Optymalizacja efektywności lokalnego przeszukiwania

Autorzy: Dominik Maćkowiak 151915, Szymon Szymankiewicz 151821

Opis zadania

Celem zadania jest przyspieszenie lokalnego przeszukiwania w wersji steepest descent z użyciem losowych rozwiązań startowych. Aby poprawić efektywność, implementujemy dwie osobne wersje algorytmu:

1. **Z uporządkowaną listą ruchów (LM)** – wykorzystuje oceny ruchów z poprzednich iteracji, uwzględniając zarówno ruchy wewnątrz-, jak i międzytrasowe.

2. **Z ruchem kandydackim** – ogranicza liczbę generowanych ruchów do najbardziej obiecujących.

Dane są dwie instancje (“kroA200” oraz kroB200”) z biblioteki TSPlib, które zawierają informacje o położeniu konkretnych wierzchołków. Celem zadania jest utworzenie z każdej instancji dwa rozłączne cykle, tak aby ich łączna długość była jak najmniejsza.

Algorytmy

Algorytm lokalnego przeszukiwania typu steepest z wymianą krawędzi z listą ruchów

Wygeneruj losowe rozwiązanie początkowe (dwa cykle).

Zainicjuj listę ruchów poprawiających (LM).

Dopóki możliwe są poprawy:
a. Wygeneruj wszystkie możliwe nowe ruchy.
b. Wybierz tylko ruchy poprawiające (`delta < 0`).
c. Posortuj je rosnąco według `delta`.
d. Dla każdego ruchu `m` z listy:
i. Pobierz usuwane krawędzie z bieżącego rozwiązania (get_removed_edges).
ii. Sprawdź ich status (is_edge_in_cycles):
• Jeśli którakolwiek krawędź nie istnieje — usuń `m` z LM.
• Jeśli wszystkie są, ale w odwróconym kierunku — pomiń `m`, ale pozostaw w LM.

• Jeśli wszystkie są w tym samym kierunku — zastosuj `m`, dodaj usunięte krawędzie i ich odwrotności do LM, przejdź do nowej iteracji.

Zwróć najlepsze znalezione rozwiązanie.

Algorytm lokalnego przeszukiwania typu steepest z wymianą krawędzi z ruchami kandydadyckimi

Oblicz początkową długość rozwiązania:
Wyznacz krawędzie kandydackie:
Ustaw flagę poprawy: improved ← True
Dopóki improved jest prawdziwe:
a. Ustaw improved ← False
b. Ustaw best_delta ← 0 oraz best_move ← None
c. Sprawdź poprawę przez zamiany między cyklami i w cyklach:
Dla każdego wierzchołka **i** sprawdź czy **j** jest jego kandydatem. Jeśli tak:
- wprowadź krawędź między i-j
- oblicz deltę
- Jeśli delta < best_delta:
• best_delta ← delta
• best_move ← move
f. Jeśli znaleziono jakikolwiek ruch poprawiający (best_move ≠ None):
i. Wykonaj ruch zgodnie z typem best_move:
• first_cycle ← new_first
• second_cycle ← new_second
ii. Zaktualizuj best_length:
• best_length ← best_length + best_delta
iii. Ustaw improved ← True
Zwróć najlepsze rozwiązanie

Funkcje pomocnicze:

Pobranie usuwanych krawędzi (get_removed_edges)

Dla podanego ruchu:
– Zidentyfikuj krawędzie, które zostaną usunięte w wyniku ruchu
Zwróć listę tych krawędzi (z kierunkiem)

Sprawdzenie obecności krawędzi (is_edge_in_cycles)

Dla danej krawędzi i aktualnych cykli:
– Jeśli krawędź występuje w cyklu w tym samym kierunku: zwróć 1

– Jeśli w przeciwnym kierunku: zwróć -1
– Jeśli brak: zwróć 0

Wyznaczenie krawędzi kandydackich (get_candidate_edges)

Inicjalizuj pusty słownik candidate_edges.
Dla każdego wierzchołka i:
a. Posortuj indeksy wierzchołków wg odległości od i:
– sorted_neighbors ← posortowane indeksy wg DistanceMatrix[i]
b. Wybierz k najbliższych sąsiadów:
– candidate_edges[i] ← pierwsze k elementów z sorted_neighbors
Zwróć candidate_edges.

Wyniki eksperymentu obliczeniowego

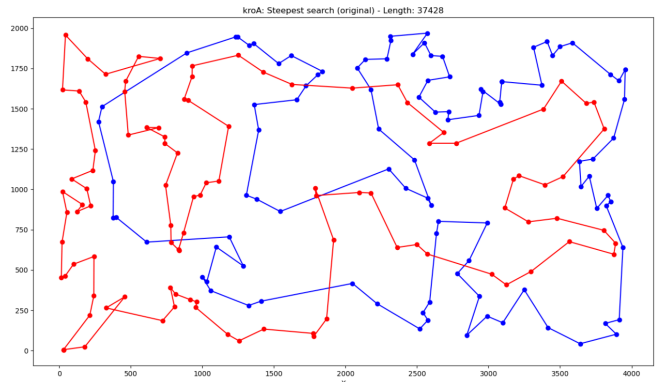
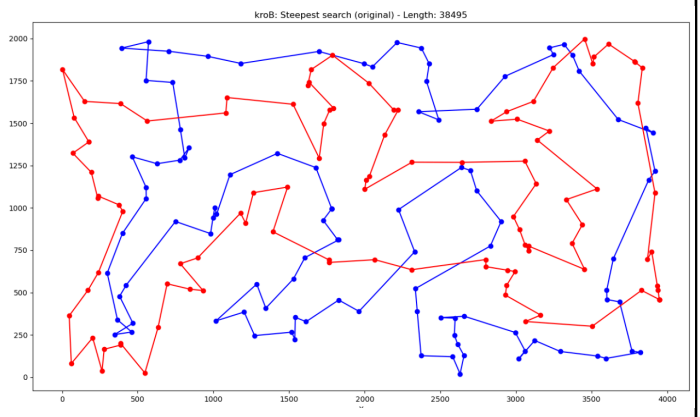
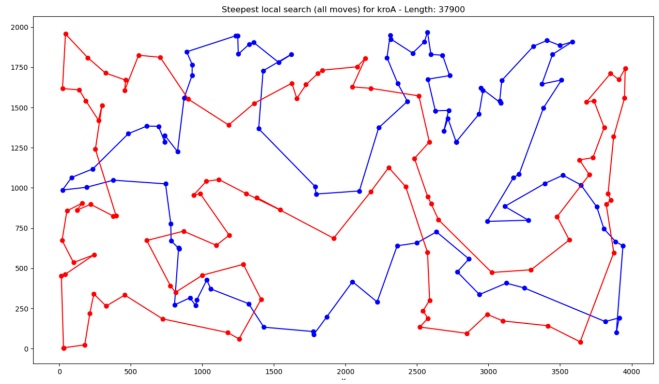
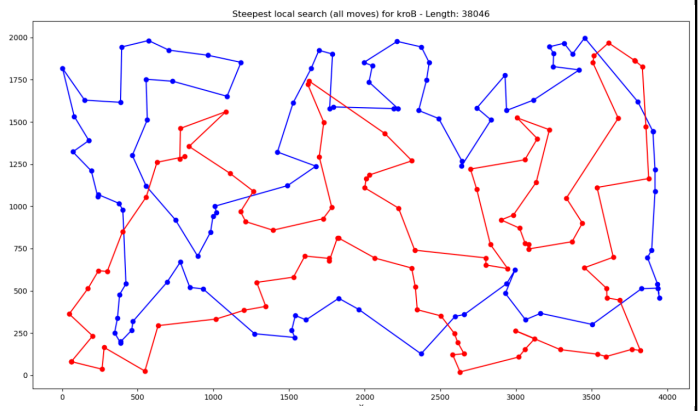
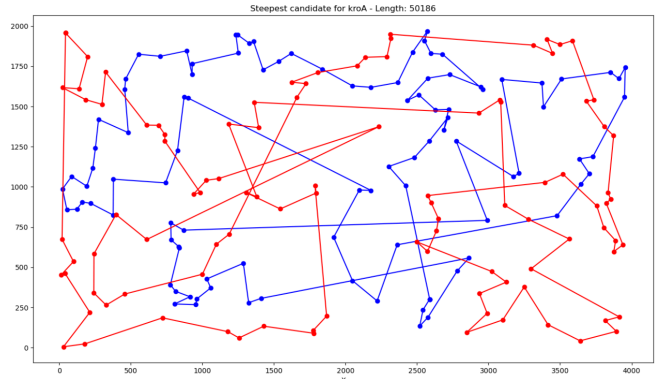
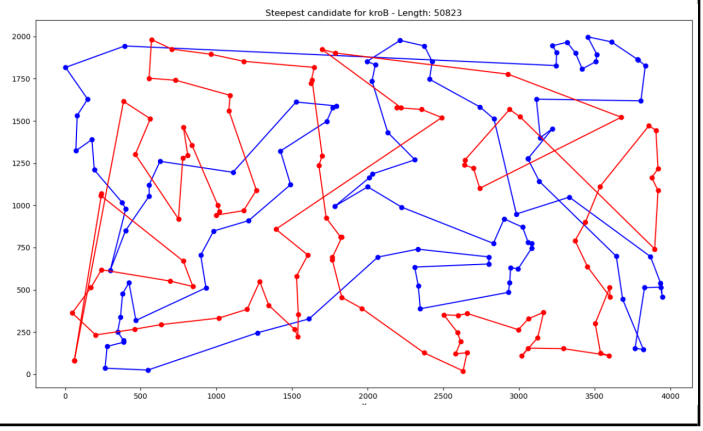
Przeprowadziliśmy eksperyment, który polegał na zmierzeniu długości dwóch utworzonych cykli. Dla każdej instancji przeprowadziliśmy 100 testów. Obliczyliśmy średnią długość z tych testów, wybraliśmy odcinek najdłuższy i najkrótszy oraz średni czas wykonania algorytmów.

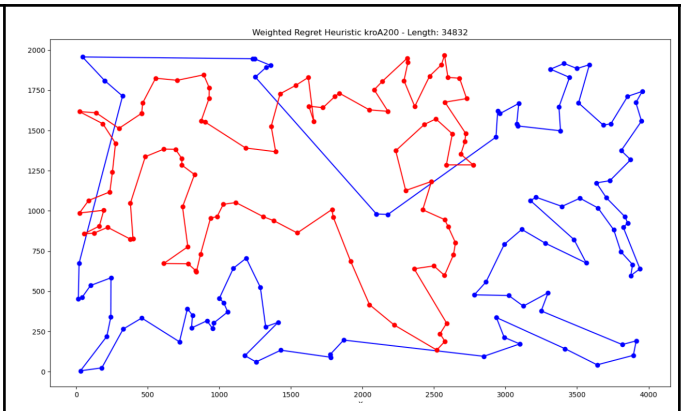
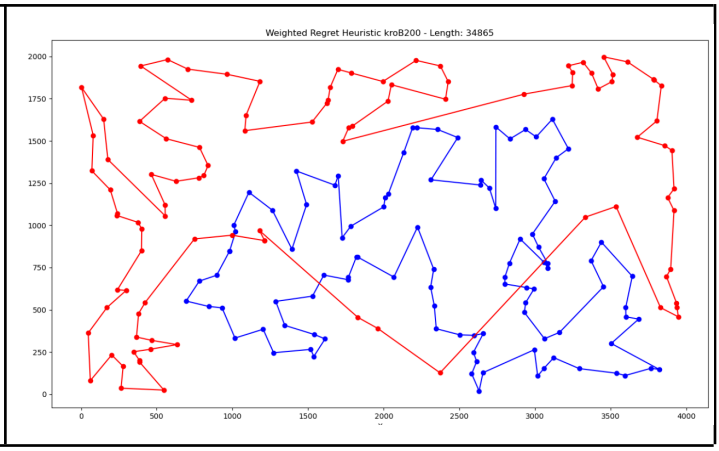
	kroA200				kroB200			
	start	średnia	min	max	start	średnia	min	max
Steepest	349555	**40115.5**	37428	42008	338017	**40188.8**	38495	41293
Steepest + lista ruchów	349555	**39918.8**	37900	41867	338017	**39893.1**	38046	41515
Steepest + ruchy kandydackie	349555	**53711.5**	50186	64655	338017	**56450.9**	50823	62951
Ważony 2-żal	BRAK	***35815.9***	33597	37041	BRAK	***36361.5***	34581	37976

	kroA200			kroB200		
	średnia (s)	min (s)	max (s)	średnia (s)	min (s)	max (s)
Steepest	**10.0549**	8.7412	11.1968	**9.4771**	8.6967	10.6124
Steepest + lista ruchów	**9.8907**	8.6757	10.5930	**9.4882**	8.8687	10.2906
Steepest + ruchy kandydackie	**5.3145**	4.7826	5.9346	**5.2988**	4.7471	5.7103
Ważony 2-żal	***1.2871***	1.2234	1.4302	***1.2765***	1.2256	1.3480

Wizualizacja dla najlepszych rozwiązań

Zestaw kroA200

	kroA200	kroB200
Steepest		
Steepest + lista ruchów		
Steepest + ruchy kandydackie		

Ważony 2-żal		

Wnioski

Przeprowadzone eksperymenty potwierdziły, że zastosowanie uporządkowanej listy ruchów oraz mechanizmu ruchów kandydackich przyczynia się do poprawy zarówno jakości uzyskiwanych rozwiązań, jak i redukcji czasu wykonywania algorytmów. Szczególnie algorytm z ruchem kandydackim charakteryzował się znaczącym skróceniem czasu obliczeń przy zachowaniu satysfakcjonującej jakości wyników. Jednak mimo wszystko ważony 2-żal pozostaje najlepszym algorytmem pod względem jakości i szybkości. Algorytm steepest byłby idealny do zastosowania mając gotowe cykle na przykład stworzone przy pomocy algorytmu z 2-żalem.

Kod programu: https://github.com/szymon240/imo-lab3