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 zapewni 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:

1. Dla każdego miasta, sprawdza czy zostało już odwiedzone
2. Jeśli nie, to sprawdza czy dystans jest mniejszy od dystansu z danego miasta do obecnie zapamiętanego jako najbliższe
3. Jeśli jest bliższe niż obecnie pamiętane jako najbliższe, zapamiętuje je jako najbliższe
4. 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⟩.

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 Nasz własny algorytm

Po zobaczeniu wyników dla pierwszych czterech opisanych tu algorytmów zdecydowaliśmy się na napisanie jeszcze jednego, własnego algorytmu. Głównym powodem było nasze ogromne niezadowolenie ze skuteczności podejścia korzystającego z dwużalu, oraz to jaki potencjał w tym podejściu zauważyliśmy. Jak da się zauważyć w tabeli 1 zamieszczonej w 3 algorytmy oparte na dwużalu nie osiągnęły zadowalających nas wyników, wobec czego wyciągnęliśmy wnioski opsisane w 5 i zdecydowaliśmy się skorzystać z dwużalu dzieląc wcześniej zbiór wierzchołków na dwa równe podzbiory, a następnie rozwiązanie standardowego problemu komiwojażera na każdym z podzbiorów. Trzymając się tematu zajęć podział na podzbiory został wykonany podejściem zachłannym:

1. Utworzenie listy pozostałych wierzchołków (wszystkie możliwe, poza startowymi)
2. Utworzenie dwóch list zawierających odpowiednio dystanse każdego wierzchołka do pierwszego i do drugiego wierzchołka startowego
3. Sortowanie utworzonych uprzednio list dystansów wierzchołków
4. Aż do przydzielenia wszystkich wierzchołków z listy pozostałych wierzchołków wykonuje:
5. Zmianę decyzji do którego zestawu wierzchołków obecnie będzie przydzielać wierzchołek (aby robić to naprzemiennie)
6. Wyszukuje pierwszy wierzchołek na liście dystansów który nie został jeszcze przydzielony do żadnego zestawu i przydziela go tam

W skutek zastosowania takiego przydziału uzyskujemy dwa równo-liczne zbiory, oraz zapewniamy że gdyby ilość badanych wierzchołków była nieparzysta, to zbiory będą różnić się długością najwyżej o 1.

Następnie wykorzystujemy tradycyjny algorytm rozbudowy cyklu w oparciu o dwużal, osobno na obu listach. Wygląda on następująco:

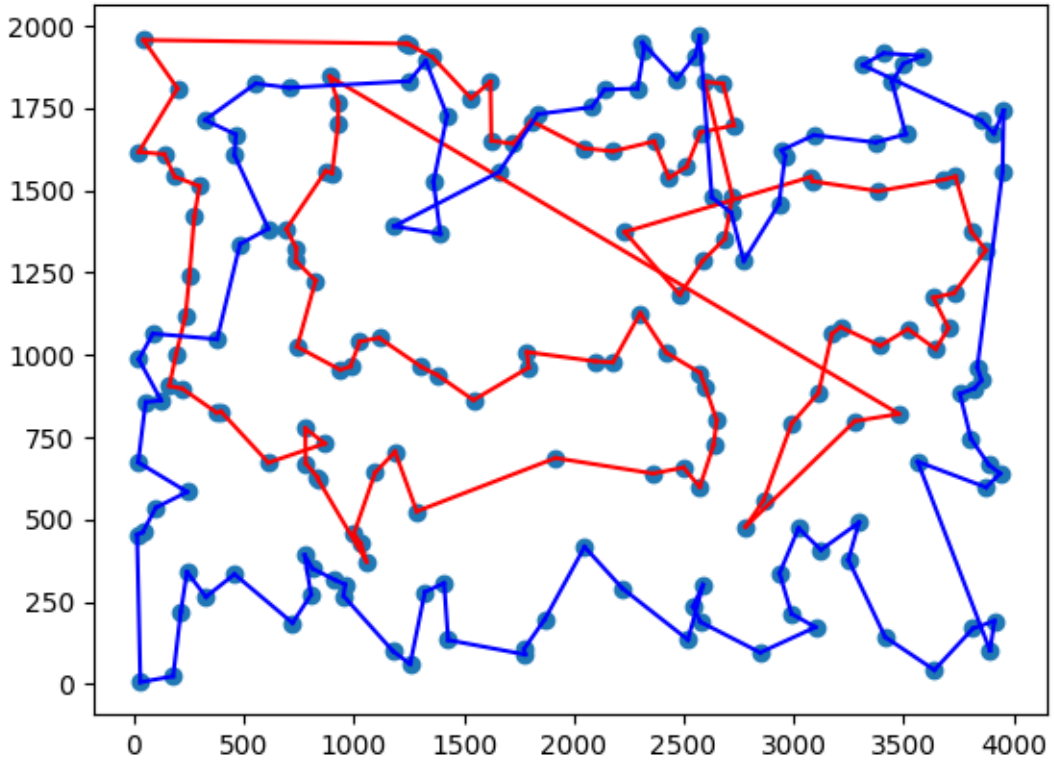
1. Algorytm zaczyna od ścieżki zawierającej wierzchołek startowy podwójnie
2. Dopóki w ścieżce nie znajdują sie wszystkie wierzchołki z zadanego mu zestawu powtarza:
3. Dla każdego nieodwiedzonego wierzchołka oblicza koszty jego wstawienia
4. Następnie oblicza żal (dwużal) dla danego wierzchołka
5. Rozbudowuje cykl o wierzchołek z największym obliczonym żalem i zaznacza go jako odwiedzonego
6. Zwraca ścieżkę

Używając takiej funkcji osobno na obu zbiorach wierzchołków uzyskujemy dwie ścieżki i zwracamy do programu głównego.

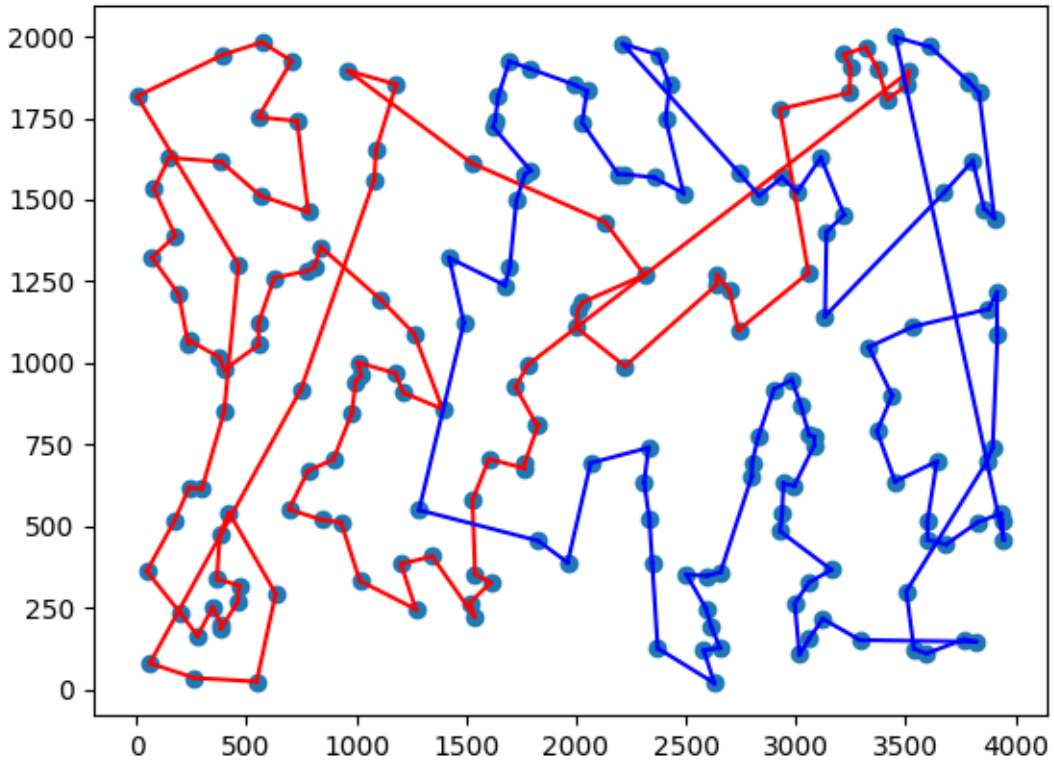
3 Wyniki eksperymentu obliczeniowego

3.1 Algorytm zachłanny - metoda najbliższego sąsiada

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



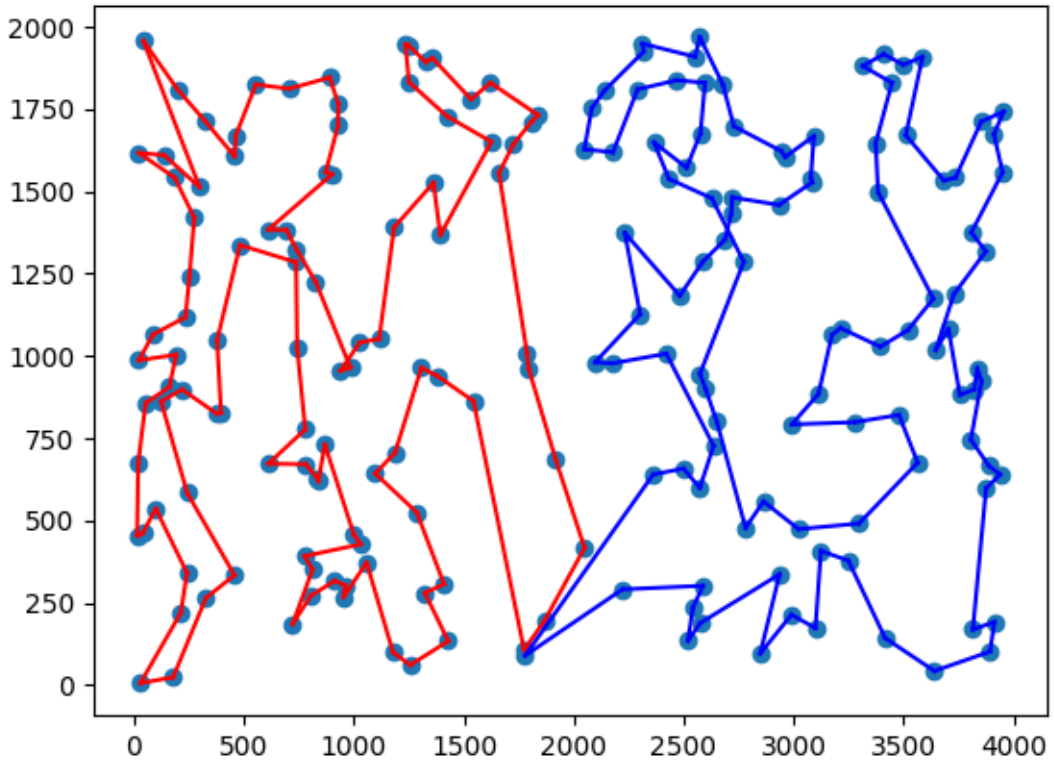
Rysunek 1: Instancja kroA200



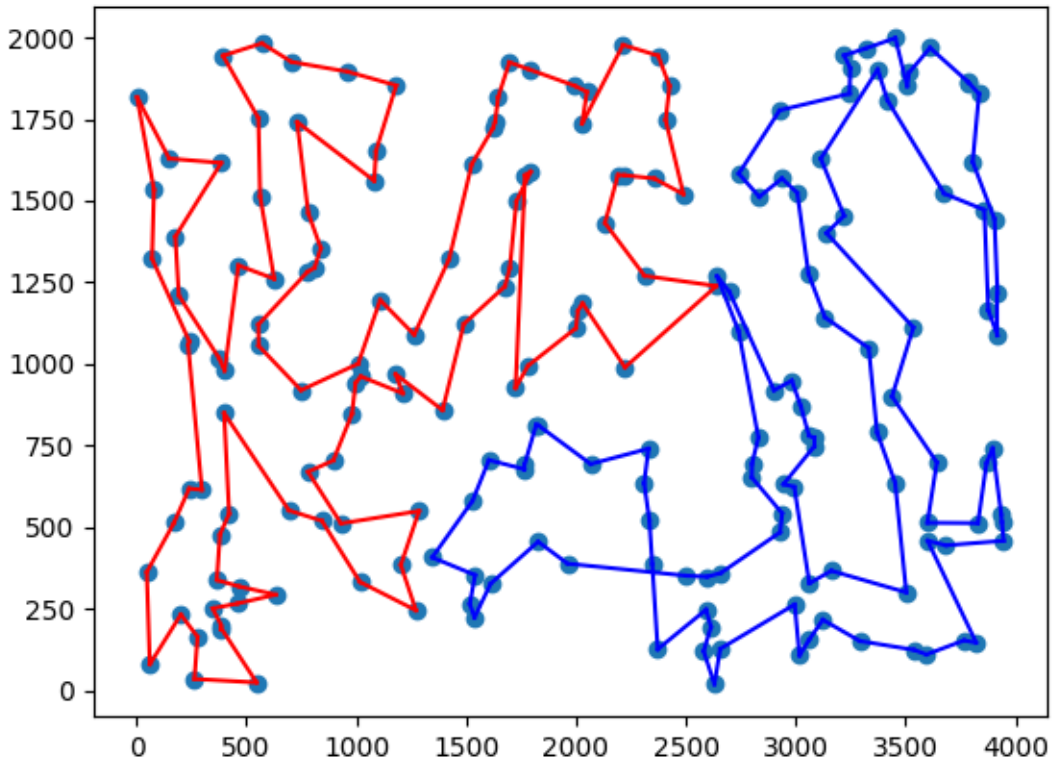
Rysunek 2: Instancja kroB200

3.2 Algorytm zachłanny - metoda rozbudowy cyklu

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

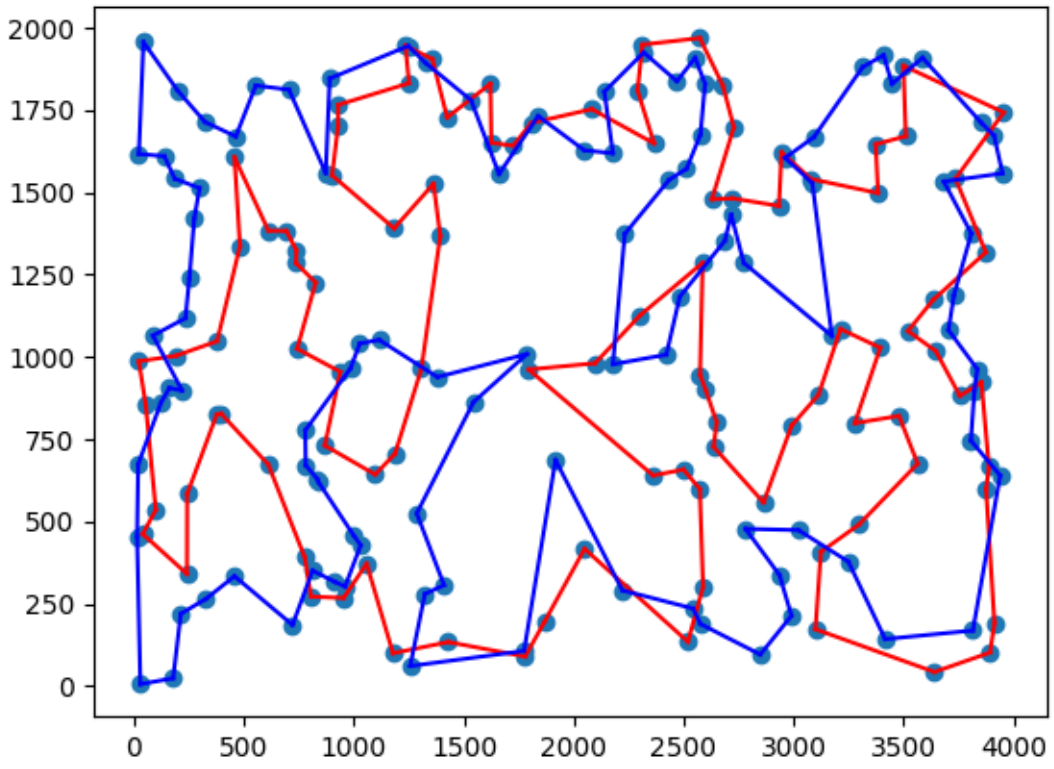


Rysunek 3: Instancja kroA200

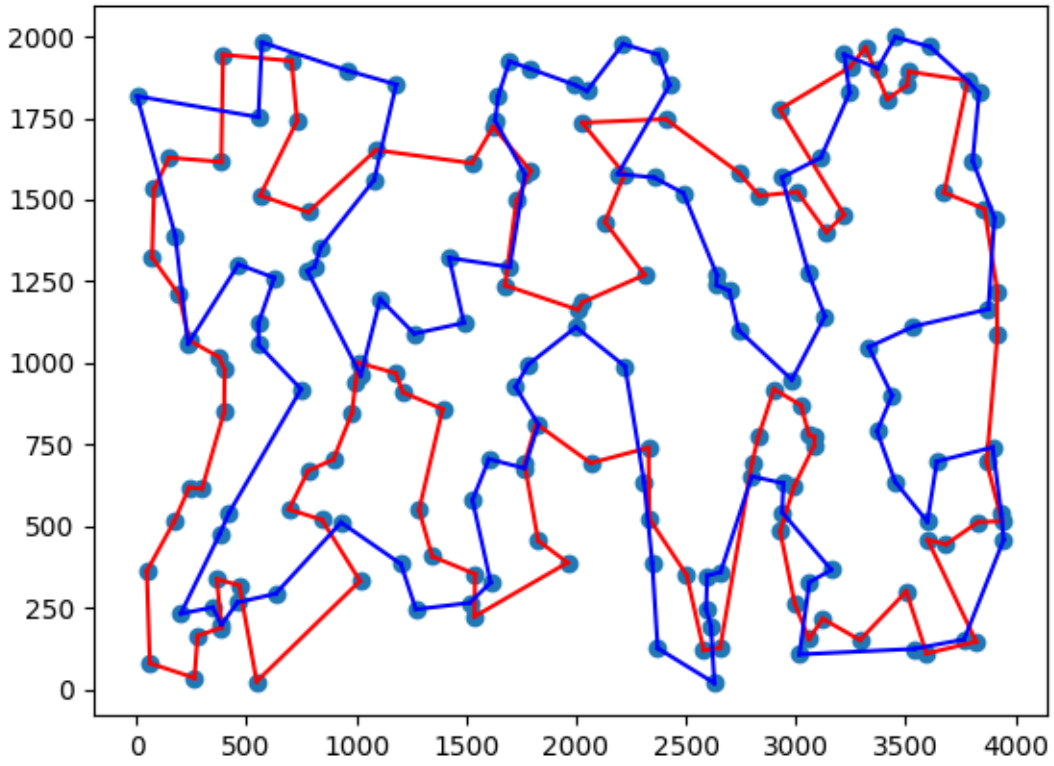


Rysunek 4: Instancja kroB200

3.3 Algorytm dwużal



Rysunek 5: Instancja kroA200

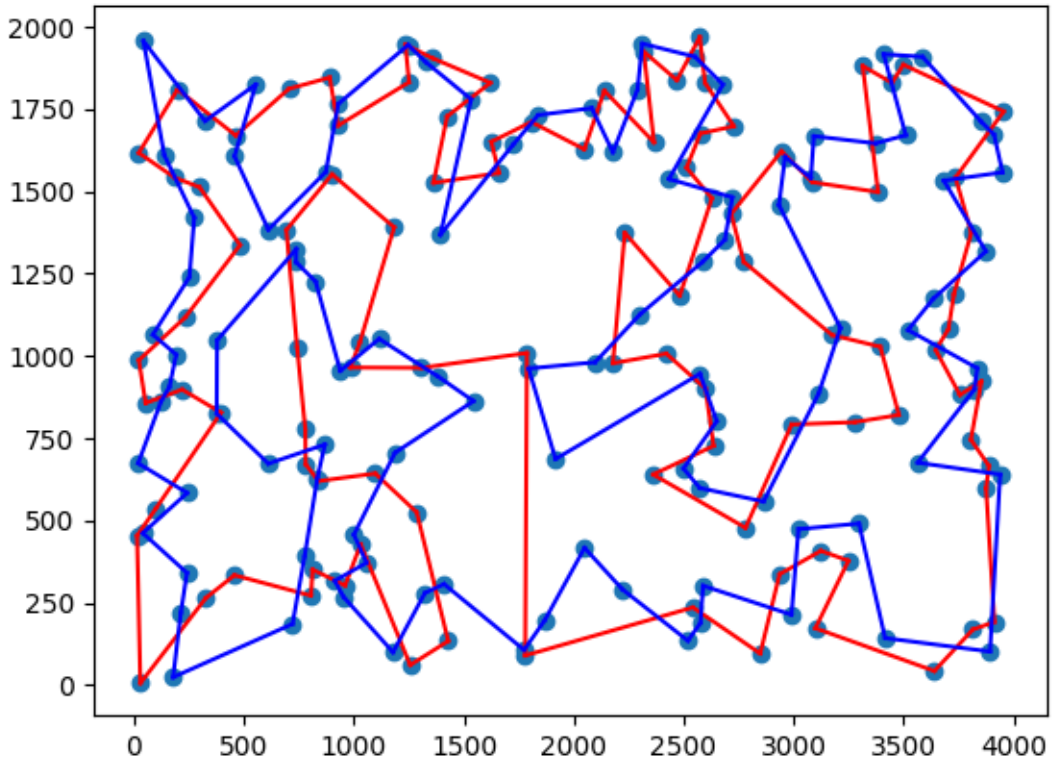


Rysunek 6: Instancja kroB200

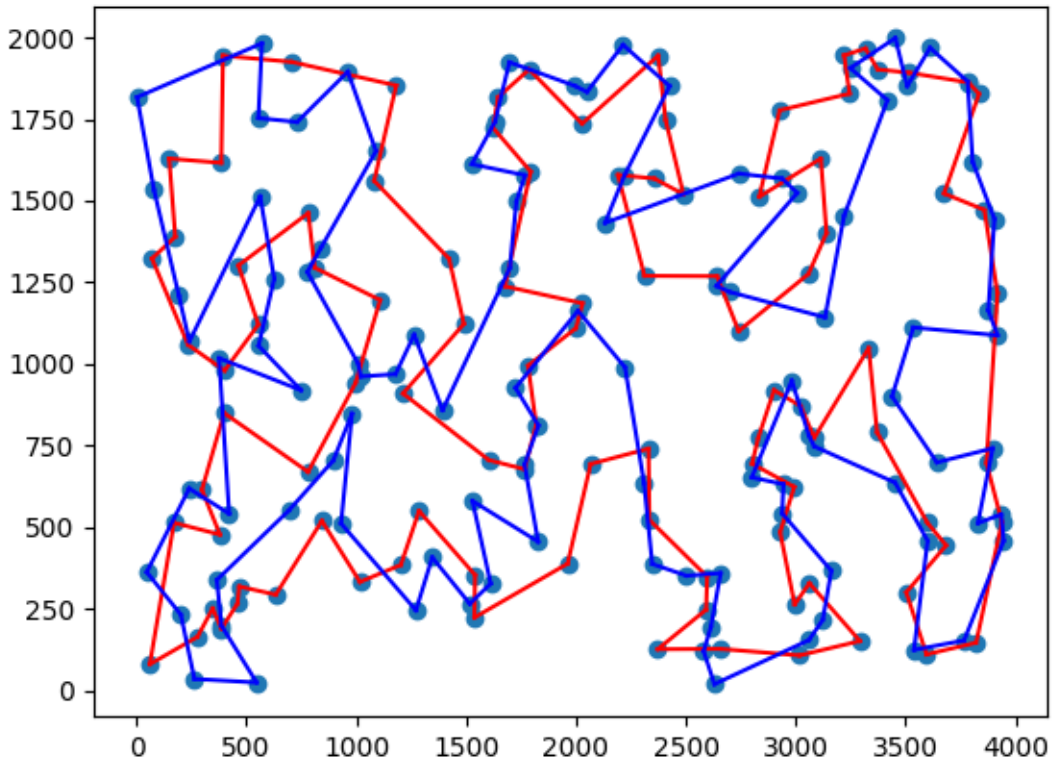
Graficzną reprezentację wyników dla algorytmu dwużalu przestawiono na **??**

3.4 Algorytm dwużal ważony

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

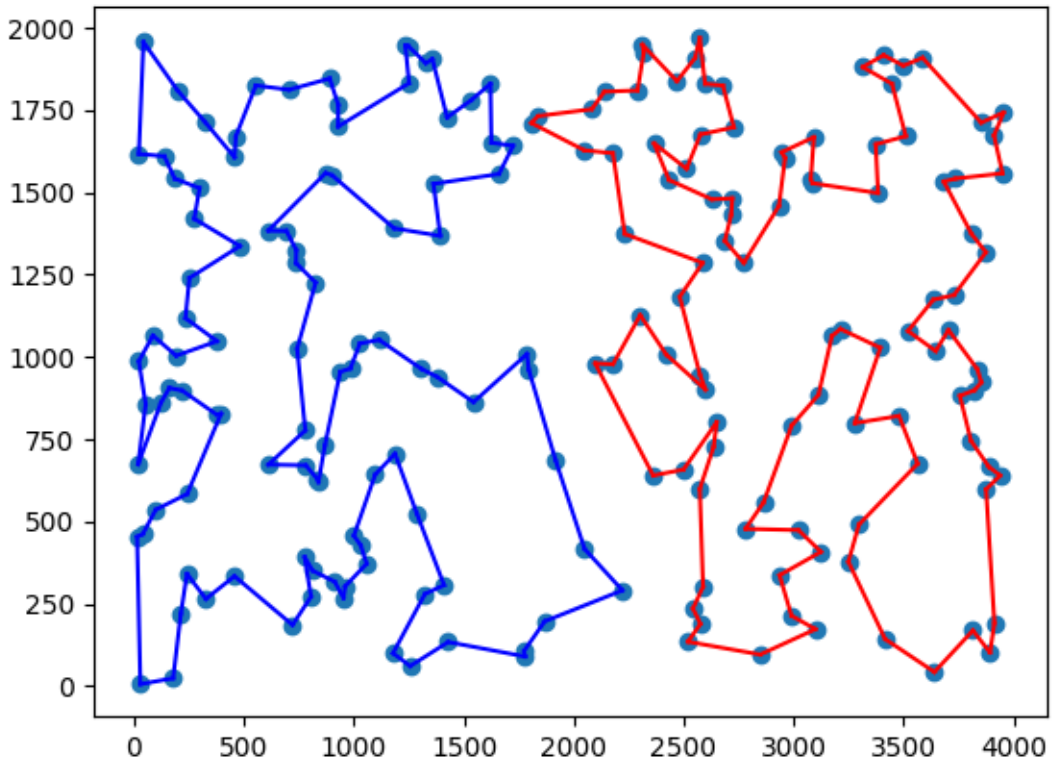


Rysunek 7: Instancja kroA200

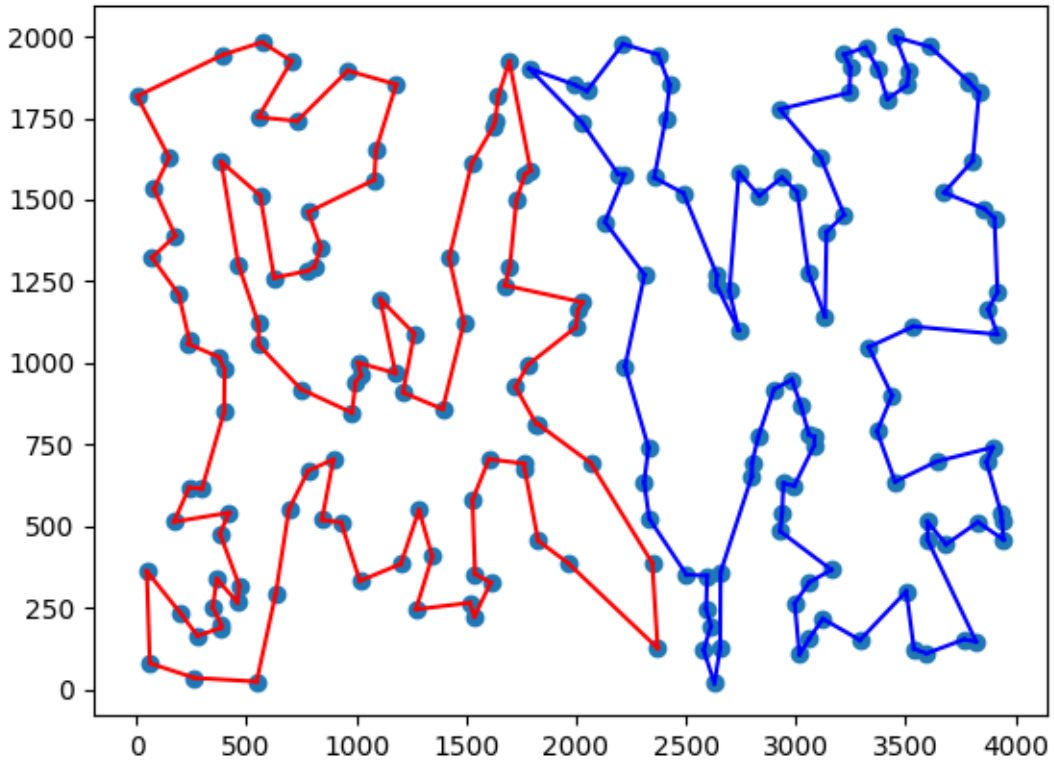


Rysunek 8: Instancja kroB200

3.5 Algorytm własny



Rysunek 9: Instancja kroA200



Rysunek 10: Instancja kroB200

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

4 Porównanie algorytmów

Algorytm	KroA200.tsp			KroB200.tsp		
	Min	Średnia	Max	Min	Średnia	Max
Nearest Neighbor Greedy	37841.00	42344.56	48120.00	38444.00	42088.28	45918.00
Cycle Extension Greedy	35323.00	38183.38	41224.00	35733.00	38778.79	40325.00
Two regret	42450.00	44358.42	46201.00	41176.00	43443.10	45065.00
Weighted Two regret	49770.00	51169.82	53146.00	48911.00	50312.95	51882.00
Split + Two regret	30591.00	33382.07	37192.00	31341.00	33223.19	35740.00

Tabela 1: Wyniki eksperymentu obliczeniowego

5 Wnioski

Obydwa podejścia wykorzystujące heurystyki związane z żalem przyniosły znacznie gorsze skutki niż znacznie prostsze od nich podejścia zachłanne, takie jak dodawanie najbliższego sąsiada do obecnie budowanego cyklu, czy rozbudowa cyklu o najtańszy węzeł. Z graficznej reprezentacji można wywnioskować potencjalne powody dlaczego tak się dzieje - przede wszystkim, obydwa algorytmy oparte na dwu-żalu skonstruowały rozłączne cykle które często się przecinały, podczas gdy algorytm rozbudowy cyklu o najtańszy wierzchołek budował cykle które "nie wchodziły sobie w drogę". Dzieje się tak przede wszystkim ze względu na to, że algorytm nie bierze pod uwagę tego że budujemy dwa rozłączne cykle i "nie musi żałować" połowy wierzchołków które sprawdza. W tym momencie pojawia się kilka pomysłów na rozwiązanie tego problemu!
Po pierwsze, można by jakoś podzielić zbiór wierzchołków na dwa równe podzbiory i wtedy wywołać na nich osobno algorytm oparty na żalu. Można pokusić sie np. o wykorzystanie algorytmu k-means, lub dla naszego problemu zmodyfikować go aby tworzył zbalansowane klastry. Po drugie, można zmodyfikować algorytm wyko-

rzystujący dwużal o to, aby dla każdego wierzchołka wyliczany był żal dołączenia go do pierwszego cyklu, jak i do drugiego jednocześnie, a następnie branie go pod uwagę tylko dla cyklu który ma mniejszy żal dołączenia go. To niestety znów powoduje problemy z balansowaniem długości obu cykli i ma tendencję do tworzenia jednego cyklu obejmującego znaczną większośc wierzchołków (>90% oraz drugiego, drobnego na kilka wierzchołków lub czasem obejmującego tylko wierzchołek startowy). Próby wymuszenia balansowania cykli niestety prowadziły do wyników bardzo zbliżonych do podstawowych algorytmów wykorzystujących dwużal oraz dwużal ważony. Znacznie lepiej jednak zadziałał algorytm z pierwszego pomysłu, mimo że nie wykorzystaliśmy w nim k-means do podziału zbioru wierzchołków a zachłanną metodę znajdywania n/2 najbliższych do wierzchołka startowego. Główny powód dla którego zdecydowaliśmy się na taką metodę podziału, to aby zostać w duchu pozostałych algorytmów i porównać ze sobą metody zachłanne.

6 Link do repozytorium

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