Inteligentne metody optymalizacji

Laboratorium nr 3

24 kwietnia 2023

Prowadzący: prof. dr hab. inż. Andrzej Jaszkiewicz

Autorzy: **Krzysztof Martin** 141275
Bartosz Paliński 145224

Zajęcia poniedziałkowe, 11:45.

Oświadczam/y, że niniejsze sprawozdanie zostało przygotowane wyłącznie przez powyższych autora/ów, a wszystkie elementy pochodzące z innych źródeł zostały odpowiednio zaznaczone i są cytowane w bibliografii.

1 Opis zadania

Celem zajęć laboratoryjnych było usprawnienie procesu lokalnego przeszukiwania w wersji stromej w celu zwiększenia jego efektywności czasowej. W tym celu wprowadzono dwa mechanizmy, które mają na celu skrócenie czasu przetwarzania:

- ruchy kandydackie
- algorytm oparty na liście ruchów

Zaimplementowane algorytmy lokalnego przeszukiwania z zastosowaniem tych mechanizmów zostały porównane z algorytmem stromym w wersji ze sąsiedztwem krawędzi oraz z algorytmem z żalem ważonym.

Efektywność algorytmów była oceniana na podstawie instancji kroaA200 i kroB200 z biblioteki TSPLib. Do uruchomienia algorytmów wykorzystano losowe rozwiązania startowe.

2 Algorytmy

2.1 Ruch aplikowalny

Dla zamiany wierzchołków pomiędzy cyklami, ruch jest aplikowalny, jeżeli:

- wierzchołki znajdują się w oddzielnych cyklach,
- sąsiadujące z nimi wierzchołki nie zostały wstawione w inne miejsce cyklu (przy zamianie wierzchołków i oraz j istnieją krawędzie $i_{prev} - i - i_{next}$ oraz $j_{prev} - j - j_{next}$)

Dla zamiany krawędzi wewnątrz cyklu, ruch jest aplikowalny, jeżeli:

- obie krawędzie istnieją,
- obie krawędzie mają ten sam kierunek.

Same istnienie połączenia między punktami nie jest wystarczające. Omawiamy sytuację, w której połączenia istnieją, ale nie mają określonego kierunku, co uniemożliwia przemieszczenie się między nimi. W takim przypadku zostawiamy to na liście.

2.2 Lokalne przeszukiwanie z pamięcią

Algorytm ten operuje na mechanizmie zapamiętywania możliwych ruchów, co umożliwia uniknięcie konieczności przeglądania całego sąsiedztwa w każdej kolejnej iteracji. Algorytm rozpoczyna swoje działanie od wygenerowania wszystkich możliwych ruchów. Następnie, na podstawie wcześniej obliczonej delty wybierany jest najlepszy ruch, który po spełnieniu warunków aplikowalności zostaje wykonany. W przypadku gdy ruch okaże się nieaplikowalny, zostaje on usunięty z listy. Wyjątkiem jest wymiana krawędzi gdy mają one przeciwne kierunki, taki ruch może okazać się aplikowalny w późniejszej iteracji, dlatego zostaje pominięty (a po wykonaniu innego ruchu z powrotem trafia na listę). Algorytm kończy działanie, gdy nie ma już więcej ruchów do rozpatrzenia.

1: *moves* = utwórz kolejkę priorytetową (z rosnącą wartością *delta*) zawierającą listę wszystkich możliwych ruchów
2: **dopóki** *moves* nie jest puste **powtarzaj**:
3: *move* = najlepszy ruch z *moves*
4: **Jeżeli** *move* jest aplikowalny:

wykonaj *move*
dodaj do kolejki *moves* wszystkie nowe możliwe ruchy
usuń *move* z kolejki *moves*
przywróć do kolejki *moves* wszystkie pominięte ruchy
Jeżeli *move* jest tymczasowo nieaplikowalny:
pomiń *move*
Jeżeli *move* jest nieaplikowalny:
usuń *move* z kolejki *moves*

2.3 Ruchy kandydackie

Algorytm rozpoczyna się od wyznaczenia dla każdego wierzchołka jego 10 najbliższych sąsiadów. Między tymi wierzchołkami zostają wyznaczone krawędzie kandydackie. Dla każdej krawędzi analizowane są przypadki wstawienia jej do istniejących cykli.

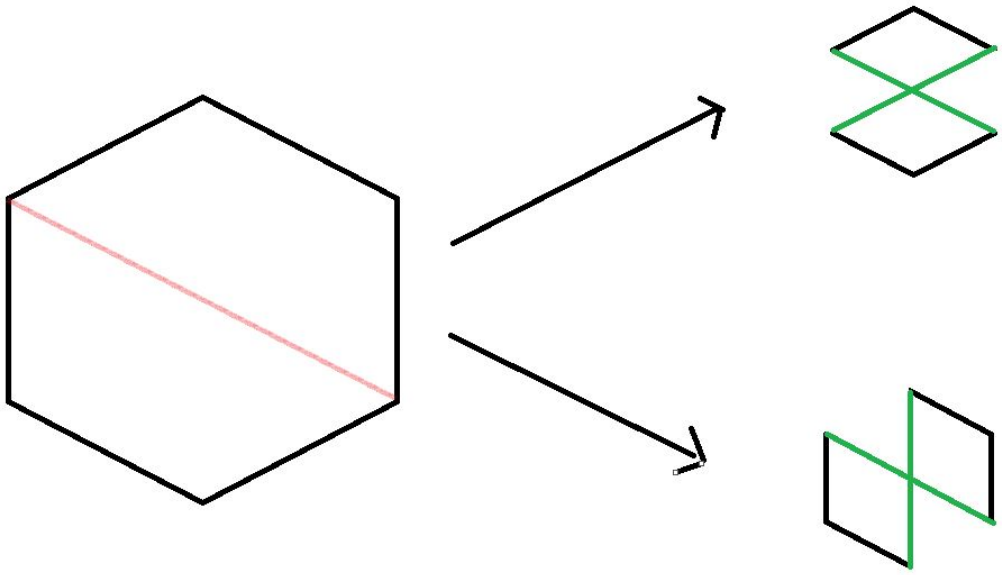
Jeżeli oba wierzchołki znajdują się w tym samym cyklu to analizujemy 2 możliwe sposoby wprowadzenia tej krawędzi do cyklu i wybieramy najkrótszą. Na rysunku 1 przedstawiono ilustrację, dlaczego istnieją 2 rodzaje wprowadzenia danej krawędzi.

Jeżeli wierzchołki należą do 2 różnych cykli to rozpatrujemy 2 sposoby zamiany wierzchołków między cyklami, które pozwalają na wprowadzenie krawędzi do cyklu.

Powtarzaj:
Wyznacz dla każdego wierzchołka 10 najbliższych jego sąsiadów.
bestMove = None
LocalBestMove = None
Dla każdego wierzchołka v1 :
    Dla każdego wierzchołka v2 z listy najbliższych sąsiadów v1
        Jeżeli v1 i v2 należą do tego samego cyklu
            Oblicz deltę funckcji celu dla pierwszej możliwości zamiany krawędzi w cyklu
            Oblicz deltę funckcji celu dla drugiej możliwości zamiany krawędzi w cyklu
            Za LocalBestMove postaw mniejszą z dwóch wartości
            Jeżeli LocalBestMove <bestMove
                bestMove = LocalBestMove

        Jeżeli v1 i v2 należą do róznych cykli
            Oblicz deltę funckcji celu dla pierwszej możliwości zamiany wierzchołków między cyklami
            Oblicz deltę funckcji celu dla drugiej możliwości zamiany wierzchołków między cyklami
            Za LocalBestMove postaw mniejszą z dwóch wartości
            Jeżeli LocalBestMove <bestMove
                bestMove = LocalBestMove

Jeżeli bestMove == None :
    Zakończ algorytm
W przeciwnym wypadku:
    Aplikuj ruch bestMove



Rysunek 1: Schemat ilustrujący 2 możliwości wprowadzenia krawędzi czerwonej do cyklu.

3 Wyniki eksperymentu obliczeniowego

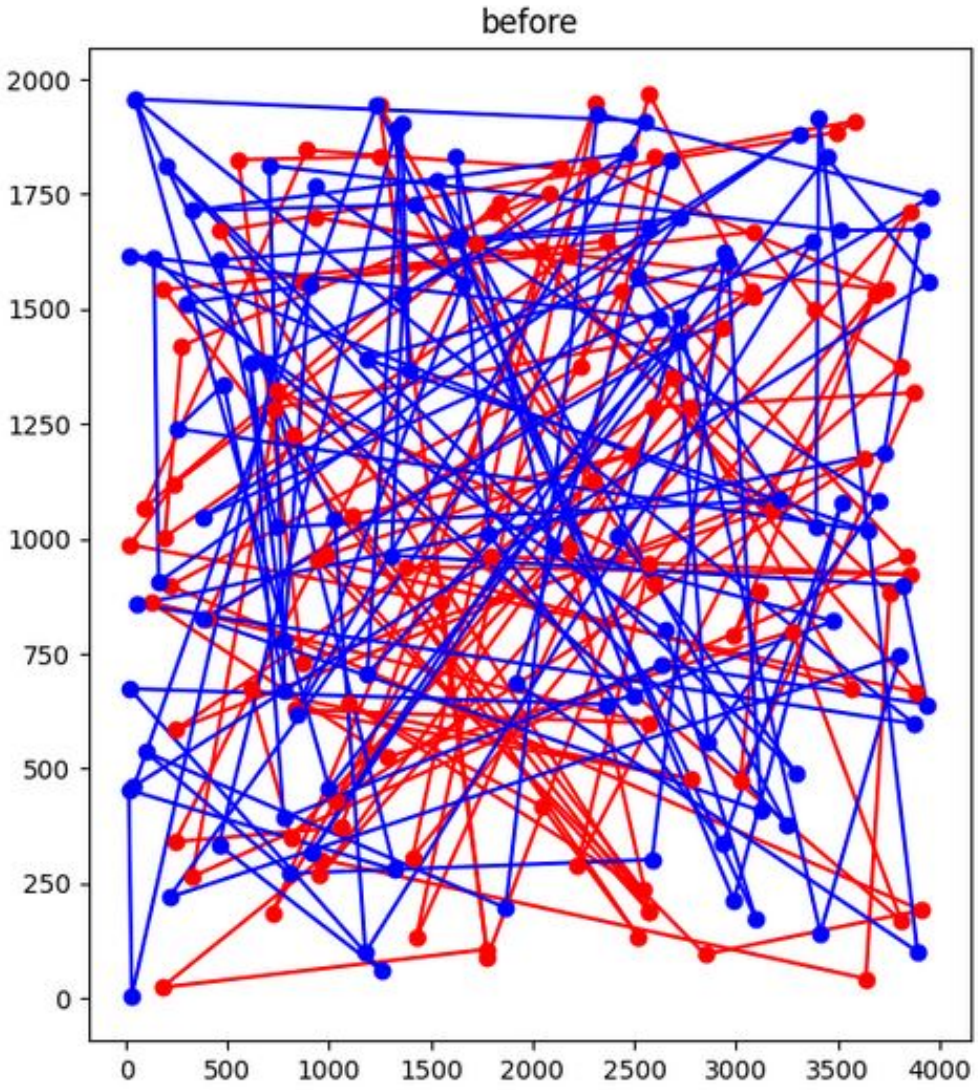
Każdy algorytm został przetestowany 100 razy dla każdej instancji, za każdym razem generując generując inne losowe rozwiązanie początkowe. Średnie, najlepsze oraz najgorsze wyniki są przedstawione w poniższej tabeli wraz z czasami wykonywania.

Tabela 1: Losowe rozwiązanie początkowe (LS - Local search)

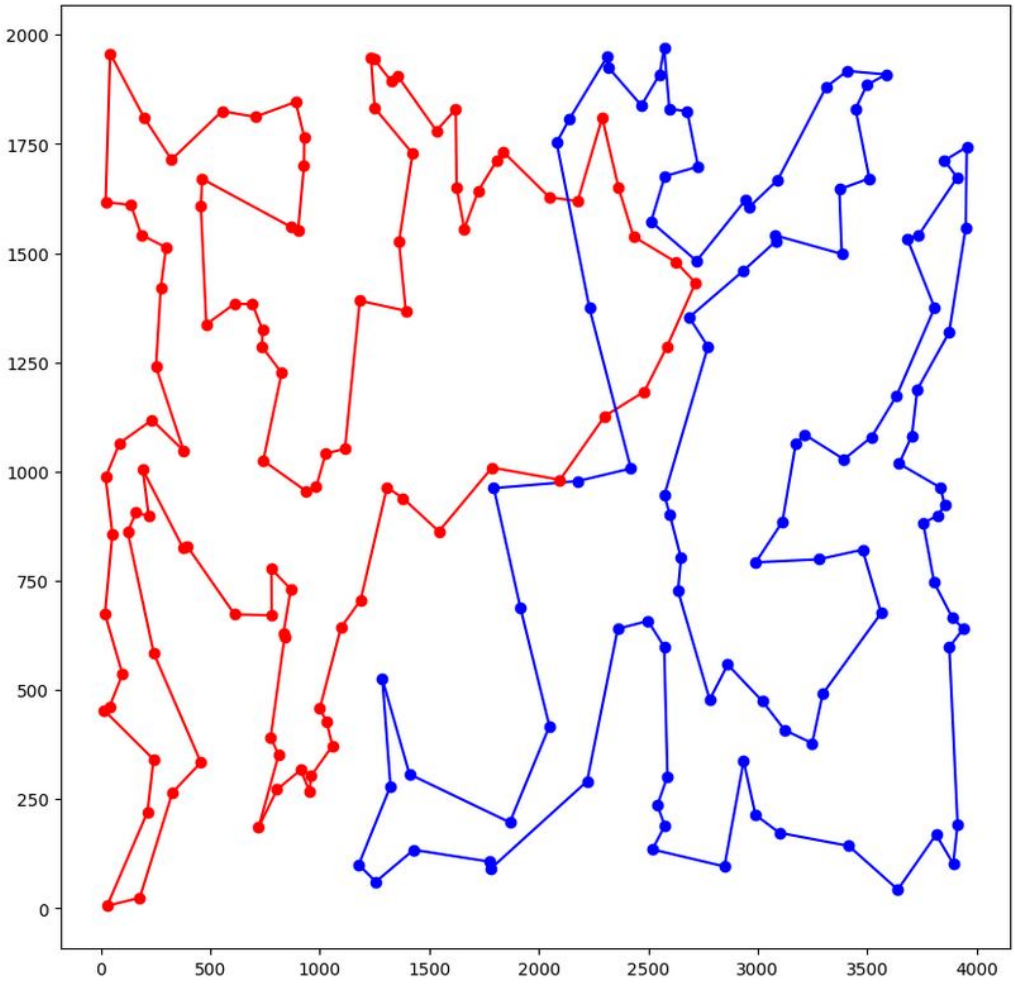
Lokalne przeszukanie	Instancja kroA200	Instancja kroB200
Random	338767 (311756 - 366842)	346899 (297563 - 386124)
Ważony 2-żal	35786 (33061 - 39211) - 208,5ms	36470 (32409 - 39274) - 220,1ms
LS wersja stroma	38544 (35987 - 40823) - 597,7 ms	38738 (36710 - 42066) - 597,7 ms
LS z pamięcią	38564 (35166 - 40916) - 260,2 ms	38576 (35636 - 41038) - 260,2 ms
Ruchy kandydackie	39069 (36518 - 41081) - 137,5 ms	39017 (36374 - 41043) - 138,8 ms

4 Wizualizacje najlepszych rozwiązań

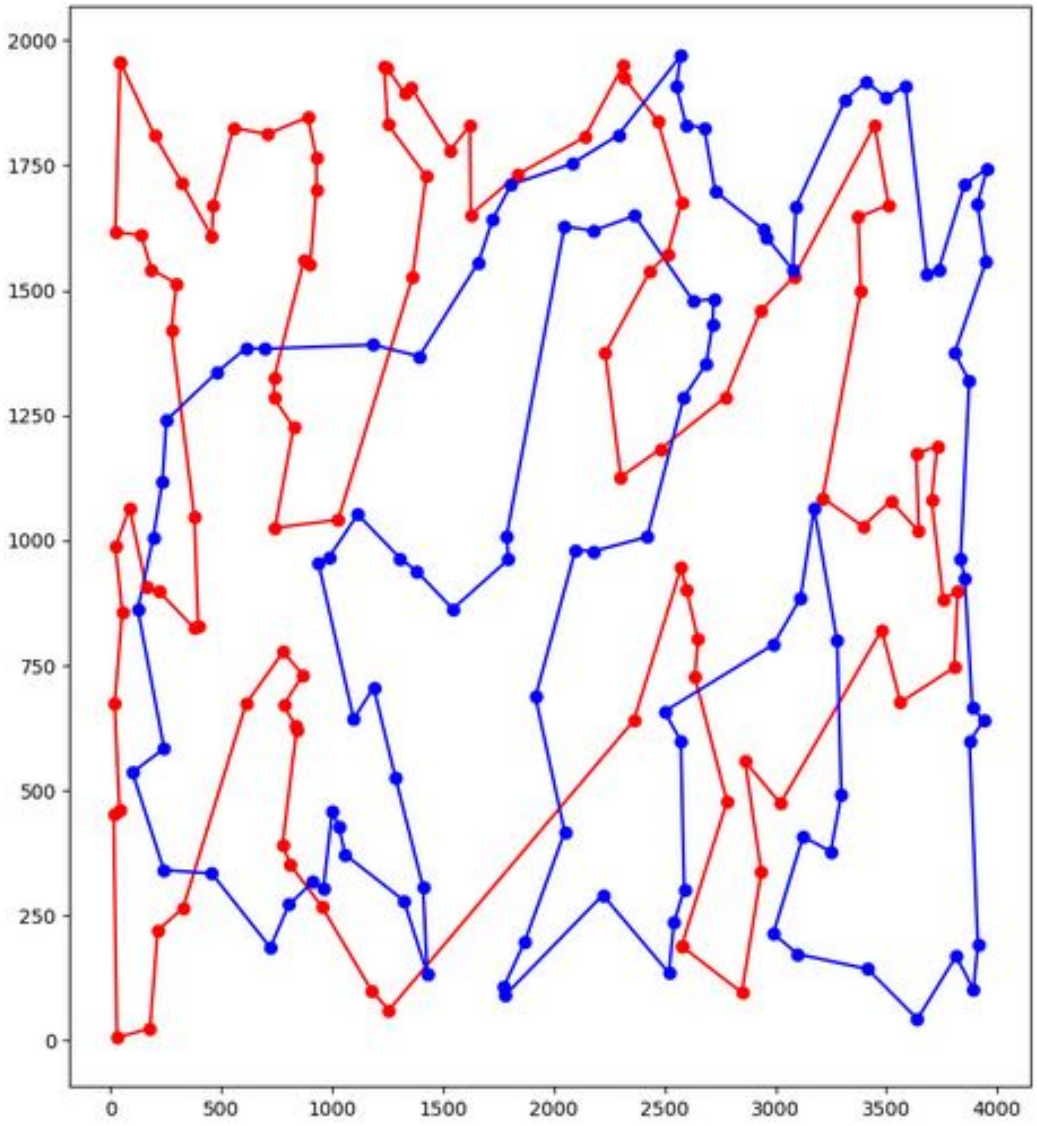
4.1 Instancja kroaA200



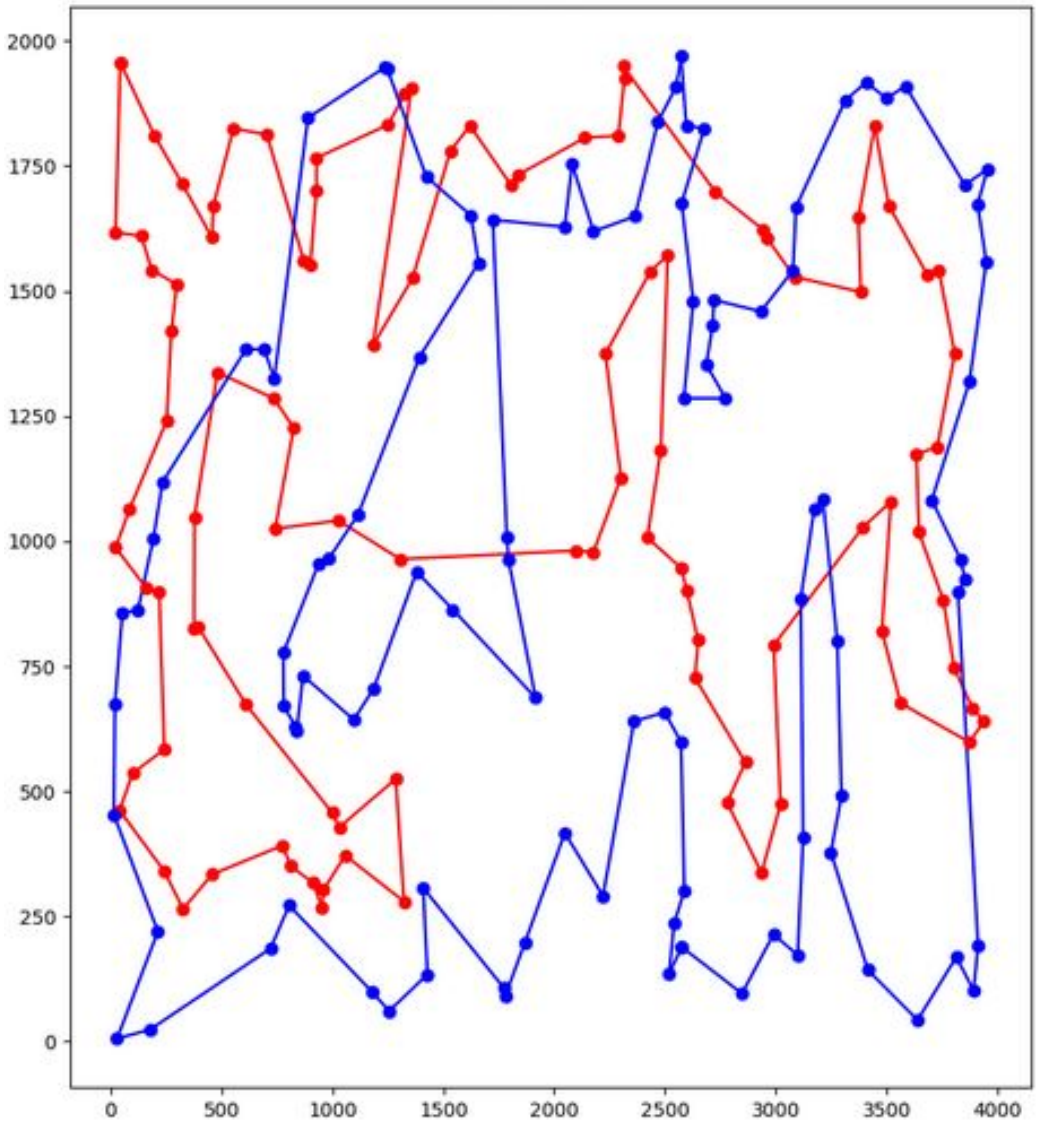
Rysunek 2: Losowe rozwiązanie



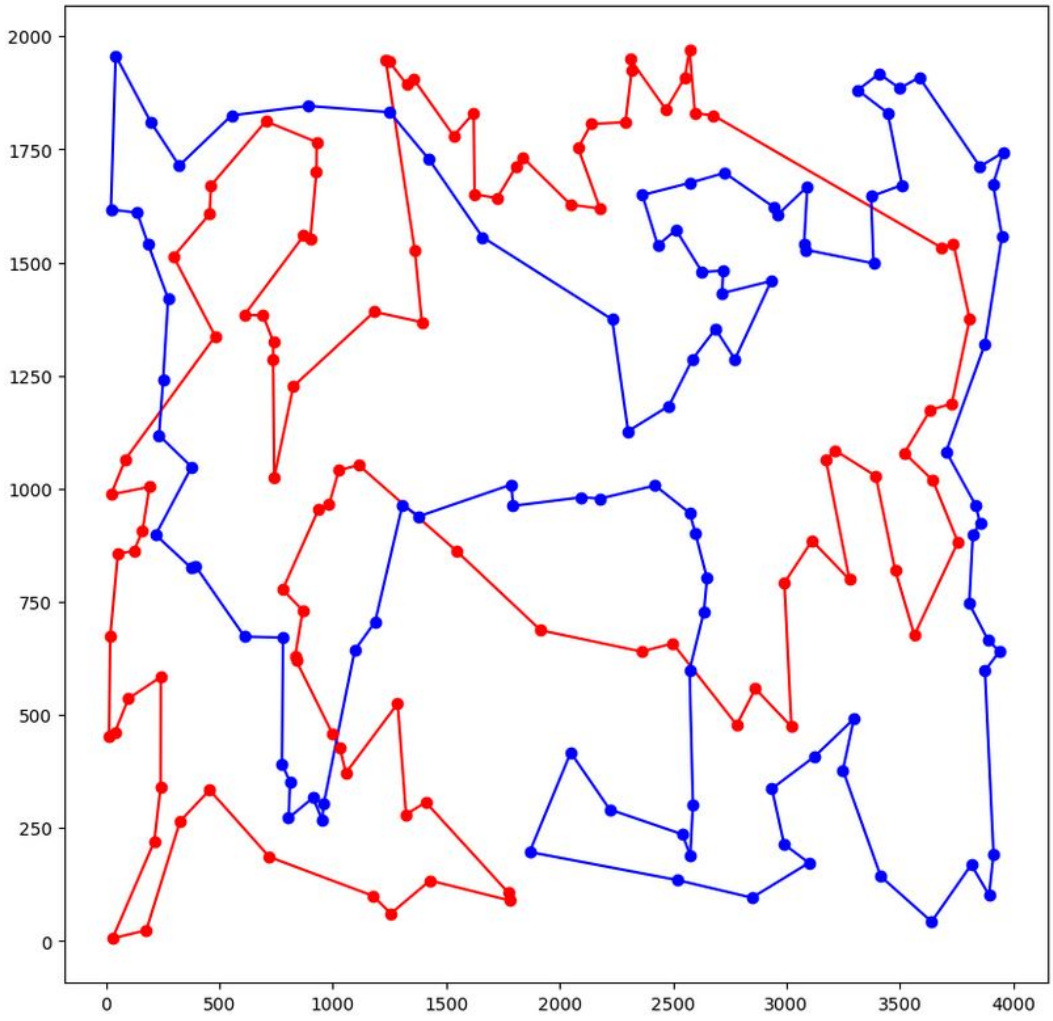
Rysunek 3: 2-żal ważony



Rysunek 4: Lokalne przeszukiwanie we wersji stromej

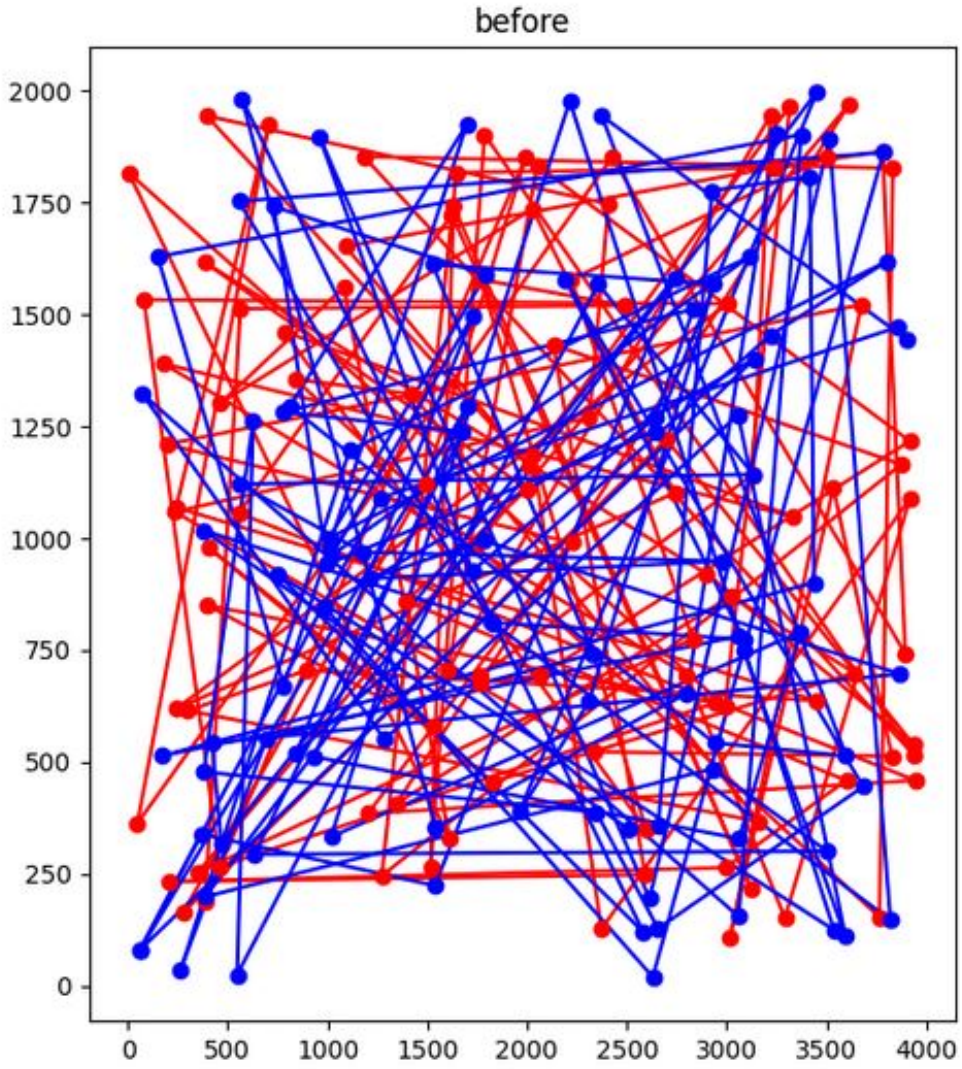


Rysunek 5: Lokalne przeszukiwanie z pamięcią ruchów

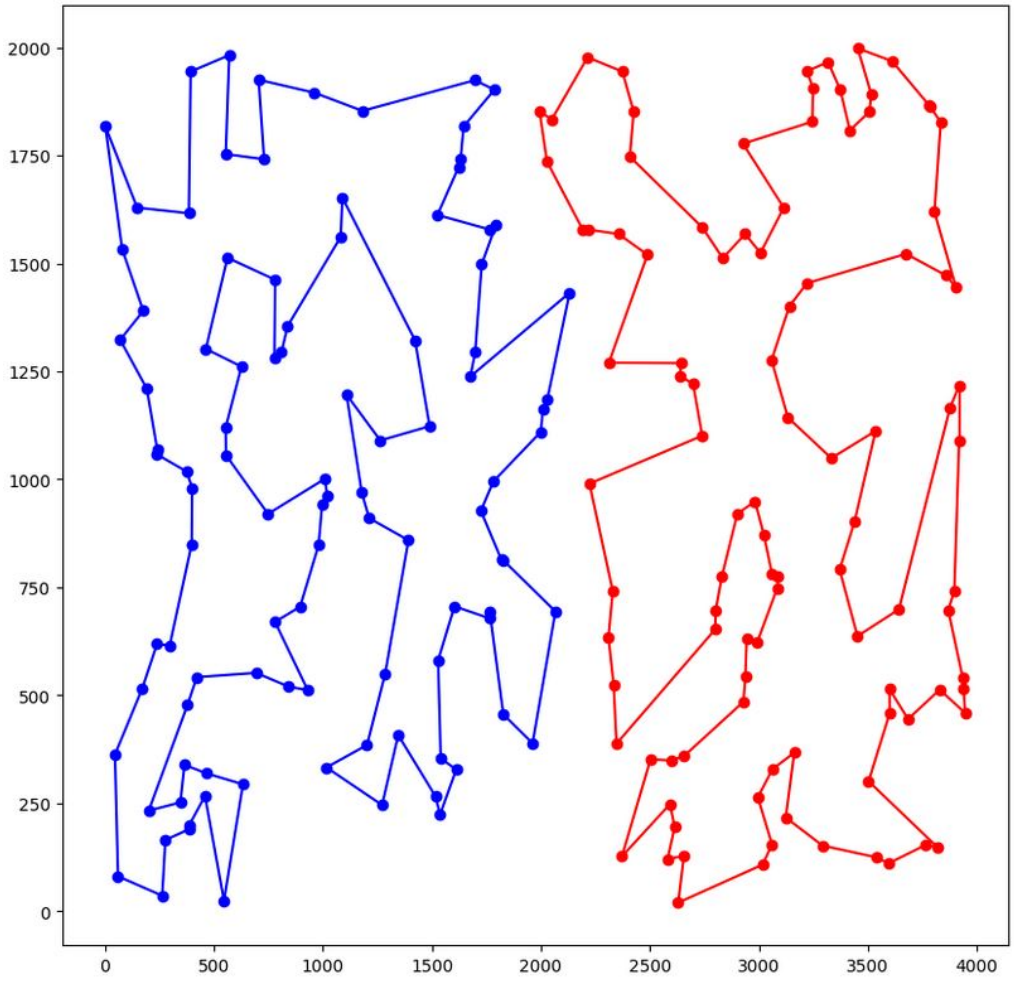


Rysunek 6: Lokalne przeszukiwanie z ruchami kandydackimi

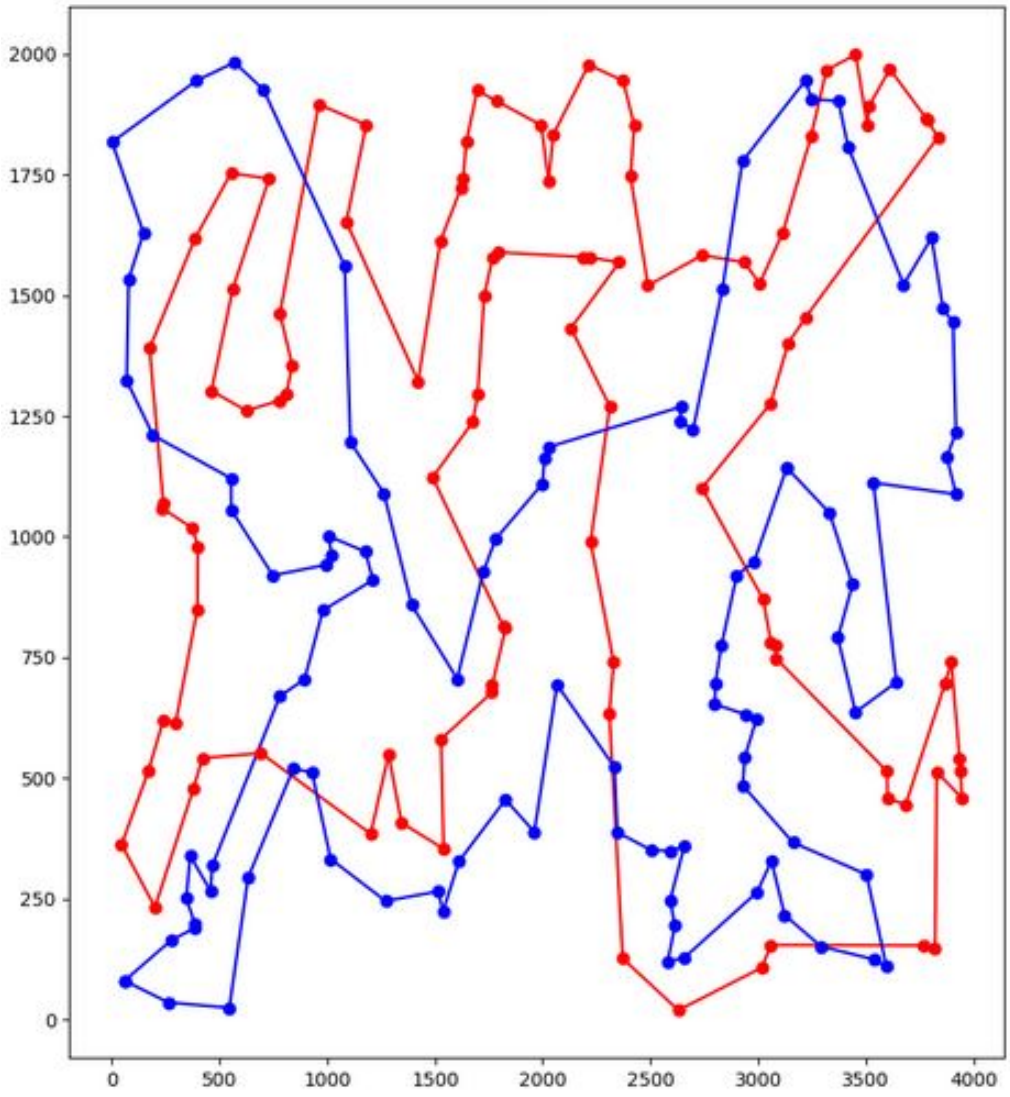
4.2 Instancja kroaB200



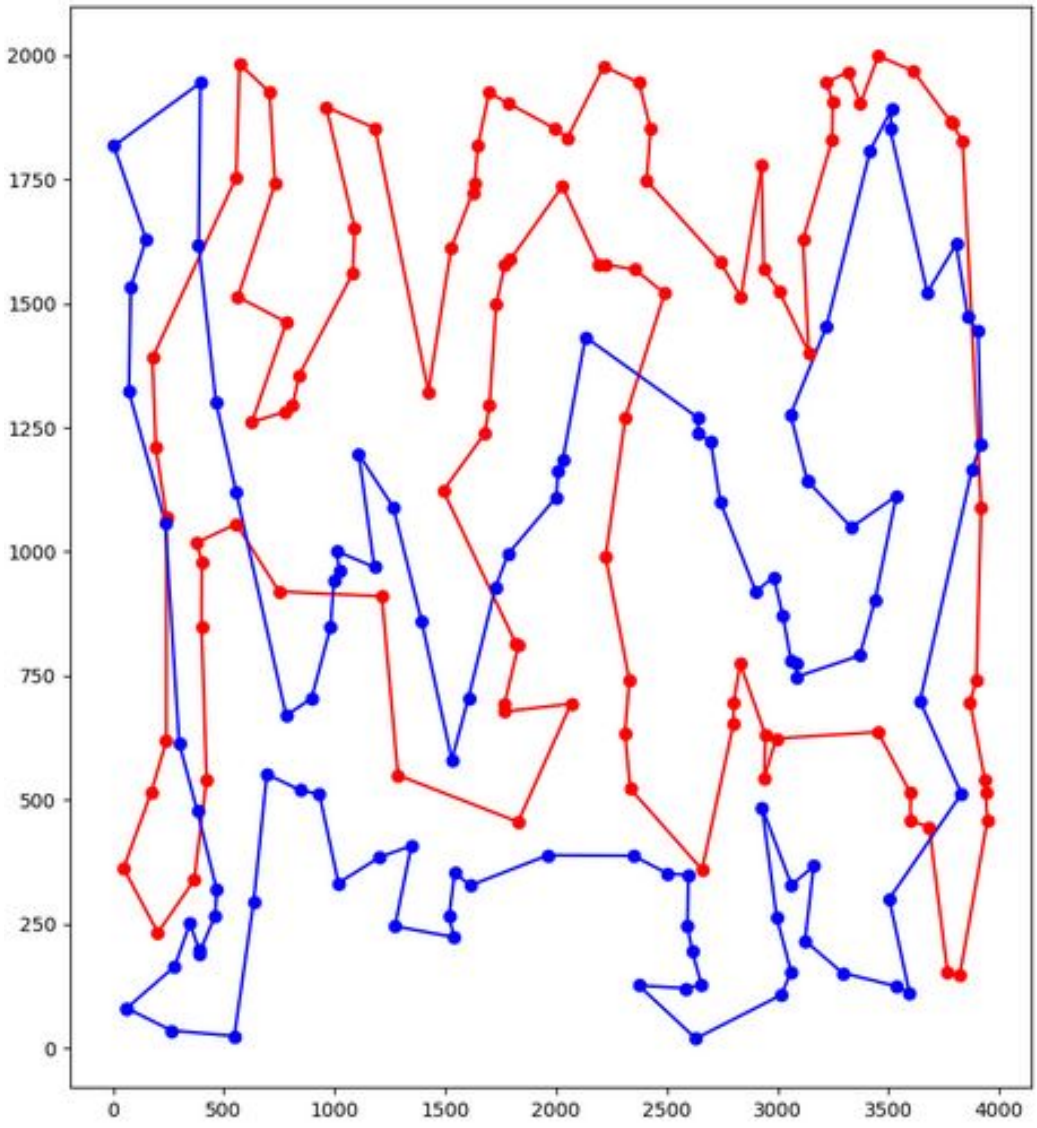
Rysunek 7: Losowe rozwiązanie



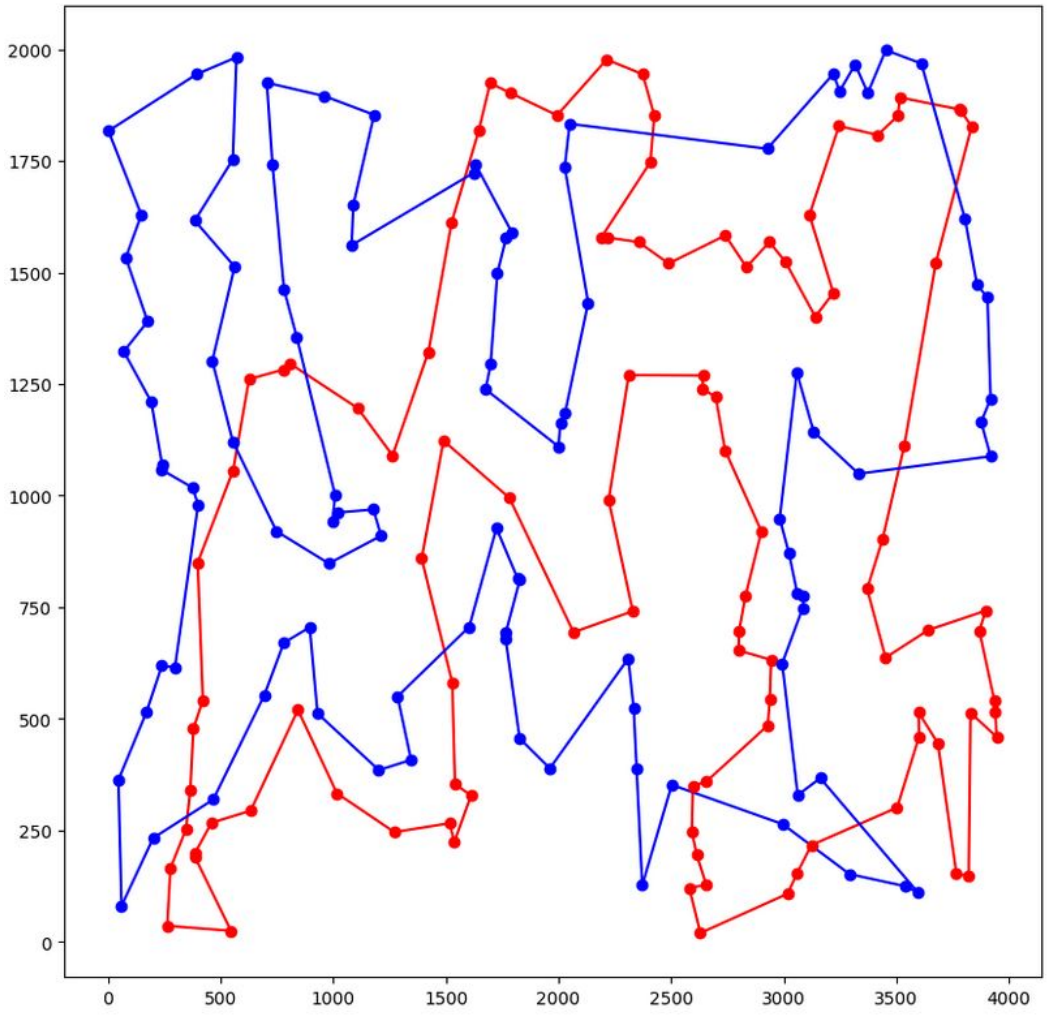
Rysunek 8: 2-żal ważony



Rysunek 9: Lokalne przeszukiwanie we wersji stromej



Rysunek 10: Lokalne przeszukiwanie z pamięcią ruchów



Rysunek 11: Lokalne przeszukiwanie z ruchami kandydackimi

5 Wnioski

- W przypadku obu przedstawionych metod optymalizacji zauważyć można skrócenie czasu trwania obliczeń. Średnio 2-krotne w przypadku przeszukiwania z pamięcią, a 4-krotne w przypadku ruchów kandydackich.
- Przeszukiwanie z pamięcią osiąga wyniki zbliżone do przeszukiwania w wersji stromej. Ruchy kandydackie osiągają wyniki minimalnie gorsze niż wersja stroma.
- Wszystkie metody przeszukiwania sąsiedztwa osiągnęły wyniki gorsze niż przedstawiona heurystyka konstrukcyjna (ważony 2-żal).

6 Kod programu

Kod źródłowy dostępny on-line: https://github.com/barosz0/IMO/tree/main/lab3