# Multiple Traveling Salesmen Problem

Marcin Juraszek
Jakub Noga

20 stycznia 2015

## 1 Wstęp

### 1.1 Opis problemu

Problem wielu komiwojażerów (MTSP) jest odmianą szeroko znanego problemu komiwojażera (TSP). Problem charakteryzuje się większą złożonością (większa liczba rozwiązań dopuszczalnych), natomiast nie powoduje to zwiększenia trudności w jego matematycznym modelowaniu.

Istnienie problemu wielu komiwojażerów możemy zaobserwować na przykład w logistyce, kiedy to dysponując ograniczoną flotą pojazdów i kierowców (w odróżnieniu do TSP, gdzie dysponujemy dokładnie jednym) chcemy dostać się do wielu miejscowości ponosząc możliwie najniższe, zdefiniowane przez nas koszty.

### 1.2 Podejście do problemu

Zdecydowaliśmy, że przystąpimy do rozważań nad problemem sprowadzając go do klasycznego problemu jednego komiwojażera, stosując jedynie drobną modyfikację: miasto - *baza* może zostać osiągnięte wielokrotnie. Spowoduje to powstanie opartych na nim podcykli, które, równoważnie, można traktować jako zakończenie trasy jednego, a rozpoczęcie drugiego komiwojażera.

## 2 Model matematyczny

### 2.1 Struktura danych

Do przedstawienia danych problemu w języku posłużyliśmy się macierzami oraz, w niektórych przypadkach, wektorami.

Podstawowe dane na temat konkretnej instancji problemu tj. koszty przejść pomiędzy kolejnymi miastami przedstawiliśmy w postaci macierzy kosztów $n \times n$:

$$C = \begin{bmatrix} 0 & c_{01} & c_{02} & \dots & c_{0n} \\ c_{10} & 0 & c_{12} & \dots & c_{1n} \\ \vdots & & \ddots & & \vdots \\ c_{n0} & & \dots & & 0 \end{bmatrix}$$

Gdzie $c_{ij}$ to koszt przejścia pomiędzy miastem $i$ a $j$.

Do przechowania rozwiązań skorzystaliśmy z dwóch równoważnych postaci zapisu sekwencji przejść z miasta do miasta: macierzy przejść $X$ oraz wektora $r$. Gdzie:

$$X = [X_{ij}] \in \{0, 1\}$$

oraz

$$r = [0, r_0, \dots, r_m, 0] : r_k \in \{0, \dots, n\}$$

Na przykład macierz:

$$X = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Odpowiada wektorowi:

$$r = [0, 1, 2, 3, 0, 4, 0]$$

## 2.2 Model

Problem wielu komiwojażerów można przedstawić jako następujący problem programowania całkowito-liczbowego:

$X$ - macierz przejść
$C$ - macierz kosztów
$n$ - rozmiar problemu (ilość miast)
$m$ - ilość komiwojażerów
$c_m$ - koszt wysłania jednego komiwojażera

$$F(X) = \sum_{i=0}^{n} \sum_{j=0}^{n} (C_{ij} X_{ij} + m c_m) \rightarrow min \tag{2.2.1}$$

$$0 \leq \sum_{j=1}^{n} X_{0j} = \sum_{i=1}^{n} X_{i0} \leq m \tag{2.2.2}$$

$$\sum_{i=1}^{n} X_{ij} = 1 : j = 1, \ldots, n \tag{2.2.3}$$

$$\sum_{j=1}^{n} X_{ij} = 1 : i = 1, \ldots, n \tag{2.2.4}$$

# 3 Adaptacja algorytmu pszczelego

## 3.1 Przebieg pracy algorytmu

### 3.1.1 Scouting

Scouting - wysłanie pszczół zwiadowców, polega na wygenerowaniu zbioru losowych rozwiązań dopuszczalnych. Klasa *ScoutBee* posiada metodę:

```
public int[][] call() throws Exception {
  int [][]transitionMatrix = new int[dimensions][dimensions];
  while(!allTownsVisited()){
    int nextTown = -1; int currentTown = stepOutOfDepot();
    transitionMatrix[0][currentTown] = 1;
    visitedTowns.put(currentTown, true);
    while(!allTownsVisited()){
      nextTown = takeStep();
      if(nextTown == 0){
        break;
      } else {
        transitionMatrix[currentTown][nextTown] = 1;
        visitedTowns.put(nextTown, true);
      }
      currentTown = nextTown;
    }
    transitionMatrix[currentTown][0] = 1;
  }
  return transitionMatrix;
}
```

Metoda ta buduję losową macierz przejść w następujący sposób:

(1) Jeżeli nie odwiedzono jeszcze wszystkich miast, wylosuj, do którego wyjść z bazy.

(2) Dopóki nie odwiedzono wszystkich miast losuj kolejne miasto do momentu powrotu do bazy.

(3) Wróć do pkt. 1.

Każda pszczoła-zwiadowca zwraca jedną losową macierz przejść. Ilość pszczół-zwiadowców ustalana jest przez użytkownika.

### 3.1.2 Wybór sąsiedztw elitarnych

Za sąsiedztwa elitarne uważamy zbiór rozwiązań (liczność - $l$ tego zbioru jest określana przez użytkownika), które charakteryzują się najwyższą wartością funkcji celu. Przypisanie odbywa się poprzez sortowanie malejące listy rozwiązń losowych wybranych w pierwszym etapie działania algorytmu, a następnie wybór pierwszych $l$ obiektów z listy.

### 3.1.3 Eksploracja sąsiedztw elitarnych

Na tym etapie algorytm korzysta z postaci równoważnej do macierzy przejść - X, czyli wektora przejść - r.

Eksploracja sąsiedztw elitarnych polega na losowej zmianie kolejności na $n$ kolejnych pozycjach wektora, liczba $n$ zmienia się proporcjonalnie do liczby pozostałych iteracji w następujący sposób:

$$n = WartoscPoczatkowa * \frac{LiczbaIteracji - AktualnaIteracja}{LiczbaIteracji} \tag{3.1.1}$$

$WartoscPoczatkowa$ - jest parametrem podawanym przez użytkownika. Każde z sąsiedztw elitarnych eksplorowane jest przez $w$ pszczół-robotnic - liczba $w$ ustalana jest przez użytkownika.

Przykład:
Dla sąsiedztwa elitarnego o reprezentacji w postaci wektora $r_0 = [0, 9, 8, 7, 6, 0, 1, 3, 2, 0, 4, 5, 0]$, oraz dla $n = 5$, przykładowym rozwiązaniem zwróconym przez funkcję eksploracji sąsiedztw elitarnych może być $r_1 = [0, 9, 8, 7, 0, 6, 2, 1, 3, 0, 4, 5, 0]$. Permutacja miała miejsce od 5. do 10. miejsca w wektorze.

### 3.1.4 Iteracje

Ilość iteracji ustalana jest przez użytkownika. Podczas każdej itaracji powstaje $l \cdot w$ sąsiedztw dlatego wysoce niewydajnym, pod względem pamięci operacyjnej, jest przechowywanie wszystkich wyników. Z tego powodu, co ustaloną liczbę iteracji $i$ następuje ponowny wybór sąsiedztw elitarnych spośród sąsiedztw powstałych z eksploracji. Po tej operacji algorytm kontynuuje standardową eksplorację.

### 3.1.5 Zbiór parametrów algorytmu

- s - liczba pszczół-zwiadowców - oznacza ilość początkowych rozwiązań losowych
- l - liczba sąsiedztw elitarnych - oznacza ilość modyfikowanych rozwiązań w czasie każdej iteracji, w celu polepszenia wartości funkcji celu
- w - liczba pszczół-robotnic przypadająca na jedno sąsiedztwo elitarne - oznacza ilość nowych rozwiązań powstałych z każdego sąsiedztwa elitarnego

- n - ilość elementów wektora przejść, które będą permutowane (wielkość sąsiedztwa) - mówi w jakim stopniu nowe rozwiązanie będzie różniło się od sąsiedztwa elitarnego, z którego się wywodzi.
- i - liczba iteracji, po których sąsiedztwa elitarne zostaną przypisane na nowo - mówi o tym jak często rozwiązania, które powstały poprzez permutację sąsiedztw elitarnych i poprawiły wartość funkcji celu, będą przejmowały rolę sąsiedztw elitarnych.
- t - ilość iteracji algorytmu

# 4 Interfejs użytkownika

## 4.1 Aplikacja Webowa

### 4.1.1 Wybór miejsc na mapie

### 4.1.2 Instancja testowa TSPLib

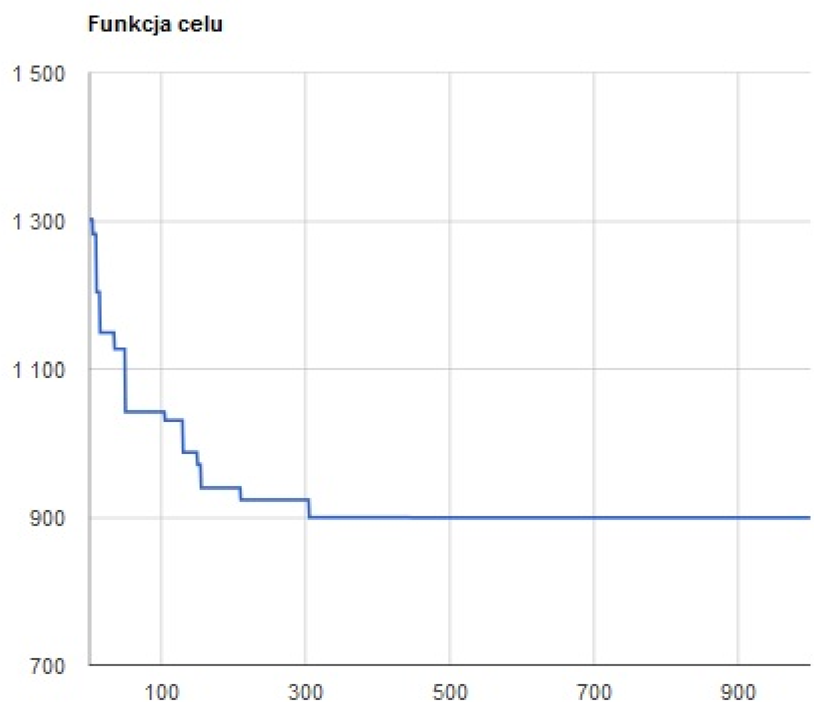
# 5 Testy

## 5.1 TSPLib Eil51

Eil51 jest instancją testową dla algorytmów rozwiązujących problem komiwojażera. Aby test naszego algorytmu był miarodajny nałożyliśmy ograniczenie wykorzystania maksymalnie jednego komiwojażera. Długość optymalnej drogi wynosi 426 jednostek.
Dla małej ilości iteracji (1000) i tym samym krótkich czasów wykonania algorytmu (rzędu 1 minuty) ciężko dobrać parametry, tak aby wyniki były zadowalające.
Poniżej prezentujemy funkcję celu dla:

- 1000 iteracji
- 20 sąsiedztw elitarnych
- Początkowej wielkości sąsiedztwa - 25
- Zmiany sąsiedztw co 5 iteracji
- Każdego sąsiedztwa elitarnego eksplorowanego przez 20 pszczół
- 10000 losowych rozwiązań początkowych

Algorytm znalazł drogę o długości 899 jednostek.




Jeśli jednak przeznaczymy na poszukiwanie odpowiednio więcej czasu i zasobów komputera możemy zbliżyć się do optimum. Poniżej prezentujemy wynik poszukiwań trwających 13 minut, o następujących parametrach:

- 10000 iteracji
- 59 sąsiedztw elitarnych
- Początkowej wielkości sąsiedztwa - 25
- Zmiany sąsiedztw po każdej iteracji
- Każdego sąsiedztwa elitarnego eksplorowanego przez 25 pszczół
- 20000 losowych rozwiązań początkowych

Algorytm znalazł drogę o długości 628 jednostek.