Rozszerzenia lokalnego przeszukiwania

Autorzy sprawozdania

Filip Marciniak 148148, Szymon Pasternak 148146

Wstęp

W ramach zadania należało zaimplementować trzy metody rozszerzające lokalne przeszukiwanie. Rozpatrywanym problemem jest zmodyfikowany problem komiwojażera. W problemie należy dla danego zbioru wierzchołków oraz symetrycznej macierzy odległości ułożyć dwa rozłączne cykle, z których każdy zawiera 50% wierzchołków. Celem jest minimalizacja łącznej długości obu cykli. W pracy rozważane są dwie instancje - `kroA200` oraz `kroB200` pochodzące z biblioteki `TSPLib`. Dla każdej z instancji zbadano:

- algorytm lokalnego przeszukiwania z różnych losowych punktów startowych,
- algorytm iteracyjnego przeszukiwania lokalnego z niewielką perturbacją,
- algorytm iteracyjnego przeszukiwania lokalnego z Large-scale neighborhood search, tj. większą perturbacją typu Destroy-Repair,
- algorytm iteracyjnego przeszukiwania lokalnego z Large-scale neighborhood bez uruchamiania lokalnego przeszukiwania po perturbacjach.

Łącznie uzyskano więc 4 różne wyniki dla każdej z instancji.

Kod programu

Kod programu dostępny jest w repozytorium: https://github.com/Johnybonny/IMO

Opis algorytmów

Każdy z algorytmów akceptuje na wejściu macierz odległości pomiędzy danymi wierzchołkami. Każdy z algorytmów uruchomiony został 10 razy z losowymi początkowymi rozwiązaniami. Jako algorytm przeszukiwania lokalnego użyta została wersja stroma, w której w ruchach wewnątrztrasowych, rozpatrywany był rodzaj sąsiedztwa polegający na wymianie krawędzi w cyklu, a w ruchach międzytrasowych, ruch polegał na wymianie wierzchołków pomiędzy dwoma cyklami. Algorytm został opisany szczegółowo w raporcie na drugim laboratorium.

Multiple start local search

```
powtórz 100 razy
{
  wygeneruj losowe rozwiązanie startowe.
  popraw rozwiązanie przy użyciu algorytmu local search.
  jeśli rozwiązanie daje wynik najlepszy z dotychczasowych, to zapamiętaj je.
}
zwróć zapamiętane rozwiązanie.
```

Iterated local search z niewielką perturbacją

W algorytmie wykorzystana została perturbacja będąca losową liczbą (2-5) losowych ruchów wykonanych na dotychczasowym rozwiązaniu. Warunkiem stopu w algorytmie jest osiągnięcie czasu równego średniemu czasowi Multiple start local search dla tej samej instancji.

```
wygeneruj losowe rozwiązanie startowe.
popraw rozwiązanie przy użyciu algorytmu local search.
powtarzaj
{
  skopiuj dotychczasowe rozwiązanie do nowego rozwiązania.
  wykonaj perturbacje na nowym rozwiązaniu.
  popraw nowe rozwiązanie przy użyciu algorytmu local search.
  jeśli nowe rozwiązanie daje wynik lepszy od aktualnego, to za aktualne
rozwiązanie podstaw nowe.
  jeśli minął czas przeznaczony na algorytm, to zakończ pętlę.
}
zwróć aktualne rozwiązanie.
```

Iterated local search z większą perturbacją

W algorytmie wykorzystana została perturbacja będąca usunięciem 30% wierzchołków następujących po sobie w losowym miejscu w każdym cyklu (destroy). Naprawienie rozwiązania było zaimplementowane poprzez algorytm korzystający z heurystyki 2-żalu z ważonym żalem opisany dokładnie w raporcie na pierwsze laboratorium (repair). Warunkiem stopu w algorytmie jest osiągnięcie czasu równego średniemu czasowi Multiple start local search dla tej samej instancji. Algorytm został również uruchomiony w wersji bez lokalnego przeszukiwania po fazie naprawy cykli (wielkoskalowe przeszukiwanie sąsiedztwa).

```
wygeneruj losowe rozwiązanie startowe.
popraw rozwiązanie przy użyciu algorytmu local search.
powtarzaj
{
  skopiuj dotychczasowe rozwiązanie do nowego rozwiązania.
  wykonaj operację destroy na nowym rozwiązaniu.
  wykonaj operację repair na nowym rozwiązaniu.
  popraw nowe rozwiązanie przy użyciu algorytmu local search. (opcjonalnie)
  jeśli nowe rozwiązanie daje wynik lepszy od aktualnego, to za aktualne
rozwiązanie podstaw nowe.
  jeśli minął czas przeznaczony na algorytm, to zakończ pętlę.
}
zwróć aktualne rozwiązanie.
```

Wyniki

W tabeli przedstawione zostały najlepsze, średnie i najgorsze wyniki dla zbadanych algorytmów.

	kroA200	kroB200

	kroA200			kroB200		
	min	mean	max	min	mean	max
MSLS	35443	35944	36362	34831	35954	36309
ILS1	33755	34621	35573	34002	34712	35863
ILS2	30599	31034	31600	30774	31238	31669
ILS2a	30535	31378	32666	29930	31990	34134

W tabeli przedstawione zostały czasy wykonywania algorytmu MSLS na obu badanych instancjach.

	kroA200			kroB200		
	min	mean	max	min	mean	max
MSLS	45	48	52	46	50	53

W tabeli poniżej przedstawione zostały liczby perturbacji wykonanych w algorytmach.

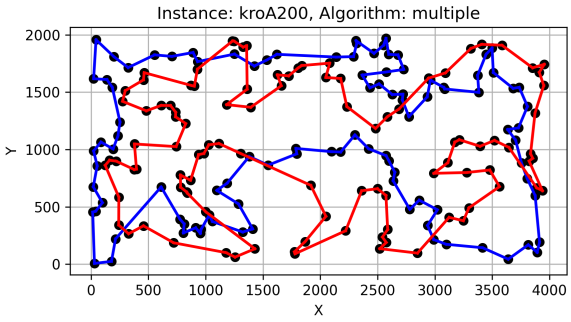
	kroA200			kroB200		
	min	mean	max	min	mean	max
ILS1	3188	3991	4656	3599	4206	4626
ILS2	760	885	950	934	973	1015
ILS2a	1465	1505	1534	1003	1234	1570

Poniżej umieszczone zostały wizualizacje najlepszych z uzyskanych wyników:

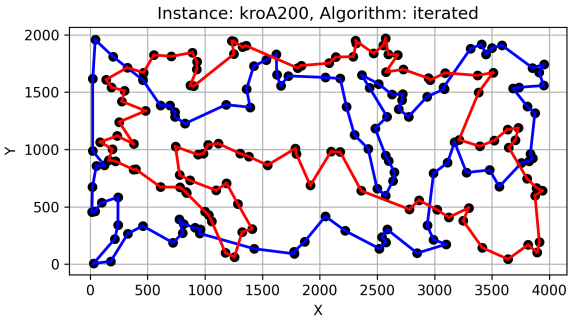
Wizualizacja kroA200.tsp

MSLS	**ILS1**

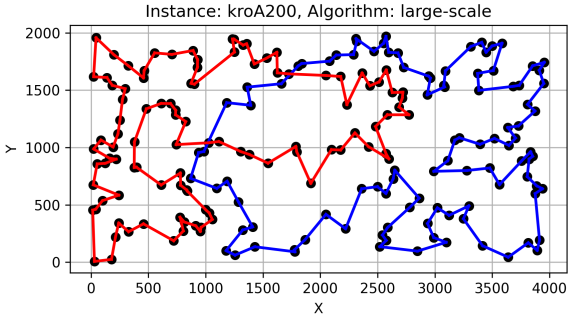
MSLS



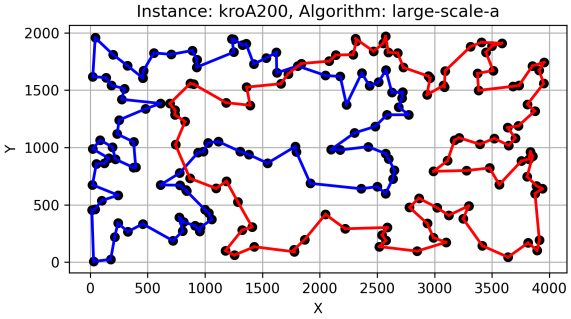
ILS1



ILS2

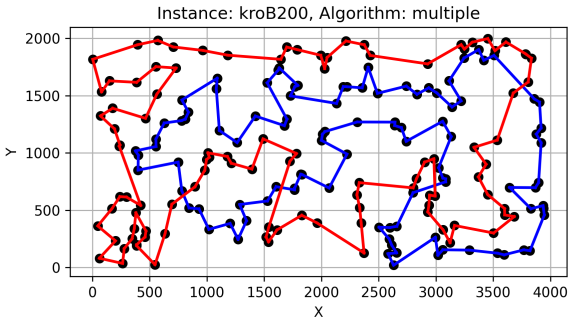


ILS2a

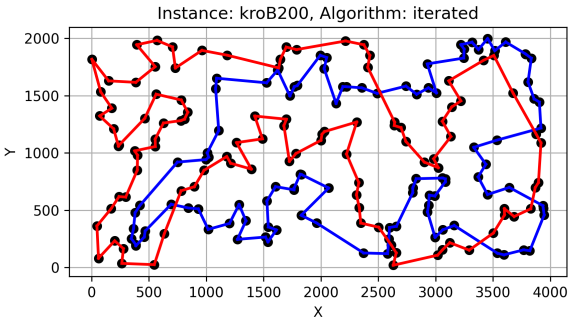


Wizualizacja kroB200.tsp

MSLS

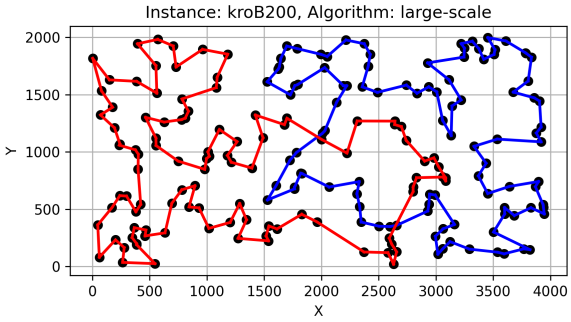
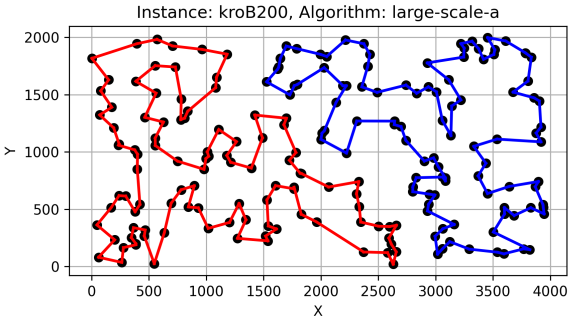


ILS1



ILS2

ILS2a

ILS2	ILS2a
	

Wnioski

1. Wszystkie algorytmy są efektywnymi metodami rozszerzenia lokalnego przeszukiwania. W każdym przypadku udało się poprawić rozwiązanie wygenerowane jako startowe.
2. Najlepiej radzi sobie algorytm Iterated local search z większą perturbacją typu Destroy-Repair w wersji z przeszukiwaniem stromym po każdej perturbacji. Wersja bez przeszukiwania stromego daje bardzo podobne rozwiązania, wykonując dużo więcej perturbacji. Iterated local search z niewielką perturbacją radzi sobie trochę gorzej, ale wykonuje zdecydowanie więcej perturbacji. Algorytm Multiple start local search radzi sobie najgorzej, ale jest wciąż dobrą metodą rozszerzenia lokalnego przeszukiwania, ponieważ daje lepsze wyniki w porównaniu z pojedynczym uruchomieniem algorytmu przeszukiwania stromego. Oczywiście dzieje się to kosztem czasu obliczeń.