

# **Techniki Optymalizacji**

## **Labolatorium nr 1**

### **Sprawozdanie**

Paulina Sadowska, Rafał Araszkiewicz

8 października 2016

## **1. Wprowadzenie**

Celem ćwiczenia było zaimplementowanie algorytmów rozwiązujących problem Komiwojażera dla zbioru 100 punktów. Algorytmy te znaleźć miały najbardziej optymalną ścieżkę łączącą 50 dowolnych punktów grafu gdzie punktem startowym miał być każdy z punktów znajdujących się w zbiorze.

## **2. Nearest Neighbour**

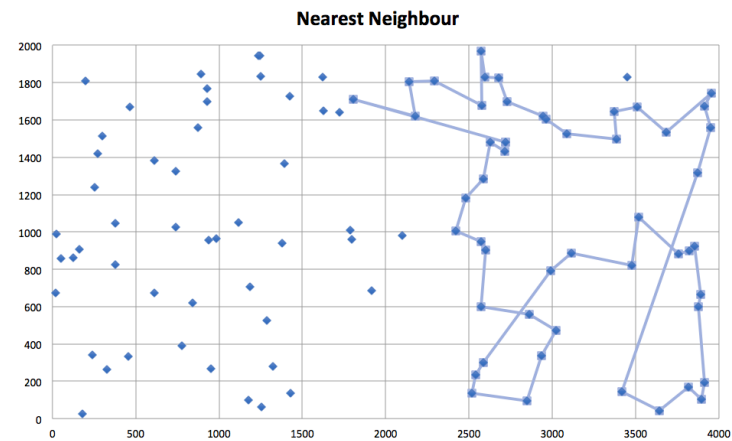
W algorytmie tym trasa budowana jest w taki sposób że w każdym punkcie szukamy trasy o najmniejszym koszcie (odległości) i punkt do którego ta trasa prowadzi dodajemy do ścieżki i dla niego szukamy kolejnego najbliższego punktu w grafie.

## 2.1. Implementacja w pseudokodzie

## 2.2. Wyniki

Tablica 1: Nearest Neighbour - wyniki

min	10298
mean	12793
max	14877
best path	80, 24, 60, 50, 86, 8, 6, 56, 19, 11, 26, 85, 34, 61, 59, 76, 22, 97, 90, 44, 31, 10, 14, 16, 73, 20, 58, 71, 9, 83, 35, 37, 23, 17, 78, 52, 87, 15, 21, 93, 69, 65, 64, 3, 96, 55, 79, 30, 88, 41, 80



Rysunek 1: Najlepsza trasa - Nearest Neighbour

### 3. Greedy Cycle

W algorytmie tym trasa jest budowana w taki sposób, aby zawsze tworzyła cykl Hamiltona. W każdej iteracji dodawany jest jeden najkrótszy łuk z pozostałych dostępnych.

#### 3.1. Implementacja w pseudokodzie

```
dla kazdego z zdefiniowanych punktow poczatkowych
    dodaj do sciezki punkt poczatkowy
    dla kazdego z punktow oprocz poczatkowego
        koszt = odleglosc pomiedzy tym punktem a poczatkowym
        jezeli koszt < dotychczasowy najmnieszy koszt
            dotychczasowy najmnieszy koszt = koszt
    koniec

    dodaj do sciezki punkt o najmnieszym koszcie (odleglosci)
    dodaj na koncu sciezki punkt poczatkowy

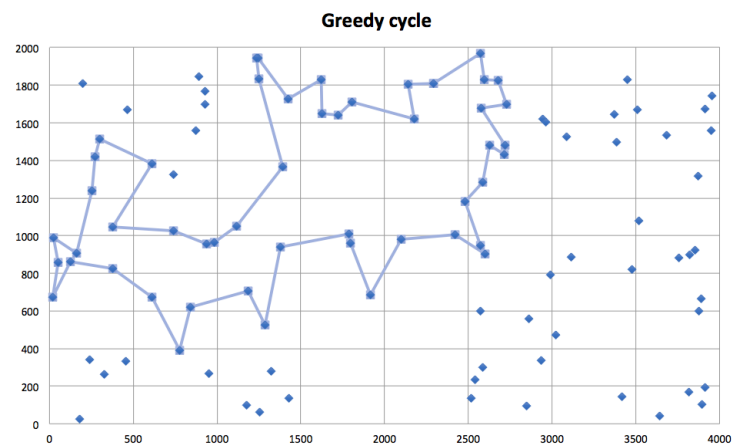
    dla pozostalych 48 krokow
        dla wszystkich par punktow w sciezce
            dla wszystkich nieodwiedzonych punktow
                koszt = odleglosc miedzy para punktow – odleglosc miedzy
                    ↪ trojka punktow
                jezeli koszt < dotychczasowy najmnieszy koszt
                    dotychczasowy najmnieszy koszt = koszt
            koniec
        koniec
        dodaj punkt powodujacy najmniesza zmiane kosztu w odpowiednie miejsce w
            ↪ sciezce
    koniec
```

koniec

### 3.2. Wyniki

Tablica 2: Greedy Cycle - wyniki

min	11095
mean	12653
max	13393
best path	61, 34, 85, 26, 11, 19, 6, 8, 56, 86, 50, 24, 80, 60, 57, 66, 27, 92, 0, 7, 91, 74, 96, 18, 52, 15, 69, 21, 93, 87, 17, 23, 37, 83, 78, 89, 48, 5, 62, 46, 10, 16, 14, 31, 44, 90, 97, 22, 76, 59, 61



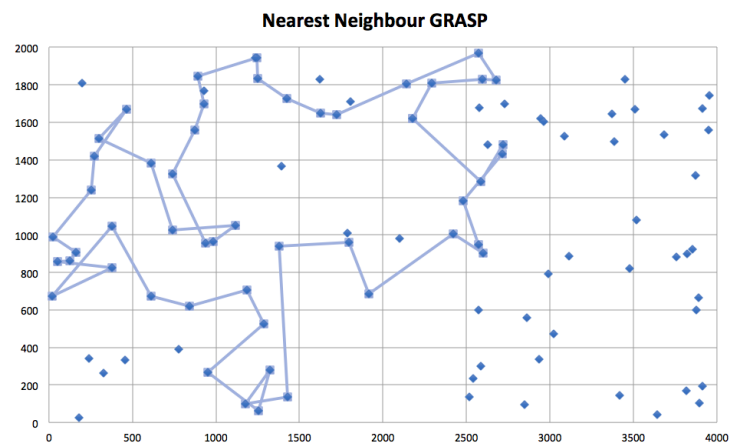
Rysunek 2: Najlepsza trasa - Greedy Cycle

## 4. Nearest Neighbour Grasp

Modyfikacja algorytmu Nearest Neighbour opisanego w punkcie 2 polegająca na tym, że w każdym kroku szukamy 3 najlepszych rozwiązań i z nich losujemy te, które będzie dodane do ścieżki.

### 4.1. Implementacja w pseudokodzie

### 4.2. Wyniki



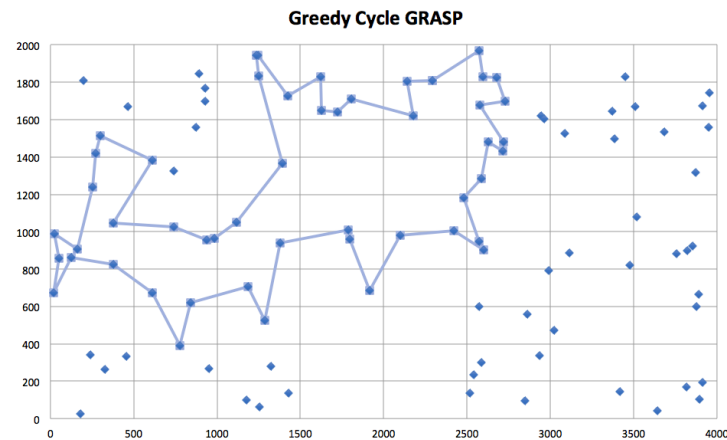
Rysunek 3: Najlepsza trasa - Nearest Neighbour Grasp

## 5. Greedy Cycle Grasp

Modyfikacja algorytmu Greedy Cycle opisanego w punkcie 3 polegająca na tym, że w każdym kroku szukamy 3 najlepszych rozwiązań i z nich losujemy te, które będzie dodane do ścieżki.

### 5.1. Implementacja w pseudokodzie

### 5.2. Wyniki



Rysunek 4: Najlepsza trasa - Greedy Cycle Grasp