Raport
Wprowadzenie do Sztucznej Inteligencji - Ćwiczenie 2.

Algorytmy genetyczne i ewolucyjne

Autor: Aleksandra Jamróz, nr albumu: 310 708

Treść zadania

Pan Mateusz kupił szczepionki na Kolanowirusa dla mieszkańców Wolski i musi objechał n=30 wielkich miast (wierzchołki w grafie) tego wspaniałego kraju, by dostarczyć je dla wszystkich. Niestety budżet pana Mateusza jest ograniczony i musi rozliczyć się z kosztów za paliwo z własnej kieszeni. Dlatego też chciałby on przejechać przez wszystkie miasta dokładnie raz jak najkrótszą / najszybszą drogą zaczynając i kończąc w tym samym mieście X (szukamy najkrótszego cyklu). Wskaż sekwencję tych miast (travelling salesman problem). Sugerowane rozkłady miast: jednorodny (taka szachownica), duże skupiska grup oraz losowy

Pseudokod:

```
P_t = init()
t = 0
ocena(P_t)
while !stop
   Tt = selekcja(Pt)
   Ot = mutacja(Tt)
   ocena(Ot)
   Pt = Ot
   t=t+1
```

Biblioteki

Rozwiązanie rozpoczęłam od zaimportowania odpowiednich bibliotek. Wykorzystałam:

- Math – do liczenia pierwiastków potrzebnych do obliczenia odległości między miastami;
- Matplotlib – efekt końcowy działania programu zwizualizowany jest na wykresie stworzonym za pomocą tej biblioteki;
- Random – do generowania listy miast, losowania pozycji z listy i mieszania zawartości listy miast;
- Timeit – mierzenia czasu wykonania programu.

Generowanie listy miast

Następnie zgodnie ze wskazówką, stworzyłam 3 funkcje generujące współrzędne miast:

1. Generowanie losowe – przyjmuje parametr określający liczbę miast. Polega na wylosowaniu parami różnych współrzędnych miast, przy czym x i y domyślnie zawierają się w zakresie od 0 do 100.
2. Generowanie na szachownicy – współrzędne miast układają się na siatce, której szerokość oczek przyjmowana jest wraz z liczbą miast jako parametr.

3. Generowanie grupowe – w tym przypadku użytkownik podaje funkcji 3 parametry. Liczba grup jednoznaczna jest z liczbą wylosowanych na początku miast głównych, tym samym aglomeracji. Liczebność każdej z tych grup wyznacza liczbę sąsiadów każdego z miast głównych. Promień sąsiedztwa określa odległość, w jakiej maksymalnie ulokowane mogą zostać pozostałe miasta dookoła miasta głównego.

Odległość euklidesowa między dwoma miastami liczona jest za pomocą biblioteki math. Wzór wynika bezpośrednio z twierdzenia Pitagorasa. Sumaryczną odległość między kolejnymi miastami na ścieżce liczy funkcja oceny – im mniejsza odległość, tym lepszy jest dany osobnik.

Algorytm ewolucyjny

Składa się on z: generowanie populacji, selekcji turniejowej, mutacji, sukcesji.

I. Generowanie populacji
   Funkcja generująca populację przyjmuje 2 parametry – listę miast, z której ma korzystać, oraz liczbę osobników do wygenerowania. Tworzy na początku pustą listę osobników, produkując następnie za pomocą funkcji *choice* z biblioteki random permutacje zbioru wszystkich miast. Dodaje je do listy populacji pod warunkiem, że żaden osobnik się nie powtarza. Jeżeli wygeneruje osobnika obecnego już na liście, nie dodaje go, ale losuje innego. Pętla działa do momentu wypełnienia listy populacji odpowiednią liczbą osobników.

II. Selekcja turniejowa
   Rozgrywanych jest tyle samo turniejów, ile jest osobników w populacji. Do każdego turnieju z populacji losowanych jest 2 zawodników, z założeniem, że losujemy ze zwracaniem, więc ten sam zawodnik może zostać wylosowany dwukrotnie i konkurować sam ze sobą. Wygrywa osobnik, dla którego funkcja oceny, w tym przypadku funkcja wyznaczająca sumaryczną odległość między miastami, przyjmuje mniejszą wartość. Kolejne zwycięskie osobniki trafiają do początkowo pustej listy nowej populacji.

III. Mutacja
   Mutacja jednego osobnika polega na wylosowaniu dwóch pozycji w liście z pomocą funkcji *randint* i zamianie współrzędnych miast na tych pozycjach miejscami. Mutowany jest określony procent populacji, współczynnik jest przyjmowany przez funkcję mutującą jako parametr. Może zawierać się w przedziale od 0 do 1, przy czym 0 oznacza, że żaden osobnik nigdy nie będzie zmutowany, a 1 – że każdy z osobników będzie mutowany przy każdym wykonaniu pętli programu.

IV. Sukcesja następuje już w zbiorczej funkcji *salesman_probleem.* Zawiera ona zmienną global_winner przechowującą najlepszą do tej pory znalezioną ścieżkę. Aktualizowana jest w momencie znalezienia lepszej drogi. Zawarta jest w niej pętla składająca się z w.w. punktów. Przyjmuje następujące parametry: wcześniej wygenerowana lista współrzędnych miast, liczebność populacji, liczba iteracji oraz współczynnik mutacji. Zmienna global_winner przechowuje osobnika, dla którego funkcja oceny przyjmowała najmniejszą wartość ze wszystkich znalezionych do tej pory. Jej wartość aktualizowana jest przy każdym wykonaniu pętli. Do wyznaczenia najlepszego osobnika spośród obecnej populacji służy dodatkowa funkcja

turnee_winner_selection. Pod koniec pętli nowo-wytworzona populacja staje się populacją bazową – zachodzi element sukcesji.

Wizualizacja

Efekt końcowy działania programu jest wizualizowany na wykresie. W tytule wykresu dla lepszej czytelności wypisują się od razu przyjęte wartości parametrów oraz czas wykonania programu. Z najlepszej znalezionej listy miast pobierane są kolejno pary punktów i łączone linią, dzięki czemu uzyskujemy obraz ścieżki.

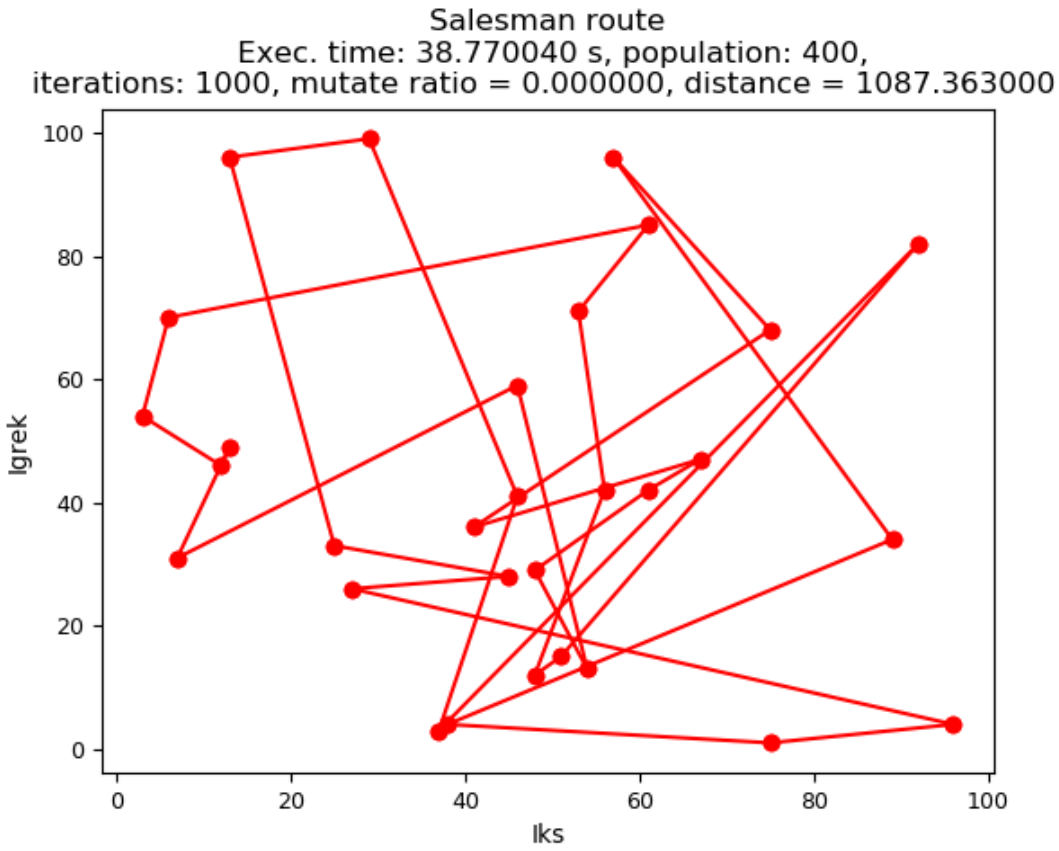
Przeprowadzanie prób

Dla każdego rodzaju ułożenia miast przeprowadziłam serię prób. Dla miast ułożonych losowo oraz na siatce liczba miast wynosiła 5, 10 lub 30, natomiast grupowo – 6, 12 oraz 30. Na początku generowałam listę miast zgodnie z typem i pozostawała ona niezmienna dla podanego rozkładu oraz liczby miast. Podczas każdej próby różniły się podane parametry: liczba populacji i iteracji oraz współczynnik mutacji. Wyniki przeprowadzonych prób umieściłam w tabelach w arkuszu kalkulacyjnym – Załącznik do raportu. Dla każdej próby program liczył czas wykonania, sumaryczny dystans między miastami oraz zapisywał wykres do pliku. Aby raport pozostał czytelny, umieszczam poniżej tylko najlepsze wyniki działania programu dla wszystkich prób. Parametry, dla których wykonała się funkcja, zapisane są w tytule każdego z wykresów.

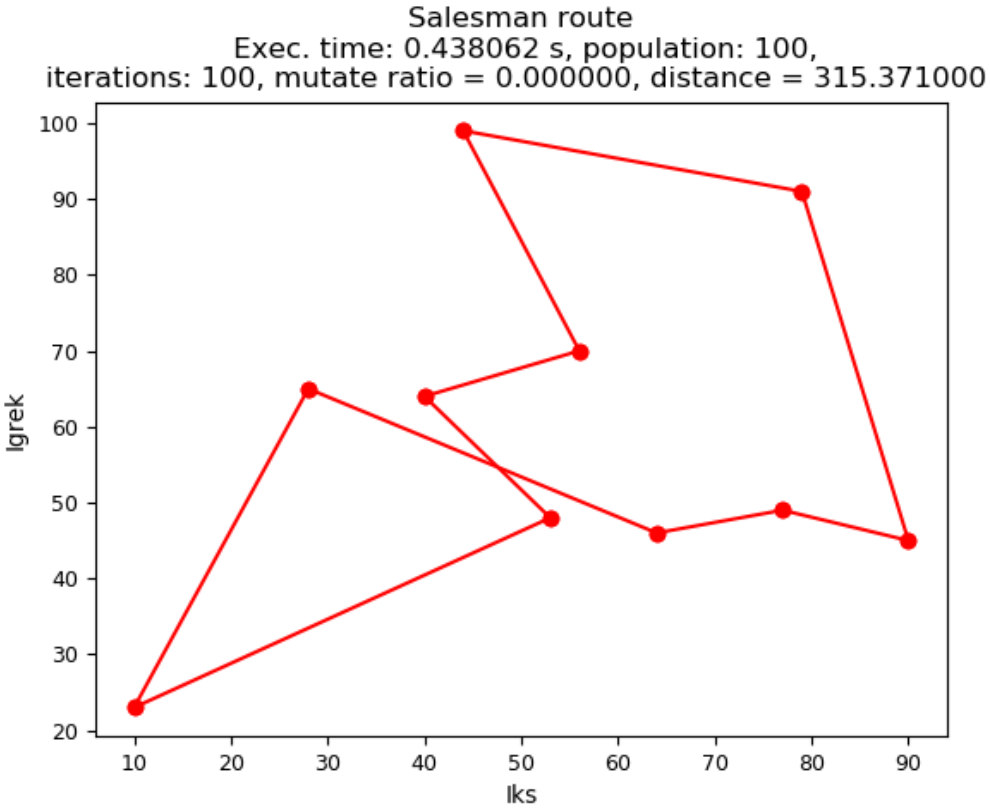
Obserwacje

- Przy dużej ilości miast, zwiększenie liczby iteracji oraz populacji ma pozytywny wpływ na precyzję i optymalność rozwiązania, jednak wydłuża jego czas.
- Czas wykonania programu jest porównywalny dla wszystkich sposobów ułożenia miast. Niewielkie różnice pojawiają się w próbach dotyczących 5 miast, kiedy ułożone są na siatce, program wykonuje się nieznacznie krócej.
- Przy większej liczbie miast często najlepszy efekt przynosi program wykonujący się najdłużej, dzięki dużej liczbie iteracji oraz populacji. Nie jest to jednak regułą, ponieważ w wielu przypadkach porównywalnie dobre efekty przynoszą próby wykonujące się znacznie szybciej. Może to zależeć od szczęścia podczas generowania populacji i losowania zawodników do turnieju.
- Dla małej liczby miast nawet niewielka wartość iteracji i populacji może znaleźć poprawne rozwiązanie, ponieważ liczba kombinacji jest wystarczająco mała.
- Lepsze wyniki osiągały próby, gdzie współczynnik mutacji wynosił 0 niż wtedy gdy był większy. Świadczy to o tym, że lepsze wyniki mogłaby spowodować modyfikacja funkcji mutującej. Zwlaszcza widoczne jest to przy próbach w których uczestniczy niewiele miast, ponieważ łatwo zepsuć poprawne wyniki i cofać się do złych rozwiązań.

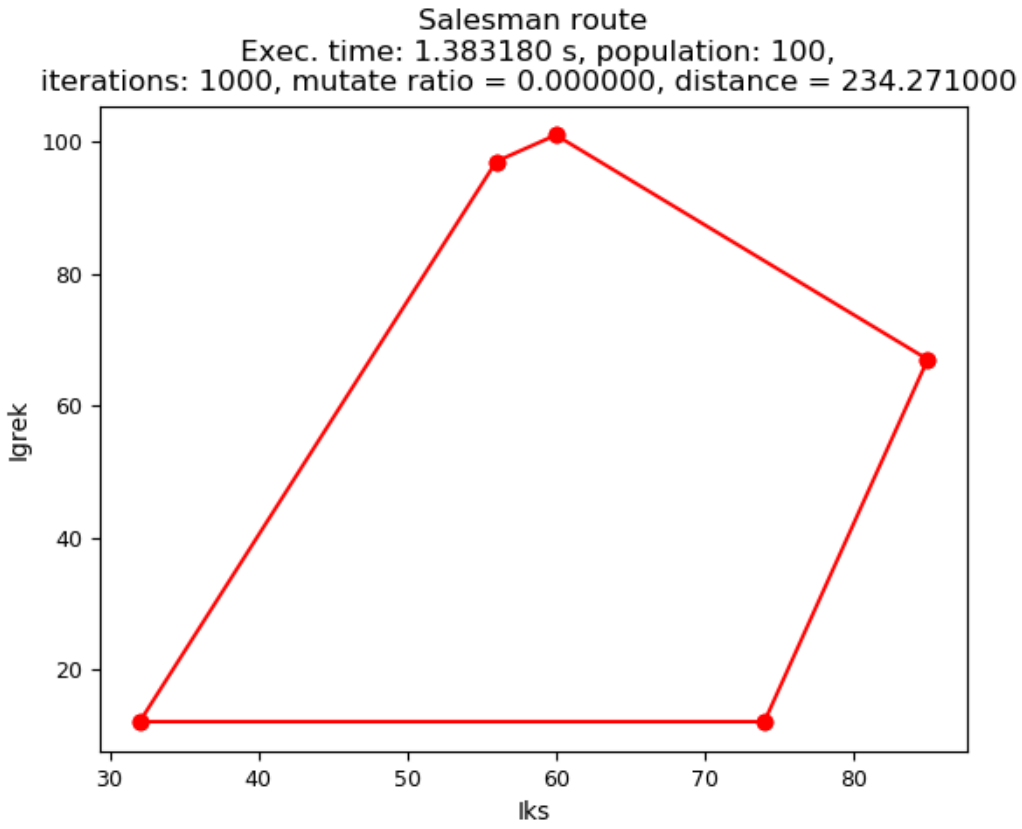
Wyżej wspomniane wykresy znajdują się na kolejnych stronach.



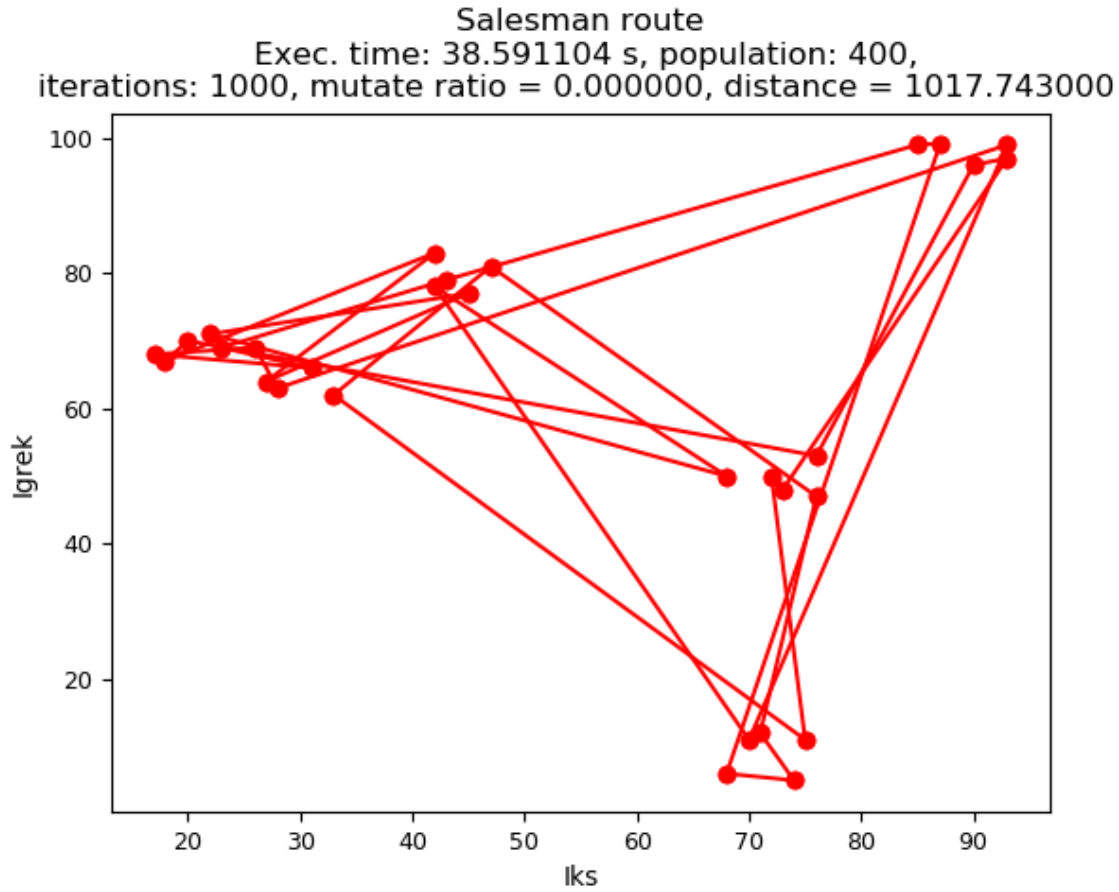
Rysunek 1. Najkrótsza droga odnaleziona dla 30 miast rozłożonych dowolnie



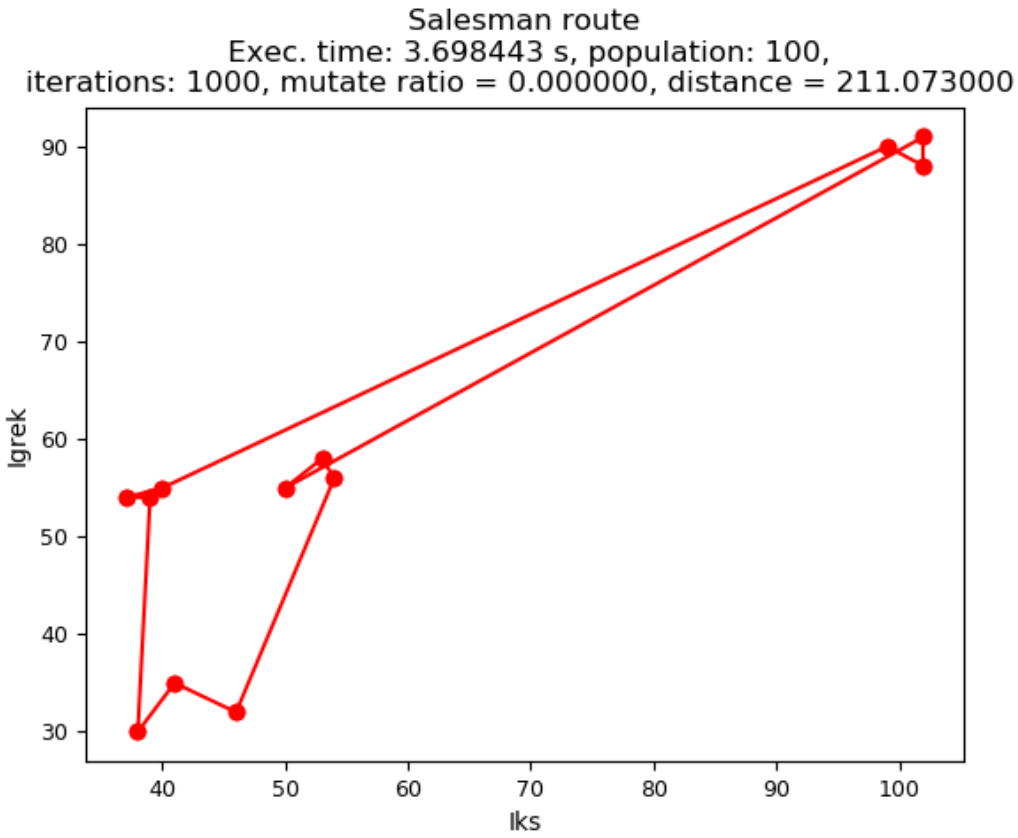
Rysunek 2 Najkrótsza droga odnaleziona dla 10 miast ułożonych dowolnie



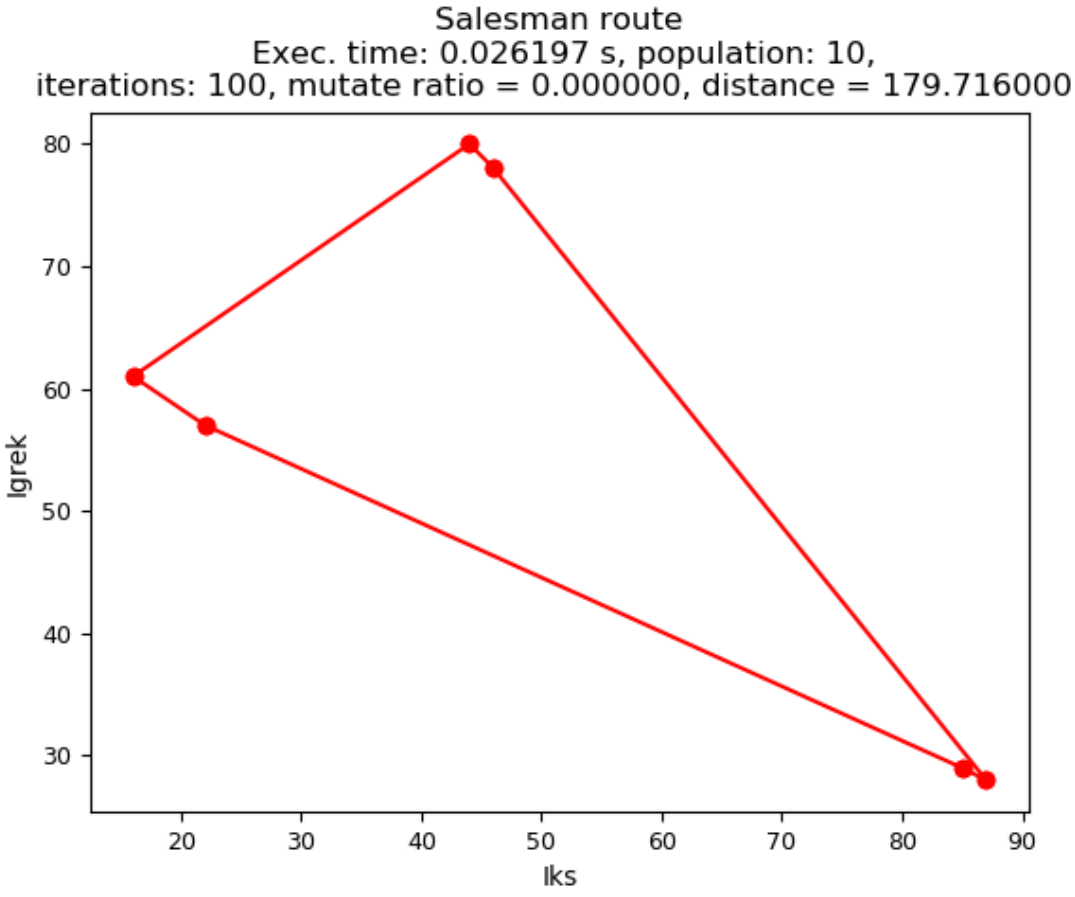
Rysunek 3 Najkrótsza droga odnaleziona dla 5 miast ułożonych dowolnie



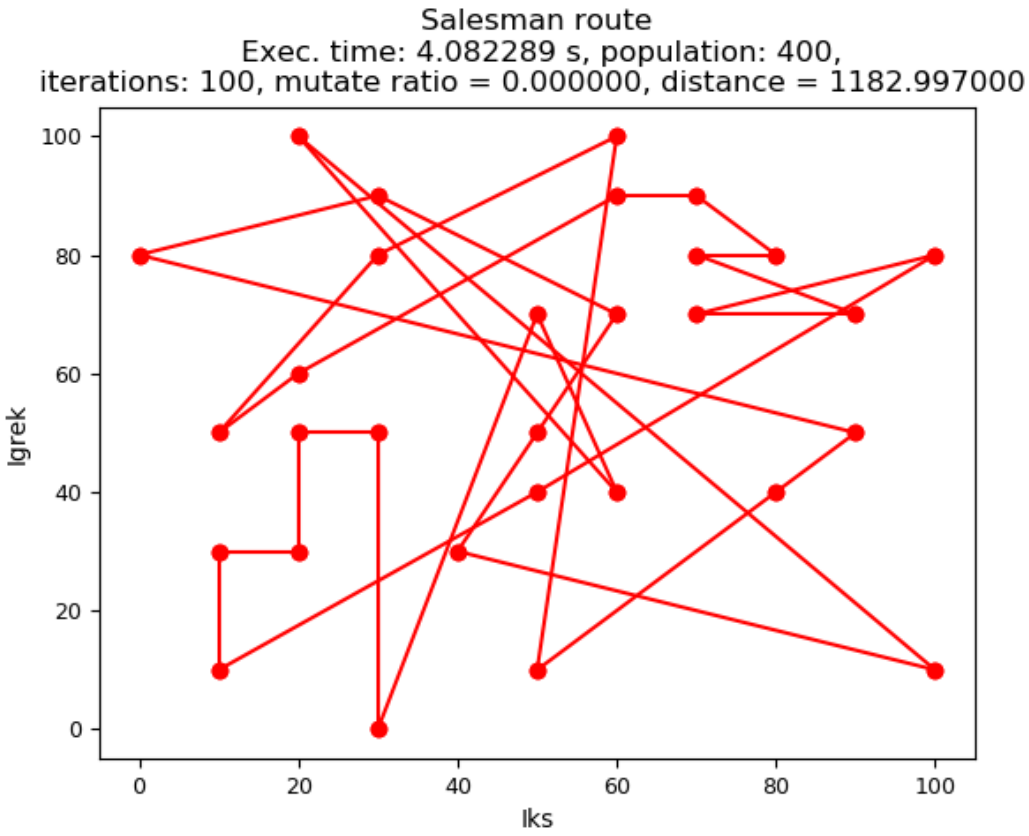
Rysunek 4 Najkrótsza droga odnaleziona dla miast ułożonych w 6 grup po 5 - w sumie 30 miast



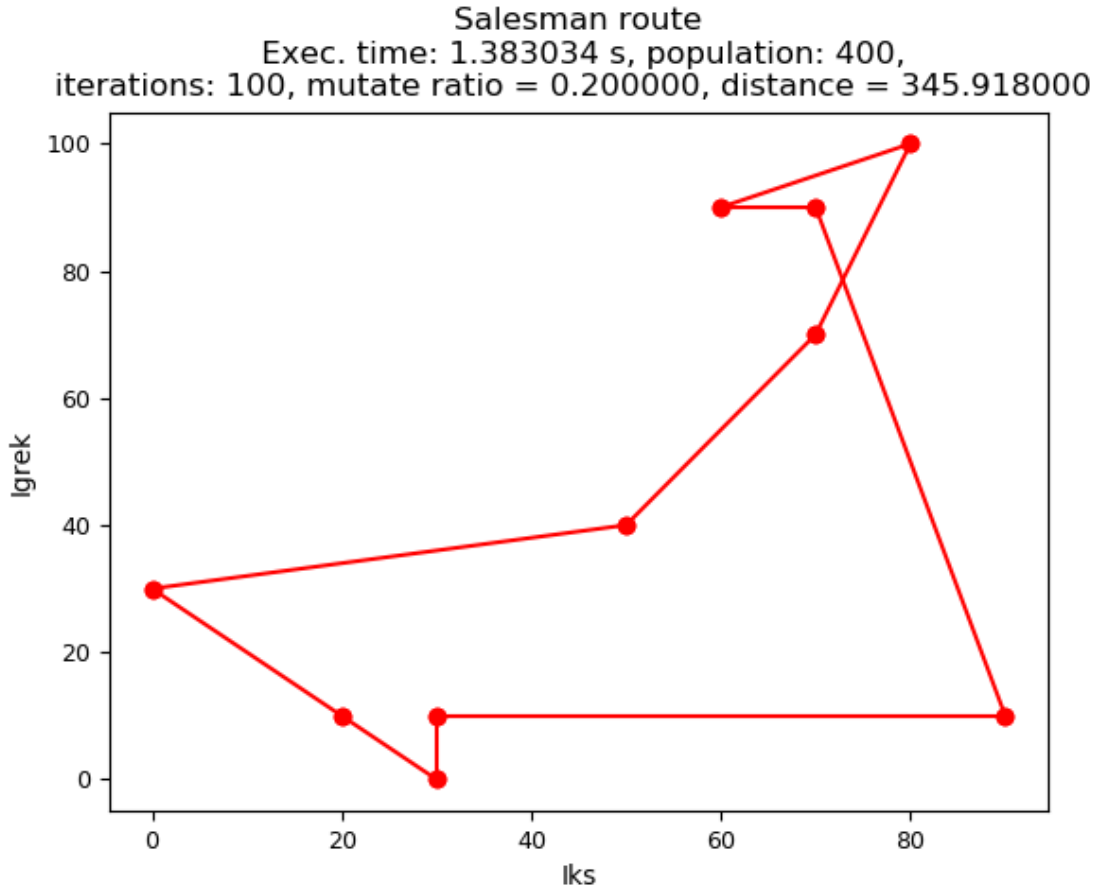
Rysunek 5 Najkrótsza droga odnaleziona dla miast ułożonych w 4 grupy po 3 miasta - w sumie 12 miast



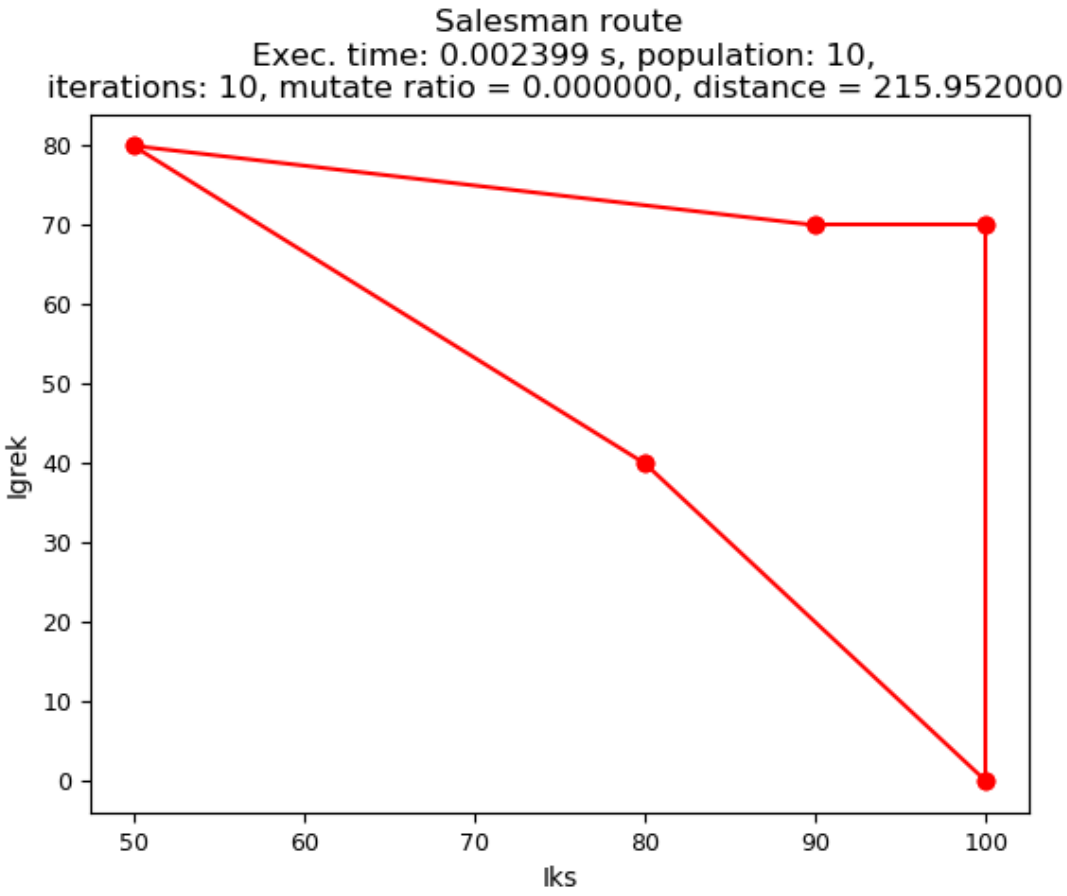
Rysunek 6. Najkrótsza droga odnaleziona dla miast ułożonych w 3 grupy po 2 miasta - w sumie 6 miast



Rysunek 7. Najkrótsza droga odnaleziona dla 30 miast ułożonych na siatce



Rysunek 8. Najkrótsza droga odnaleziona dla 10 miast ułożonych na siatce



Rysunek 9. Najkrótsza droga odnaleziona dla 5 miast ułożonych na siatce