Zadanie 4. Rozszerzenia lokalnego przeszukiwania

Oskar Kiliańczyk 151863 & Wojciech Kot 151879

1 Opis zadania

Celem eksperymentu jest poprawa i rozszerzenie lokalnego przeszukiwania o trzy nowe metody:
- MSLS - Multiple Start local search
- ILS - Iterated local search
- LNS - Large neighborhood search
Porównujemy wersję bazową, znaną z poprzedniego eksperymentu z tymi trzema rozszerzeniami.

2 Opisy algorytmów

2.1 Multiple Start Local Search (MSLS)

1. Zainicjuj zmienną obecnie najlepszego kosztu jako MAXINT
2. dla i=1,2,:.ile_iteracji
   (a) Wygeneruj losowe rozwiązanie początkowe
   (b) Wykonaj lokalne przeszukiwanie
   (c) Jeśli obecnie znalezione rozwiązanie ma mniejszy koszt niż obecnie najlepszy
      i. zapisz obecnie znalezione rozwiązanie jako najlepsze
      ii. zapisz koszt tego rozwiązania jako obecnie najlepszy
3. Zwróć obecnie najlepsze rozwiązanie

2.2 Iterated Local Search (ILS)

2.2.1 Główny algorytm

1. Wygeneruj losowe rozwiązanie początkowe
2. Wykonaj lokalne przeszukiwanie
3. zapisz obecnie znalezione rozwiązanie jako najlepsze
4. zapisz koszt tego rozwiązania jako obecnie najlepszy
5. Dopóki czas wykonywania $<$ limit_czasu:
   (a) Wykonaj funkcję Perturbacji* na obecnie najlepszym rozwiązaniu
   (b) Jeśli obecnie znalezione rozwiązanie ma mniejszy koszt niż obecnie najlepszy
      i. zapisz obecnie znalezione rozwiązanie jako najlepsze
      ii. zapisz koszt tego rozwiązania jako obecnie najlepszy

2.2.2 Funkcja Perturbacji

1. Parametry: cykl1, cykl2, ile_zmian_wierzcholkow, ile_zmian_krawędzi
2. dla i=1,2,:.,ile_zmian_wierzcholkow:
   (a) Zamień losowe wierzchołki między cyklami
3. dla i=1,2,:.,ile_zmian_krawędzi:
   (a) Zamień losowo wybrane krawędzi w pierwszym cyklu
   (b) Zamień losowo wybrane krawędzi w drugim cyklu
4. Zwróć cykl1, cykl2

2.3 Large neighborhood Search (LNS)

2.3.1 Główny algorytm

1. Wygeneruj losowe rozwiązanie początkowe
2. Wykonaj lokalne przeszukiwanie
3. zapisz obecnie znalezione rozwiązanie jako najlepsze
4. zapisz koszt tego rozwiązania jako obecnie najlepszy
5. Dopóki czas wykonywania $<$ limit_czasu:
   (a) Wykonaj funkcję destroy_repair na obecnie najlepszym cyklu
   (b) Jeśli ustawiona jest flaga wykonaj_LS:
      i. Wykonaj lokalne przeszukiwanie na obecnym rozwiązaniu
   (c) Oblicz koszt obecnego rozwiązania
   (d) Jeśli obecnie znalezione rozwiązanie ma mniejszy koszt niż obecnie najlepszy
      i. zapisz obecnie znalezione rozwiązanie jako najlepsze
      ii. zapisz koszt tego rozwiązania jako obecnie najlepszy
6. Zwróć najlepsze rozwiązanie

2.3.2 Funkcja destroy-repair

1. Parametry: cykl1, cykl2, destroy_frac, n (ilość wierzchołków)
2. Liczba wierzchołków na region oraz promień regionu:

$$\text{nodes\_per\_region} = \lfloor \text{destroy\_frac} \cdot (|cycle_1| + |cycle_2|) \div 3 \rfloor$$
$$\text{region\_size} = \max\left(1, \frac{\text{nodes per region} - 1}{2}\right)$$

3.
4. Wybierz najdłuższą krawędź z cykl1, która jest blisko cykl2
5. jeśli znaleziono taką krawędź:
   (a) region1 = sąsiedztwo tej krawędzi o promieniu region_size
6. w przeciwnym wypadku:
   (a) region1 = sąsiedztwo tej krawędzi o promieniu region_size
7. Wykonaj analogiczną procedurę, jak kroki 3–5, ale dla krawędzi z cykl2
8. Utwórz region3, jako losowo wybrany węzeł i jego sąsiedztwo
9. Utwórz listę do_usuniecia jako sumę zbiorów region1, region2 i region3
10. oblicz maksymalną ilość elementów do usunięcia jako

$$\lfloor \text{destroy\_frac} \cdot n \rfloor$$

11. Jeśli lista do_usuniecia zawiera więcej elementów niż obliczona ilość, losowo usuń nadmiarowe wierzchołki
12. Usuń wierzchołki obecne na liście do_usuniecia z cykl1 oraz cykl2
13. Dla każdego wierzchołka na liście do_usuniecia:
    (a) Znajdź najlepsze miejsce wstawienia w cykl1 lub cykl2, minimalizując wzrost kosztu
    (b) Wstaw wierzchołek w znalezione miejsce

3 Wyniki

3.1 Tabela wynikowa

Algorytm	Best	Avg	Worst	Best Time	Avg Time	Worst Time	Avg Perturbations
MSLS	34630	35375.7	35862	235.639	269.287	327.604	-
ILS	31016	31919.8	33193	270.753	270.851	270.946	4977.8
LNS	29690	30559.4	32006	272.054	272.242	272.595	5714.6
LNS bez LS	30573	32590.4	37300	270.371	270.611	270.96	64054.4

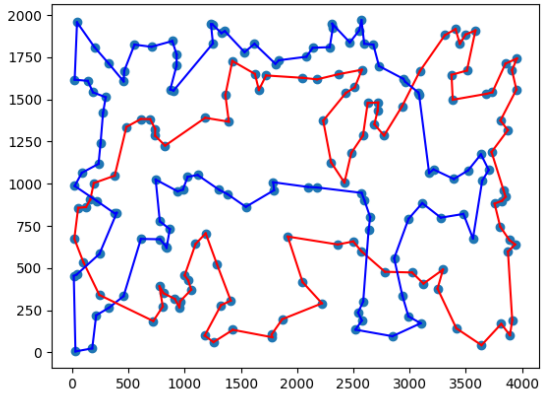
Tabela 1: Wyniki dla kroA200

Algorytm	Best	Avg	Worst	Best Time	Avg Time	Worst Time	Avg Perturbations
MSLS	34819	35501.6	36081	256.952	279.947	380.768	-
ILS	31475	32454.2	33343	281.034	281.280	281.655	6057.1
LNS	30092	30770.9	31762	282.524	282.865	283.275	5768.0
LNS bez LS	31255	32949.6	34546	281.116	281.292	281.755	72515.3

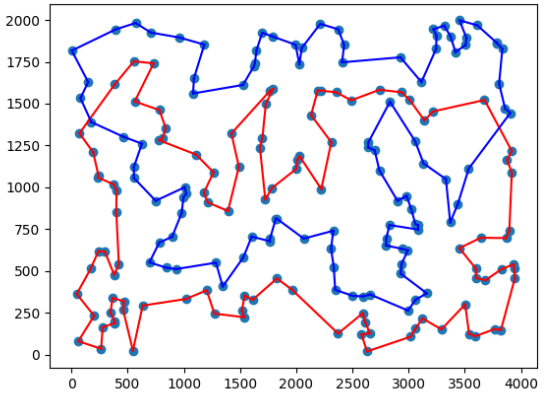
Tabela 2: Wyniki dla kroB200

3.2 Wizualizacja wyników

3.2.1 ILS

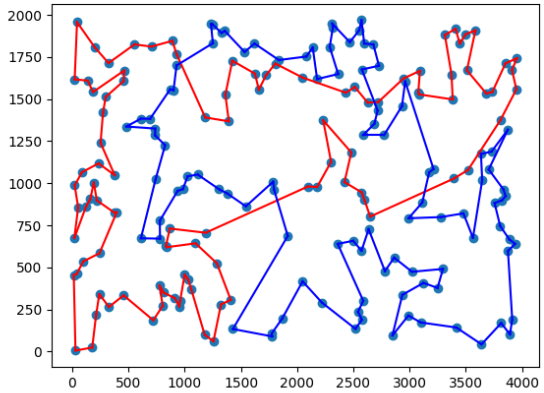


Rysunek 1: kroA200, losowy start

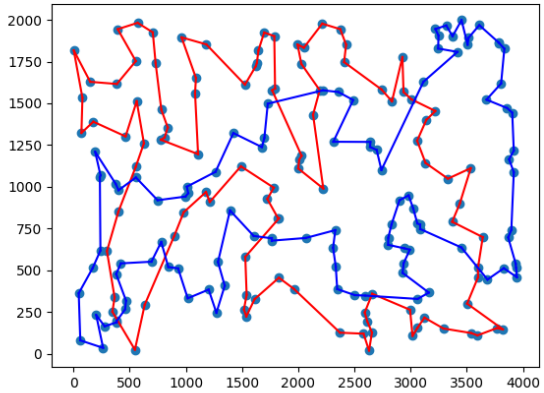


Rysunek 2: kroB200, losowy start

3.2.2 MSLS

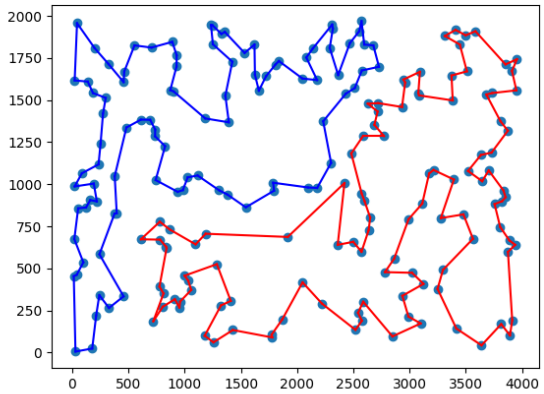


Rysunek 3: kroA200, losowy start

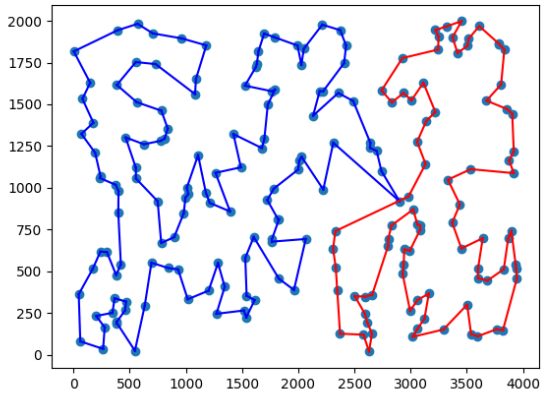


Rysunek 4: kroB200, losowy start

3.2.3 LNS

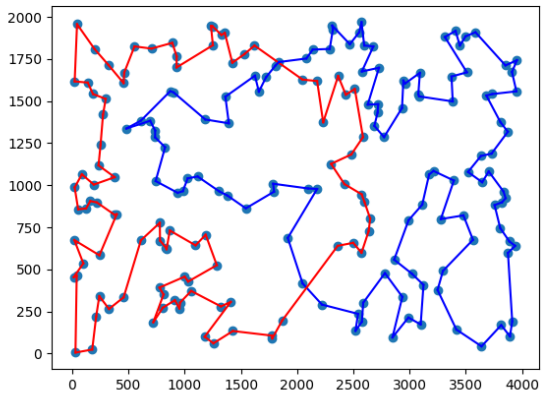


Rysunek 5: kroA200, losowy start

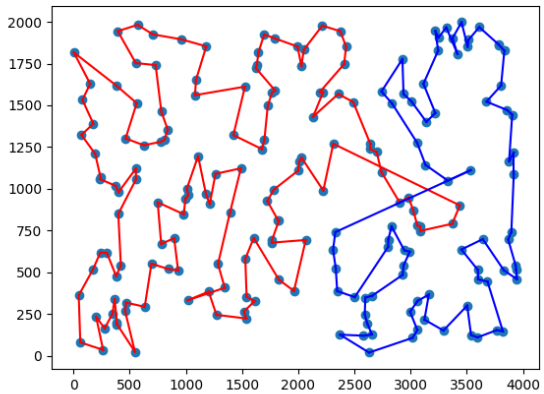


Rysunek 6: kroB200, losowy start

3.2.4 LNS bez LS



Rysunek 7: kroA200, losowy start



Rysunek 8: kroB200, losowy start

4 Wnioski i analiza wyników

Najlepsze wyniki końcowe jakościowo osiągnęła metoda wykorzystująca Large Neighborhood Search (LNS) z lokalnym przeszukiwaniem (LS). Najgorsze natomiast wyniki były dla Multiple Start Local Search (MSLS), co było do przewidzenia, patrząc na wyniki zwykłego LS z poprzednich eksperymentów i to że jest to jedynie wielokrotne powtórzenie LS. LNS nie korzystajce z LS osiagnelo znacznie gorsze rezultaty pomimo znacznie wiekszej ilosci iteracji.

5 Link do repozytorium

Kod źródłowy w repozytorium GitHub dostępny pod linkiem:
Repozytorium.