Projektowanie Efektywnych Algorytmów

Projekt

18.10.2023

263900 Aleksandra Chrustek

(1) Brute force

1 Sformułowanie zadania

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

1. Graf pełny to zbiór wierzchołków, przy czym między każdymi dwoma wierchołkami istnieje krawędź je łącząca. [1]
2. 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 rozumiemy jako zadanie polegające na znalezienu najlepszej drogi dla podróżującego chcącego odwiedzić n miast i skończyć podróż w miejscu jej rozpoczęcia. Połączenie między każdym miastem ma swój koszt określający efektywnosć jej przebycia. Najlepsza droga to taka, której całkowity koszt (suma kosztów przebycia wszystkich połączeń między miastami na 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 przebycia połączenia z A do B, jak z B do A, czyli dane połączenie ma 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 wszystkich dopuszczalnych rozwiązań 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 w 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.

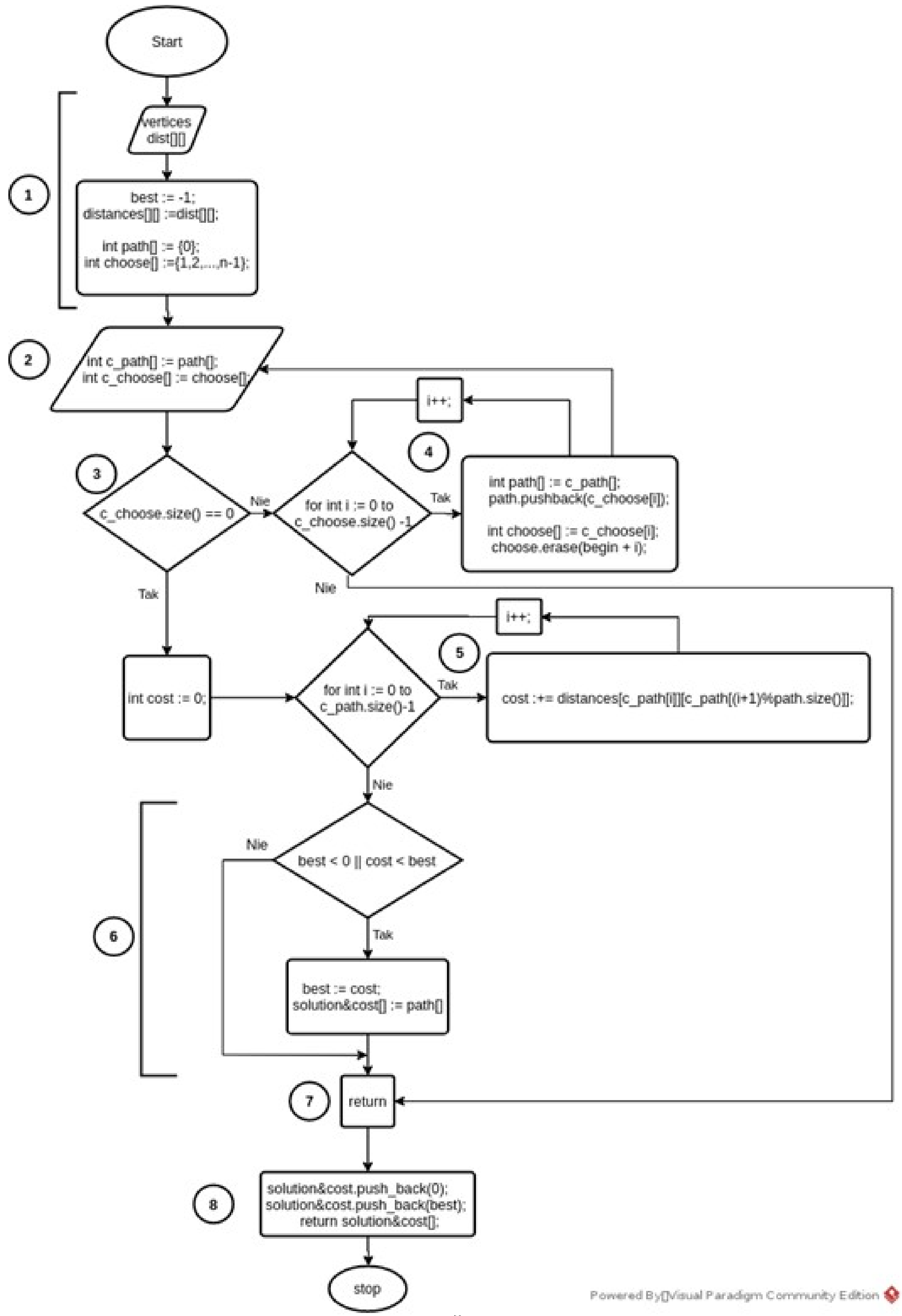
Tabela 1: Permutacje zbioru 0,1,2,3,4

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 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 wierzchołek 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 oraz tablice wierzchołków 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 wszystkich 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 wierzchołek 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 kompletny i można przejść do obliczenia kosztu i sprawdzenia czy jest optymalny.

4. Jeśli tablica c_choose[] zawiera wierzchołki 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 wierzchołka, 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 wierzchołkami 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
tsp_14.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 20 132 0 1 2 3 4 5 0
tsp_6_2.txt 20 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 5 264 0 1 8 4 6 2 11 9 7 5 3 10 0
tsp_13.txt 1 269 0 10 3 5 7 9 11 2 6 4 8 1 12 0
tsp_14.txt 1 282 0 10 3 5 7 9 13 11 2 6 4 8 1 12 0
tsp_dr_mierzwa_6_14.csv

Każda z instancji rozwiązywana była zgodnie z liczbą jej wykonań, np. tsp_12.txt wykonana została 5 razy. Do pliku wyjściowego tsp_dr_mierzwa_6_14.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 zawartość pliku wyjściowego. Kolorem czerwonym zaznaczone zostały wyniki odbiegające od normy - anomalie - nie były one uwzględniane w obliczaniu średniej.

tsp_6_1.txt 20 132 0 1 2 3 4 5 0
304
300
292
292
291
292
291
298
277
269
268
269
268
269
268
268
275
269
268
268

tsp_6_2.txt 20 80 0 5 1 2 3 4 0
275
268
267
267
268
267
268
274
283
285
274
269
273
321
267
268
267
280
268
267
tsp_10.txt 10 212 0 3 4 2 8 7 6 9 1 5 0
932923
876057
871363
846115
836886
836527
832811
834557
832569
834337
tsp_12.txt 5 264 0 1 8 4 6 2 11 9 7 5 3 10 0
92559085
92024111
92794163
94003758
92639226
tsp_13.txt 1 269 0 10 3 5 7 9 11 2 6 4 8 1 12 0
1147277683
tsp_14.txt 1 282 0 10 3 5 7 9 13 11 2 6 4 8 1 12 0
15136627937
tsp_dr_mierzwa_6_14.csv

Pomiary zostały wykonane na platformie sprzętowej:
procesor: Intel® Core™ i5-01400F CPU @ 2.90GHz × 6
pamięć operacyjna: 16 GiB
system operacyjny: Windows 11 64-bit

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

Po każdym powtórzeniu wykonania 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 znalezienia innej ścieżki, program informuje o znalezieniu innej ścieżki i/lub kosztu. Sytuacja, w której opracowany algorytm znalazłby inną ścieżkę i/lub koszt niż w pliku konfiguracyjnym nie miała miejsca.

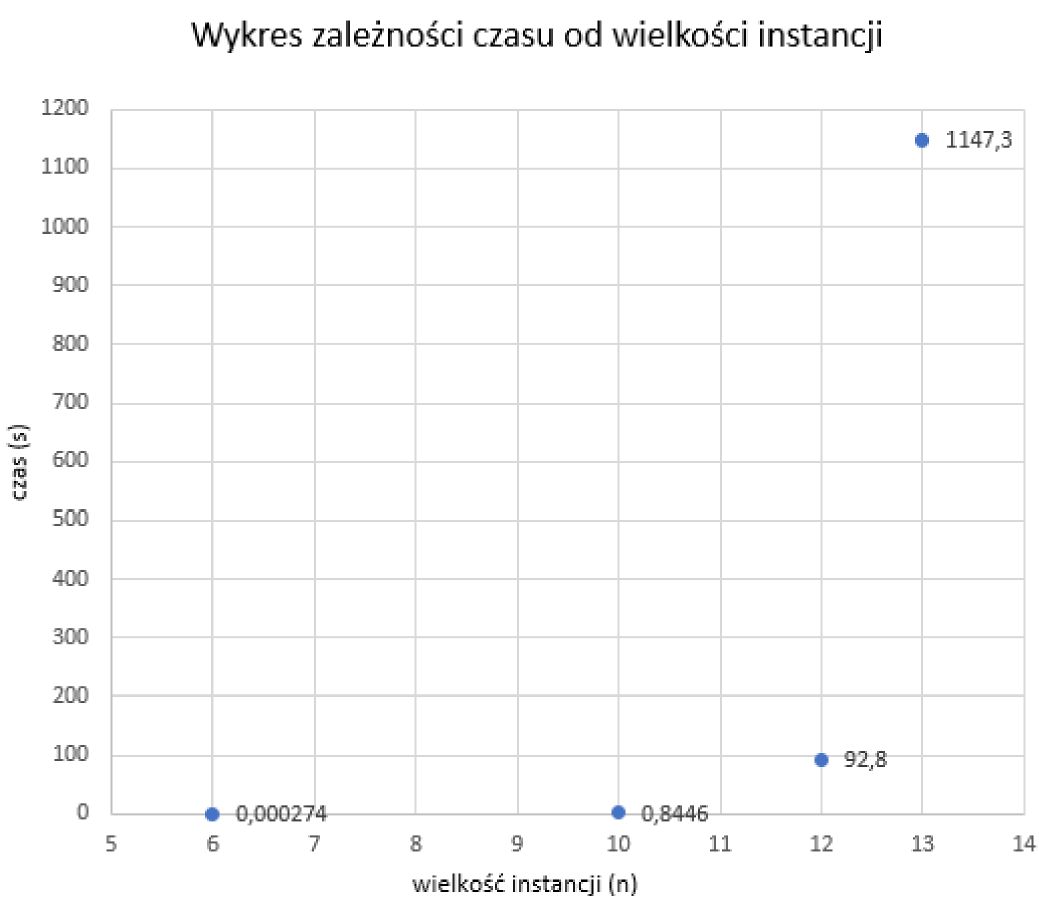
Wyniki zostały opracowane w programie Microsoft Excel.

6 Wyniki

Wyniki zgromadzone zostały w pliku: tsp_dr_mierzwa_6_14.csv

Tabela 2: Tabela wyników

wielkość instancji (n)	*czas t(s)*
6	0,0002743
10	0,8446
12	92,8
13	1147,3
14	15136,6



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

Rysunek 3: Wykres n!

7 Analiza wyników i wnioski

Wzrost czasu względem wielkości instancji ma charakter wykładniczy (rysunek 2). 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!). Algorytm brute force rozwiązujący problem komiwojażera jest efektywny dla instancji do 12. Dla większych instancji czas rozwiązania znacznie wzrasta. Na wykresie uwzględniono instancje do 13 z pominięciem 14, w celu zwiększenia czytelności wykresu.

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