Piotr Małek, nr 199322

Filip Piechocki, nr 199334

**Projekt z przedmiotu GIS**

**„Problem komiwojażera”**

**Sprawozdanie nr. 3**

# Opis problemu:

Problem komiwojażera (ang. Traveling Salesman Problem, w skrócie TSP) polega na wyznaczeniu najkrótszej ścieżki przebiegającej przez wszystkie wierzchołki w grafie, czyli znalezieniu minimalnego cyklu Hammiltona w grafie spójnym. Nazwa nawiązuje do najbardziej nasuwającego się zastosowania algorytmów wyznaczających taką ścieżkę, czyli wyliczania najkrótszej trasy dla samochodu dostawczego między lokalizacjami do których musi dostarczyć towar, tak aby zużyć jak najmniej benzyny.

Mimo że problem jest powszechny i wykracza daleko poza zagadnienia logistyczne, nie wynaleziono jeszcze optymalnego algorytmu działającego w rozsądnym czasie i wyznaczającego najlepszą możliwą ścieżkę. Wymyślono za to rozmaitość rozwiązań heurystycznych, działąjących w akceptowalnie długim czasie i dających bardzo dobre wyniki, choć bez gwarancji że nie ma rozwiązań lepszych.

Problem komiwojażera ma sens dla grafu spójnego, skierowanego bądź nie skierowanego, ważonego.

Jako przykład rozwiązania heurystycznego zostanie w ramach projektu zaimplementowany algorytm kopca mrówek, oraz naiwny do celów porównawczych.

# Założenia projektowe:

W ramach projektu zostaną zaimplementowane dwie metody wyznaczania minimalnego cyklu Hammiltona w grafie: naiwy oraz heurystyczny wykorzystujący system kopca mrówek.

Projekt zostanie zrealizowany w C++, z wykorzystaniem biblioteki Qt do celów prezentacji wyników.

Program który będzie wynikiem tego projektu będzie umożliwiał zaczytanie grafu z pliku tekstowego oraz zapewni mechanizm do prezentacji wyników. Efektem tego projektu będzie aplikacja okienkowa implementująca algorytm heurystyczny z wykorzystaniem systemu kopca mrówek i oraz algorytm naiwny.

# Opis algorytmu kopca mrówek:

Rozwiązanie zostanie zaimplementowane zgodnie ze specyfikacją zawartą w publikacji *„Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem”* [[1]](#footnote-1) .

## Ogólna koncepcja:

Koncepcja tego rozwiązania sprowadza się do zaimplementowania rozwiązania bazującego na tym jakie wykorzystują w naturze mrówki. Chodzi o to aby graf zasiedlić populacją botów (mrówek). Każda z mrówek zostawia na swojej drodze feromony, a każda kolejna mrówka przy podjęciu decyzji którą drogą pójść decyduje się na nią zgodnie z prawdopodobieństwem bazującym na ilości pozostawionych na niej feromonów. Czyli teraz jeżeli rozpatrzymy dwie ścieżki między dwoma wierzchołkami, jedną o długości 10 a drugą 20 to w pierwszej fazie działania algorytmu na żadnej nie będzie feromonów dlatego połowa mrówek pójdzie ścieżką krótszą, a połowa dłuższą. Jednak ponieważ na krótszej ścieżce odłożone zostananie więcej feromonów (przy wyborze ścieżki promowane są ścieżki krótsze), każda kolejna mrówka z większym prawdopodobieństwem wybieże ścieżkę krótszą. Po zakończeniu przez mrówki swoich cykli następuje aktualizacja ilości feromonów. Po kilku takich cyklach prawdopodobnie któraś z mrówek znajdzie rozwiązanie bliskie optymalnemu.

## Właściwy algorytm

Inicjalnie mrówki rozmieszczane są losowo po poszczególnych węzłach grafu i każda z krawędzi jest spryskiwana inicjalną ilością feromonu [parametr „pheromone0”]. Następnie rozpoczyna się seria iteracji przejść przez graf. W ramach każdego z takich koków mrówki przechodzą cały graf, a następnie rozpoczyna się procedura „globalUpdate” polegająca na „wyparowaniu” feromonów ze wszystkich krawędzi i wzmocnienia tych krawędzi z ścieżki najkrótszej. Dzięki temu istnieje większa szansa że ta najkrótsza ścieżka zostanie ponownie wykorzystana do budowy następnych, jeszcze krótszych ścieżek. W ramach takiego pełnego przejścia grafu dzieje się jeszcze jedna ciekawa rzecz. Pełne przejście składa się z N kroków (gdzie N jest równe ilości wieżchołków w grafie), w ramach takiego kroku każda z mrówek wybiera sugerując się prawdopodobieństwem i ilością feromonów na krawędzi kolejna krawędź. Wybrana krawędź jest lekko pozbawiana feromonów, tak aby nie stało się tak że każda mrówka przejdzie tylko tą jedną ścieżką. Czyli po każdym kroku następuje lekkie tłumienie wybranych krawędzi, aby uniknąć chodzenia cały czas tą samą ścieżką, a dopiero po przejściu całego grafu, wzmacniana jest ta najlepsza ścieżka.

## Weryfikacja poprawności

W celu weryfikacji poprawności działania algorytmu ACS (Ant Colony System), został zaimplementowany algorytm Brute Force, znajdujący rozwiązanie dokładne. Algorytm zakłada, że graf jest grafem pełnym (a jest, gdyż przed działaniem algorytmów TSP, zamieniamy graf na pełny, co jest opisane poniżej). Dlatego też znajduje wszystkie permutacje wierzchołków grafu, które są ścieżkami w grafie pełnym a po połączeniu początku z końcem, staja się cyklem obejmującym wszystkie wierzchołki. Następnie spośród tych cykli (które są wszystkimi możliwymi cyklami w przechodzącymi przez wszystkie wierzchołki w danym grafie) wybierany jest najkrótszy, który jest rozwiązaniem problemu TSP. Niestety metoda ta, ma złożoność obliczeniową O(n!) więc, na przeciętnym komputerze PC, przestaje być przydatna już dla grafów o ilości wierzchołków przekraczającej 10.

## Założenia projektowe:

Inicjalny graf będzie pobierany z pliku w formacie xml o następującej strukturze:

<graph>

<edge>

<vertex>A</vertex>

<vertex>B</vertex>

<weight>3</weight>

</edge>

<edge>

<vertex>C</vertex>

<vertex>B</vertex>

<weight>8</weight>

</edge>

<edge>

<vertex>A</vertex>

<vertex>C</vertex>

<weight>12</weight>

</edge>

</graph>

Gdzie tag graph przechowuje informacje o całym grafie, czyli :

1. edge – informacja o krawędziach:
   1. vertex – etykieta wierzchołka pierwszego
   2. vertex – etykieta wierzchołka drugiego
   3. weight – waga krawędzi (odległość pomiędzy wierzchołkami)

Wynik przetwarzania zostanie przedstawiony w programie jako porównanie ścieżki najlepszej wyznaczonej metodą bruteforce oraz efektu przetwarzania z wykorzystaniem wirtualnego kopca mrówek.

Zakładamy że graf na którym będziemy pracować będzie grafem spójnym. Dlatego przed poddaniem grafu działaniu algorytmu zostanie przetestowany pod kątem spójności przy użyciu zasady DFS (wybieramy losowy wierzchołek i przechodzimy graf w głąb).

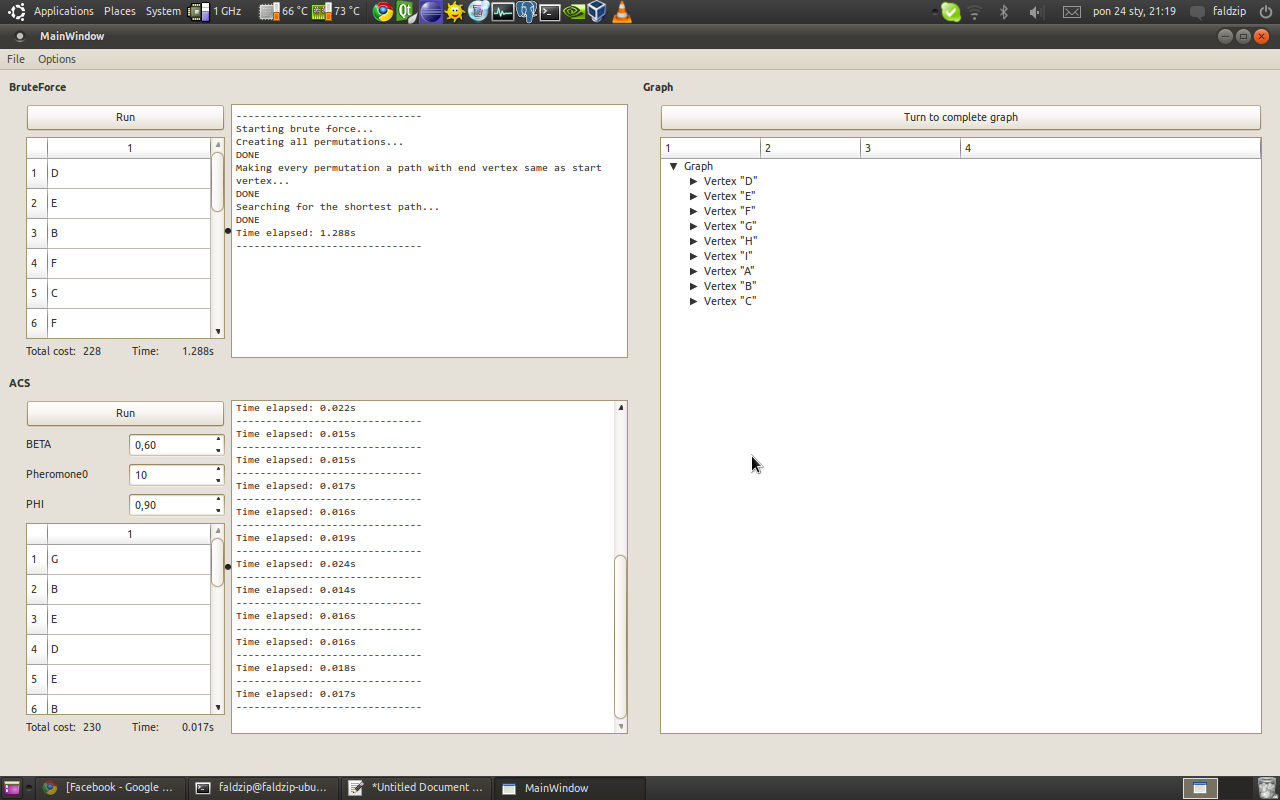
Ponieważ problem komiwojażera zakłada odwiedzanie każdego z miast tylko raz, to spójność grafu nie jest jedynym warunkiem na istnienie cyklu Hamiltona. Np. z wierzchołka będącego liściem nie da się już wrócić, wiec przed poddaniem grafu działaniu algorytmu, zostanie on zamieniony w graf pełny, gdzie każda dodana podczas tej operacji krawędź grafu pomiędzy dwoma wierzchołkami będzie miała wagę równą wadze najkrótszej ścieżki z znalezionej pomiędzy tymi wierzchołkami. Do wyznaczenia najkrótszych ścieżek został użyty algorytm Floyda-Warshalla. Dodatkowo jest możliwe określenie z przejścia przez jakie krawędzie powstała ta najkrótsza ścieżka, tak więc wynikowa ścieżka zawiera tylko połączenia które istniały w inicjalnym grafie.

## Struktury danych:

W celu jak najlepszej optymalizacji graf będzie jako tablica wierzchołków. Wierzchołek będzie przechowywał informację o wychodzących z niego krawędziach (słownik, o kluczu reprezentowanym wierzchołek docelowy). Każda krawędź z kolei będzie przechowywać informacje o swojej długości.

Dla ACS będzie przypisana specjalna struktura przechowująca Zestawienie krawędź-poziom feromonu, oraz zestaw parametrów sterujących działaniem algorytmu.

# Wykonany program



Przeprowadziliśmy także testy, których wyniki prezentowane są poniżej:

|  |  |  |  |
| --- | --- | --- | --- |
| Ilość wierzchołków | Algorytm | Czas (s) | Uwagi |
| 5 | Bruteforce | 0.004-0.008 |  |
| ACS | 0.009-0.016 |  |
| 6 | Bruteforce | 0.005-0.010 |  |
| ACS | 0.009-0.016 | 1/5 ścieżka dłuższa |
| 7 | Bruteforce | 0.018-0.027 |  |
| ACS | 0.013-0.021 | 1/2 ścieżka dłuższa |
| 8 | Bruteforce | 0.099-0.180 |  |
| ACS | 0.014-0.024 | 1/4 ścieżka najkrótsza |
| 9 | Bruteforce | 0.928-1.468 |  |
| ACS | 0.018-0.022 | 1/10 ścieżka najkrótsza |
| 10 | Bruteforce | 9.822-14.682 |  |
| ACS | 0.018-0.022 | 1/20 ścieżka najkrótsza |

1. Publikacja dostępna pod adresem http://www.idsia.ch/~luca/acs-ec97.pdf [↑](#footnote-ref-1)