

## Zadanie 2. Lokalne przeszukiwanie

Oskar Kiliańczyk 151863 & Wojciech Kot 151876

# 1 Opis zadania

Zadanie polega na implementacji lokalnego przeszukiwania w wersjach stromej (steepest) i zachłannej (greedy), z dwoma różnymi rodzajami sąsiedztwa, startując albo z rozwiązań losowych, albo z rozwiązań uzyskanych za pomocą jednej z heurystyk opracowanych w ramach poprzedniego zadania. W sumie 8 kombinacji — wersji lokalnego przeszukiwania. Jako punkt odniesienia należy zaimplementować algorytm losowego błędzenia, który w każdej iteracji wykonuje losowo wybrany ruch (niezależnie od jego oceny) i zwraca najlepsze znalezione w ten sposób rozwiązanie. Algorytm ten powinien działać w takim samym czasie jak średnio najwolniejsza z wersji lokalnego przeszukiwania.

## 1.1 Sąsiedztwa

W przypadku rozważanego problemu potrzebne będą dwa typy ruchów:

- ruchy zmieniające zbiory wierzchołków tworzące dwa cykle,
- ruchy wewnątrztrasowe, które jedynie zmieniają kolejność wierzchołków na trasie.

Stosujemy dwa rodzaje ruchów wewnątrztrasowych (jeden albo drugi, stąd dwa rodzaje sąsiedztwa). Jeden to wymiana dwóch wierzchołków wchodzących w skład trasy, drugi to wymiana dwóch krawędzi. Dla ruchów wewnątrz cykli wykonujemy lokalne zamiany elementów w obrębie jednej ścieżki.

## 1.2 Randomizacja kolejności przeglądania dla algorytmów zachłannych

W obu wersjach algorytmów zachłannych stosujemy randomizację wyboru. Tworzymy listę możliwych par punktów do zamiany wewnątrz cyklu lub między dwoma cyklami. Losowo przetasowujemy te możliwe pary, dzięki czemu kolejność przetwarzania nie jest deterministyczna. Wybieramy pary do wymiany aż nie znajdziemy takiej, dla której zamiana daje poprawę funkcji celu. Jeśli taka istnieje, przeprowadzamy zamianę i powtarzamy procedurę, jeśli nie ma takiej pary - kończymy przetwarzanie.

## 1.3 Algorytmy startowe

Napisane na potrzeby tego zadania algorytmy wykorzystują dwa różne algorytmy startowe (generujące rozwiązanie bazowe, które napisane obecnie algorytmy lokalnego przeszukiwania będą ulepszać). Pierwszy z nich, nazwaliśmy randomstart - zwraca on dwa losowe zbiory wierzchołków, tworząc w pełni losowo wygenerowane bazowe rozwiązanie, które będzie relatywnie proste do polepszenia. Drugi z nich natomiast, to nasz własny algorytm (nie zaproponowany w zadaniu) który łączył heurystykę żalu z zachłannym podziałem wierzchołków na dwa podzbiory.

# 2 Opisy algorytmów

## 2.1 Zachłanny

1. Dla każdego z dwóch cykli:
  - (a) Dopóki możliwa jest poprawa\*:
    - i. Wygeneruj listę możliwych par wierzchołków  $(i, j)$  wewnątrz cyklu.
    - ii. Permutuj listę i sprawdzaj kolejne pary:
      - A. Oblicz zmianę kosztu po ruchu lokalnym (zamiana wierzchołków lub odwrócenie fragmentu).
      - B. Jeśli koszt się zmniejsza, wykonaj ruch.
      - C. Przerwij sprawdzanie i wróć na początek pętli (\*).
2. Dopóki możliwa jest poprawa międzycyklowa:
  - (a) Wygeneruj losową permutację par wierzchołków  $(i, j)$  z dwóch cykli.
  - (b) Dla każdej pary:
    - i. Oblicz zmianę kosztu po zamianie wierzchołków między cyklami.
    - ii. Jeśli koszt się zmniejsza, wykonaj zamianę i zaznacz poprawę.
  - (c) Jeżeli wykonano jakąkolwiek zamianę, wróć do lokalnej optymalizacji (punkt 1).
3. Zakończ, gdy nie ma już żadnych lokalnych ani międzycyklowych poprawek.

## 2.2 Steepest

1. Dopóki możliwa jest poprawa:
  - (a) Przeszukaj wszystkie możliwe modyfikacje ścieżek:
    - zmiany lokalne w jednej ścieżce (zamiana dwóch wierzchołków lub odwrócenie fragmentu),
    - wymiany wierzchołków między ścieżkami.
  - (b) Wybierz modyfikację dającą największą poprawę.
  - (c) Wprowadź ją do odpowiedniej ścieżki lub ścieżek.
2. Zwróć ulepszone ścieżki.

## 3 Wyniki

### 3.1 Tabela wynikowa

Instance Avg Diff	Algorytm	Best	Avg	Worst	Avg Time	Best Diff
kroA200 297870	greedy_edge	38683	43477.2	46597	0.0911878	323996
kroA200 262294	greedy_vertex	65591	79580.1	89485	0.227998	291579
kroA200 298626	steepest_edge	38261	42868	49455	1.38019	335537
kroA200 263108	steepest_vertex	65990	79327.8	91208	2.42133	300881
kroA200 8544.31	random	301648	332569	354465	2.42166	38018

Tabela 1: Wyniki dla kroA200 z algorytmem randomstart

Instance Avg Diff	Algorytm	Best	Avg	Worst	Avg Time	Best Diff
kroA200 203.38	greedy_edge	30293	32567.1	37008	0.0146494	1159
kroA200 285.68	greedy_vertex	30310	32608.6	36369	0.022078	2613
kroA200 479.23	steepest_edge	30426	32421.1	36148	0.0430189	3109
kroA200 324.68	steepest_vertex	30293	32475.8	37008	0.0496467	2552
kroA200 0	random	30293	33024.9	36098	2.42141	0

Tabela 2: Wyniki dla kroA200 z naszym algorytmem

Instance Avg Diff	Algorytm	Best	Avg	Worst	Avg Time	Best Diff
kroB200 291367	greedy_edge	38866	43465.6	46581	0.0910784	321463
kroB200 253942	greedy_vertex	70839	78826.1	88908	0.232368	291703
kroB200 291400	steepest_edge	38556	43203.3	48639	1.36221	313027
kroB200 254883	steepest_vertex	67856	78534	90919	2.35443	280540
kroB200 8461.69	random	295224	324648	351050	2.35485	38466

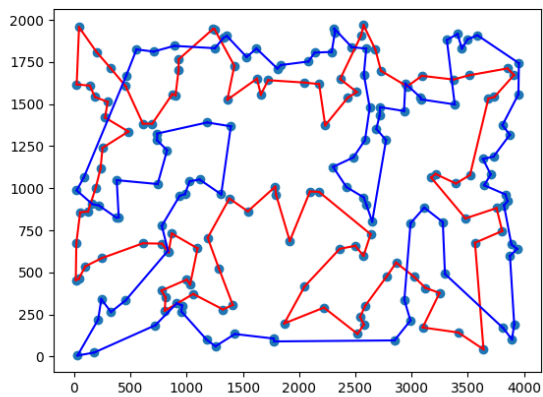
Tabela 3: Wyniki dla kroB200 z algorytmem randomstart

Instance Avg Diff	Algorytm	Best	Avg	Worst	Avg Time	Best Diff
kroB200 501.29	greedy_edge	31009	33192.1	36480	0.0174221	3629
kroB200 330.03	greedy_vertex	31178	33141.9	36479	0.0225539	2879
kroB200 495.55	steepest_edge	31133	32820.3	36479	0.0346969	4046
kroB200 254.42	steepest_vertex	31368	33197.1	36010	0.0385854	2879
kroB200 0	random	31526	33517.7	36555	2.35449	0

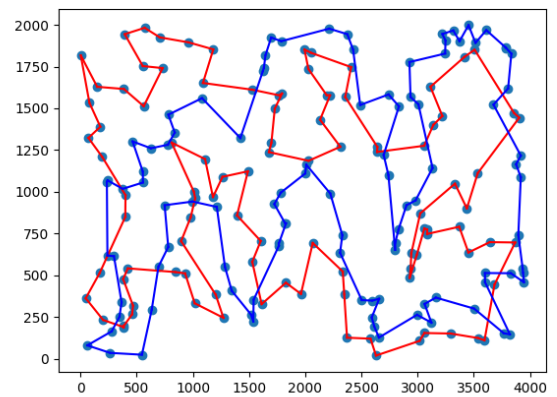
Tabela 4: Wyniki dla kroB200 z naszym algorytmem

## 3.2 Wizualizacja wyników

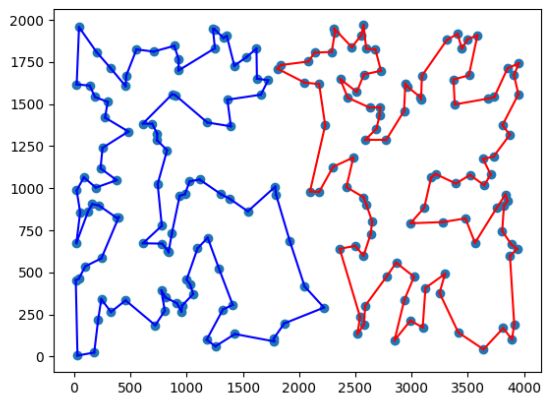
### 3.2.1 Algorytm wymiany krawędzi (zachłanny)



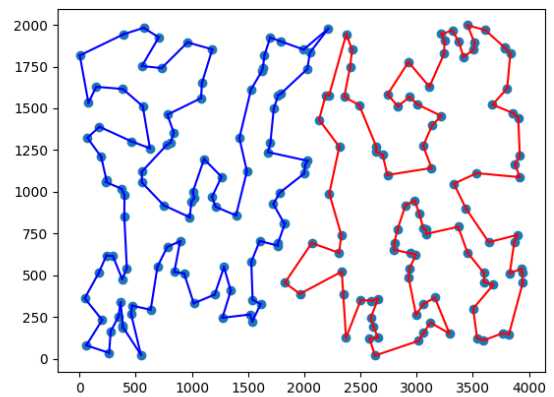
Rysunek 1: kroA200, losowy start



Rysunek 2: kroB200, losowy start

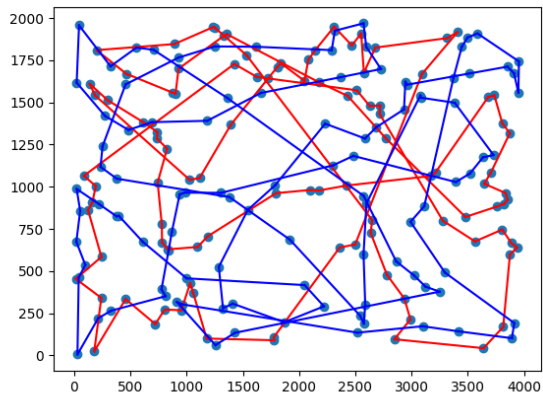


Rysunek 3: kroA200, własny algorytm startowy

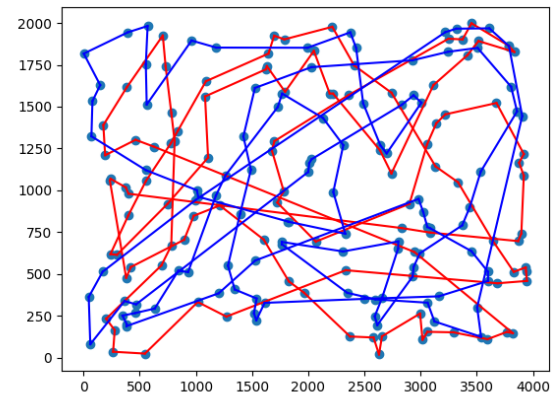


Rysunek 4: kroB200, własny algorytm startowy

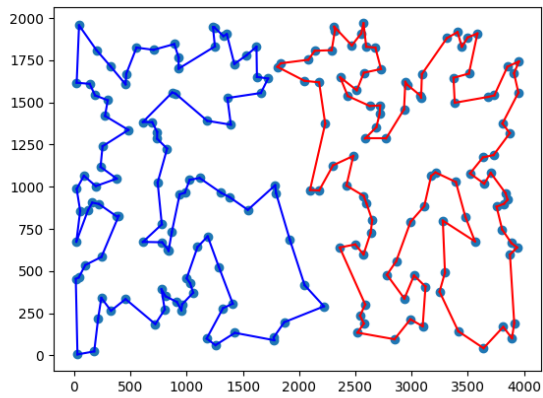
### 3.2.2 Algorytm wymiany wierzchołków (zachłanny)



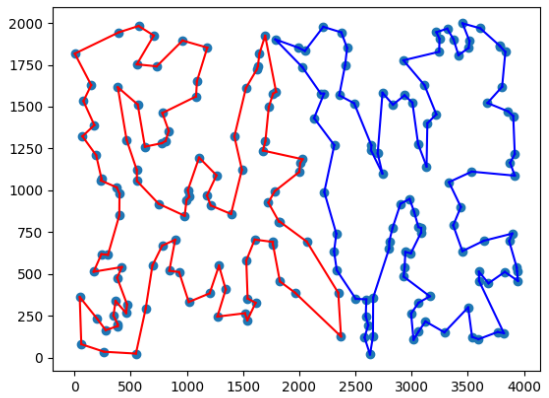
Rysunek 5: kroA200, losowy start



Rysunek 6: kroB200, losowy start

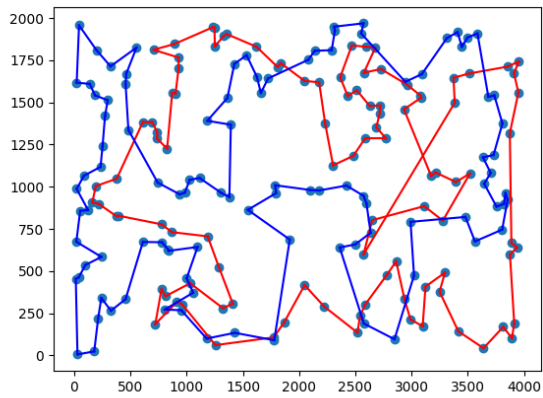


Rysunek 7: kroA200, własny algorytm startowy

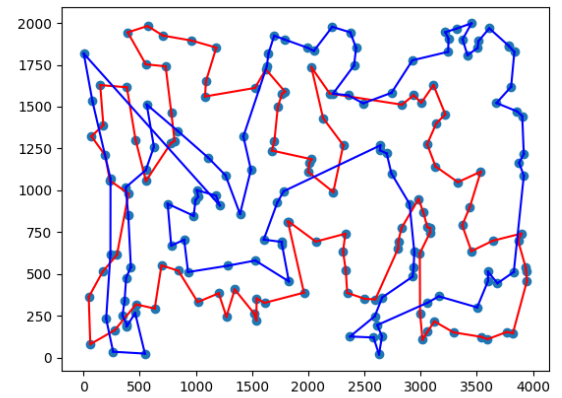


Rysunek 8: kroB200, własny algorytm startowy

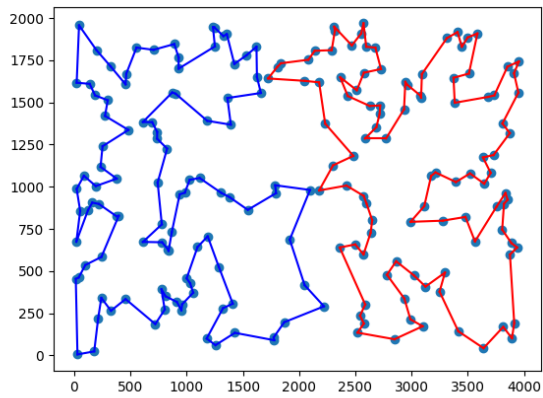
### 3.2.3 Algorytm wymiany krawędzi (steepest)



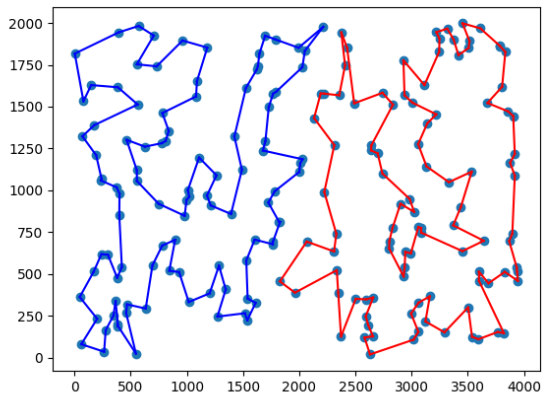
Rysunek 9: kroA200, losowy start



Rysunek 10: kroB200, losowy start

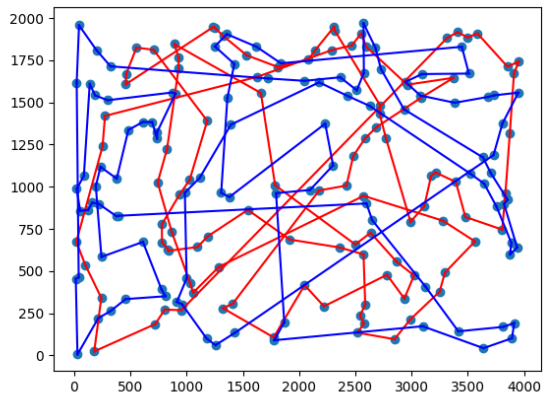


Rysunek 11: kroA200, własny algorytm startowy

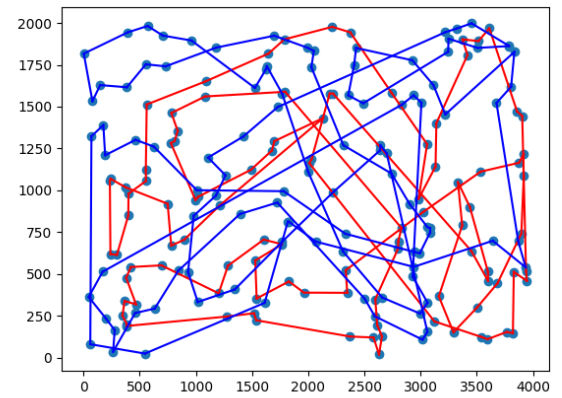


Rysunek 12: kroB200, własny algorytm startowy

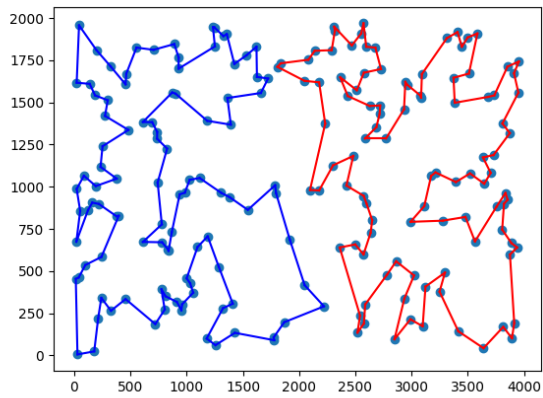
### 3.2.4 Algorytm wymiany wierzchołków (steepest)



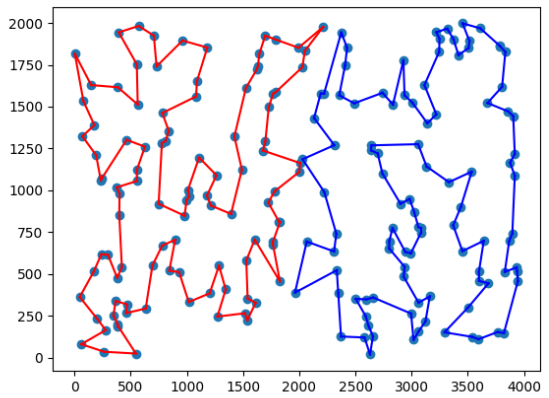
Rysunek 13: kroA200, losowy start



Rysunek 14: kroB200, losowy start



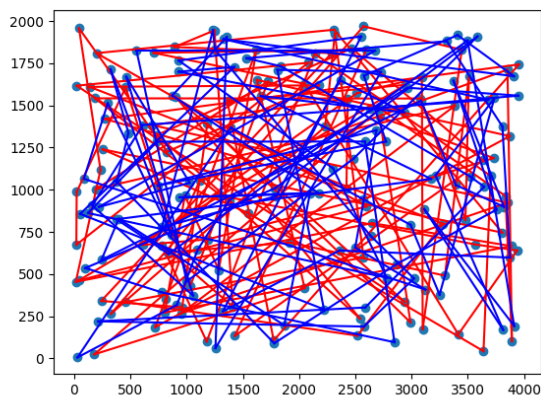
Rysunek 15: kroA200, własny algorytm startowy



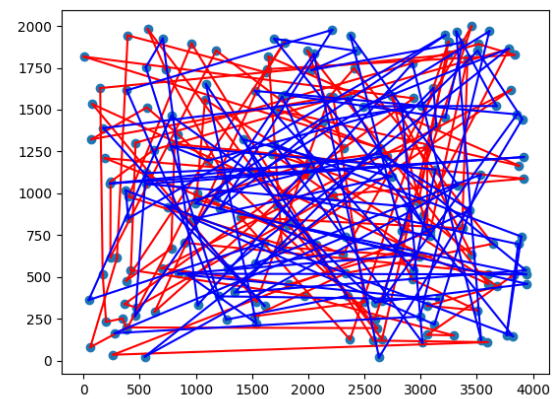
Rysunek 16: kroB200, własny algorytm startowy



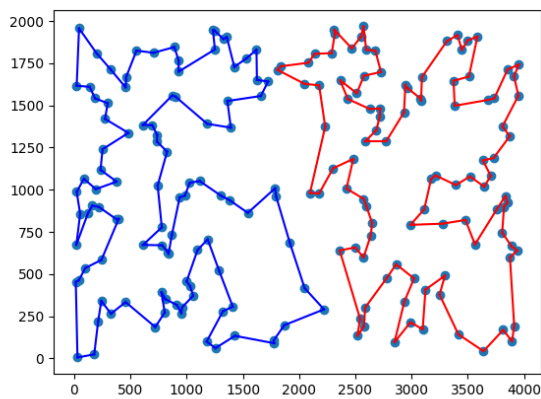
### 3.2.5 Algorytm losowego błędzenia w obu typach sąsiedztwa (random)



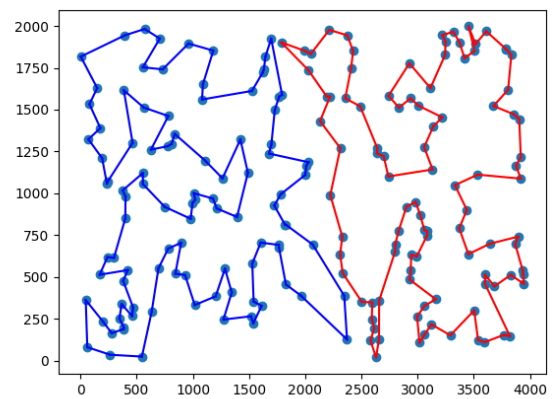
Rysunek 17: kroA200, losowy start



Rysunek 18: kroB200, losowy start



Rysunek 19: kroA200, własny algorytm startowy



Rysunek 20: kroB200, własny algorytm startowy

## 4 Link do repozytorium

Kod źródłowy w repozytorium GitHub dostępny pod linkiem:  
Repozytorium Local Search.