Zadanie 1. Heurystyki konstrukcyjne

Oskar Kiliańczyk 151863 & Wojciech Kot 151876

1 Opis zadania

Podczas zajęć rozważamy zmodyfikowany problem komiwojażera. Początkowo, obliczamy macierz odległości pomiędzy danymi miastami. Obliczona macierz odległości między wierzchołkami grafu będzie podstawą dla każdego algorytmu, a celem jest wyznaczenie dwóch rozłącznych zamkniętych ścieżek (cykli), z których każda zawiera 50% wierzchołków. Jeśli liczba wierzchołków jest nieparzysta, jedna ścieżka zawiera jeden wierzchołek więcej. Kryterium optymalizacji jest minimalizacja łącznej długości obu cykli.

Rozważane instancje problemu pochodzą z biblioteki TSPLib, a są to kroa200 oraz krob200. Są to instancje dwuwymiarowe euklidesowe, w których każdemu wierzchołkowi przypisane są współrzędne na płaszczyźnie. Odległość między wierzchołkami liczona jest jako odległość euklidesowa, zaokrąglana do najbliższej liczby całkowitej. W implementacji algorytmów wykorzystywana będzie wyłącznie macierz odległości, co zapewnia możliwość zastosowania kodu do innych instancji problemu.

2 Zaimplementowane algorytmy

2.1 Algorytm zachłanny - metoda najbliższego sąsiada

Algorytm ten wykorzystuje funkcję znajdującą najbliższego sąsiada dla danego wierzchołka (miasta) Działa ona w następujący sposób:
dla każdego miasta, sprawdza czy zostało już odwiedzone
jeśli nie, to sprawdza czy dystans jest mniejszy od dystansu z danego miasta do obecnie zapamiętanego jako najbliższe
jeśli jest bliższe niż obecnie pamiętane jako najbliższe, zapamiętuje je jako najbliższe
jeśli nie ma żadnego miasta obecnie pamiętanego jako najbliższe, przypisuje to miasto

Główny algorytm natomiast, wygląda następująco:

1. Przydziela wierzchołki startowe do cykli pierwszego i drugiego.
2. Tworzy tablicę indeksów miast, zaznaczając wszystkie poza startowymi jako nieodwiedzone.
3. Dopóki istnieją jakieś nieodwiedzone miasta, powtarza następujące kroki:
   - Znajduje najbliższego nieodwiedzonego sąsiada do ostatniego wierzchołka cyklu 1.
   - Dodaje go do cyklu 1 oraz zapisuje w tablicy jako odwiedzony.
   - Znajduje najbliższego nieodwiedzonego sąsiada do ostatniego wierzchołka cyklu 2.
   - Dodaje go do cyklu 2 oraz zapisuje w tablicy jako odwiedzony.
4. Kiedy już nie ma żadnych nieodwiedzonych miast, dopisuje na koniec cykli ich wierzchołki startowe.
5. Zwraca oba cykle jako znalezione ścieżki.

2.2 Algorytm zachłanny - metoda rozbudowy cyklu

Algorytm ten korzysta z funkcji znajdywania najlepszego wstawienia, a działa ona w następujący sposób:

1. Przyjmuje jako argumenty:
   - obecny cykl,
   - macierz dystansów,
   - tablicę indeksów odwiedzonych miast.
2. Tworzy listę możliwości (nieodwiedzonych wierzchołków).
3. Ustawia najtańszy koszt na maksymalnie dużą wartość.
4. Dla każdej możliwości z listy możliwości:
   (a) Dla każdego możliwego wstawienia w cykl:

- Oblicza wzrost dystansu, jaki spowoduje wstawienie (a więc przy wstawianiu między a i b wierzchołka c, oblicza dystans $|ac| + |bc| - |ab|$).
- Zapisuje najmniejszy znaleziony wzrost dystansu oraz miejsce jego wstawienia.

(b) Jeśli koszt wstawienia obecnie znalezionego wierzchołka jest mniejszy niż obecnie pamiętany najtańszy koszt wstawienia:

- Zapisuje ten koszt jako obecnie najtańszy.
- Zapisuje wierzchołek oraz miejsce jego wstawienia.

5. Po przejrzeniu wszystkich możliwości zwraca parę ⟨wierzchołek, miejsce wstawienia⟩.
6. Dwukrotnie przydziela wierzchołki startowe do cykli pierwszego i drugiego jako początek i koniec cyklu.
7. Tworzy tablicę indeksów miast, zaznaczając wszystkie poza startowymi jako nieodwiedzone.
8. Dopóki istnieją jakieś nieodwiedzone miasta, powtarza następujące kroki:
   (a) Znajduje najtańsze wstawienie, czyli parę ⟨wierzchołek, miejsce w cyklu⟩ dla cyklu 1.
   (b) Wstawia w odpowiednie miejsce cyklu 1 znaleziony wierzchołek.
   (c) Zapisuje wierzchołek jako już odwiedzony.
   (d) Znajduje najtańsze wstawienie dla cyklu 2.
   (e) Jeśli ono nie istnieje (np. bo ostatni wierzchołek został już wstawiony), to kończy pętlę.
   (f) Wstawia w odpowiednie miejsce cyklu 2 znaleziony wierzchołek.
   (g) Zapisuje wierzchołek jako już odwiedzony.
9. Kiedy już nie ma żadnych nieodwiedzonych miast, zwraca oba cykle jako znalezione ścieżki.

Sam algorytm:

1. Dwukrotnie przydziela wierzchołki startowe do cykli pierwszego i drugiego jako początek i koniec cyklu.
2. Tworzy tablicę indeksów miast, zaznaczając wszystkie poza startowymi jako nieodwiedzone.
3. Dopóki istnieją jakieś nieodwiedzone miasta, powtarza następujące kroki:
   (a) Znajduje najtańsze wstawienie, czyli parę ⟨wierzchołek, miejsce w cyklu⟩ dla cyklu 1.
   (b) Wstawia w odpowiednie miejsce cyklu 1 znaleziony wierzchołek.
   (c) Zapisuje wierzchołek jako już odwiedzony.
   (d) Znajduje najtańsze wstawienie dla cyklu 2.
   (e) Jeśli ono nie istnieje (np. bo ostatni wierzchołek został już wstawiony), to kończy pętlę.
   (f) Wstawia w odpowiednie miejsce cyklu 2 znaleziony wierzchołek.
   (g) Zapisuje wierzchołek jako już odwiedzony.
4. Kiedy już nie ma żadnych nieodwiedzonych miast, zwraca oba cykle jako znalezione ścieżki.

2.3 Algorytm z żalem - metoda rozbudowy cyklu

1. Dwukrotnie przydziela wierzchołki startowe do cykli pierwszego i drugiego jako początek i koniec cyklu.
2. Tworzy tablicę indeksów miast, zaznaczając wszystkie poza startowymi jako nieodwiedzone.
3. Dopóki istnieją jakieś nieodwiedzone miasta, powtarza następujące kroki:
   (a) Dla każdego cyklu:
      i. Oblicz koszt rozbudowy cyklu o każde możliwe nieodwiedzone dotąd miasto dla każdego możliwego punktu rozbudowy cyklu.
      ii. Posortuj listę kosztów (rosnąco).
      iii. Oblicz dwużal jako różnicę pomiędzy drugim najlepszym a najlepszym wynikiem z listy kosztów.

iv. Jeżeli wyliczony żal jest większy niż dla jakiegokolwiek innego wstawienia miasta, to uznaj go za najlepszy.

(b) Rozszerz cykl w miejscu wskazanym dla najlepszego wyliczonego dwużalu.

(c) Zapisz wierzchołek jako już odwiedzony.

4. Kiedy już nie ma żadnych nieodwiedzonych miast, zwraca oba cykle jako znalezione ścieżki.

2.4 Algorytm z żalem - metoda rozbudowy cyklu z żalem ważonym

1. Dwukrotnie przydziela wierzchołki startowe do cykli pierwszego i drugiego jako początek i koniec cyklu.
2. Tworzy tablicę indeksów miast, zaznaczając wszystkie poza startowymi jako nieodwiedzone.
3. Dopóki istnieją jakieś nieodwiedzone miasta, powtarza następujące kroki:
   (a) Dla każdego cyklu:
      i. Oblicz koszt rozbudowy cyklu o każde możliwe nieodwiedzone dotąd miasto dla każdego możliwego punktu rozbudowy cyklu.
      ii. Posortuj listę kosztów (rosnąco).
      iii. Oblicz dwużal ważony jako różnicę pomiędzy drugim najlepszym a najlepszym wynikiem z listy kosztów z odpowiednimi wagami (domyślnie $waga_2 = (-1) \cdot waga_1$).
      iv. Jeżeli wyliczony żal jest większy niż dla jakiegokolwiek innego wstawienia miasta, to uznaj go za najlepszy.
   (b) Rozszerz cykl w miejscu wskazanym dla najlepszego wyliczonego dwużalu.
   (c) Zapisz wierzchołek jako już odwiedzony.
4. Kiedy już nie ma żadnych nieodwiedzonych miast, zwraca oba cykle jako znalezione ścieżki.

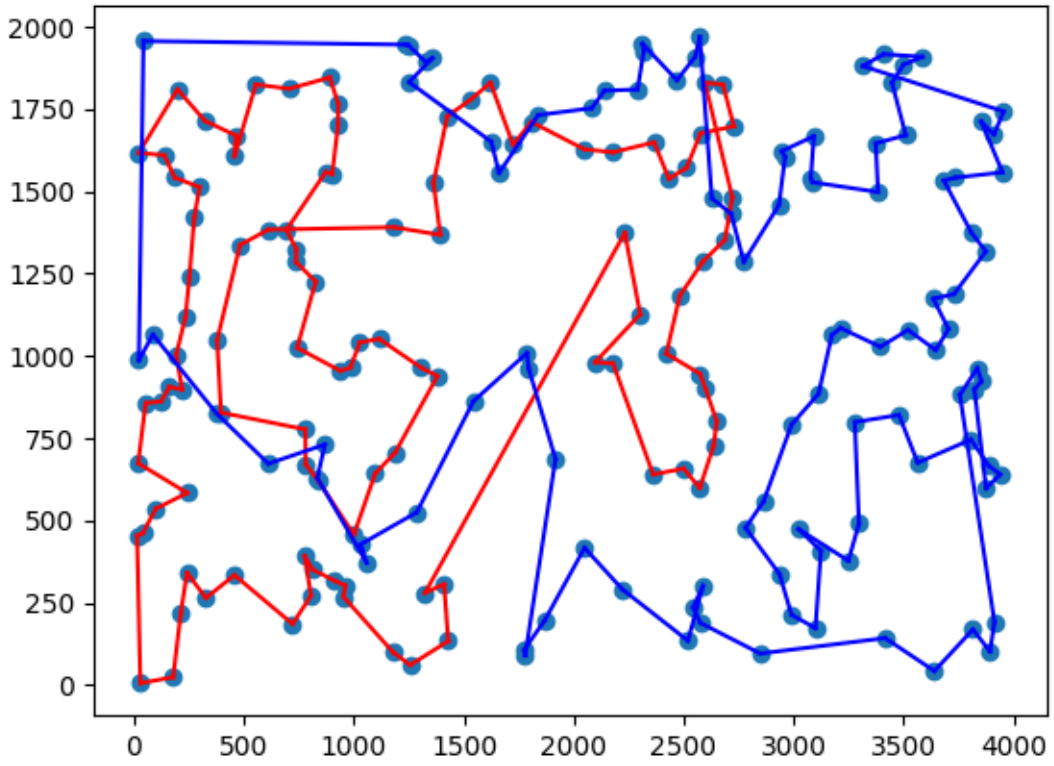
2.5 Własny jakiś dziki algorytm, jak nam się będzie chciało, bo imo bym zrobił coś

```
kodkodkod
```

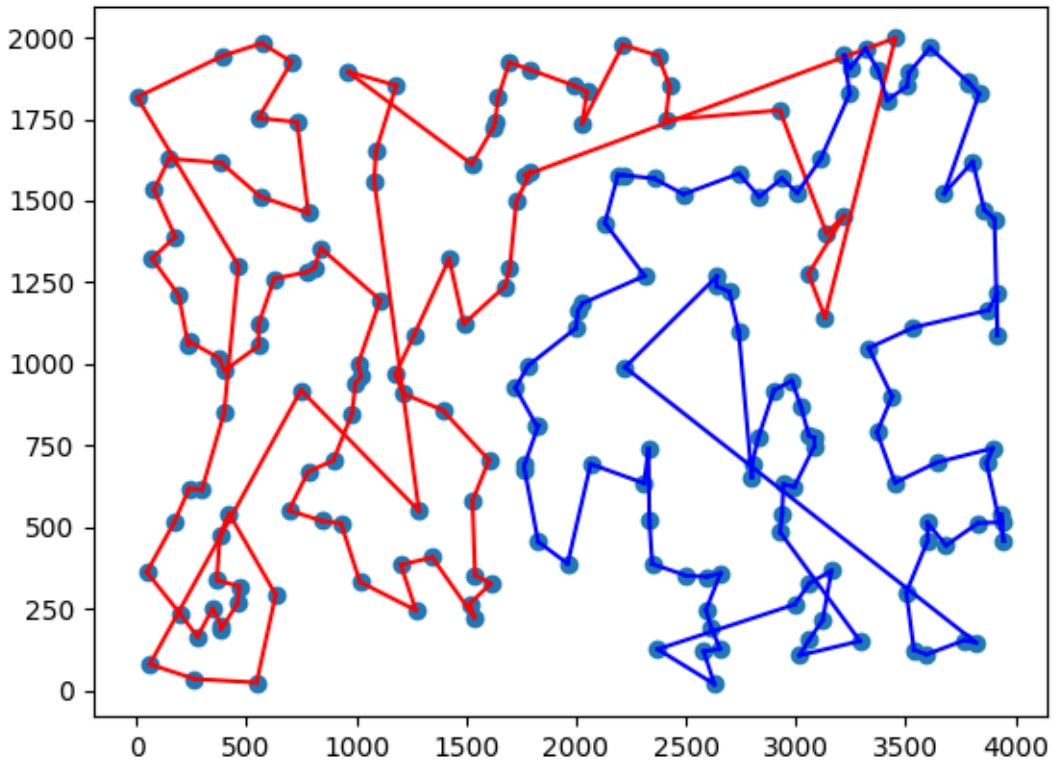
3 Wyniki eksperymentu obliczeniowego

3.1 Algorytm zachłanny - metoda najbliższego sąsiada

Graficzną reprezentację wyników dla algorytmu zachłannego metodą najbliższego sąsiada przestawiono na **??**



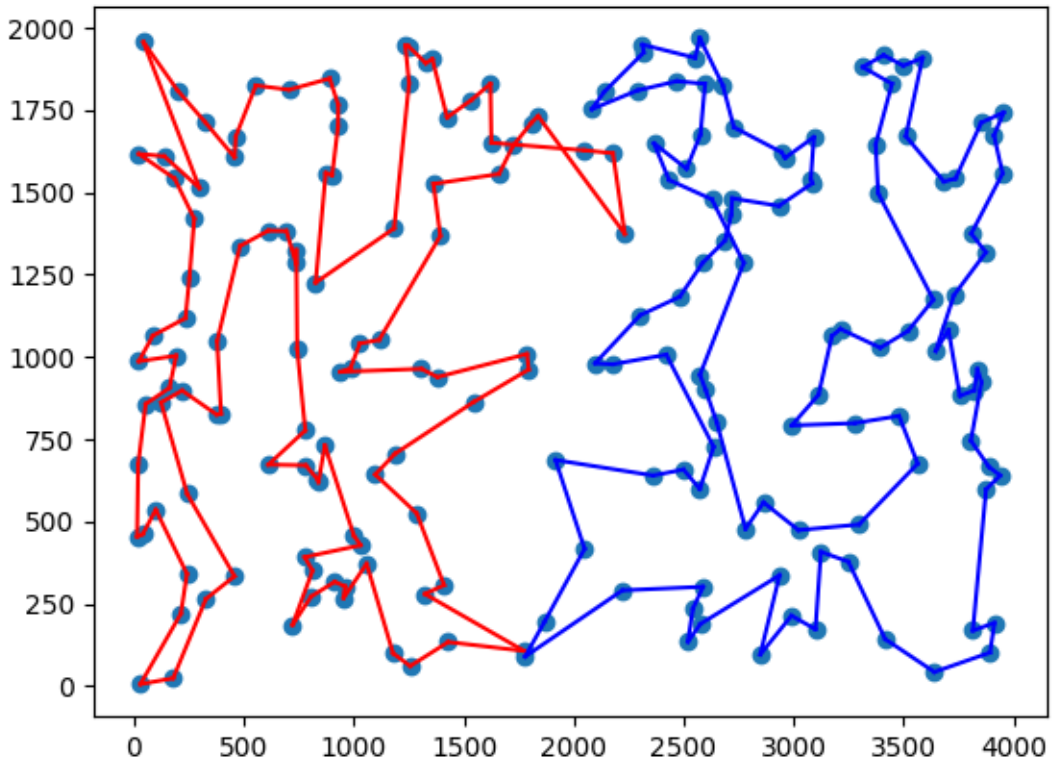
Rysunek 1: Instancja kroA200



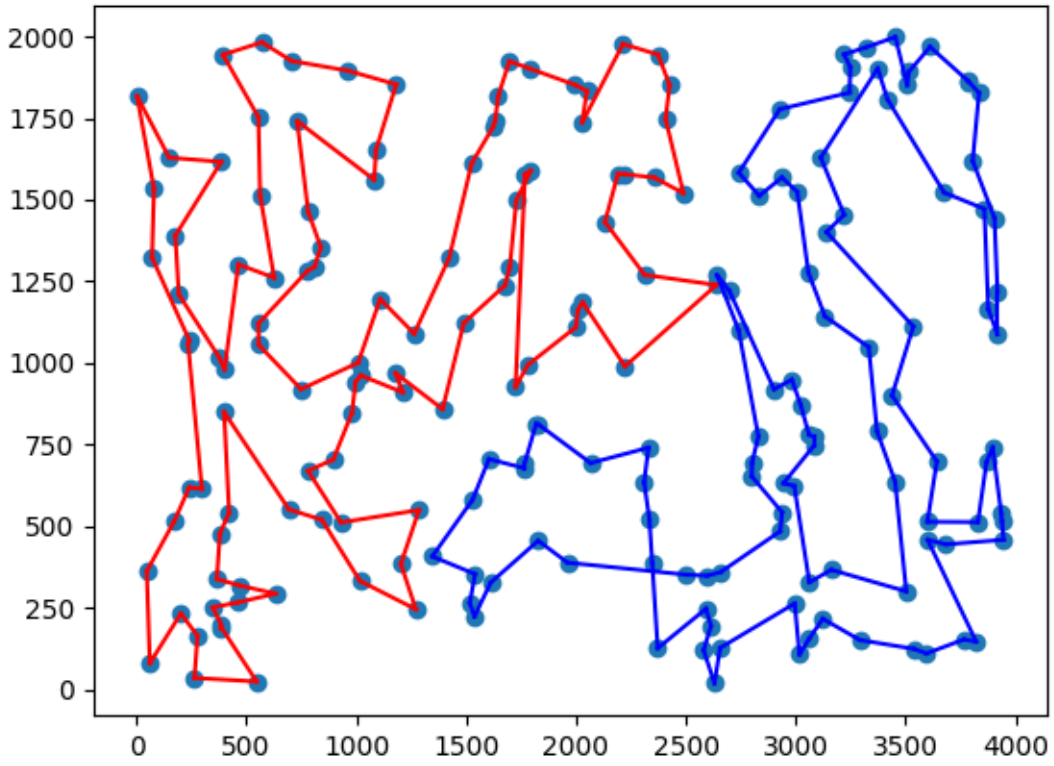
Rysunek 2: Instancja kroB200

3.2 Algorytm zachłanny - metoda rozbudowy cyklu

Graficzną reprezentację wyników dla algorytmu zachłannego metodą rozbudowy cyklu przestawiono na **??**



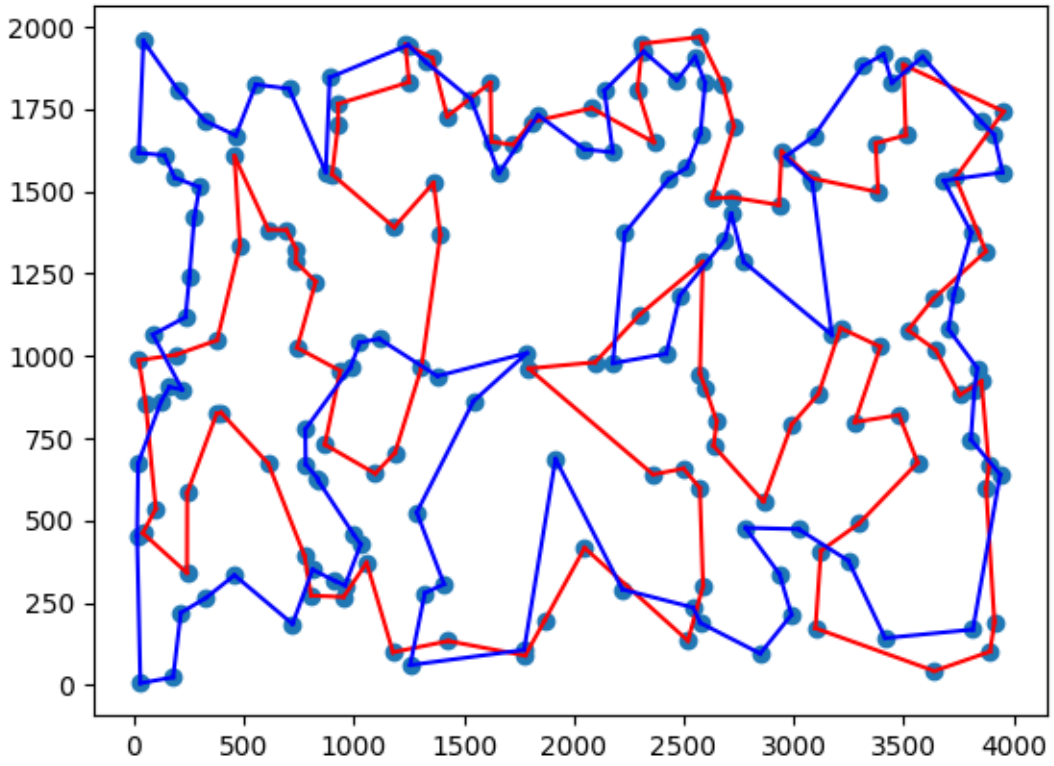
Rysunek 3: Instancja kroA200



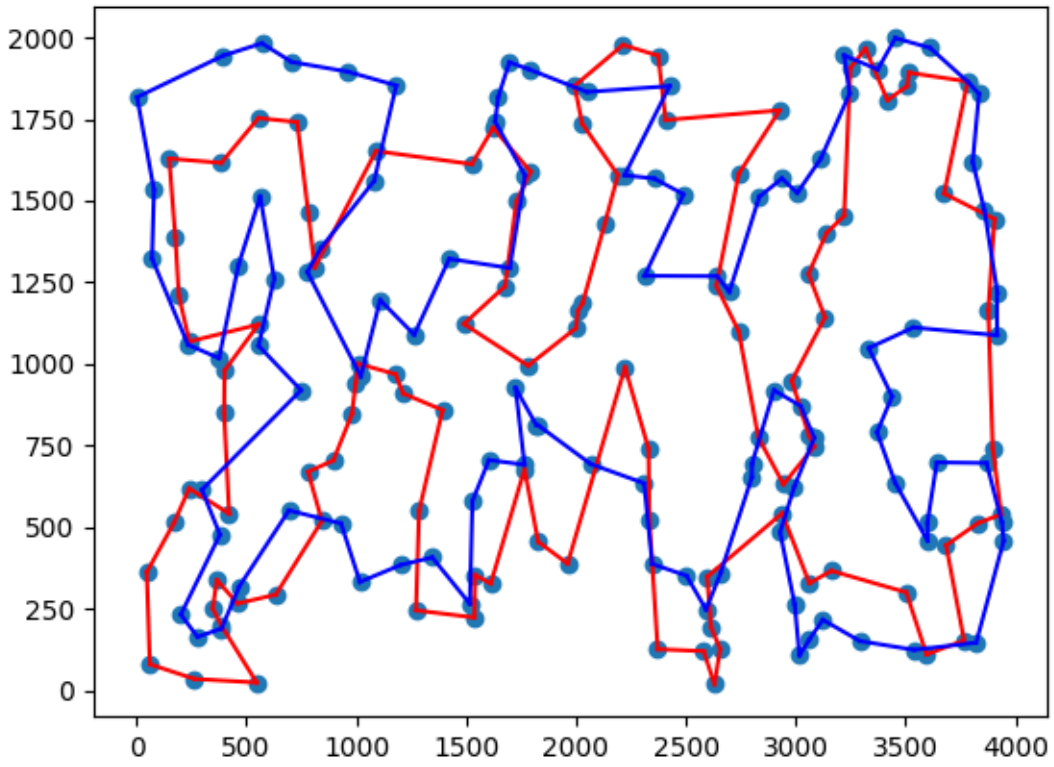
Rysunek 4: Instancja kroB200

3.3 Algorytm dwużal

Graficzną reprezentację wyników dla algorytmu dwużalu przestawiono na **??**



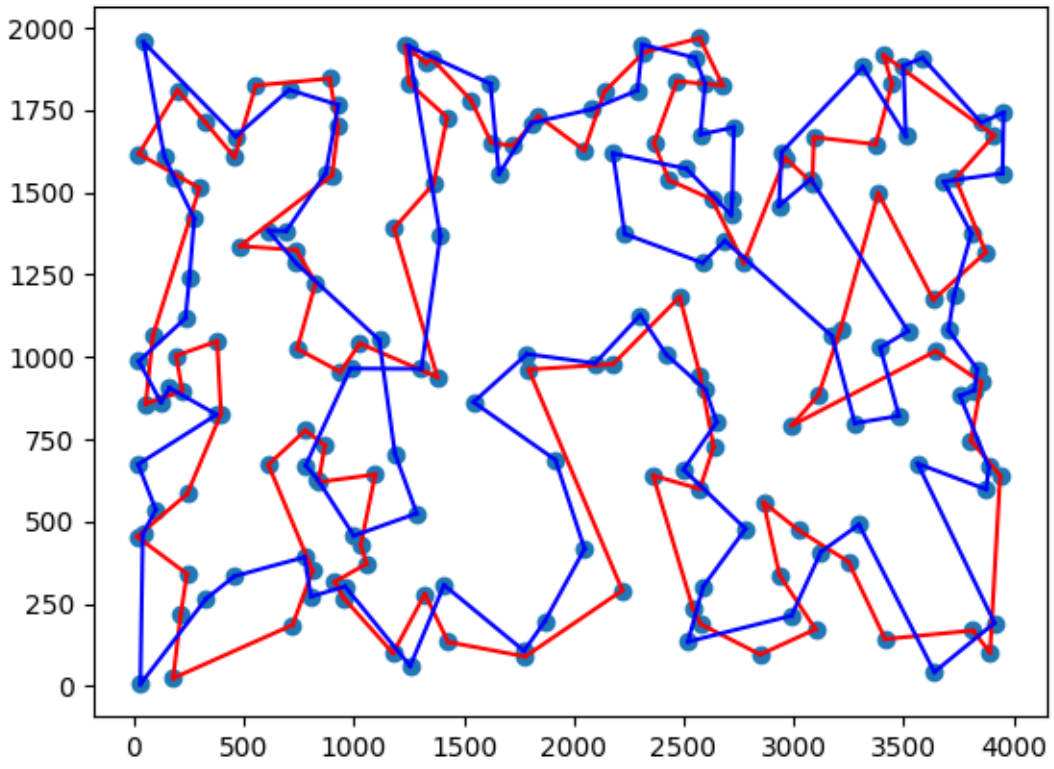
Rysunek 5: Instancja kroA200



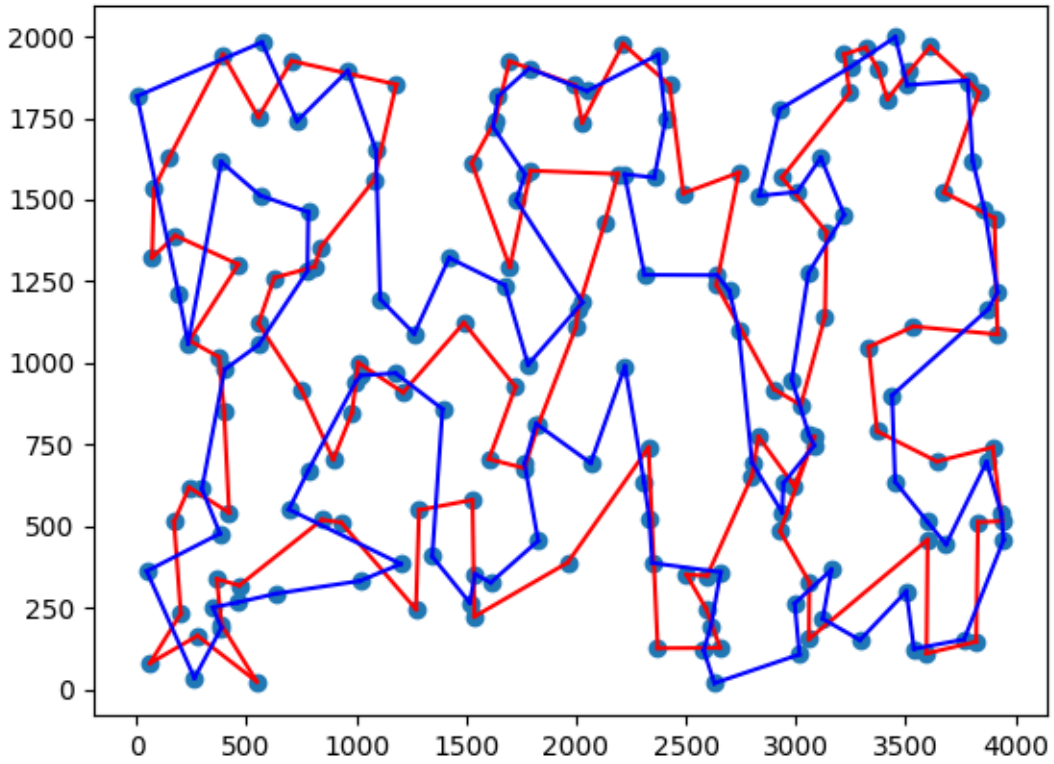
Rysunek 6: Instancja kroB200

3.4 Algorytm dwużal ważony

Graficzną reprezentację wyników dla algorytmu dwużalu ważonego przestawiono na **??**



Rysunek 7: Instancja kroA200



Rysunek 8: Instancja kroB200

4 Porównanie algorytmów

Algorytm	KroA200.tsp			KroB200.tsp		
	Min	Średnia	Max	Min	Średnia	Max
Nearest Neighbor Greedy	36755.00	42701.26	47080.00	38121.00	42122.12	47313.00
Cycle Extension Greedy	35396.00	37966.75	41224.00	35733.00	38455.32	39991.00
Two regret	42450.00	44536.57	46038.00	42029.00	43420.64	45065.00
Weighted Two regret	50016.00	51307.46	53146.00	48945.00	50277.95	52187.00

Tabela 1: Wyniki eksperymentu obliczeniowego

5 Wnioski

Żal nie działa, chociaż byśmy chcieli, ale żal rozwiązuje zwykły TSP. Może w połączeniu z algorytmem klastrującym, jakimś k-means zadziałałby lepiej, ale strasznie krzywdzące dla niego jest też to że ścieżki MUSZĄ być równej długości

6 Link do repo

Kod źródłowy w repozytorium GitHub dostępny pod linkiem:
Repozytorium TSP Heuristics.