Obliczenia inżynierskie w chmurze – projekt

Rozwiązanie problemu komiwojażera z wykorzystaniem algorytmu najbliższego sąsiada

Marek Sadło – 304479

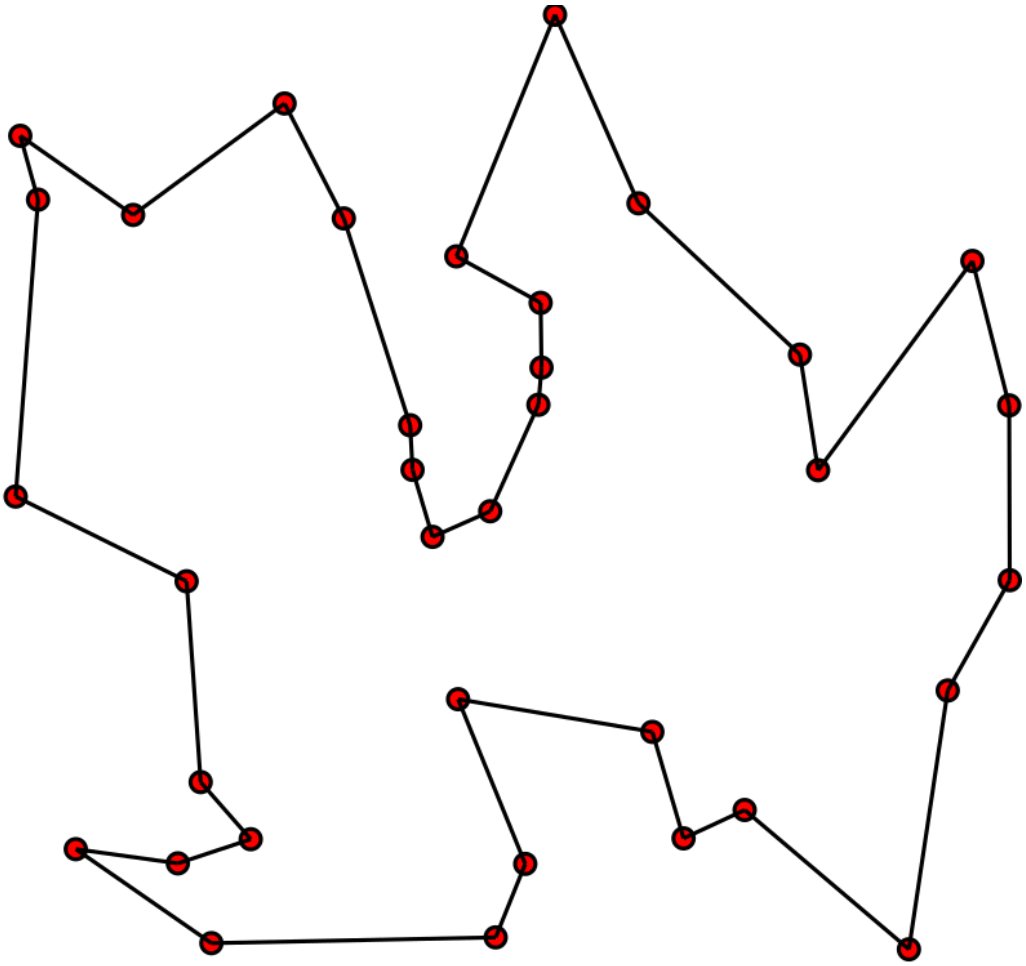
Prowadzący: dr inż. Mateusz Żbikowski

1. Cel projektu

Celem projektu było przetestowanie działania algorytmu najbliższego sąsiada do rozwiązywania problemu komiwojażera. Równie istotnym elementem było wykorzystanie narzędzi chmury Microsoft Azure do przeprowadzenia niezbędnych obliczeń. W eksperymencie zbadano też czas wykonywania obliczeń w porównaniu do zwykłego komputera osobistego.

2. Opis problemu

Problem komiwojażera (ang. *Traveling Salesman Problem*, TSP) to klasyczny problem optymalizacji w matematyce i informatyce. Polega na znalezieniu najkrótszej możliwej trasy, która pozwoli komiwojażerowi (wędrownemu sprzedawcy) odwiedzić każde z określonych miast dokładnie jeden raz i wrócić do miasta początkowego. Jest to znany i dość wiekowy problem i ma zastosowanie w logistyce, planowaniu tras dostaw, optymalizacji produkcji i w wielu innych dziedzinach wymagających minimalizacji kosztów podróży lub transportu.



Rys. 1 - Ilustracja problemu komiwojażera

Jest to problem NP-trudny, co oznacza, że dla dużej liczby miast rozwiązanie optymalne wymaga przeszukiwania bardzo dużej liczby możliwości (liczba możliwych tras rośnie wykładniczo).

3. Algorytm najbliższego sąsiada

W praktyce do rozwiązywania problemu komiwojażera stosuje się zarówno algorytmy dokładne (np. algorytm podziału i ograniczeń, programowanie dynamiczne) jak i heurystyczne/metaheurystyczne (np. algorytmy genetyczne, symulowane wyżarzanie, algorytmy mrówkowe). Problem ten wciąż jest nierozwiązany, to znaczy, że do tej pory nie znaleziono szybkiego algorytmu, który by zawsze znajdował optymalną trasę.

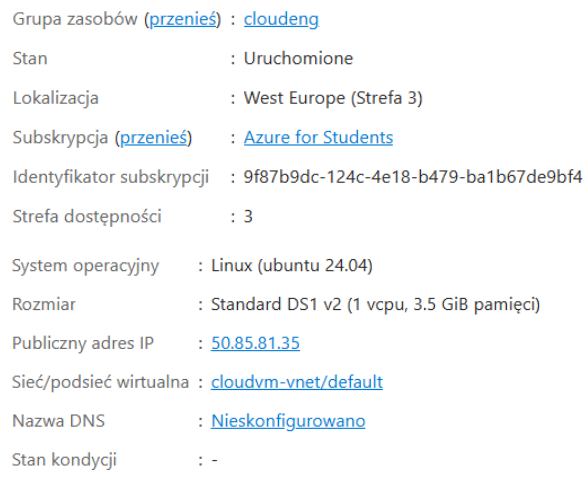
Jednym z prostych i szybkich algorytmów heurystycznych jest algorytm najbliższego sąsiada. Choć nie gwarantuje znalezienia optymalnego rozwiązania, często daje przyzwoite wyniki i jest wykorzystywany w sytuacjach, gdzie czas obliczeń jest kluczowy. Działanie algorytmu jest następujące:

1) Wybierz miasto początkowe jako bieżące miasto.
2) Spośród wszystkich nieodwiedzonych miast wybierz to, które jest najbliżej bieżącego miasta (czyli ma najmniejszy koszt/dystans).
3) Przejdź do wybranego miasta i oznacz je jako odwiedzone.
4) Powtarzaj kroki 2 i 3, aż wszystkie miasta zostaną odwiedzone.
5) Po odwiedzeniu wszystkich miast wróć do miasta początkowego, zamykając trasę.

Warto zaznaczyć, iż rzadko kiedy trasa wyznaczona w ten sposób jest najkrótsza. Algorytm działa dobrze dla nieskomplikowanych zadań, bardziej złożone problemy powinny być rozwiązywane lepszymi narzędziami, jak na przykład algorytmy genetyczne.

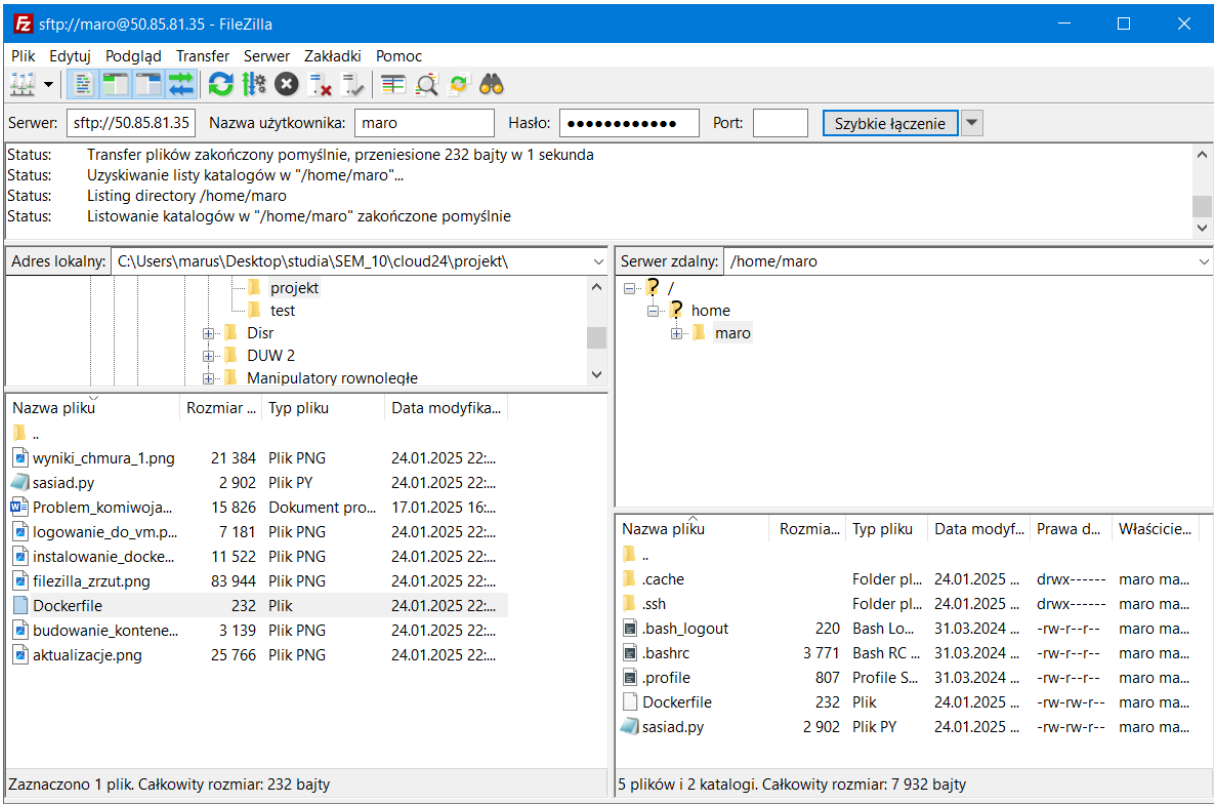
4. Przygotowanie do obliczeń

W celu wykonania projektu napisano program *sasiad.py*, w którym zaimplementowano algorytm najbliższego sąsiada do rozwiązania problemu komiwojażera. Przygotowano również plik *Dockerfile*, aby określić zasady budowy kontenera na wirtualnej maszynie. Następnie uruchomiono wirtualną maszynę na platformie Microsoft Azure korzystając z dostępnych środków w ramach subskrypcji studenckiej. Zdecydowano wybrać prostą maszynę z zainstalowanym Linuxem, 1 CPU i 3.5 GB pamięci, ponieważ było to jedno z najtańszych rozwiązań.



Rys. 2 - Parametry witualnej maszyny

Po uruchomieniu wirtualnej maszyny przeniesiono na jej dysk pliki *sasiad.py* oraz *Dockerfile* za pomocą programu Filezilla. Na Rys.3 po prawej stronie widać foldery i pliki znajdujące się na zdalnie połączonej maszynie.



Rys. 3 - Widok z przenoszenia plików w programie Filezilla

Następnie konieczne było zalogowanie się do wirtualnej maszyny z pomocą wiersza poleceń Windows i protokołu SSH, wykorzystując adres IP oraz nazwę użytkownika i hasło ustalone podczas jej konfiguracji.

```
C:\Users\marus>ssh maro@50.85.81.35
The authenticity of host '50.85.81.35 (50.85.81.35)' can't be established.
ED25519 key fingerprint is SHA256:J6Z/oF5MLyrtRZqWyAfwZlUY3emAB7EqEfjuIOU6IpQ.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '50.85.81.35' (ED25519) to the list of known hosts.
maro@50.85.81.35's password:
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-1020-azure x86_64)
```

Rys. 4 - Logowanie do wirtualnej maszyny

Kolejnym krokiem była aktualizacja wszystkich niezbędnych pakietów.

```
maro@cloudvm:~$ sudo apt-get update
```

Rys. 5 - Aktualizacja pakietów

Po dokonaniu aktualizacji zainstalowano aplikację Docker używając odpowiedniej komendy.

```
maro@cloudvm:~$ sudo apt install docker.io
```

Rys. 6 - Instalowanie Dockera

Gdy instalacja Dockera dobiegła końca zbudowano obraz o nazwie python39 na podstawie pliku *Dockerfile*, który został wcześniej odpowiednio napisany tak, aby w naszym kontenerze były dostępne wszystkie potrzebne narzędzia, w tym również biblioteka *matplotlib*.

```
maro@cloudvm:~$ sudo docker build -t python39 .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  15.36kB
```

Rys. 7 - Budowanie obrazu kontenera

Po zbudowaniu wystarczyło już tylko uruchomić kontener, który po chwili automatycznie uruchamiał program *sasiad.py* i przeprowadzał obliczenia.

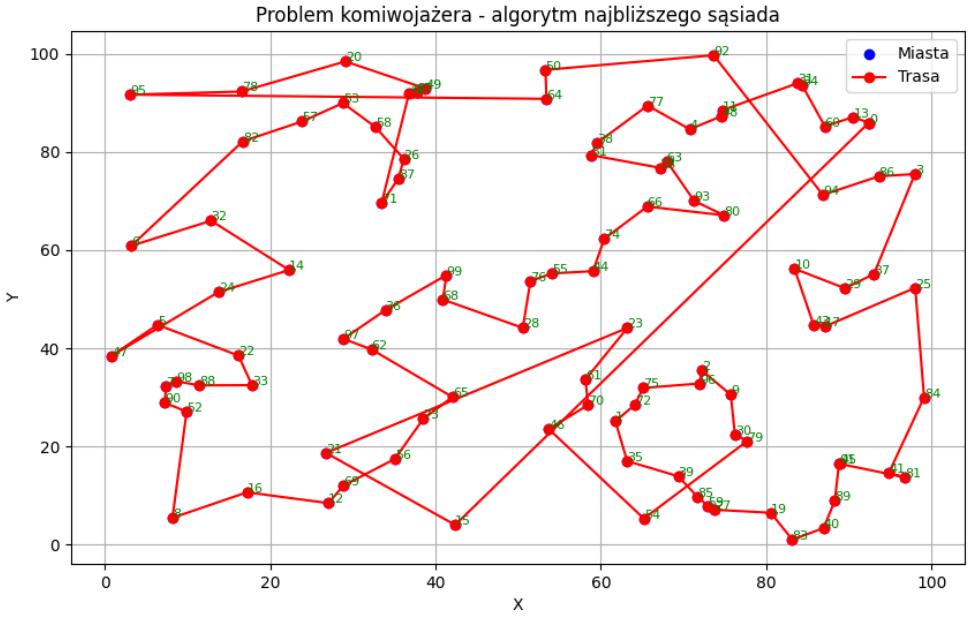
```
maro@cloudvm:~$ sudo docker run --name python39 -it python39
```

Rys. 8 - Uruchomienie kontenera

Następnie dokonano kilku testowych uruchomień programu oraz 3 które posłużyły do weryfikacji działania. Po skończeniu pracy z kontenerem został on usunięty z pomocą komendy *sudo docker rm*, a w przeglądarce zakończono działanie maszyny wirtualnej, upewniając się że wszystkie zasoby zostały poprawnie usunięte.

5. Wyniki obliczeń

Dokonano łącznie 6 testów działania programu – 3 na wirtualnej maszynie w chmurze oraz 3 na osobistym komputerze do porównania. Pierwszy test na każdej z maszyn polegał na wyznaczeniu trasy między wylosowanymi 100 miastami, drugi obejmował 1000 miast a trzeci 5000. Wyniki liczbowe oraz graficzne przedstawiono poniżej.



Rys. 9 - Wykres znalezionej trasy - 100 miast

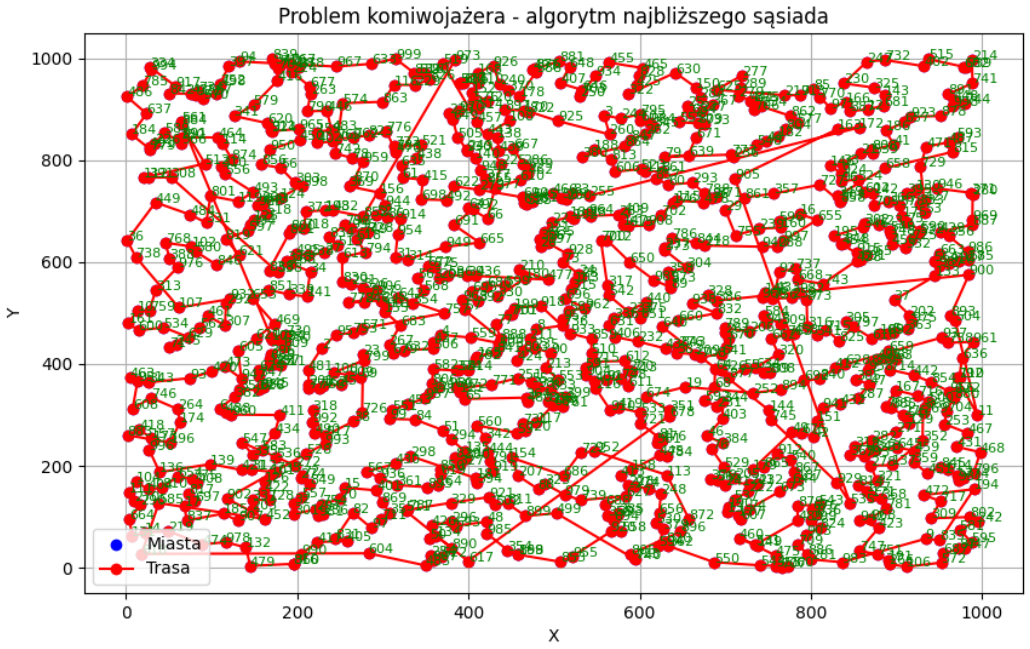
```
maro@cloudvm:~$ sudo docker run --name python39 -it python39
Optymalna kolejnosc odwiedzania miast: [0, 49, 89, 78, 58, 37, 81, 80, 13, 33, 65, 77, 31, 29, 69, 18, 51, 4, 98, 21, 88
, 9, 86, 64, 84, 72, 7, 52, 85, 42, 91, 46, 90, 27, 12, 26, 20, 38, 53, 10, 23, 11, 87, 2, 39, 14, 34, 95, 76, 8, 94, 63
, 43, 32, 75, 79, 17, 99, 1, 92, 15, 41, 25, 62, 45, 97, 66, 22, 30, 56, 67, 57, 71, 93, 55, 47, 68, 3, 60, 28, 82, 16,
5, 24, 70, 73, 61, 6, 50, 36, 96, 83, 40, 48, 44, 54, 19, 59, 35, 74, 0]
Calkowita droga: 905.2075356728081
Czas obliczen: 0.0057 sekund
Wykres zapisano w pliku 'wykres_drogi.png'.
```

Rys. 10 - Wyniki obliczeń 100 miast – chmura

```
C:\Users\marus\Desktop\studia\SEM_10\cloud24\projekt>python sasiad.py
Optymalna kolejnosc odwiedzania miast: [0, 13, 60, 34, 31, 11, 48, 4, 77, 38, 51, 18, 63, 93, 80, 66, 74, 44, 55, 76, 28
, 68, 99, 36, 97, 62, 65, 73, 56, 69, 12, 16, 8, 52, 90, 7, 98, 88, 33, 22, 5, 47, 24, 14, 32, 6, 82, 57, 53, 58, 26, 87
, 71, 42, 67, 49, 20, 78, 95, 64, 50, 92, 94, 86, 3, 37, 29, 10, 43, 17, 25, 84, 41, 81, 45, 91, 89, 40, 83, 19, 27, 59,
 85, 39, 35, 1, 72, 75, 96, 2, 9, 30, 79, 54, 46, 70, 61, 23, 21, 15, 0]
Calkowita droga: 1005.0494520837221
Czas obliczen: 0.0050 sekund
Wykres zapisano w pliku 'wykres_drogi.png'.
```

Rys. 11 - Wyniki obliczeń 100 miast – PC

Łatwo zauważyć, że czas obliczeń na obu jednostkach jest porównywalny, zobaczymy jak będzie to wyglądać dla większej liczby miast.



Rys. 12 - Wykres znalezionej trasy - 1000 miast

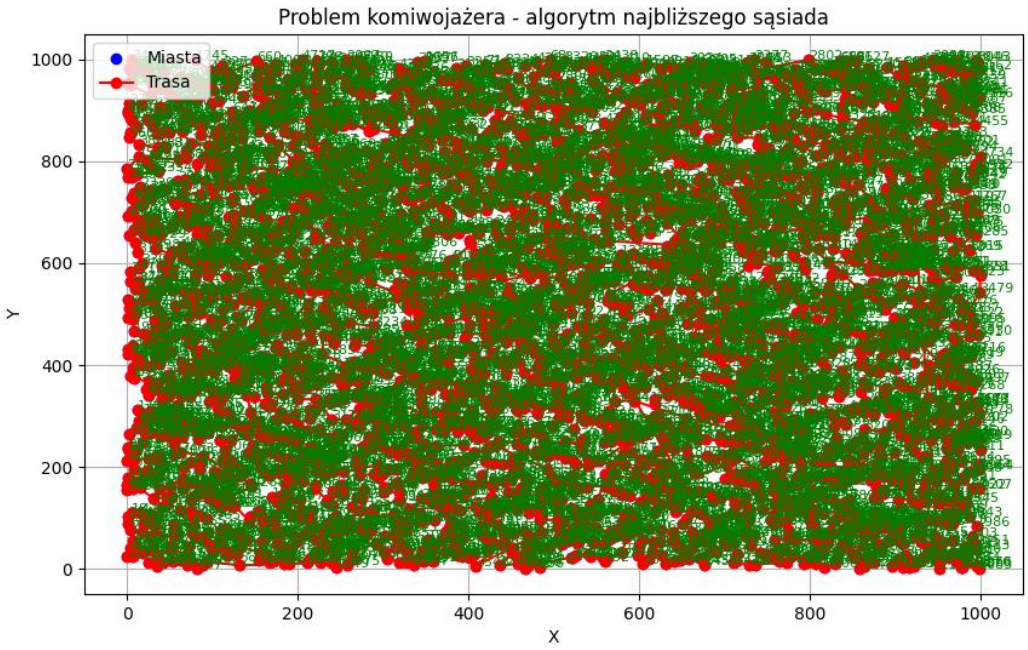
```
Calkowita droga: 29085.474475791136
Czas obliczen: 0.5913 sekund
Wykres zapisano w pliku 'wykres_drogi.png'.
```

Rys. 13 - Wyniki obliczeń 1000 miast – chmura

```
Calkowita droga: 29117.75363692491
Czas obliczen: 0.5619 sekund
Wykres zapisano w pliku 'wykres_drogi.png'.
```

Rys. 14 - Wyniki obliczeń 1000 miast – PC

Dla 1000 miast ponownie PC okazał się minimalnie szybszy niż chmura.



Rys. 15 - Wykres znalezionej trasy - 5000 miast

Przy tak dużej liczbie miast jest już niemożliwe by prześledzić znalezioną trasę, można oczywiście przybliżyć widok, ale analizowanie całej trasy zajęło by setki godzin.

```
6, 4178, 518, 4948, 1170, 165, 1158, 4912, 994, 1355, 2396, 3376, 3177, 9, 3222, 740, 3117, 3766, 2950, 195, 2016, 4
23, 25, 673, 4986, 3812, 3634, 0]
Calkowita droga: 63395.6340811301
Czas obliczen: 15.0731 sekund
/home/neighbor/sasiad.py:73: UserWarning: Creating legend with loc="best" can be slow with large amounts of data.
  plt.savefig(filename)
Wykres zapisano w pliku 'wykres_drogi.png'.
```

Rys. 16 - Wyniki obliczeń 5000 miast – chmura

```
5, 4189, 4025, 4544, 943, 4018, 1170, 4066, 4083, 2413, 2451, 403, 1986, 1843, 3422, 1607, 2918, 2553, 2432, 814, 3755,
1556, 4322, 3858, 3009, 1764, 3527, 419, 1155, 2810, 1305, 3008, 4545, 2954, 213, 2801, 3486, 2235, 1355, 2675, 4030, 17
67, 1107, 0]
Calkowita droga: 62469.04924773204
Czas obliczen: 14.0728 sekund
Wykres zapisano w pliku 'wykres_drogi.png'.
```

Rys. 17 - Wyniki obliczeń 5000 miast – PC

Również i tym razem różnice w czasie wykonywania obliczeń są niewielkie i wciąż na korzyść komputera osobistego mimo że trwało to już koło 15 sekund. Można by się zastanowić czy w takim razie chmura jest do czegoś potrzebna. Przede wszystkim nie obciążamy w ten sposób swojego komputera osobistego i możemy robić na nim inne rzeczy podczas gdy w chmurze przeprowadzane są obliczenia. Poza tym wybrana maszyna wirtualna była jedną z najtańszych i najsłabszych dostępnych maszyn, a wiele z nich jest znacznie wydajniejsza niż przeciętny komputer zwykłego człowieka.

6. Podsumowanie

Można wysnuć wnioski, iż metoda najbliższego sąsiada całkiem szybko rozwiązuje problem komiwojażera nawet dla dużej liczby miast. Niestety trasa przez nią wyznaczona nie jest najlepsza, choć z pewnością zazwyczaj jest jedną z lepszych opcji. Ten problem wciąż jest debatowany wśród naukowców i ciągle powstają nowe propozycje jego rozwiązania. Większość dobrych algorytmów wymaga sporej mocy obliczeniowej, którą można by pozyskać z przetestowanej maszyny wirtualnej na platformie Microsoft Azure, wybierając droższe, ale bardziej wydajne jednostki obliczeniowe.