Projektowanie Efektywnych Algorytmow

Projekt

21/10/2022

259193 Kacper Wróblewski

(1) Brute force

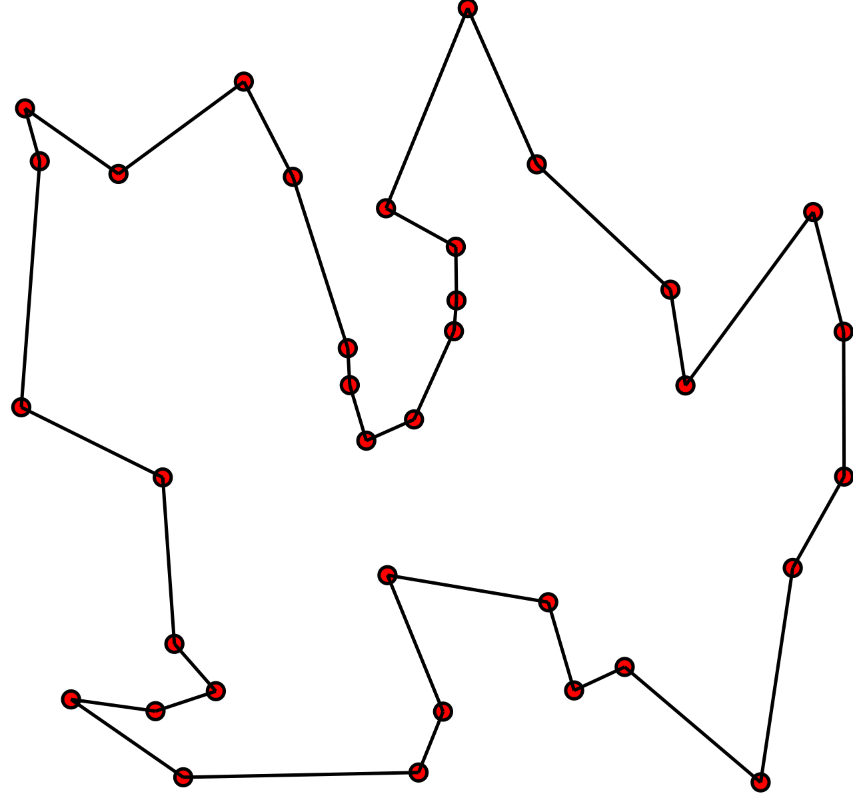
|  |  |
| --- | --- |
| Spis treści | strona |
| Sformułowanie zadania | 2 |
| Opis metody | 3 |
| Opis algorytmu | 4 |
| Dane testowe | 5 |
| Procedura badawcza | 6 |
| Wyniki | 7 |
| Analiza wyników i wnioski | 8 |

# Sformułowanie zadania

Zadanie polega na opracowaniu, implementacji i zbadaniu efektywnosci algorytmu przeglądu zupełnego rozwiązującego problem komiwojażera w wersji optymalizacyjnej. Algorytm był realizowany na gotowym grafie stworzonym z danych do opracowania.

Problem komiwojażera, czyli TSP (*travelling salesman problem*) polega na znalezieniu cyklu Hamiltona w grafie, który ma najmniejszy koszt. Sprowadza się do wyznaczenia najkrótszej ścieżki pomiędzy wierzchołkami przedstawianymi jako miasta (stąd problem podróżującego sprzedawcy). Dana jest określona ilość miast i odległość albo cena podróży pomiędzy nimi i podróżujący musi odwiedzić wszystkie płacąc jak najmniej lub podróżując jak najmniejszą odległość.

Problem TSP należy do klasy problemów NP-trudnych, co znaczy, że rozwiązanie nie zawsze jest jasne lub zajmuje zwyczajnie zbyt długo, aby brać je pod uwagę przez co wymagane jest częste zawężanie kryteriów.



*https://pl.wikipedia.org/wiki/Problem\_komiwojażera#/media/Plik:GLPK\_solution\_of\_a\_travelling\_salesman\_problem.svg*

*Rys. 1. Przykładowe rozwiązanie problemu TSP*

# Metoda

Metoda przeglądu zupełnego, tzw. przeszukiwanie wyczerpujące *(eng. exhaustive search)* bądź metoda siłowa *(eng. brute force)*, 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 tym przypadku zależy nam na znalezieniu możliwie małego kosztu przejścia pomiędzy wszystkimi wierzchołkami, więc rozwiązujemy problem minimalizacyjny.

Sposób ten jest trywialny i sprowadza się do sprawdzenia wszystkich permutacji wierzchołków grafu, ich kosztów oraz wyłonienia rozwiązania optymalnego. Swoją nazwę zawdzięcza naiwności, gdyż jest to rozwiązanie bardzo proste i może wpaść na nie praktycznie każdy, bez większego wysiłku intelektualnego.

Jednakże poprzez swoją prostotę jest to sposób mało optymalny, gdyż zajmuje bardzo długo i wymaga rozpatrzenia dużej ilości przypadków.

# Algorytm

# 

*Rys. 2. Schemat blokowy algorytmu*

# Do algorytmu rozpoczyna się z zainicjalizowanym grafem po czym za pomocą moduły pythonowego wytwarza wszystkie możliwe permutacje kolejności wierzchołków. Potem w pierwszej pętli rozpatruje kolejno permutacje, a w drugiej jej parametry (koszt). Po wyjściu z zagnieżdżonej pętli algorytm aktualizuje wagę oraz sprawdza czy aktualna waga jest lepsza od dotychczasowej i jeśli tak to ją nadpisuje. Jeśli rozpatrzono wszystkie permutacje, algorytm kończy działanie i zapisuje wynik.

# Dane testowe

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

1. *tsp\_6\_1.txt*
2. *tsp\_6\_2.txt*
3. *tsp\_10.txt*
4. *tsp\_12.txt*
5. *tsp\_13.txt*
6. *tsp\_14.txt*
7. *tsp\_15.txt*
8. *tsp\_17.txt*

<http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/>

Do programu został dołączony plik *config.ini* aby sterować parametrami programu w sposób zastępujący:

*Nazwa pliku:*

*tsp\_12.txt //tutaj zawarta nazwa pliku wywoła odpowiedni graf w programie.*

# 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, którego struktura została opisana wyżej.

Dla problemów o mniejszych rozmiarach można było wykonać algorytm sto razy. Jednak przy wielkościach przekraczających dwanaście rozwiązanie trwało zbyt długo, aby stwierdzić estymowany wynik, a tym bardziej otrzymać gotowy cykl Hamiltona. W związku z tym zmniejszono próbkę do dziesięciu, jednak to nie zrobiło większej różnicy.

Wyniki zostały zgromadzone w pliku wyjściowym *dane.csv*. Brane pod uwagę było optymalne rozwiązanie problemu oraz średni czas wykonania w sekundach.

Wyniki opracowane zostały w programie MS Excel.

# Wyniki

Wyniki zgromadzone zostały w pliku *dane.csv.* Wszystkie ww. pliku zostały dołączone do raportu i znajdują się na dysku Google pod adresem: https://drive.google.com/drive/folders/1N8be7vT8iL9-Xmcs7t0\_Q6BUi8diObOs?usp=sharing.  
Wyniki przedstawione zostały w postaci wykresu zależności czasu uzyskania rozwiązania problemu od wielkości instancji (rysunek 3).

|  |  |
| --- | --- |
| **droga** | **czas[s]** |
| [0, 1, 2, 3, 4, 5] | 0,0010011 |
| [0, 5, 1, 2, 3, 4] | 0,0010006 |
| [0, 3, 4, 2, 8, 7, 6, 9, 1, 5] | 2,2400355 |
| [0, 1, 8, 4, 6, 2, 11, 9, 7, 5, 3, 10] | 338,786 |

*tab. 1. Fragment tabeli wynikowej*

*Rys. 3. Wykres wynikowy*

# Analiza wyników i wnioski

Krzywa wzrostu czasu względem wielkości instancji ma charakter wykładniczy *(Rysunek 3)*. Nałożenie lini trendu eksponecjalnej potwierdza, że badany algorytm wyznacza rozwiązania problemu komiwojażera dla badanych instancji w czasie *n!* zależnym względem wielkości instancji (obie krzywe są zgodne co do kształtu). Złożoność czasowa opracowanego algorytmu wynosi O(n!).

Wykres potwierdza małą praktyczność i naiwność zastosowanej metody. Mimo stosunkowo małego wzrostu liczby instancji (+4, +6) złożoność czasowa rośnie najpierw o ok. 100%, a w większych grafach o ok. 100000%.