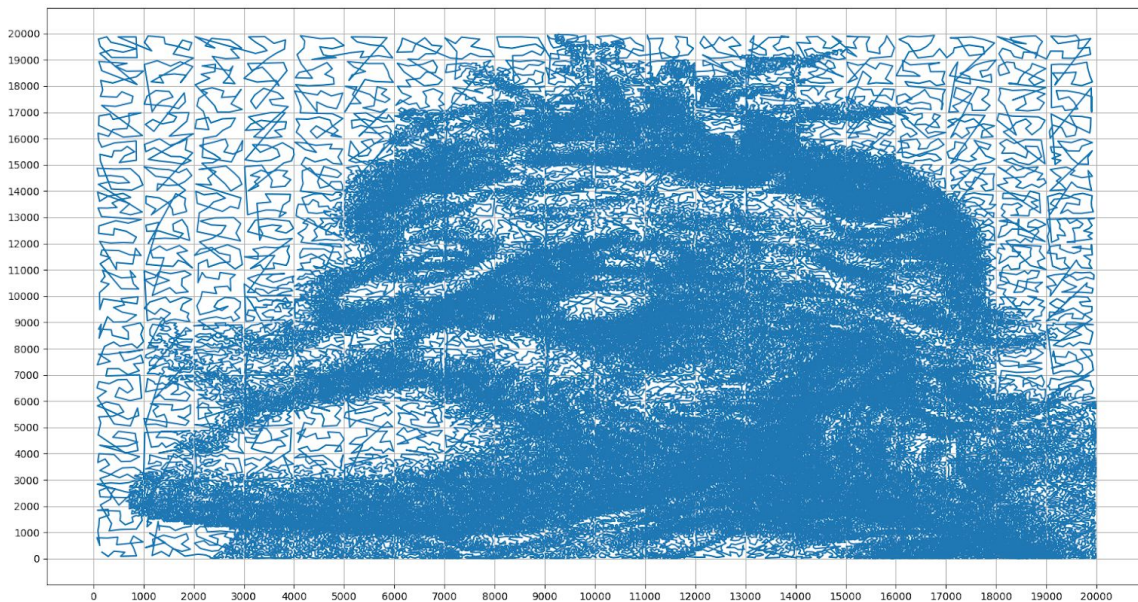


# Algorytmy heurystyczne

## Dokumentacja końcowa

Ireneusz Wróbel, Grzegorz Nowacki



### 1. Temat

Traveling santa problem

### 2. Treść zadania

Mikołaj potrzebuje pomocy w wyborze trasy po której dostarcza prezenty. Każdego roku musi odwiedzić każde dziecko z listy. Nasze zadanie nie jest dokładnie takie samo jak znany problem komiwojażera. Każdego roku ścieżka musi być inna.

Uszczegóławiając nasze zadanie polega konkretnie na znalezieniu dwóch krawędziami rozłącznych ścieżek, które są możliwie najkrótsze. Jeśli jedna ze ścieżek zawiera krawędź od A do B to druga nie może zawierać ani krawędzi od A do B ani od B do A. Za wynik naszego zadania przyjmujemy długość dłuższej ze ścieżek.

### 3. Przyjęte założenia

- ograniczenie liczby punktów, które bierzemy pod uwagę w celu skrócenia czasu obliczeń do 10% całości - punkty zostaną wylosowane ze zbioru wszystkich punktów
- punkt początkowy oraz punkt końcowy ścieżki może być losowy
- punkt końcowy nie musi być taki sam jak punkt początkowy
- znalezione ścieżki zostaną przedstawione jak w przykładowym pliku umieszczonym na stronie kaggle - plik CSV zawierający 2 kolumny odpowiadające dwóm ścieżkom a kolejne wiersze zawierają ID kolejnych odwiedzonych punktów w ścieżce

#### 4. Dane wejściowe

*santa\_cities.csv* - zawiera zbiór 150000 punktów. Każdy wiersz oznacza 1 punkt. W pierwszej kolumnie znajduje się identyfikator punktu, w dwóch kolejnych kolumnach są współrzędne x i y.

| id     | x     | y     |
|--------|-------|-------|
| 0      | 4360  | 6178  |
| 1      | 10906 | 14956 |
| 2      | 5071  | 8963  |
| 3      | 13853 | 4105  |
| 4      | 18885 | 3168  |
| ...    | ...   | ...   |
| 149999 | 9141  | 13286 |

#### 5. Zastosowane algorytmy

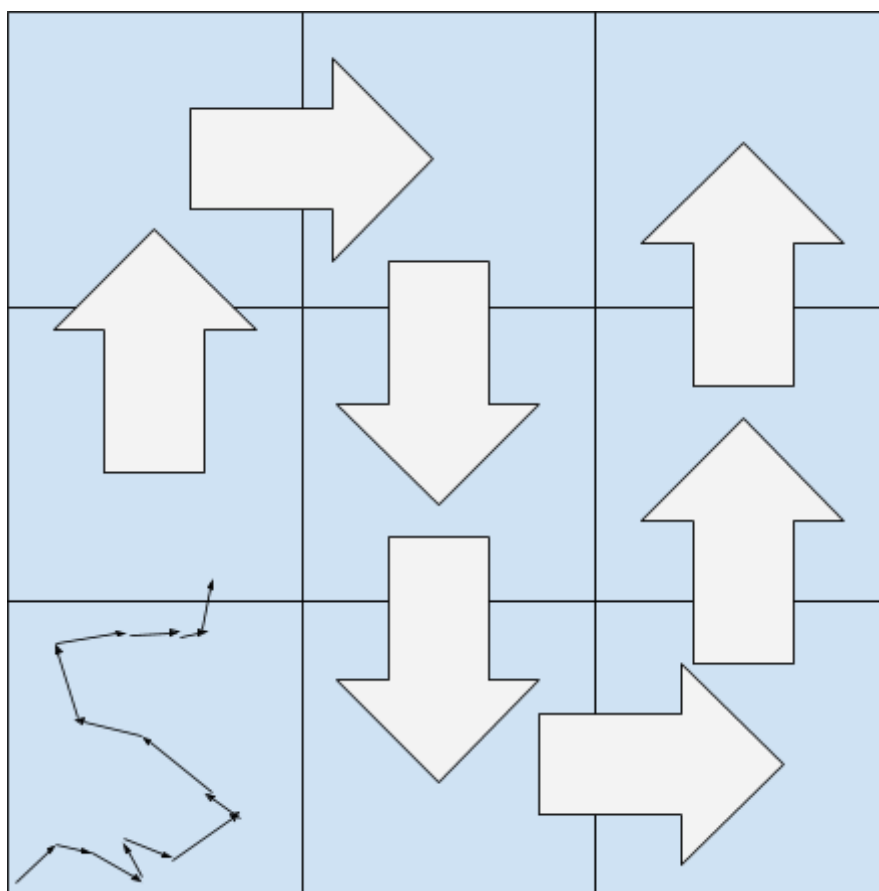
### **5.1. Algorytm najbliższego sąsiada z tabu**

- Wylosowanie punktu początkowego pierwszej ścieżki.  
Oznaczenie go jako roboczego.
- W pętli dopóki pierwsza ścieżka nie składa się ze wszystkich punktów:
  - Wybranie sąsiada punktu roboczego, który znajduje się najbliżej. Dodanie go do ścieżki pierwszej i oznaczenie jako roboczy. Dodanie do listy tabu krawędzi od poprzedniego punktu roboczego do aktualnego.
- Wylosowanie punktu początkowego drugiej ścieżki.  
Oznaczenie go jako roboczy.
- W pętli dopóki druga ścieżka nie składa się ze wszystkich punktów:
  - Wybranie sąsiada punktu roboczego, który znajduje się najbliżej oraz droga z aktualnego punktu roboczego do niego nie znajduje się w liście tabu. Dodanie tego punktu do ścieżki drugiej oraz oznaczenie jako roboczy.

### **5.2. Lokalne poprawki ścieżek wygenerowanych przez algorytm najbliższego sąsiada**

#### **5.2.1. Podział zbioru punktów na płaszczyźnie na mniejsze fragmenty**

Cały zbiór punktów na płaszczyźnie umieszczamy w kwadracie, który następnie dzielimy na mniejsze kwadraty. W każdym z nich wyznaczamy ścieżkę przy użyciu algorytmu opisanego powyżej. Jako punkty początku i końca ścieżek w każdym podkwadracie wybieramy punkty znajdujące się blisko wierzchołków. Następnie łączymy wyznaczone ścieżki poprzez te skrajne punkty.



### 5.2.2 Lokalne sprawdzanie wszystkich permutacji na określonym zbiorze punktów

Dla obu ścieżek stworzonych przez algorytm najbliższego sąsiada dzielimy wykonujemy iterację po fragmentach ścieżki zawierających  $n$  punktów. Wyszukujemy takich elementów przestrzeni przeszukiwań, które mają najmniejszą wartość funkcji celu.

Z powodu dużej liczby możliwych permutacji wybierane fragmenty ścieżki będą maksymalnie długości około 10 punktów. Punkt początkowy oraz końcowy fragmentu ścieżki nie będzie podlegał permutacji.

#### **Przestrzeń przeszukiwań**

Wszystkie możliwe permutacje  $n$  punktów z fragmentu ścieżki z wyłączeniem takich permutacji, które będą miały na

sąsiadujących pozycjach punkty, które są elementami jakiegokolwiek krawędzi z drugiej ścieżki.

### Funkcja celu

Funkcją celu jest długość ścieżki, która składa się z krawędzi pomiędzy kolejnymi sąsiadującymi punktami w permutacji czyli elemencie przestrzeni przeszukiwań.

$$f(z) = f(p_0, p_1, \dots, p_n) = \sum_{i=0}^{n-1} d(p_i, p_{i+1}) = \sum_{i=0}^{n-1} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}$$

gdzie:

$z$  - punkt w przestrzeni przeszukiwań

$p_0, p_1, \dots, p_n$  - punkty z permutacji

$n$  - liczba punktów w permutacji

$x, y$  - współrzędne punktu

## 6. Pomiary

Ze względu na czasochłonność obliczeń dla całego zbioru 150000 punktów postanowiliśmy przeprowadzić testy na ograniczonej liczbie punktów. Pomiary zostały przeprowadzone na 5%, 10% i 20% całego zbioru (tj. 7500, 15000 i 30000 punktów).

W algorytmie wykorzystującym permutacje, również z powodu czasochłonności obliczeń, ograniczyliśmy długość permutowanego fragmentu do 10.

### 7500 punktów

| Algorytm                | Długość ścieżki 1. | Długość ścieżki 2. | Czas obliczeń [s] |
|-------------------------|--------------------|--------------------|-------------------|
| Najbliższego sąsiada    | 1,291,337          | 1,775,806          | 276               |
| Podział na kwadraty (4) | 1,343,511          | 1,822,160          | 83                |
| Podział na kwadraty (9) | 1,382,976          | 1,885,921          | 41                |
| Podział na kwadraty     | 1,393,946          | 1,891,164          | 25                |

|                     |           |           |     |
|---------------------|-----------|-----------|-----|
| (16)                |           |           |     |
| Permutacje (dł. 8)  | 1,283,794 | 1,668,771 | 291 |
| Permutacje (dł. 10) | 1,276,545 | 1,624,090 | 537 |

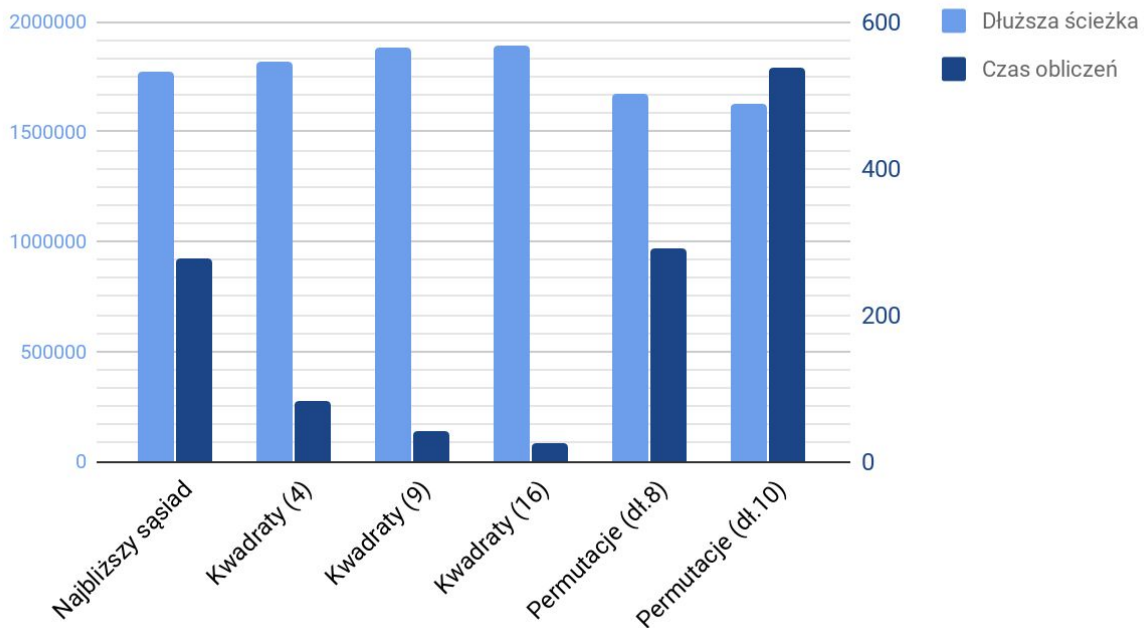
### 15000 punktów

| Algorytm                 | Długość ścieżki 1. | Długość ścieżki 2. | Czas obliczeń [s] |
|--------------------------|--------------------|--------------------|-------------------|
| Najbliższego sąsiada     | 1,837,303          | 2,524,555          | 1135              |
| Podział na kwadraty (4)  | 1,901,695          | 2,578,740          | 328               |
| Podział na kwadraty (9)  | 1,925,609          | 2,621,011          | 179               |
| Podział na kwadraty (16) | 1,955,350          | 2,684,890          | 102               |
| Permutacje (dł. 8)       | 1,836,577          | 2,371,962          | 1095              |
| Permutacje (dł. 10)      | 1,835,243          | 2,302,222          | 1594              |

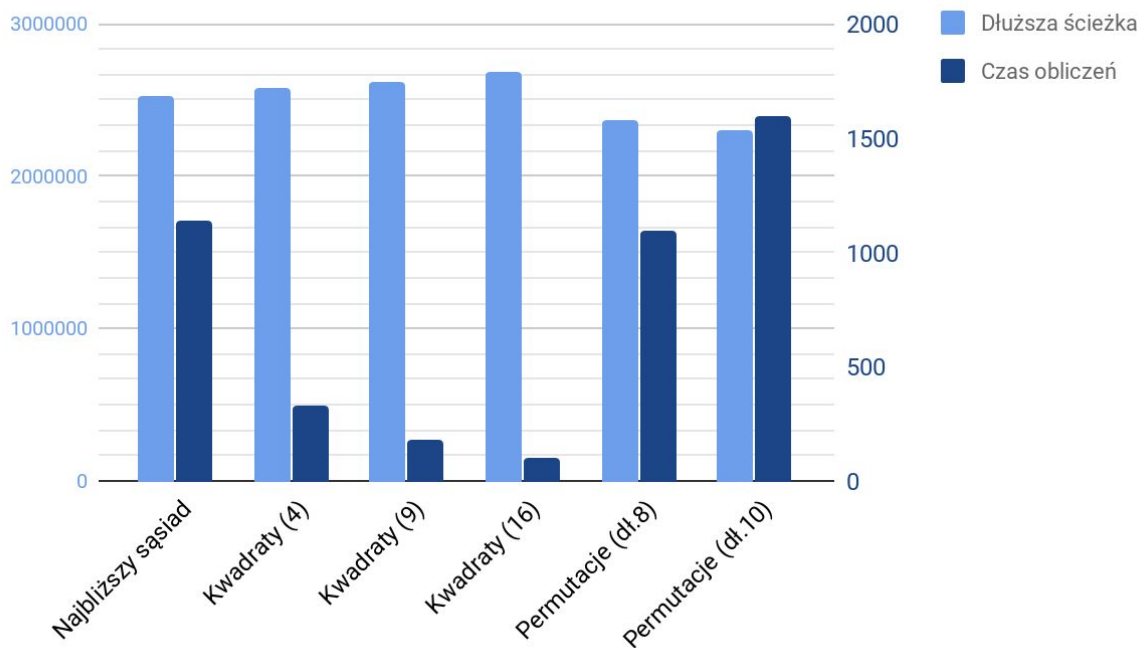
### 30000 punktów

| Algorytm                 | Długość ścieżki 1. | Długość ścieżki 2. | Czas obliczeń [s] |
|--------------------------|--------------------|--------------------|-------------------|
| Najbliższego sąsiada     | 2,683,231          | 3,574,681          | 4391              |
| Podział na kwadraty (4)  | 2,741,080          | 3,674,075          | 1318              |
| Podział na kwadraty (9)  | 2,757,650          | 3,691,133          | 694               |
| Podział na kwadraty (16) | 2,801,247          | 3,751,652          | 406               |
| Permutacje (dł. 8)       | 2,628,787          | 3,350,077          | 4367              |
| Permutacje (dł. 10)      | 2,622,313          | 3,328,352          | 6015              |

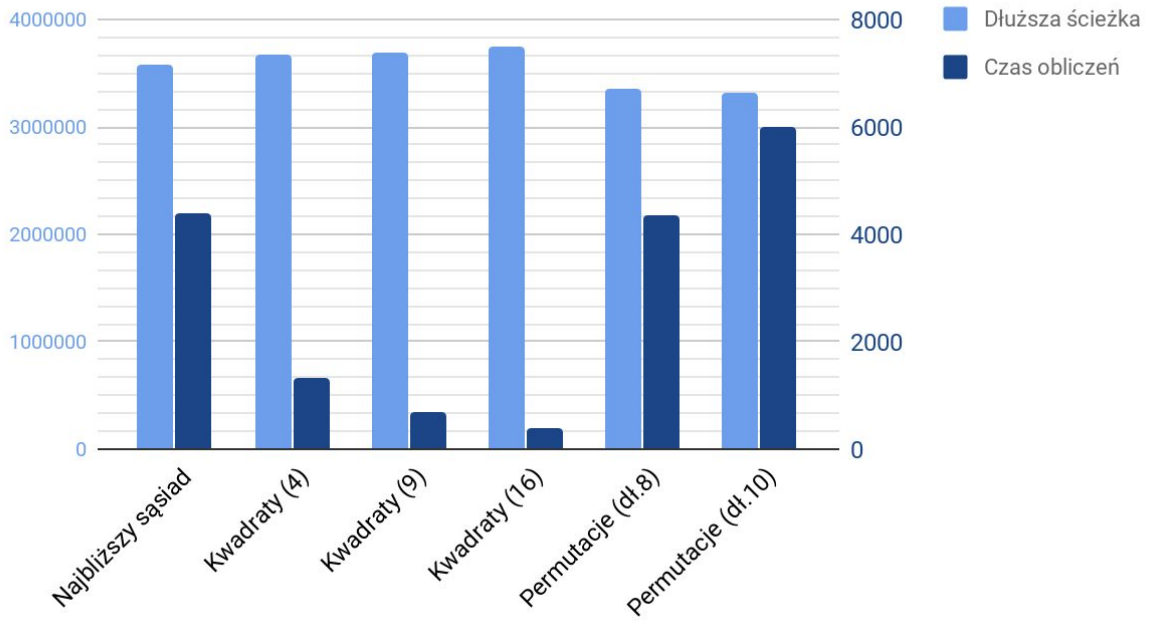
## 7500 punktów



## 15000 punktów



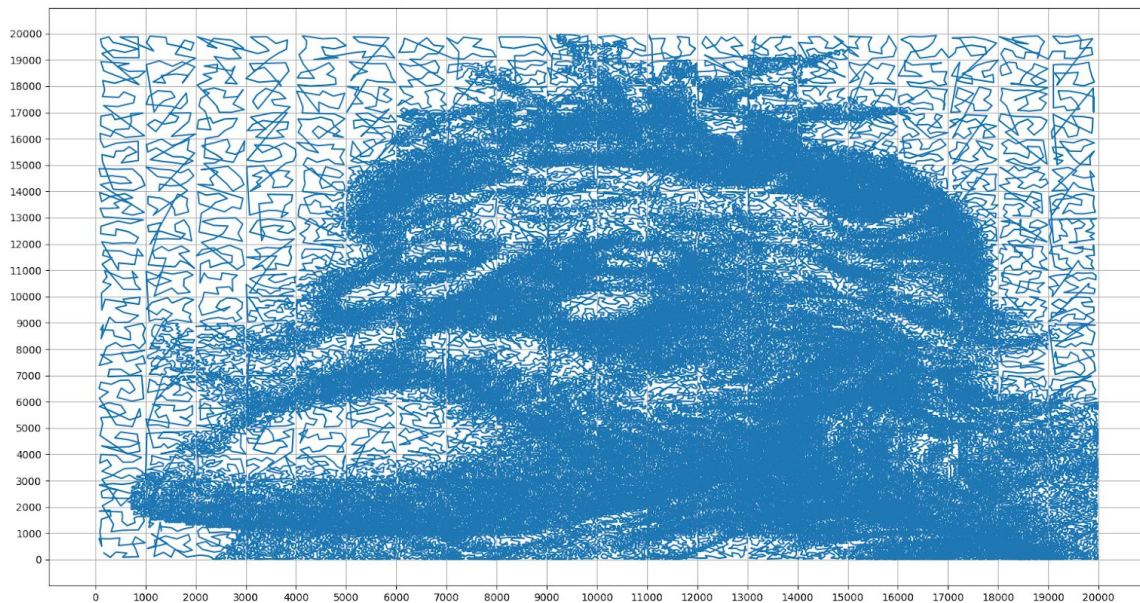
## 30000 punktów



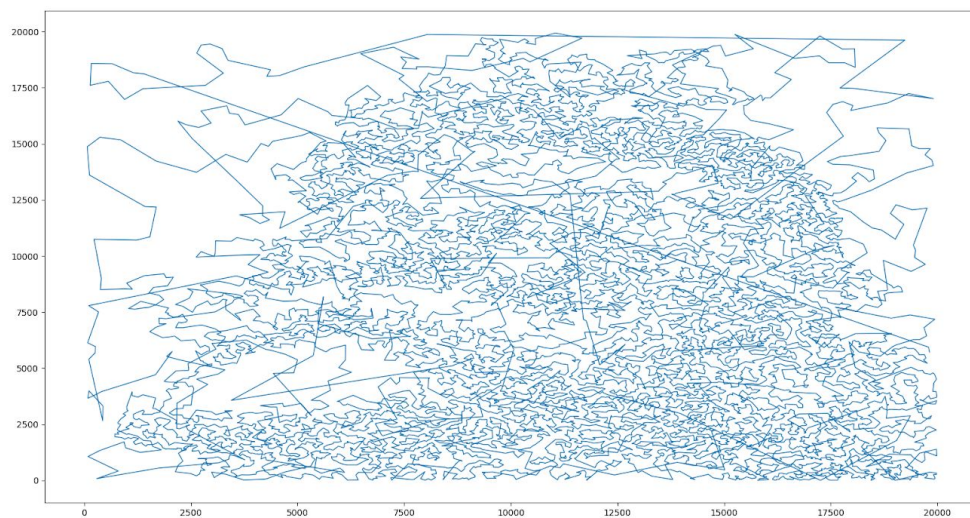


## 7. Wizualizacja ścieżek

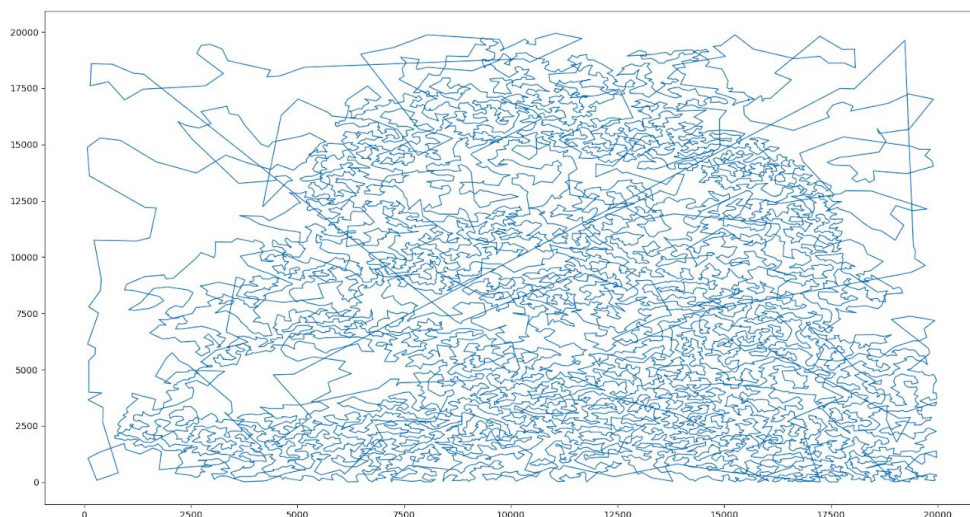
Jedna z wynikowych ścieżek algorytmu najbliższego sąsiada z podziałem na kwadraty (400) na zbiorze wszystkich 150000 punktów. Czas obliczeń to tylko około 11 minut, a długość ścieżki wyniosła około 8,500,000.



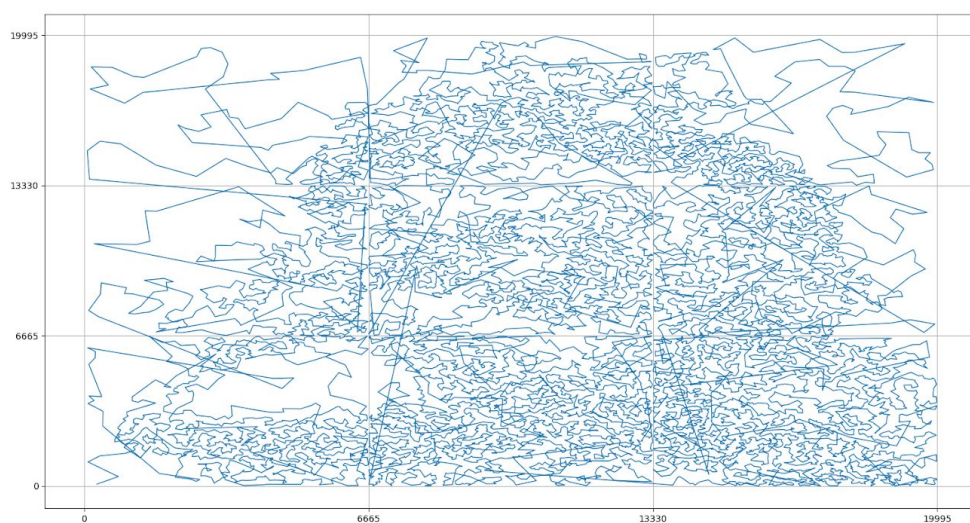
Poniżej wynik podstawowego algorytmu najbliższego sąsiada wykonany dla 15000 punktów.



Poprawki ścieżek o długości 15000 poprzez permutacje o długości 10.



Podział przestrzeni na 9 mniejszych obszarów.



## 8. Analiza wyników

Najlepsze wyniki zostały osiągnięte przy wykorzystaniu algorytmu najbliższego sąsiada z późniejszym, lokalnym poprawianiem ścieżek poprzez permutacje. Przy czym im dłuższe były fragmenty ścieżek brane w okno permutacji, tym wynikowa długość całej ścieżki była mniejsza. Wynika to z faktu, że biorąc pod uwagę dłuższe fragmenty ścieżek algorytm jest w stanie naprawić coraz większe lokalne nieprawidłowości w przeprowadzonych ścieżkach. Taka nieprawidłowość to na przykład łączenie dwóch punktów w takich sposób, że krawędź przecina wcześniejsze fragmenty ścieżki - taka sytuacja w większości przypadków wydłuża końcową długość całej drogi.

Wraz z permutowaniem dłuższych fragmentów zwiększa się diametralnie czas obliczeń. Jest to spowodowane dużą przestrzenią przeszukiwań.

Jeżeli chcemy osiągnąć jak najkrótsze ścieżki wynikowe i nie ogranicza nas czas obliczeń to spośród przedstawionych algorytmów najlepszym wyborem jest algorytm najbliższego sąsiada z późniejszym, lokalnym poprawianiem ścieżek poprzez permutacje.

Jeżeli chcemy ograniczyć czas obliczenia wynikowych ścieżek to lepszym wyborem jest podstawowy algorytm najbliższego sąsiada lub wersja tego algorytmu z podziałem na podzadania (mniejsze obszary).

Podział punktów na mniejsze obszary pozwala na skrócenie czasu potrzebnego do wyszukiwania najbliższego sąsiada, ponieważ algorytm operuje wtedy na mniejszych zbiorach danych. Wiąże się to niestety z nieco gorszymi długościami wynikowych ścieżek, ponieważ połączenie pomiędzy ostatnim punktem ścieżki z jednego obszaru, a pierwszym punktem ścieżki z kolejnego obszaru, mogą tworzyć dodatkowy narzut w długości ostatecznej drogi. Narzut ten zwiększa się wraz ze wzrostem ilości podobszarów na które dzielimy cały wejściowy obszar punktów.

Podstawowy algorytm najbliższego sąsiada nie ma natomiast tego narzutu, ale w związku z operowaniem na zbiorze wszystkich punktów, czas wykonania jest dłuższy. Zaś w końcowym etapie przebiegu

algorytmu tworzy on krawędzie, które są bardzo długie. Jest to spowodowane tym, że na końcu zostaje stosunkowo mało punktów, ale rozproszonych po całym obszarze. Podobny problem, ale o innym natężeniu występuje w pozostałych algorytmach.

## 9. Implementacja

Projekt został podzielony na 6 modułów.

- **algorithms** - zawierający implementację głównych algorytmów wykorzystywanych w obliczeniach
  - **nearest\_neighbour\_alg\_for\_first\_path**
  - **nearest\_neighbour\_alg\_for\_second\_path**
  - **permutations\_fix\_for\_first\_path**
  - **permutations\_fix\_for\_second\_path**
  - **square\_fix\_for\_first\_path**
  - **square\_fix\_for\_second\_path**
- **algorithms\_utils** - zawierający pomocnicze funkcje wykorzystywane przez główne algorytmy
- **utils** - zawierający pomocnicze funkcje, które nie są powiązane z algorytmami takie jak wczytywanie danych itp
- Moduły odpowiedzialne za wywołanie algorytmów, stworzenie wyjściowych zbiorów, przedstawienie wyników i pomiary czasów
  - **main\_nearest\_neighbour**
  - **main\_permutations**
  - **main\_squares**
- **config** - moduł zawierający zmienne konfiguracyjne