

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 przetwarzamy 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.

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 | Algorytm | Best | Avg | Worst | Avg Time | Best Diff | Avg Diff |
|-------------|--------------------------|--------|---------|--------|----------|-----------|----------|
| kroA200.tsp | traverse_greedy_edge | 39458 | 43643.3 | 46577 | 0.0010 | 320561 | 297093 |
| kroA200.tsp | traverse_greedy_vertex | 69190 | 80372.9 | 94789 | 0.0026 | 286837 | 258182 |
| kroA200.tsp | traverse_steepest_edge | 39408 | 43325.3 | 50129 | 0.0137 | 325160 | 297835 |
| kroA200.tsp | traverse_steepest_vertex | 65888 | 79686.9 | 91978 | 0.0325 | 287177 | 259582 |
| kroA200.tsp | traverse_random | 302110 | 331199 | 370208 | 0.0325 | 41606 | 9761.55 |
| kroB200.tsp | traverse_greedy_edge | 38993 | 43099.6 | 45870 | 0.0010 | 317848 | 289345 |
| kroB200.tsp | traverse_greedy_vertex | 66490 | 79094.8 | 92281 | 0.0017 | 294355 | 252270 |
| kroB200.tsp | traverse_steepest_edge | 39541 | 42884.2 | 47293 | 0.0146 | 315923 | 289159 |
| kroB200.tsp | traverse_steepest_vertex | 66197 | 78607.4 | 93626 | 0.0286 | 287842 | 254715 |
| kroB200.tsp | traverse_random | 293102 | 323943 | 349289 | 0.0325 | 30386 | 9234.74 |

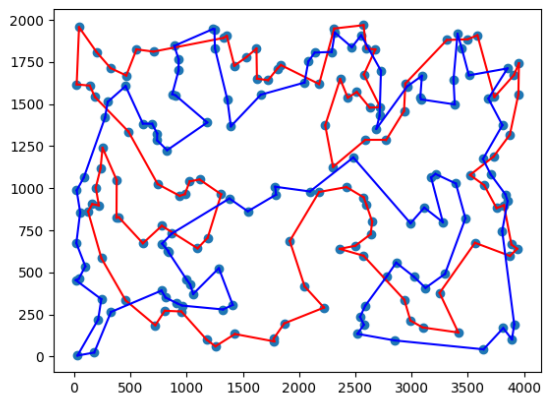
Tabela 1: Wyniki działania algorytmów dla startu typu `randomstart`

| Instance | Algorytm | Best | Avg | Worst | Avg Time | Best Diff | Avg Diff |
|-------------|--------------------------|-------|---------|-------|----------|-----------|----------|
| kroA200.tsp | traverse_greedy_edge | 30293 | 32806.3 | 37011 | 0.0157 | 2503 | 291.26 |
| kroA200.tsp | traverse_greedy_vertex | 30426 | 32858.5 | 37011 | 0.0239 | 3061 | 339.52 |
| kroA200.tsp | traverse_steepest_edge | 30512 | 32486.2 | 36552 | 0.0333 | 3109 | 315.47 |
| kroA200.tsp | traverse_steepest_vertex | 30595 | 32659.5 | 36552 | 0.0466 | 2503 | 222.65 |
| kroA200.tsp | traverse_random | 30435 | 32934.9 | 36432 | 0.1472 | 0 | 0 |
| kroB200.tsp | traverse_greedy_edge | 30863 | 32967.3 | 36040 | 0.0166 | 3437 | 456.04 |
| kroB200.tsp | traverse_greedy_vertex | 31175 | 32955 | 36480 | 0.0229 | 2848 | 349.09 |
| kroB200.tsp | traverse_steepest_edge | 30934 | 32867.7 | 36480 | 0.0351 | 4046 | 591.07 |
| kroB200.tsp | traverse_steepest_vertex | 31175 | 33043.4 | 36480 | 0.0413 | 2878 | 328.22 |
| kroB200.tsp | traverse_random | 31218 | 33461.5 | 36604 | 0.1584 | 0 | 0 |

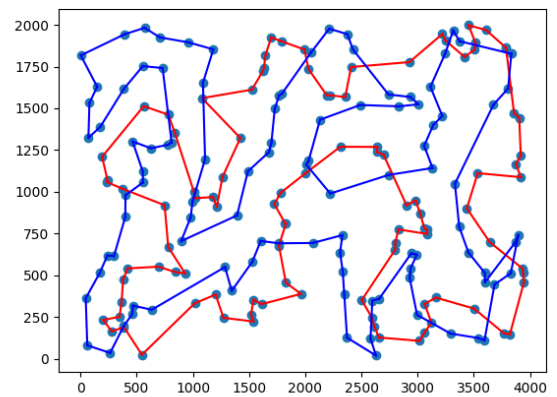
Tabela 2: Wyniki działania algorytmów dla startu typu `split_paths_regret_TSP`

3.2 Wizualizacja wyników

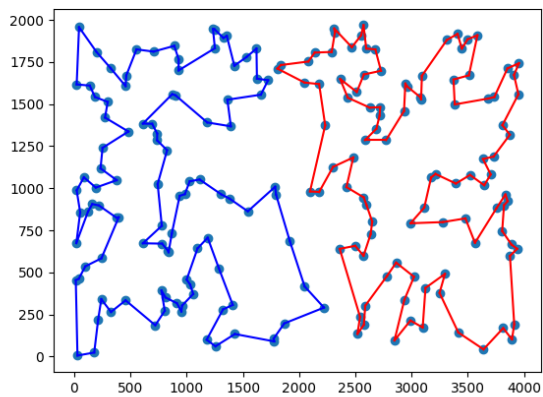
3.2.1 Algorytm wymiany krawędzi (zachłanny)



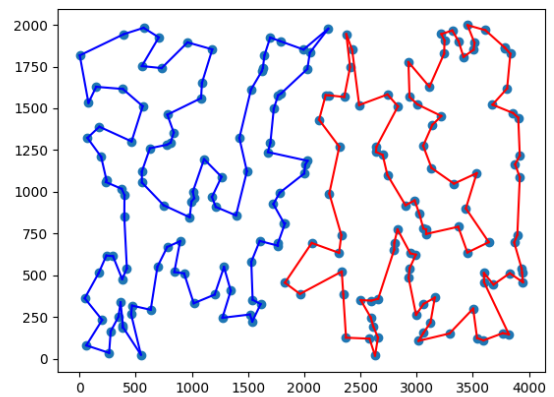
Rysunek 1: kroA200, losowy start



Rysunek 2: kroB200, losowy start

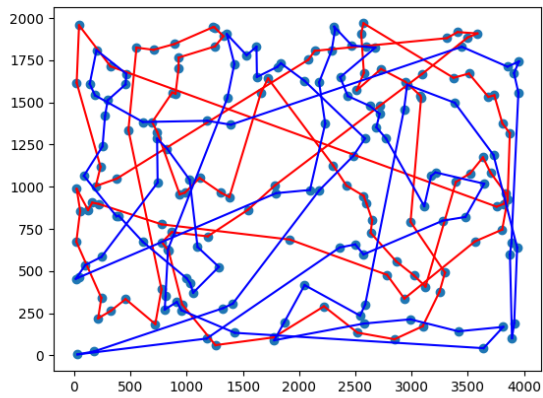


Rysunek 3: kroA200, własny algorytm startowy

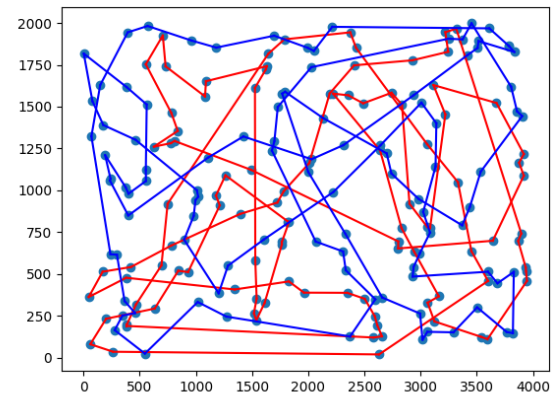


Rysunek 4: kroB200, własny algorytm startowy

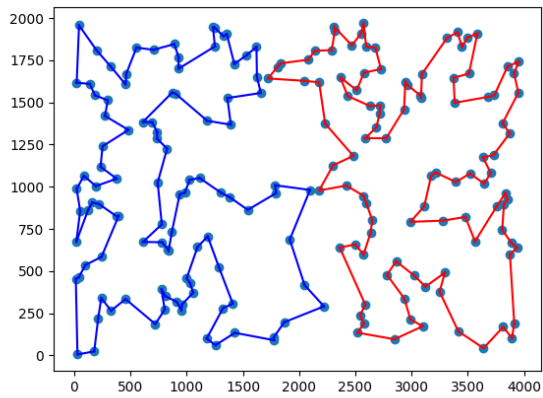
3.2.2 Algorytm wymiany wierzchołków (zachłanny)



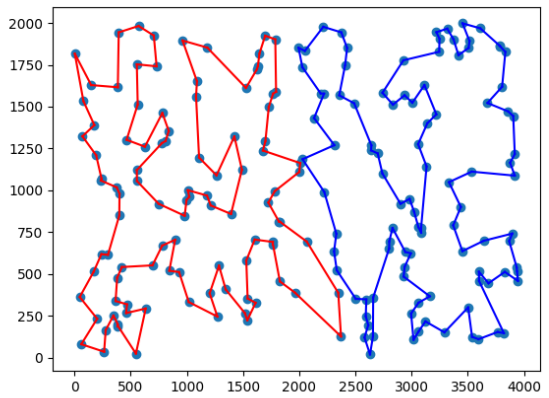
Rysunek 5: kroA200, losowy start



Rysunek 6: kroB200, losowy start

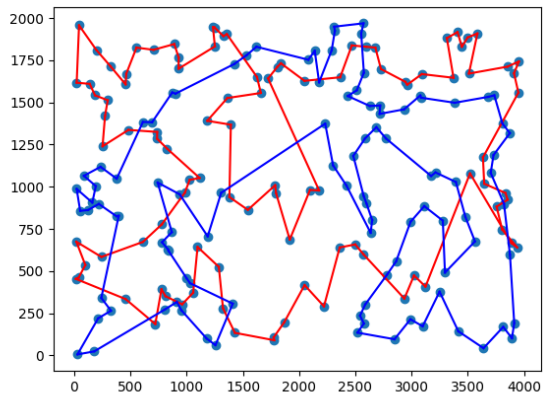


Rysunek 7: kroA200, własny algorytm startowy

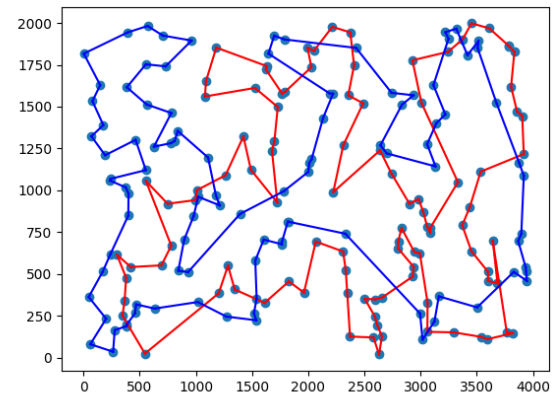


Rysunek 8: kroB200, własny algorytm startowy

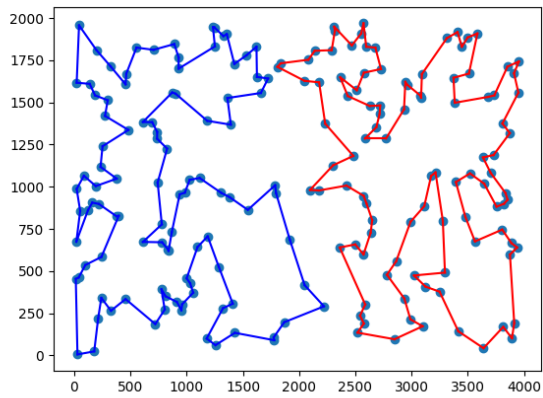
3.2.3 Algorytm wymiany krawędzi (steepest)



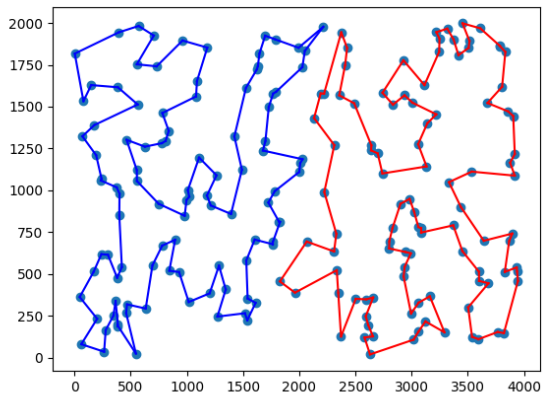
Rysunek 9: kroA200, losowy start



Rysunek 10: kroB200, losowy start

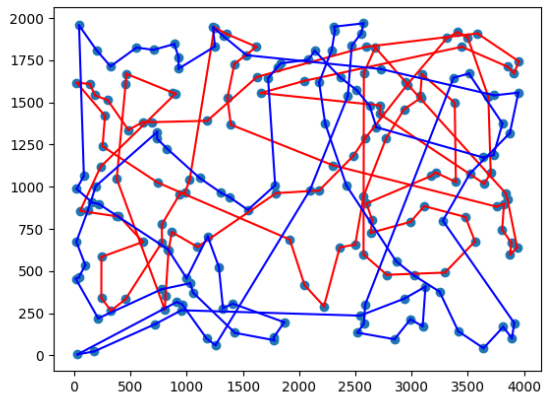


Rysunek 11: kroA200, własny algorytm startowy

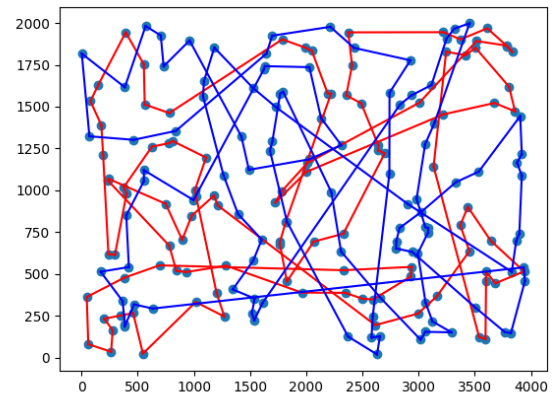


Rysunek 12: kroB200, własny algorytm startowy

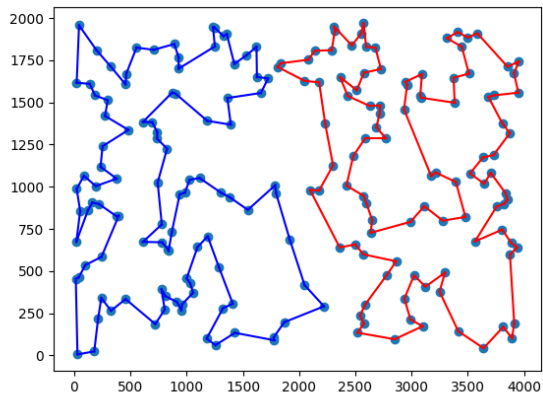
3.2.4 Algorytm wymiany wierzchołków (steepest)



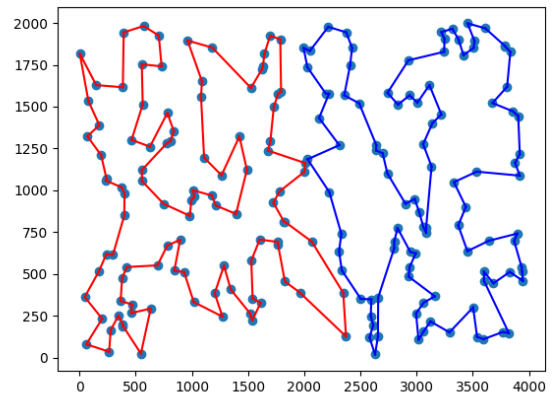
Rysunek 13: kroA200, losowy start



Rysunek 14: kroB200, losowy start

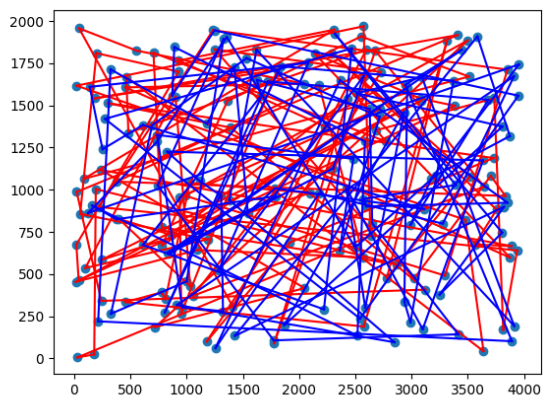


Rysunek 15: kroA200, własny algorytm startowy

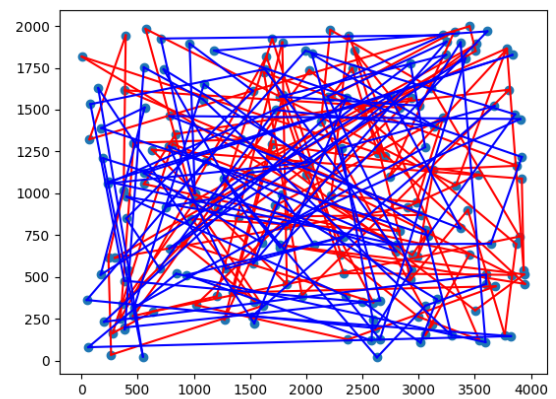


Rysunek 16: kroB200, własny algorytm startowy

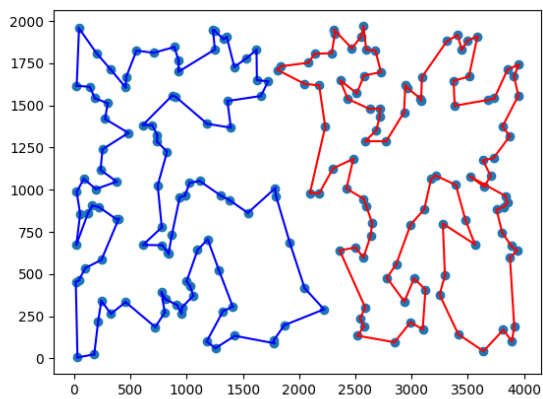
3.2.5 Algorytm losowego błędzenia w obu typach sąsiedztwa (random)



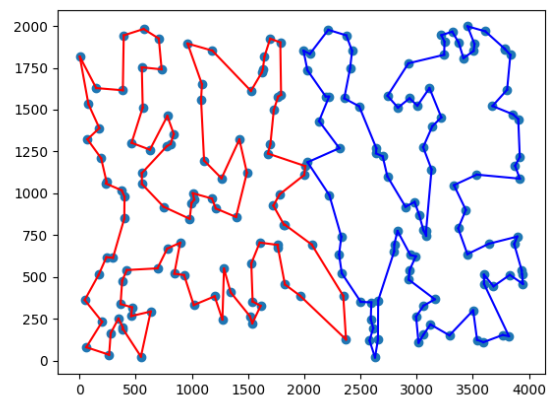
Rysunek 17: kroA200, losowy start



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.