Projektowanie Efektywnych Algorytmów

Projekt

19/01/2023

259193 Kacper Wróblewski

(7) Algorytm mrówkowy

|  |  |
| --- | --- |
| Spis treści | strona |
| Sformułowanie zadania | 2 |
| Opis metody | 3 |
| Opis algorytmu | 4 |
| Dane testowe | 7 |
| Procedura badawcza | 8 |
| Wyniki | 9 |
| Analiza wyników i wnioski | 15 |

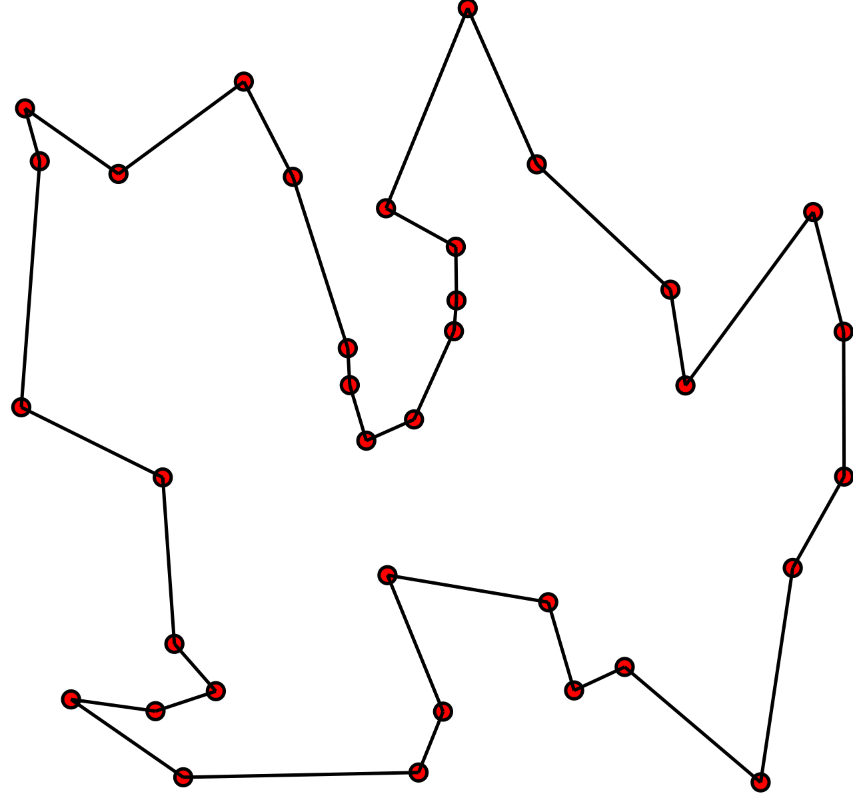
# Sformułowanie zadania

Zadanie polega na opracowaniu, implementacji i zbadaniu efektywności algorytmu mrówkowego rozwiązującego problem komiwojażera w wersji optymalizacyjnej. Algorytm był realizowany na gotowym grafie stworzonym z danych do opracowania. Ze względu na specyfikę metody wśród danych do opracowania zamieszczono również dokładny wynik najkrótszej drogi dla każdej instancji jako punkt orientacyjny dla wyliczania wartości błędu.

Należało zbadać zależność czasową od wielkości instancji (jak w zadaniu 1. i 2.), pamięciową (jak w zadaniu 2.) oraz od parametrów algorytmu.

Problem komiwojażera, czyli TSP (*travelling salesman problem*) polega na znalezieniu cyklu Hamiltona w grafie, który ma najmniejszy koszt*.* Sprowadza się to 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. Podróżujący musi odwiedzić wszystkie z nich 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

# Algorytm

# *Rys. 2. Schemat blokowy całego programu*

Program, jak przedstawiono na Rys. 2., rozpoczyna się od wczytania danych zawartych w pliku konfiguracyjnym. Potem należy rozpatrzeć przypadek rozszerzenia pliku, gdyż część z badanych zbiorów ma inny format niż *.txt*, który wymaga nieco innej metody odczytu (pliki *.tsp/.astp* można łatwo odczytać za pomocą biblioteki *tsplib95* w pythonie). Następnie program wchodzi w pętlę realizującą zaimplementowany algorytm. Liczba iteracji tej pętli określona jest w pliku konfiguracyjnym. Gdy program zakończy iterację, zapisuje zwrócone dane do pliku, którego ścieżka i nazwa również znajdują się w pliku sterującym, czym powinien zakończyć się program. Opcjonalnie, na końcu programu można wypisać   
w konsoli interesujące programistę dane, w celu szybszej weryfikacji poprawności metody.

*Rys. 3. Schemat blokowy implementacji algorytmu mrówkowego*

# Dane testowe

Do sprawdzenia poprawności działania algorytmu i przeprowadzenia badań wybrano następujący zestaw instancji (wyniki zebrano w punkcie 6.):

<http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/>

*Plik: Optymalne wartości (koszt):*

1. *gr96.tsp 55209*
2. *kroa100.tsp 21282*
3. *krob150.tsp 26130*
4. *krob200.tsp 29437*
5. *pr152.tsp 736382*
6. *ftv170.atsp 2755*
7. *rbg323.atsp 1326*

Pliki testowe są w następującym formacie:

Pliki sformatowano do *.txt* oraz dodano optymalną drogę w pierwszej linii celem wczytania do programu danych potrzebnych do wyliczenia błędu pomiaru. Kolejną linię zajmuje wielkość grafu. W kolejnych znajduje się macierz sąsiedztwa.

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

*Plik wejściowy:*

*./input/ftv170.txt //wybór instancji po nazwie z folderu input*

*Plik wyjściowy:*

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

*Ilość powtórzeń:*

*1 //ilość iteracji pętli z algorytmem*

*Alfa: //poniżej dobór parametrów algorytmu*

*1.0 // mrówkowego*

*Beta:*

*5.0*

*Rho:*

*0.1*

*Ilość mrówek: //domyślnie równa wielkości instancji*

*170*

*Ilość iteracji:*

*500*

*Aktualizacja feromonu: (DAS, QAS, CAS) //sposób aktualizacji rozkładu feromonu*

*QAS*

*Heurystyka wyboru: (visibility, frequency factor) //wybór heurystyki* visibility *(podana na*

*visibility // wykładzie) lub* frequency factor *(wymyślona)*

# Procedura badawcza

Należało zbadać zależność czasu rozwiązania problemu i złożoność pamięciową od wielkości instancji dla algorytmu mrówkowego. W przypadku tego algorytmu w przestrzeni rozwiązań dopuszczalnych 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*, wraz z podaniem żądanych parametrów metody (jego struktura została opisana powyżej).

Ponadto, mierzono średnie zużycie pamięci procesu dla tego algorytmu, celem porównania złożoności pamięciowej z programowaniem dynamicznym (które znane jest z wysokiej złożoności pamięciowej).

Wyniki zostały zgromadzone w plikach wyjściowych *.csv* znajdujących się w katalogu *output*.

Wyniki opracowane zostały w programie MS Excel.

# Wyniki

Wyniki zgromadzone zostały w plikach *.csv* w katalogu *output.* Przedstawione zostały w postaci wykresu zależności czasu uzyskania rozwiązania problemu od wielkości instancji (wykres 1.).

# Analiza wyników i wnioski