Projektowanie Efektywnych Algorytmow
Projekt
20/10/2020

248820 Przemysław Rychter

(2) Algorytm Helda-Karpa

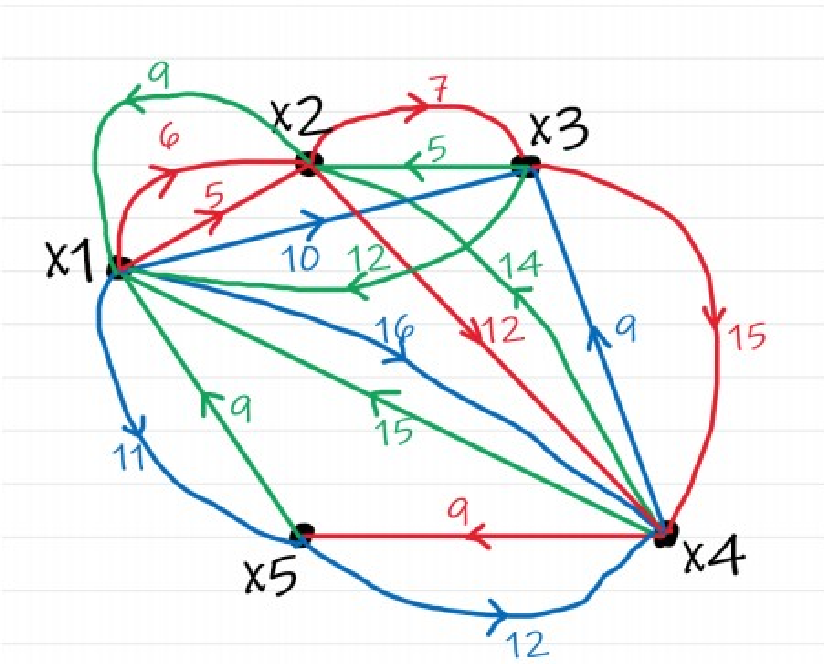
1. Sformułowanie zadania

Zadanie polega na opracowaniu, implementacji i zbadaniu efektywności algorytmu Helda-Harpa opartego na programowaniu dynamicznym, 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 efektywnosć 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.



Rysunek 1: Przykładowy graf reprezentujący asymetryczny problem komiwojażera

Macierz reprezentująca koszty przebycia dróg między sąsiednimi węzłami może wyglądać następująco: (symbol nieskonczoności oznacza brak bezpośredniego połaczenia między wierzchołkami)

$$\begin{bmatrix} 0 & 5 & 10 & 16 & 11 \\ 9 & 0 & 7 & 12 & \infty \\ 12 & 5 & 0 & 15 & \infty \\ 15 & 14 & 9 & 0 & 9 \\ 9 & \infty & \infty & 12 & 0 \end{bmatrix}$$

Rysunek 2: Przykładowa macierz odległości

2. Metoda

Algorytm Helda-Karpa jest oparty na metodzie znanej pod nazwą „programowanie dynamiczne” jest to technika projektowania algorytmów polegająca na rozwiązywaniu podproblemów i zapamiętywaniu ich wyników. Problem jest dzielony na mniejsze podproblemy. Wyniki rozwiązania podproblemów są zapisywane, dzięki czamu w przypadku natrafienia na ten sam podproblem nie trzeba go ponownie rozwiązywać.

Algorytm działa na podstawie równiania rekurencyjnego opisującej minimalny koszt przejścia z wierzchołka x_0 poprzez dany podzbiór $S \subset V \setminus \{x_0\} \equiv \{x_1, x_2, \ldots, x_{(n-1)}\}$ koncząc w wybranym wierzchołku x_i należącym do S, $x_i \in S$,V to zbiór wszystkich wierzchołków instancji, $V = \{x_0, x_1, \ldots, x_{(n-1)}\}$

$$cost(x_i, S) = min_{(x_j)}\{cost(x_j, S \setminus \{xi\}) + d_{(ji)}\} \quad ,$$

$d_{(ji)}$ to odlległość krawędzi łączącej bezpośrednio dwa wierzchołki, $cost(x_i, S)$ to koszt przejścia od wierzchołka x_0 poprzez wszystkie wierzchołki z S koncząc w x_i . Kiedy S ma tylko 1 wierzchołek to $cost(x_i, S) = d_{(0i)}$

Używając powyższego równiania możemy wyliczyć koszty dla podzbiorów S o rozmiarach od 1 do n – 2. Kiedy dojdziemy do S o rozmiarze n-1 czyli $S = V \setminus \{x_0\}$ Koszt optymalnej ścieżki zostaje obliczony wzorem:

$$najkrótszy\ cykl\ Hamiltona = min_{(x_i)}\{\ cost(x_j, V \setminus \{x_0\}) + d_{(j0)}\}$$

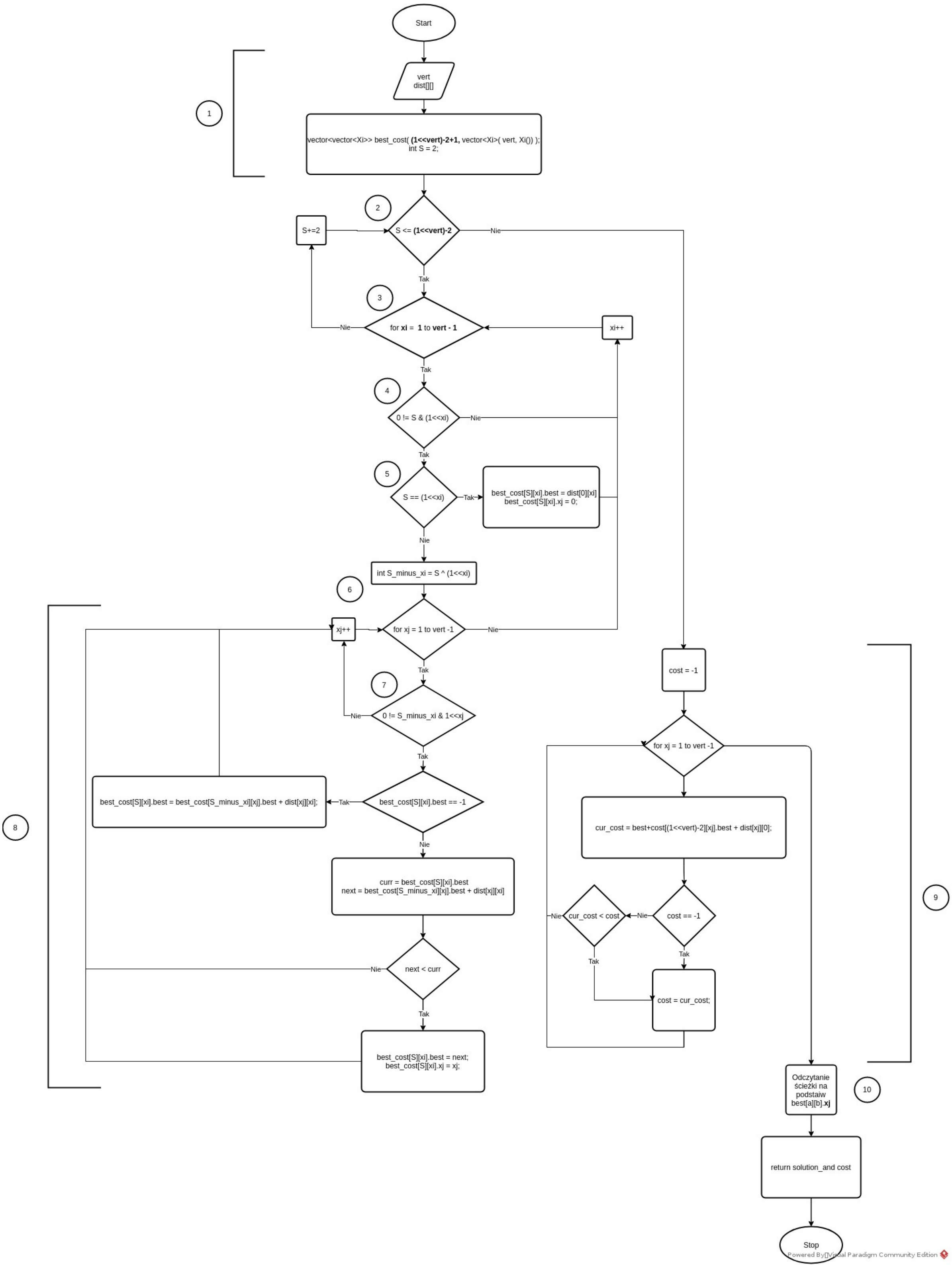
Algorytm Helda-Karpa posiada złożoność czasową $O(n^2 2^n)$ oraz pamięciową $O(n 2^n)$.

Powyższe dane pochodzą z [3][4][5]

Wiele implementacji korzysta z reprezentacji zbioru jako liczba binarna, jest to bardzo pomocne, zaimplementowany algorytm używa takiego kodowania podzbiorów V. Tworząc algorytm oparty na programowaniu dynamicznym można wykorzystać metodę „zstępującą z zapamiętywaniem” lub metodę wstępującą.

3. Algorytm

Podczas implementacji zostałą wykorzystana metoda wstępująca, polega ona na rozwiązaniu wszystkich możliwych podproblemów zaczynając od najmiejszych. Rozwiązując iteracyjnie kolejne podproblemy mamy pewność, że mniejsze podproblemy potrzebne do rozwiązania danego podproblemu zostały rozwiązane.



Rysunek 3: Schemat blokowy algorytmu opartego na programowaniu dynamicznym

1. Tablica best_cost[S][xi] reprezentuje najlepszy koszt przejścia od wierzchołka. x0 przez wszystkie wierzchołki w S, kończąc na xi, xi zawiera się w S. S to konkretny podzbiór wszystkich wierzchołków oprócz x0, podzbiory są kodowane jako liczba binarna w której dany bit reprezentuje obecność danego wierzchołka. S = 2, ponieważ ustalamy sztywno pierwszy wierzchołek zerowy więc podzbiory nie zawierają x0 - pierwszego bitu w danym podzbiorze S. Tablica zawiera Xi (składające się z 2 liczb jedna reprezentuje najlepszy koszt druga, poprzedni wezeł) zamiast int w celu zapamietania poprzedniego wezła. Liczba (1<<vert)-2 reprezentuje zbiór wszystkich wierzchołków oprócz 0 ,+ 1 ponieważ indeksowanie od 0

2. Iteracja po podzbiorach zbioru wszystkich wierzchołków bez x0, w każdej iteracji dodajemy 2 do S '10' bo pomijamy wierzchołek x0. Konczymy na liczbie opisacej podzbior 2^vert-2 czyli to są wszystkie wierzchołki oprócz x0.

3. Dla każdego wierzchołka znajdującego się w S znajdź podścieżkę o najlepszym koszcie

4. Czy xi zawiera się w S ? Jeśli nie, weź następne xi. Jeśli tak kontynuuj iteracje.

5. Kiedy S zawiera tylko xi nie ma możliwości wybrania podścieżki, jest bezpośrednie połączenie.

6. Znajdz wszystkie xj - węzeł bezpośrednio przed xi, x0 -> S\{xi,xj} -> xj -> xi ,zbiór S_minus_xi to zbór S bez wierzchołka xi – S\{xi}, dla takiego zbioru szukamy minimum kosztu w przejściu x0 -> S\{xi} -> xi w zależności od xj.

7. xj musi się oczywiście zawierać w S_minus_xi, jeśli nie weź następne xj.

8. Jeśli znaleziono lepszy wynik dla kolejnego xj, należy go zapisać.

9. Obliczenie kosztu najkrótszej ścieżki, przyjeliśmy że zaczynamy w 0 , obliczono najlepszy koszt dla podzbioru n-1 elementowego. Teraz obliczany jest koszt najlepszego cyklu hamiltona.

10. Odczytanie ścieżki

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
tsp_14.txt
tsp_15.txt
tsp_17.txt

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

Algorytm dla wszystkich instancji zwrócił koszty zgodne z podanymi kosztami optymalnymi, dla wszystkich instancji zwrócił ścieżki zgodne z podanymi ścieżkami optymalnymi, lub ścieżki symetryczne, dla instancji symetrycznych. Dla instancji tsp_17.txt algorytm znalazł inną ścieżkę o koszcie optymalnym zgodnym z podanym, ścieżka ta była poprawna (została sprawdzona ręcznie).

Zostały stworzone 3 wersje głownego programu:

- program po każdym wykonaniu algorytmu sprawdza znalezioną ścieżkę oraz koszt z podanymi w pliku conf.ini i informuje w przypadku braku zgodności kosztu lub/i ścieżki, dodatkowo, w przypadku braku zgodności ścieżki program sprawdza czy nie jest ona symetryczna i zwraca o tym informację(ta wersja została wykorzystana do sprawdzenia poprawności algorytmu)
- program sprawdza po każdym wykonaniu algorytmu tylko **koszt.** Dla **ATSP,** czyli instancji ze strony: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/atsp/index.html zostały podane tylko koszty optymalne, dlatego program dla każdego powtórzenia sprawdza czy zwrócony koszt zgadza się z optymalnym, w przypadku niezgodności zwraca informacje. Dane do pliku .csv są przekazywane standardowo.
- Program sprawdza po każym wykonaniu tylko **ścieżkę**. Dla **STSP**, czyli instancji ze strony: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html zostały podane tylko ścieżki optymalne, dlatego program dla każdego powtórzenia sprawdza czy zwrócona ścieżka zgadza się z podaną optymalną, w przypadku niezgodności zwaraca informację. Dane do pliku .csv są przekazywane standardowo.

5. Procedura badawcza

Należało zbadać zależność czasu rozwiązania problemu od wielkości instancji oraz ilość używanej pamięci operacyjnej. W przypadku algorytmu Helda-Karpa 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).

Instancje testowe pochodziły ze stron:

- http://elib.zib.de/pub/mp-testdata/tsp/tsplib/atsp/index.html (**ATSP**)
- http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html (**TSP)**
- http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/

Instancje z dwóch pierwszych powyższych adresów zostały pobrane ze strony:

- http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/pea.php

Na powyższej stronie można było znaleźć instancje w standardowym formacie oraz koszty optymalne które nie były pierwotnie określone dla **TSP**, dlatego ostatecznie koszty optymalne przed rozpoczęciem badania były określone dla każdej instancji, natomiast nie dla każdej zostały podane ścieżki, więc do wykonania badań została użyta wersja programu sprawdzająca tylko poprawność kosztu po każdym wykonianu algorytmu. Poniżej treść pliku „conf.ini”.

tsp_**6**_2.txt 100 80 0 5 1 2 3 4 0
tsp_**10**.txt 100 212 0 3 4 2 8 7 6 9 1 5 0
tsp_**12**.txt 100 264 0 1 8 4 6 2 11 9 7 5 3 10 0
tsp_**13**.txt 100 269 0 10 3 5 7 9 11 2 6 4 8 1 12 0
burma**14**.tsp 100 3323
tsp_**15**.txt 100 291 0 12 1 14 8 4 6 2 11 13 9 7 5 3 10 0
br**17**.atsp 100 39
gr**21**.tsp 100 2707
gr**24**.tsp 20 1272 0 15 10 2 6 5 23 7 20 4 9 16 21 17 18 14 1 19 13 12 8 22 3 11 0
fri**26**.tsp 10 937 0 24 23 22 25 21 20 16 17 19 18 15 10 12 11 14 13 9 8 7 6 4 5 3 2 1 0
Held_Karp_6_26.csv

Każda instancji rozwiązywana była zgodnie z liczbą jej wykonań, np. burma14.tsp wykonana została 100 razy. Do pliku wyjściowego Held_Karp_6_26.csv zapisywane były informacje o instancji: jej nazwa, liczba wykonań algorytmu, koszt znalezionej ścieżki optymalnej oraz znaleziona scieżka optymalna. Następnie zapisywane były czasy wykonań algorytmu dla tej instancji. Plik wyjściowy zapisywany był w formacie csv. Poniżej przedstawiono fragment zawartości pliku wyjściowego.

fri26.tsp 10 937 0 24 23 22 25 21 20 16 17 19 18 15 10 12 11 14 13 9 8 7 6 4 5 3 2 1 0
228576692
229799772
249397499
261096273
228204259

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 [6].

Po każdym powtórzeniu wykoniania algorytmu dla danej instancji, w programie głównym „main.cpp” sprawdzana była zgodność znalezionego kosztu z kosztem podanym w pliku konfiguracyjnym „conf.ini” W przypadku znalezienia innego kosztu program informuje o znalezieniu innego kosztu. Sytuacja w której opracowany algorytm znalazłby inny koszt miała miejce dla instancji „burma14.tsp”

Wyniki zostały opracowane w programie LibreOffice Calc.

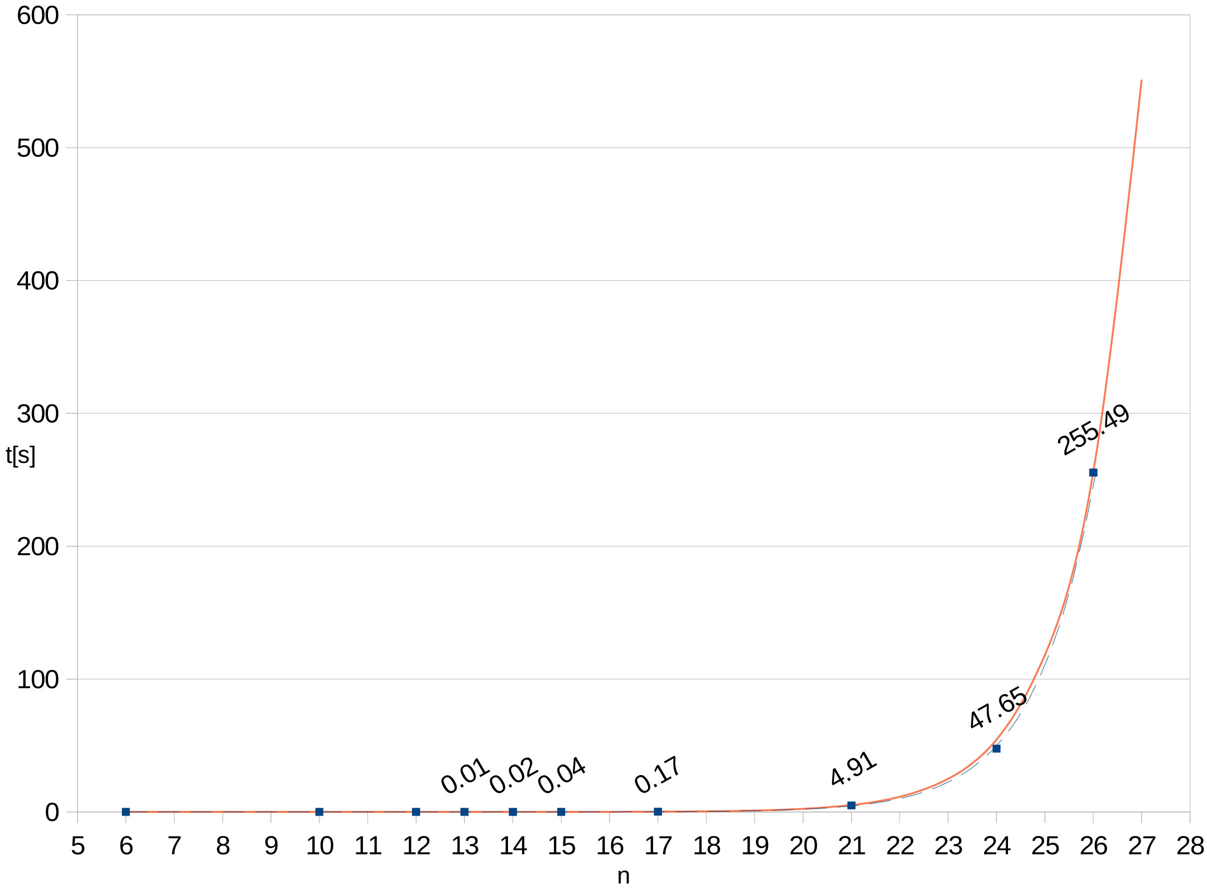
6. Wyniki

Wyniki zgromadzone zostały w plikach:

Held_Karp_6_26.csv

Held_Karp_6_26.ods

Pliki został dołączone 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 żródłowe, folder z instancjami, plik konfiguracyjny „conf.ini” oraz instrukcja kompilacji w systemie Linux.



Rysunek 4: Wpływ wielkości instancji n na czas rozwiązania problemu komiwojażera metodą programowania dynamicznego

Wyniki przedstawione zostały w postaci wykresu zależności czasu uzyskania rozwiązania problemu od wielkości instancji (rysunek 4).Na wykresie zostały przedstawione uśrednione pomiary czasu (niebieskie punkty) dla instancji o danym rozmiarze oraz krzywa wykresu $O(n^2 2^n)$ (pomarańczowa)- została ona określona funkcją $f(n)=n^2 2^n c$, gdzie c oznacza stałą, która została obliczona w celu dopasowania teoretycznej złożoności algorytmu do pomiarów.

7. Analiza wyników i wnioski

Wzrostu czasu względem wielkości instancji ma charakter wykładniczy (rysunek 4). Nałożenie krzywej $O\left(n^2 2^n\right)$ potwierdza, że badany algorytm wyznacza rozwiązania problemu komiwojażera dla badanych instancji w czasie $n^2 2^n$ zależnym względem wielkości instancji (obie krzywe są zgodne co do kształtu). Złożoność czasowa opracowanego algorytmu wynosi $O\left(n^2 2^n\right)$

Źródła

[1] https://pl.wikipedia.org/wiki/Graf_pe%C5%82ny
[2] https://pl.wikipedia.org/wiki/Cykl_Hamiltona
[3] http://algorytmy.ency.pl/artykul/algorytm_helda_karpa
[4] http://www.math.nagoya-u.ac.jp/~richard/teaching/s2020/Quang1.pdf
[5] https://en.wikipedia.org/wiki/Held%E2%80%93Karp_alxgorithm
[6] https://en.cppreference.com/w/cpp/chrono

spis rysunków