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 różnym 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 błądzenia, 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 potrzebę 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 Opisy dostępnych ruchów

2.1.1 Obliczanie zmiany kosztu przy zamianie wierzchołków

Dla dwóch pozycji i i j w cyklu:

1. Jeśli i i j są sąsiadami:
   (a) Wyznacz wierzchołki poprzedzające i następujące po parze (i, j).
   (b) Oblicz sumę długości krawędzi przed i po zamianie miejscami i i j.
2. W przeciwnym wypadku:
   (a) Wyznacz wierzchołki poprzedzające i następujące po i i po j.
   (b) Oblicz sumę długości czterech krawędzi: przed i po zamianie i i j.
3. Zwróć różnicę: *nowy koszt* − *stary koszt*.

2.1.2 Obliczanie zmiany kosztu przy zamianie krawędzi

Dla dwóch pozycji i i j w cyklu:

(a) Wyznacz wierzchołki poprzedzające i następujące po i i po j.
(b) Oblicz sumę długości krawędzi: przed zamianą i i j.
    i. długość $i-1->i$ + długość $j->j+1$
(c) Oblicz sumę długości po zamianie krawędzi:
    i. długość $i-1->j-1$ + długość $i->j$

1. Zwróć różnicę: *nowy koszt − stary koszt.*

2.2 Algorytmy

2.2.1 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.

2.2.2 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.

2.2.3 Random Traverse

1. Zainicjuj wartość rozwiązania jako 0.
2. Dopóki czas przetwarzania jest mniejszy niż limit:
   (a) Wybierz losowy ruch z całej puli dostępnych ruchów.
   (b) Oblicz zmianę funkcji celu dla tego ruchu.
   (c) Wykonaj ruch i dodaj zmianę do ogólnej wartości rozwiązania.
   (d) Jeśli wartość rozwiązania jest obecnie najlepsza:
       i. Zapisz rozwiązanie jako najlepszego
3. Po upływie czasu zwróć najlepsze znalezione rozwiązanie.

3 Wyniki

3.1 Tabela wynikowa

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

Tabela 1: Wyniki dla kroA200 z algorytmem randomstart

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

Tabela 2: Wyniki dla kroA200 z naszym algorytmem

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

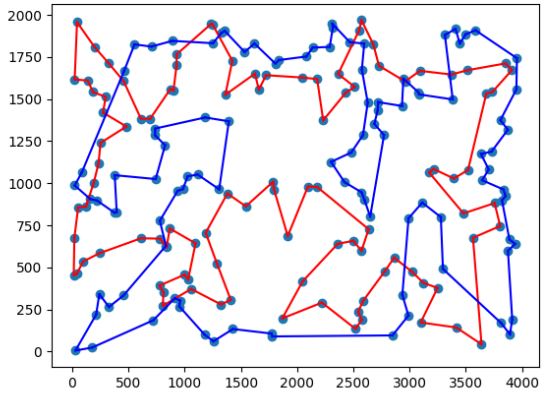
Tabela 3: Wyniki dla kroB200 z algorytmem randomstart

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

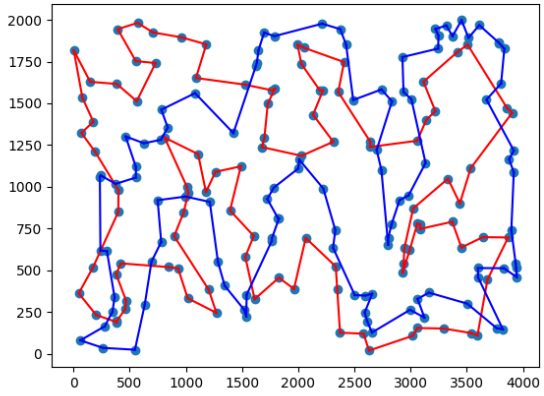
Tabela 4: Wyniki dla kroB200 z naszym algorytmem

3.2 Wizualizacja wyników

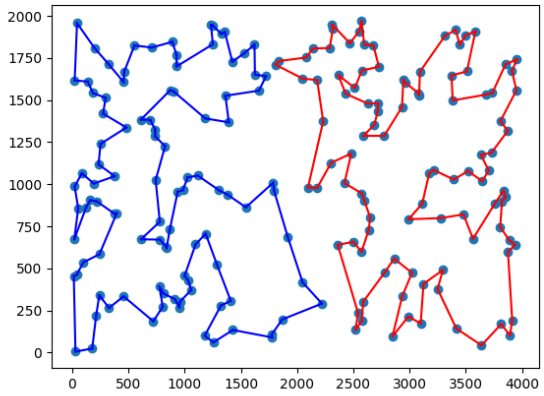
3.2.1 Algorytm wymiany krawędzi (zachłanny)



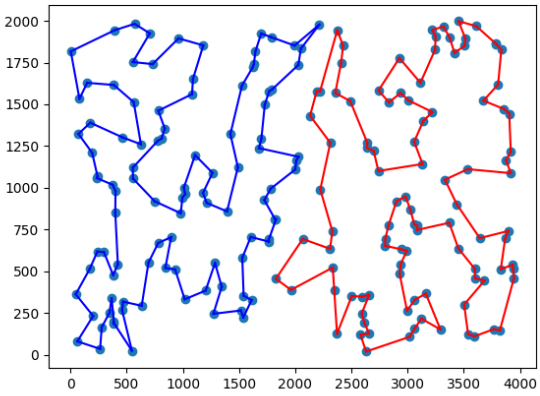
Rysunek 1: kroA200, losowy start



Rysunek 2: kroB200, losowy start

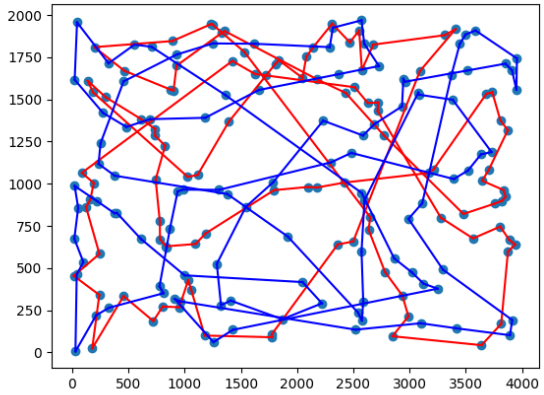


Rysunek 3: kroA200, własny algorytm startowy

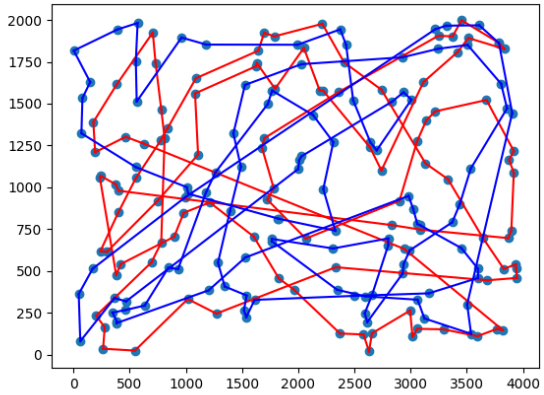


Rysunek 4: kroB200, własny algorytm startowy

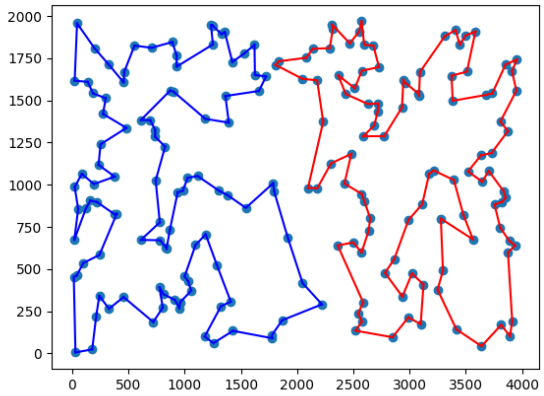
3.2.2 Algorytm wymiany wierzchołków (zachłanny)



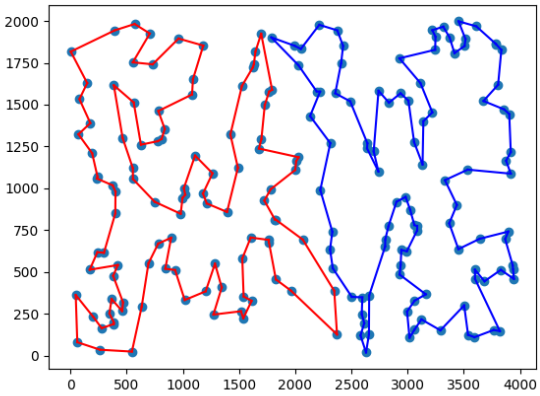
Rysunek 5: kroA200, losowy start



Rysunek 6: kroB200, losowy start

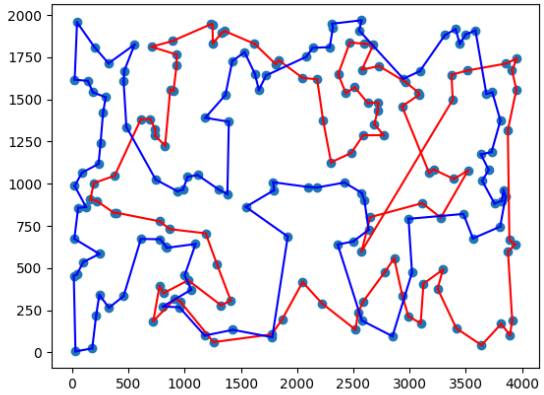


Rysunek 7: kroA200, własny algorytm startowy

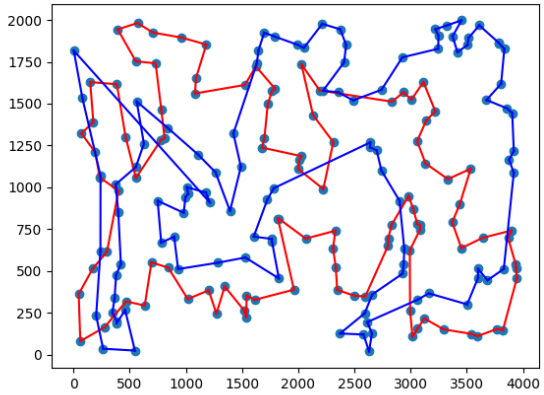


Rysunek 8: kroB200, własny algorytm startowy

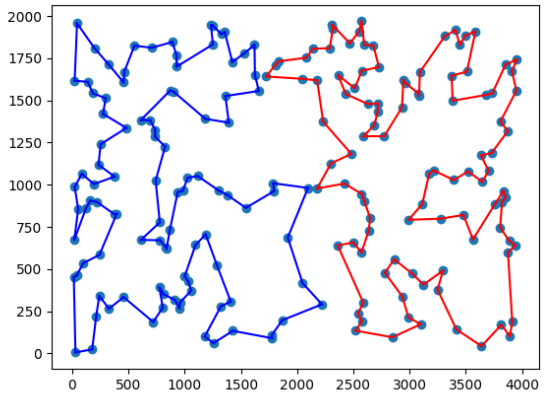
3.2.3 Algorytm wymiany krawędzi (steepest)



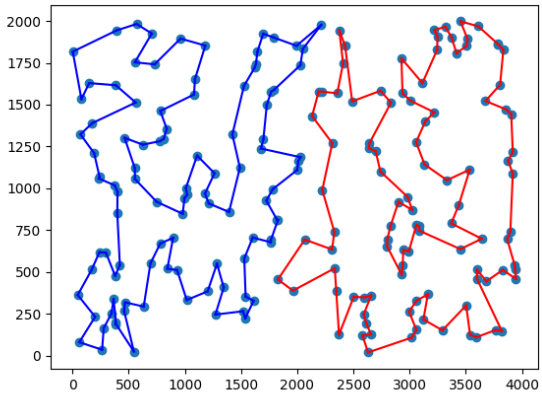
Rysunek 9: kroA200, losowy start



Rysunek 10: kroB200, losowy start

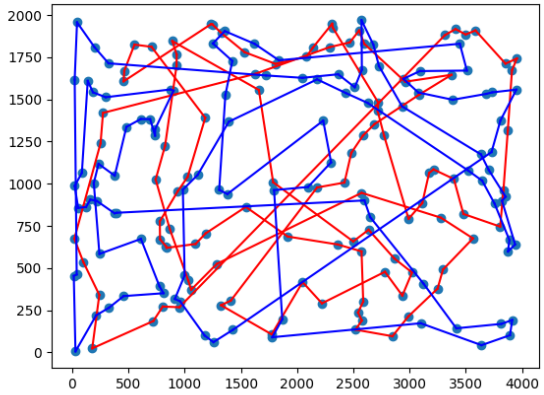


Rysunek 11: kroA200, własny algorytm startowy

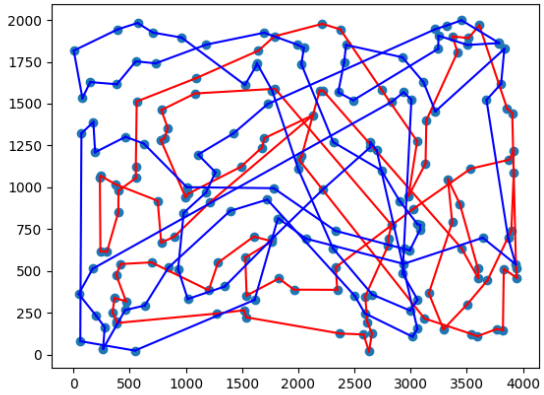


Rysunek 12: kroB200, własny algorytm startowy

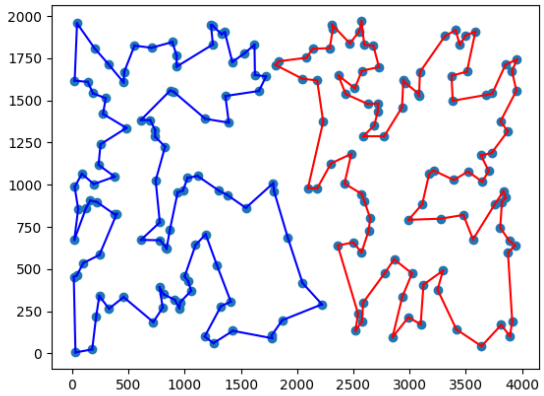
3.2.4 Algorytm wymiany wierzchołków (steepest)



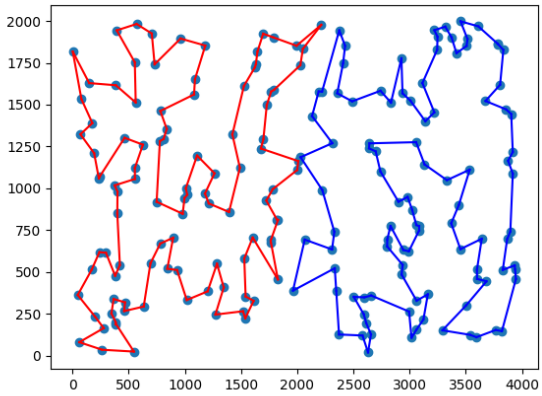
Rysunek 13: kroA200, losowy start



Rysunek 14: kroB200, losowy start

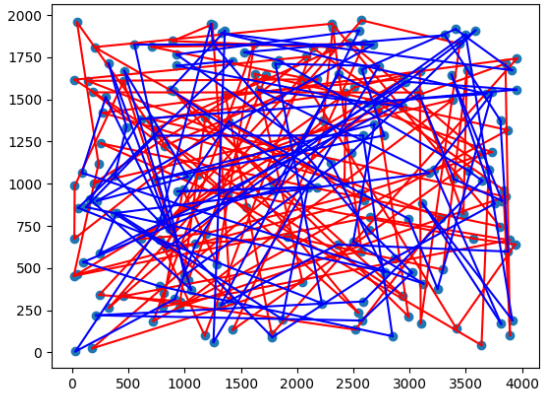


Rysunek 15: kroA200, własny algorytm startowy

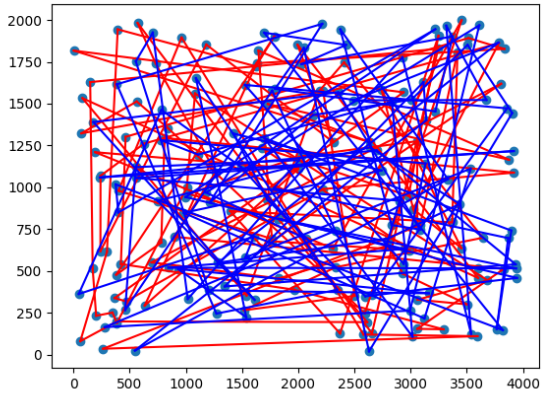


Rysunek 16: kroB200, własny algorytm startowy

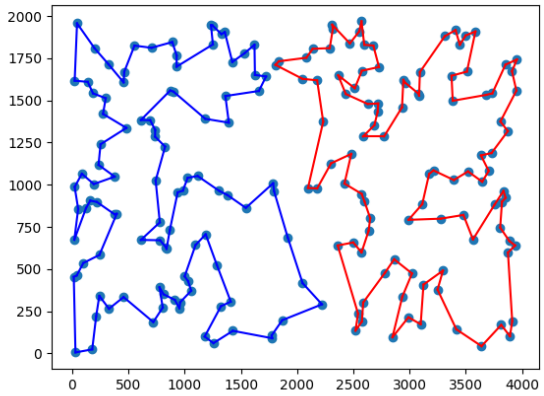
3.2.5 Algorytm losowego błądzenia w obu typach sąsiedztwa (random)



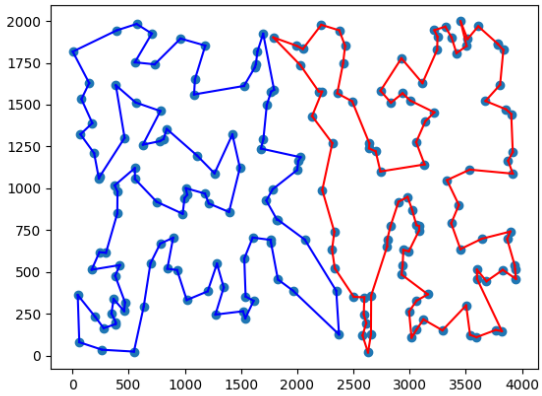
Rysunek 17: kroA200, losowy start



Rysunek 18: kroB200, losowy start



Rysunek 19: kroA200, własny algorytm startowy



Rysunek 20: kroB200, własny algorytm startowy

4 Wnioski i analiza wyników

1. Algorytmy oparte na sąsiedztwie z zamianą krawędzi osiągnęły znacznie lepsze wyniki niż te oparte na zamianie wierzchołków.

2. Algorytmy steepest osiągały znacznie dłuższe czasy wykonywania, przynosząc jedynie niewielką poprawę co do jakości.

3. Pomimo świetnych wyników 2 tygodnie temu, okazuje się że nasz algorytm nadal da się poprawić, choć nie umiał zrobić tego algorytm losowego błądzenia. Mimo to, wszystkie algorytmy osiągnęły jedynie niewielkie poprawy na startowym rozwiązaniu pozyskanym z naszego algorytmu.

5 Link do repozytorium

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