

### Zadanie 3. Wykorzystanie ocen ruchów z poprzednich iteracji i ruchów kandydackich w lokalnym przeszukiwaniu

Oskar Kiliańczyk 151863 & Wojciech Kot 151879

# 1 Opis zadania

Celem eksperymentu jest poprawa efektywności czasowej lokalnego przeszukiwania w wersji stromej, wykorzystującego najlepsze sąsiedztwo z poprzedniego zadania. Porównujemy wersję bazową z dwiema modyfikacjami: uporządkowaną listą ruchów oraz ruchem kandydackim. Każdy algorytm uruchamiany jest 100 razy na każdej instancji, startując z losowych rozwiązań. Dla porównania uwzględniamy również najlepszą heurystykę konstrukcyjną z zadania 1.

## 2 Opisy algorytmów

### 2.1 Te same co w poprzednich sprawozdaniach

#### 2.1.1 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.

#### 2.1.2 Nasz własny algorytm

Nasza heurystyka konstrukcyjna cechuje się podejściem zachłannym, ale wykorzystuje pewną wiedzę dziedzinową, a mianowicie to, że znacznie lepsze wyniki da się uzyskać kiedy wpierw dobierze się mądry podział wierzchołków startowych.

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 się wszystkie wierzchołki z danego mu zestawu powtarza:
3. Dla każdego nieodwiedzanego wierzchołka oblicza koszty jego wstawienia
4. Następnie oblicza żal (dwiużal) dla danego wierzchołka
5. Rozbudowuje cykl o wierzchołek z największym obliczonym żalem i zaznacza go jako odwiedzanego
6. Zwraca ścieżkę

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

## 2.2 Nowe pseudokody

### 2.2.1 Steepest wykorzystujący listę ruchów z poprzednich iteracji

### 2.2.2 Steepest wykorzystujący ruchy kandydackie

1. Dopóki możliwa jest poprawa:

(a) Przeszukaj wszystkie możliwe modyfikacje:

- **Zamiana między ścieżkami:**

- Dla każdej pary wierzchołków z różnych ścieżek
- Rozważ tylko wierzchołki będące kandydatami (z listy *candidate\_edges*)
- Oblicz zmianę kosztu po zamianie wierzchołków

- **Modyfikacje wewnętrzz ścieżki:**

- Dla każdej pary wierzchołków w ścieżce:
  - i. Sprawdź możliwość odwrócenia fragmentu między wierzchołkami
  - ii. Oblicz zmianę kosztu dla każdej operacji

(b) Wybierz modyfikację dającą największą poprawę (najmniejszy koszt)

(c) Wprowadź zmianę:

- Jeśli zamiana między ścieżkami - zamień odpowiednie wierzchołki
- Jeśli modyfikacja wewnętrzna - odwróć odpowiedni fragment ścieżki

(d) Zaktualizuj całkowity koszt rozwiązania

2. Zwróć ulepszone ścieżki

## 3 Wyniki

### 3.1 Tabela wynikowa

Algorytm	Best	Avg	Worst	Best Time	Avg Time	Worst Time	Best Diff	Avg Diff
split_paths_regret_TSP	30426	32893	36854	0.0943059	0.105313	0.206992		
traverse_steepest_edge	35949	38818.8	41812	3.70818	4.1397	4.53379	325405	301537
steepest_LM	34643	38673	41570	1.02937	1.30407	1.92433	327271	301518
steepest_kandydackie	36656	39723.4	43659	0.889616	1.04703	1.90562	323577	298848

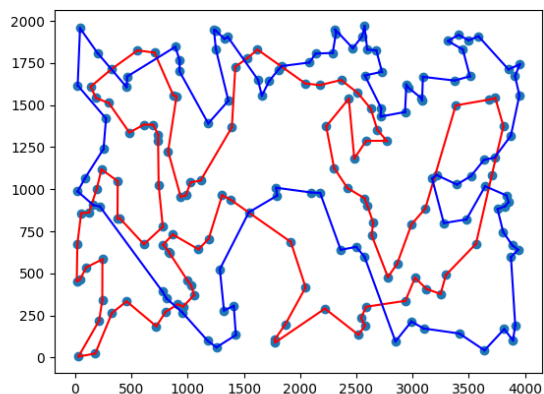
Tabela 1: Wyniki dla kroA200

Algorytm	Best	Avg	Worst	Best Time	Avg Time	Worst Time	Best Diff	Avg Diff
split_paths_regret_TSP	31218	33102.7	36913	0.0939048	0.0999451	0.131889		
traverse_steepest_edge	36556	38791.3	41839	3.94789	4.60979	6.5536	329070	293891
steepest_LM	36309	38794.7	41938	0.97498	1.25639	1.55901	322191	293913
steepest_kandydackie	37372	39808.4	41720	0.861845	0.965179	1.08236	325756	292558

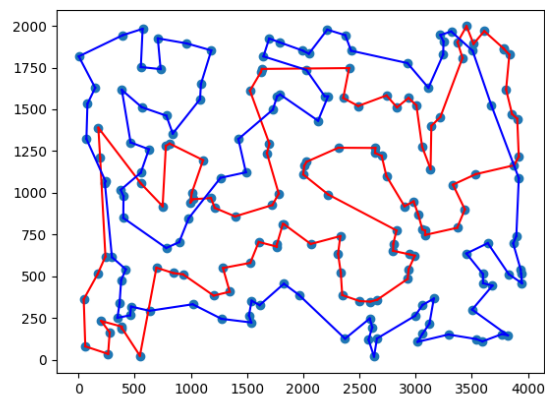
Tabela 2: Wyniki dla kroB200

## 3.2 Wizualizacja wyników

### 3.2.1 Algorytm stromy, bazowy

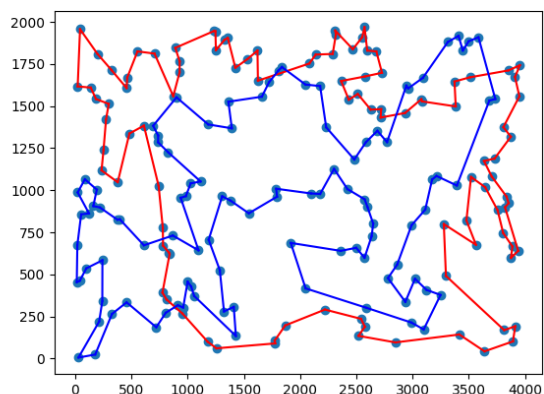


Rysunek 1: kroA200, losowy start

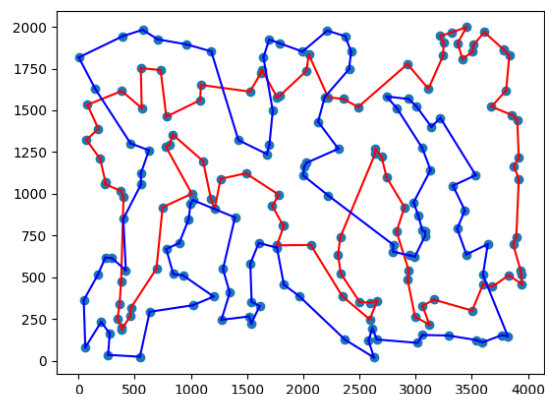


Rysunek 2: kroB200, losowy start

### 3.2.2 Algorytm stromy z listą ruchów

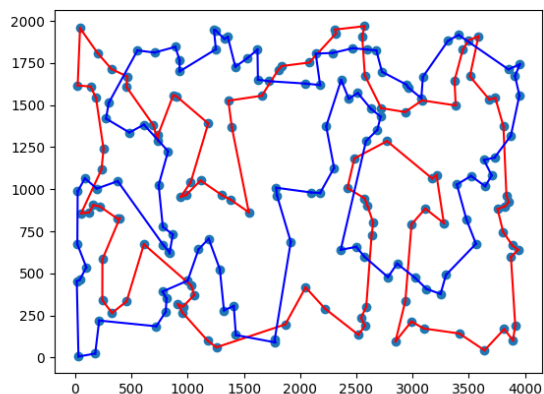


Rysunek 3: kroA200, losowy start

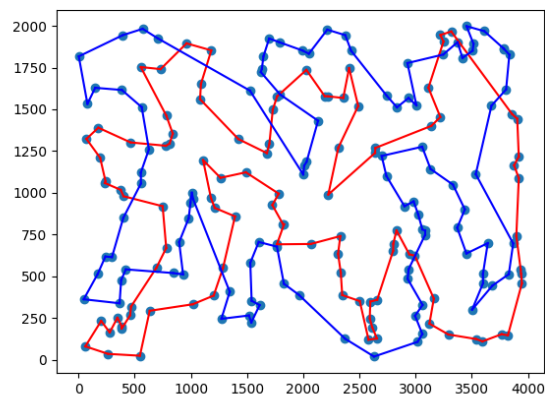


Rysunek 4: kroB200, losowy start

### 3.2.3 Algorytm stromy z mechanizmem ruchów kandydackich

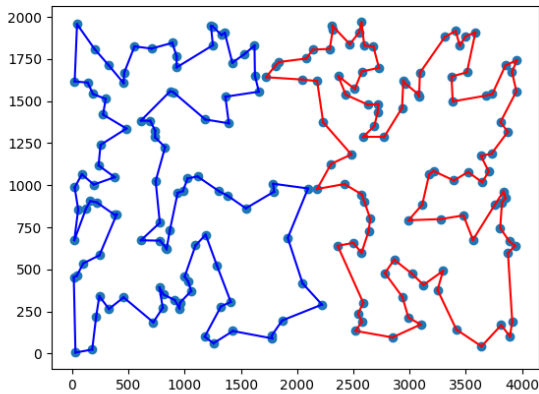


Rysunek 5: kroA200, losowy start

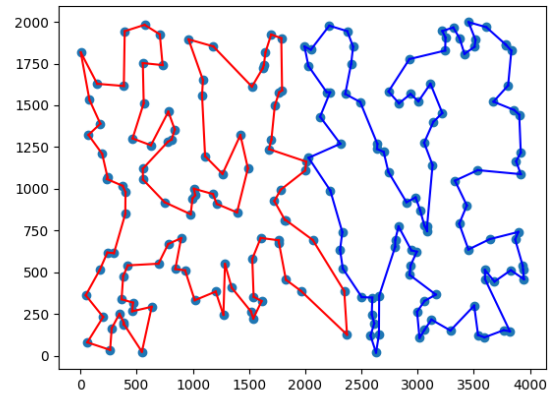


Rysunek 6: kroB200, losowy start

### 3.2.4 Heurystyka konstrukcyjna



Rysunek 7: kroA200, losowy start



Rysunek 8: kroB200, losowy start

## 4 Wnioski i analiza wyników

Na podstawie wyników można zauważyć, że algorytm stromy z listą ruchów oraz algorytm stromy z mechanizmem ruchów kandydackich są znacznie bardziej efektywne niż algorytm bazowy. W przypadku instancji kroA200 algorytm stromy z listą ruchów osiągnął lepszy wynik, natomiast w przypadku instancji kroB200 algorytm stromy z mechanizmem ruchów kandydackich okazał się lepszy. Niezależnie od instancji algorytm korzystający z ruchów kandydackich okazał się odrobinę bardziej efektywny czasowo. Wyniki naszej heurystyki konstrukcyjnej są wciąż najlepsze, jednak jest to przede wszystkim zasługa dobrego startowego podziału, czyli zastosowania swego rodzaju wiedzy dziedzinowej, a to sugeruje, że algorytmy lokalnego przeszukiwania mogą być użyte do poprawy wyników heurystyki konstrukcyjnej, ale nie jako osobna metoda znajdowania optimum rozpoczynając z rozwiązań losowych. Poprzednie zadanie pokazało, że algorytmy lokalnego przeszukiwania są bardziej efektywne w przypadku, gdy startujemy z rozwiązań konstrukcyjnych.

## 5 Link do repozytorium

Kod źródłowy w repozytorium GitHub dostępny pod linkiem:  
Repozytorium.