Projektowanie Efektywnych Algorytmów

Projekt

21/10/2022

259193 Kacper Wróblewski

(2) Algorytm Helda-Karpa

|  |  |
| --- | --- |
| 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 |

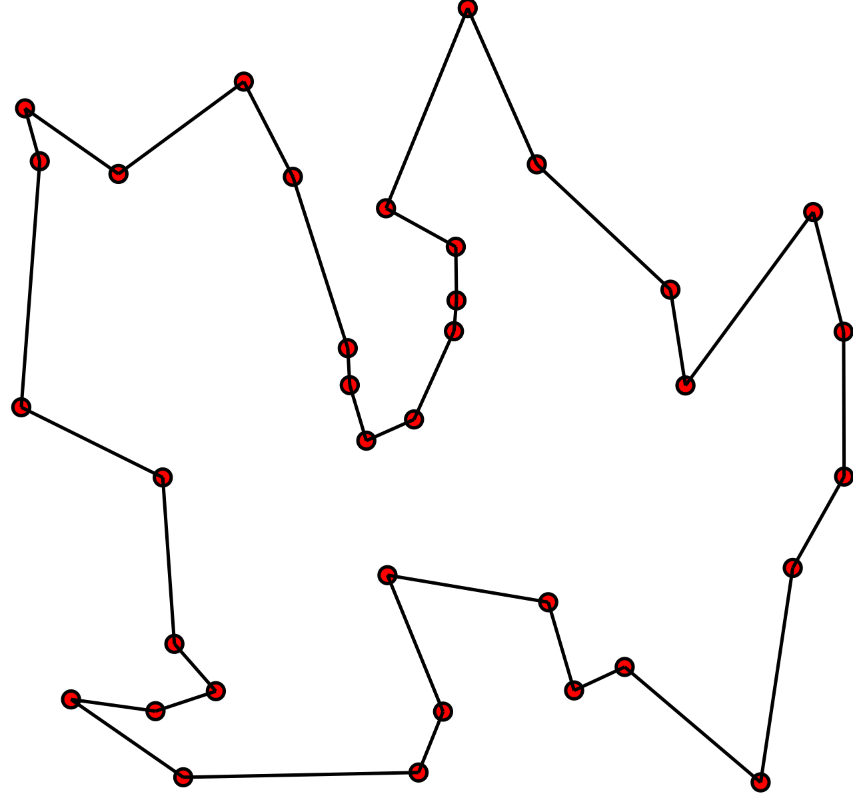
# Sformułowanie zadania

Zadanie polega na opracowaniu, implementacji i zbadaniu efektywności algorytmu Helda-Karpa rozwiązującego problem komiwojażera w wersji optymalizacyjnej. Algorytm był realizowany na gotowym grafie stworzonym z danych do opracowania.

Należało zbadać zależność czasową od wielkości instancji (jak w zadaniu 1.) oraz pamięciową.

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. Jak przedstawiono na Rys. 1, rozwiązaniem jest cykl w grafie zupełnym w postaci listy kolejnych wierzchołków oraz całkowity koszt przebytej drogi.

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 lub kontemplacja jakości algorytmu w danym kontekście.



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

*Rys. 1. Przykładowe rozwiązanie problemu TSP*

# Metoda

Programowanie dynamiczne *(dynamic programming)* jest to sposób projektowania algorytmów rozwiązujących zagadnienia optymalizacyjne (takie jak *TSP*). Jest to alternatywa dla metod zachłannych. Strategię programowania dynamicznego opracował amerykański matematyk Richard Bellman.

Podstawą tej metody jest podział problemu na mniejsze części, zwane podproblemami, względem kilku parametrów. Przez dużą złożoność pamięciową zastosowanie w programowaniu dynamicznym naiwnych rozwiązań jakim jest np. przegląd zupełny *(ang. brute force)* prowadzi do znacznego wzrostu złożoności pamięciowej problemu.

Kluczem do zaprojektowania algorytmu programowania dynamicznego jest znalezienie funkcji rekurencyjnej, która optymalnie opisuje funkcję celu dzieląc problem na części (zazwyczaj od najmniejszej do największej). Dzięki świadomości, że postępowanie jest optymalne mamy pewność, że kolejne wywołania funkcji również takie będą. W tym podejściu pamiętane są wszystkie poprzednie rozwiązania podproblemu, więc rozbijanie zadania na części nie tylko ma sens w kwestii przydatności, ale również optymalności algorytmów programowania dynamicznego.

Algorytm Helda-Karpa (czasami określany również jako algorytm Bellmana-Helda-Karpa) jest to algorytm oparty na programowaniu dynamicznym służący do rozwiązywania problemu komiwojażera. Jego złożoność czasowa to *O(n22n)*  a pamięciowa *O(n2n).* Pomimo tego, że te wartości są gorsze od najczęściej spotykanych wielomianowych, jest to złożoność lepsza od *O(n!)* prezentowanej przez przegląd zupełny.

W skrócie, metoda polega na przeglądzie wszystkich podzbiorów (kombinacji) zbioru wierzchołków w rosnącej kolejności wynikowo przypisując kolejne optymalne wartości. Szczegóły implementacji zaprezentowano w punkcie 3.

# Algorytm

# *Rys. 2. Schemat blokowy całego programu*

*Rys. 3. Schemat blokowy implementacji algorytmu Helda-Karpa*

Algorytm

# Dane testowe

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

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

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://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html>

1. *gr17.tsp*
2. *gr21.tsp*
3. *gr24.tsp*
4. *bays29.tsp*

<http://elib.zib.de/pub/mp-testdata/tsp/tsplib/atsp/index.html>

1. *br17.atsp*

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

*Plik wejściowy:*

*./input/tsp\_6\_1.txt //tutaj zawarta nazwa pliku wywoła odpowiedni graf w programie.*

*Plik wyjściowy:*

*./output/data.csv //ścieżka pliku wyjściowego*

*Ilość powtórzeń:*

*100 //ilość razy, z jaką wykona się algorytm*

# Procedura badawcza

Należało zbadać zależność czasu rozwiązania problemu od wielkości instancji. W przypadku algorytmu realizującego programowanie dynamiczne w 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.* Wyniki przedstawione zostały w postaci wykresu zależności czasu uzyskania rozwiązania problemu od wielkości instancji (rysunek 3).

# Analiza wyników i wnioski