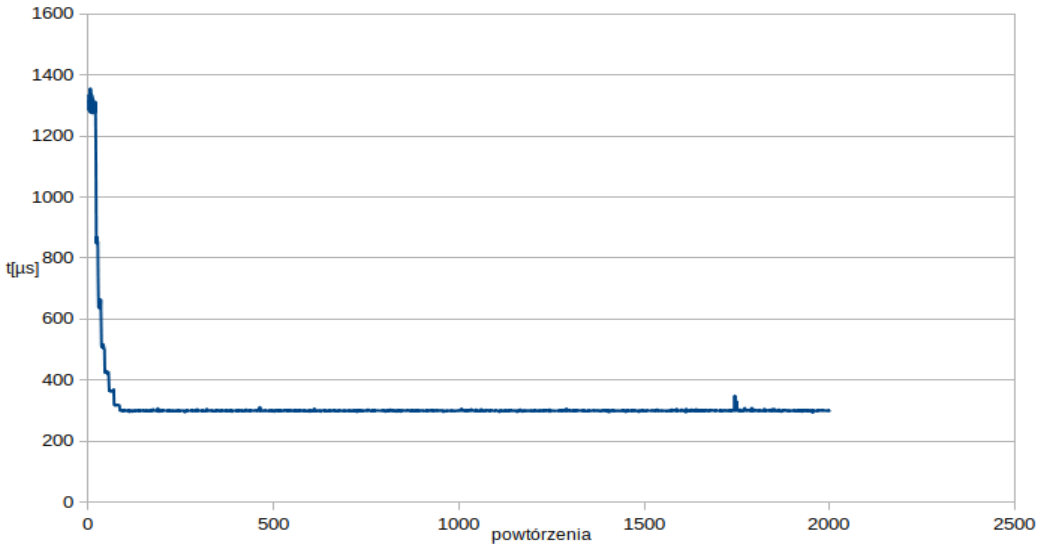
Dla instancji 6-elementowych dla wszystkich 200 powtórzeń wykonania algorytmu zauważono spadek czasu wykonywania algorytm, który się ustabilizował na poziomie 300 mikrosekund (rysunek 3).



Rysunek 3: Czas wykonania algorytmu dla instancji 6-elem. w zależności od numeru powtórzenia

Podejrzewa się, że jest to związane z cechą procesora na którym zostały wykonane badania. Procesor ma zmienną częstotliwością taktowania.

W celu potwierdzenia powyższego podejrzenia pomiary dla dwóch 6-elementowej instancji zostały przeprowadzone jeszcze raz ale z ilością powtórzeń = 1000.



Rysunek 4: Czas wykonania algorytmu dla instancji 6-elem. w zależności od numeru powtórzenia

Na wykresie (rysunek 4) widać, że czas ustablizował się. Kod programu został sprawdzony i po każdym wykonaniu algorytmu instancja testowa dla algorytmu jest niszczona.

7. Analiza wyników i wnioski

Wzrostu czasu względem wielkości instancji ma charakter wykładniczy (rysunek 2). Nałożenie punktów wykresu O(n!) potwierdza, że badany algorytm wyznacza rozwiązania problemu komiwojażera dla badanych instancji w czasie n! zależnym względem wielkości instancji (oba wykresy są zgodne co do kształtu). Złożoność czasowa opracowanego algorytmu wynosi O(n!).

Źródła

[1] https://pl.wikipedia.org/wiki/Graf_pe%C5%82ny

[2] https://pl.wikipedia.org/wiki/Cykl_Hamiltona

[3] https://en.wikipedia.org/wiki/Hamiltonian_path#:~:text=The%20number%20of%20different%20Hamiltonian,point%20are%20not%20counted%20separately.

[4] https://en.cppreference.com/w/cpp/chrono

spis rysunków

Spis tabel