Projektowanie efektywnych algorytmów

Projekt

26.10.2020

248821 Michał Rajkowski

(1) Brute Force

Spis treści

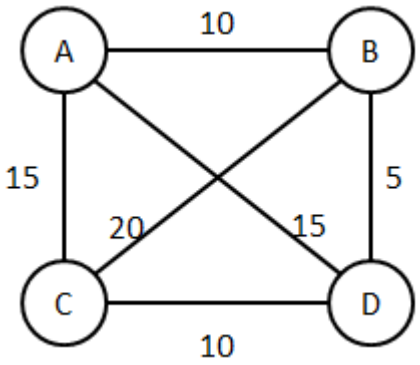
1. Sformułowanie zadania

Zadanie polega na opracowaniu, implementacji i zbadaniu efektywności algorytmu przeglądu zupełnego rozwiązującego problem komiwojażera w wersji optymalizacyjnej.

Problem komiwojażera (ang. Travelling salesman problem) jest to zagadnienie optymalizacyjne polegające na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym.

Zagadnienie to można podzielić na dwa podtypy ze względu na wagi ścieżek łączących wierzchołki grafu. W symetrycznym problemie komiwojażera (STSP) dla dowolnych dwóch wierzchołków A B krawędź łącząca wierzchołek A z B ma taką samą wagę jak ta łącząca wierzchołek B z A. Natomiast w asymetrycznym problemie komiwojażera (ATSP) wagi te mogą się różnić.

Główną trudnością problemu jest duża liczba danych do analizy. W przypadku problemu symetrycznego dla n miast liczba kombinacji wynosi $\frac{(n-1)!}{2}$. Dla problemy asymetrycznego jest to aż n!.



Przykładowy graf ważony pełny nieskierowany (STSP)

2. Metoda

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

W przypadku naszego problemu szukamy wszystkich możliwych cykli Hamiltona i wybieramy z nich ten o najniższej łącznej sumie wag krawędzi. Do znajdowania kolejnych cykli Hamiltona korzystamy z tablicy wierzchołków grafu, a następnie za pomocą odpowiedniej metody przechodzimy kolejno przez wszystkie jej permutacje. Dla każdej permutacji obliczamy sumę dróg między kolejnymi wierzchołkami tablicy. Dodatkowo zakładamy, iż każda droga rozpoczyna się i kończy w tym samym wierzchołku (wierzchołku startowym 0) i doliczamy odległość między wierzchołkiem startowym a pierwszym i ostatnim wierzchołkiem danej permutacji aby otrzymać całkowitą wagę cyklu Hamiltona.

Jako iż korzystamy z metody przeglądu zupełnego złożoność obliczeniowa czasu naszego algorytmu winna przystawać do O(n!).

3. Algorytm

3.1 Opis słowny

Opracowany algorytm TSP bazuje na metodzie bruteforce. Znajduje on wszystkie możliwe cykle Hamiltona rozpoczynające się i kończącę wierzchołkiem 0. Dla każdego takiego cyklu oblicza jego długość i porównuje z aktualnie zapamiętaną długością najkrótszego cyklu. Mechanizm ten bazuje na algorytmie minimum, przez co w wyniku zostaje zwrócony najkrótszy cykl Hamiltona (pod względem wagi dróg).

W pierwszym kroku następuje utworzenie tablicy wierzchołków grafu. Kolejność wierzchołków w tablicy może być dowolna, tak długo jak nie zawiera ona wierzchołka startowego. Zatem dla uproszczenia algorytm rozpoczyna się z tablicą zawierającą kolejne wierzchołki grafu według ich numeracji (to znaczy 1., 2., 3., 4.,).

Następnie zostaje utworzona ścieżka, w postaci tablicy n+1. Będzie ona przechowywać w sobie minimalną ścieżkę znalezioną przez algorytm. W tym samym kroku przypisujemy długość minimalnej ścieżki na nieskończoność (w naszym przypadku max int, gdyż jest to maksymalna możliwa wartość dal naszej zmiennej).

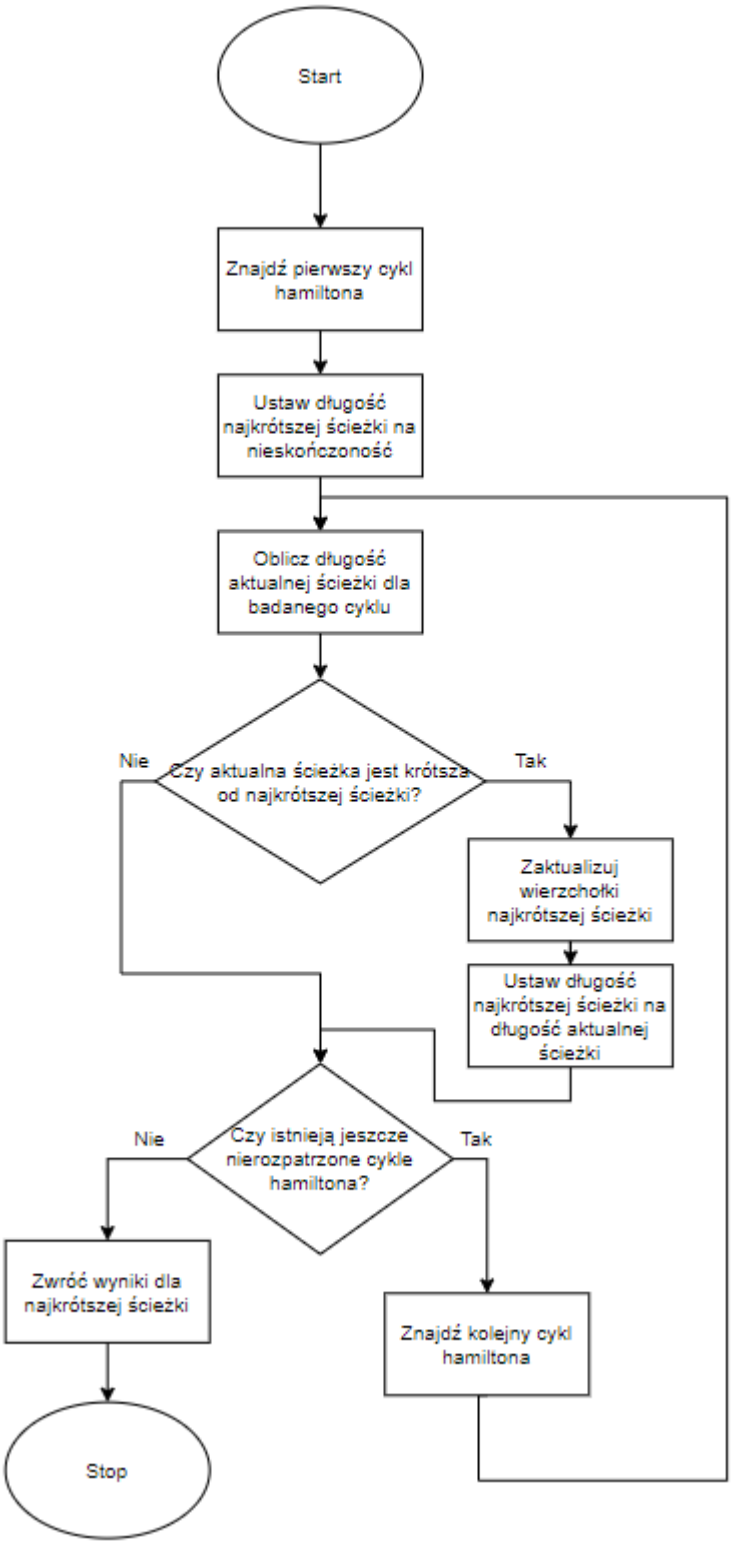
Następnie dla aktualnej permutacji tablicy wierzchołków grafu (aktualny cykl hamiltona) obliczamy długość ścieżki. Jest to suma długości krawędzi między kolejnymi wierzchołkami aktualnego cyklu.

Jeżeli obliczona długość ścieżki jest mniejsza od obliczonej minimalnej ścieżki, aktualną ścieżkę zapamiętujemy jako ścieżkę minimalną oraz aktualizujemy wartość zmiennej przechowującej długość minimalnej ścieżki.

Jeżeli istnieje następna permutacja tablicy wierzchołków (istnieją jeszcze nierozpatrzone cykle hamiltona), powtarzamy procedurę obliczenia długości ścieżki i porównania ją z minimalną dla kolejnego cyklu.

Dodatkowo program korzysta z autorskiej struktury grafu w postaci macierzowej przechowywanej jako obiekt, oraz metod umożliwiających operacje na tej strukturze. Metody te służą do tworzenia budowania/usuwania grafu oraz wczytywania grafu z pliku tekstowego. Nie wpływają one w żaden sposób na czas pracy algorytmu TSP, z tego powodu nie umieszczam w sprawozdaniu objaśnienia co do ich działania.

3.2 Schemat blokowy:



3.3 Algorytm w postaci kodu

```
int TSP(){

    //algorym zaczyna sie zawsze w wierzcholku  0

    int ver[cnt-1];
    for(int i = 0; i < cnt - 1; i++){ //Znajdź pierwszy cykl hamiltona
        ver[i] = i + 1;
    }
    path = new int[cnt + 1];
    path[0] = 0;
    path[cnt] = 0;


    int minimum_path = INT_MAX;   //ustaw długość najkrótszej ścieżki na nieskończoność

    while(true){
        int current_path = 0;
        int current_node = 0;

        for(int i = 0; i < cnt - 1; i++){              //oblicz długość aktualej ścieżki dla badanego cyklu
            current_path += tab[current_node][ver[i]];
            current_node = ver[i];
        }
        current_path += tab[current_node][0];

        if(current_path < minimum_path){   //Czy aktualna ścieżka jest krótsza od najkrótszej ścieżki?
            minimum_path = current_path;   //Ustaw długość najkrótszej ścieżki na długość aktualnej ścieżki

            for(int i = 0; i < cnt-1; i++){ //zaktualizuj wierzchołki najkrótszej ścieżki
                path[i+1] = ver[i];
            }
        }
        if(!next_permutation(ver, ver + cnt-1)){ //czy istnieją jeszcze nierozpatrzone cykle hamiltona? + znajdź kolejny cykl hamiltona
            break;
        }
    }


    return minimum_path; //Zwróć wyniki dla najkrótszej ścieżki

}
```

4. Dane testowe

Do sprawdzenia poprawności działania algorytmu oraz do 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

Źródło: 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 20 132 0 1 2 3 4 5 0

tsp_10.txt 20 212 0 3 4 2 8 7 6 9 1 5 0

tsp_12.txt 20 264 0 1 8 4 6 2 11 9 7 5 3 10 0

tsp_13.txt 20 269 0 10 3 5 7 9 11 2 6 4 8 1 12 0

tsp_14.txt 5 282 0 10 3 5 7 9 13 11 2 6 4 8 1 12 0

final_test.csv

Każda instancji rozwiązywana była zgodnie z liczbą jej wykonań, np. tsp_6_1.txt wykonana została 20 razy. Do pliku wyjściowego final_test.csv zapisywany był czas wykonania, otrzymane rozwiązanie (koszt ścieżki) oraz ścieżka (numery kolejnych węzłów). Plik wyjściowy zapisywany był w formacie csv. Poniżej przedstawiono fragment zawartości pliku wyjściowego:

tsp_10.txt 20 212 0 3 4 2 8 7 6 9 1 5 0

41

43

33

47

(…)

tsp_12.txt 20 264 0 1 8 4 6 2 11 9 7 5 3 10 0

5185

5236

5150

(…)

tsp_13.txt 20 269 0 10 3 5 7 9 11 2 6 4 8 1 12 0

(…)

Wyniki opracowane zostały w programie MS Excel.

5.1 Metoda pomiaru czasu

Pomiary czasu uzyskane zostały przy pomocy funkcji clock pochodzącej z biblioteki time.h.

Funkcja clock zwraca czas zużyty przez procesor w „clock ticks”. Następnie otrzymane tiki zostają przekonwertowane na milisekundy przy pomocy makra CLOCKS_PER_SEC. Za ostateczne obliczanie czasu odpowiedzialna jest następująca linia kodu:

cpu_time = ((double) (end - start)) / (CLOCKS_PER_SEC/1000);

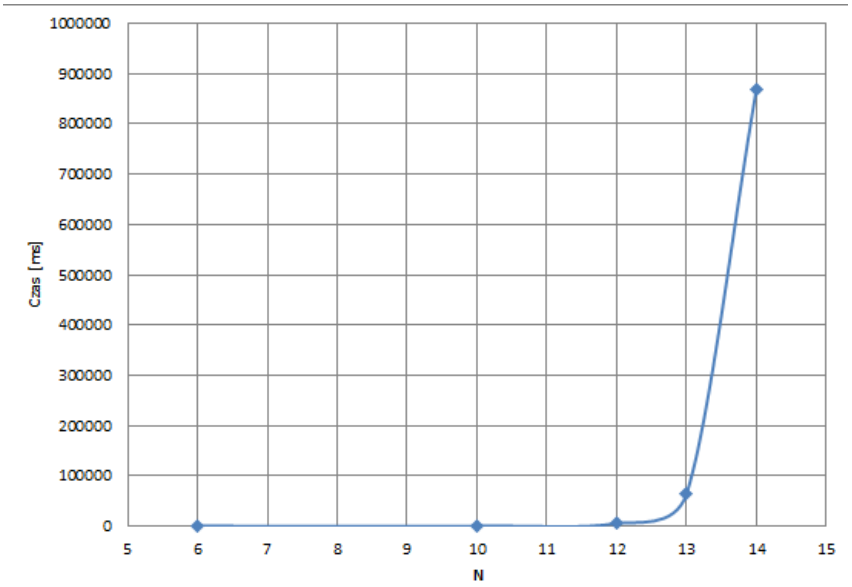
Metoda pomiaru czasu okazała się nie wystarczająco dokładna dla najmniejszej instancji. Sprawdza się ona jednak dobrze dla kolejnych instancji. W związku z tym dla najmniejszej instancji średni czas został wyliczony osobno i wynosi on około 10ms. Został on obliczony poprzez wywołanie wielokrotne metody TSP() a następnie podzielenia łącznego otrzymanego czasu przez liczbę jej wywołań.

6. Wyniki

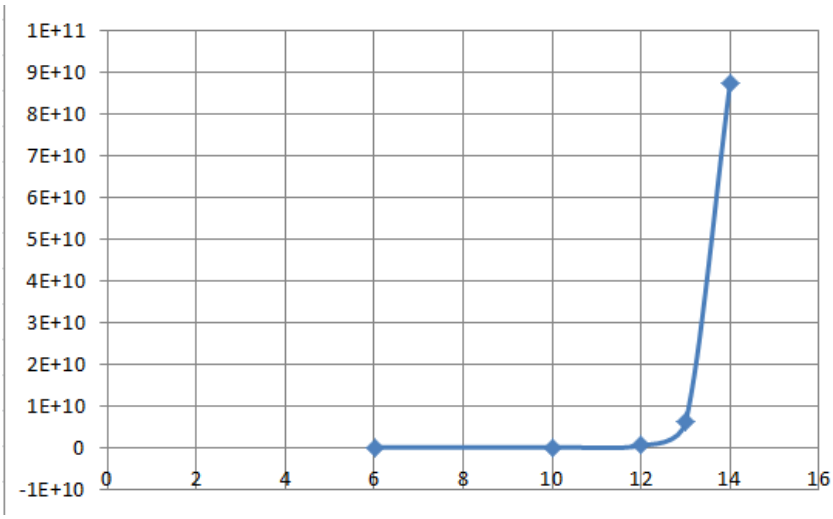
Wyniki zwrócone w wyniku działania programu są przechowane w pliku final_test.csv.(jest to czysta wersja wyników zwróconych przez program). Natomiast ich wersja przedstawiona w przystępniejszej formie, wraz z wykresami została zawarta w pliku wyniki.xlsx.

Wyniki zostały przedstawione w postaci wykresu złożoności czasu uzyskania rozwiązania problemu od wielkości instancji (rysunek 1.).

Został przedstawiony także wykres funkcji y(n) = n!. (rysunek 2.)



Rysunek 1: wpływ wielkości instancji n na czas uzyskania rozwiązania problemu komiwojażera metodą brute force



Rysunek 2: Wykres funkcji y(n) = n!

7. Analiza wyników i wnioski

Krzywa wzrostu czasu względem wielkości instancji ma charakter wykładniczy (rysunek 1). Przy porównaniu wykresu z wykresem funkcji y(n) = n! (rysunek 2.) potwierdza się fakt iż badany algorytm wyznacza rozwiązania problemu komiwojażera dla badanej instancji w czasie n! zależnym względem wielkości instancji (obie krzywe są zgodne co do kształtu). Złożoność czasowa opracowanego algorytmu wynosi zatem O(n!).