

Lokalne przeszukiwanie

Autorzy sprawozdania

Filip Marciniak 148148, Szymon Pasternak 148146

Wstęp

W ramach zadania należało przeprowadzić badania dotyczące lokalnego przeszukiwania. 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 - **kroA100** oraz **kroB100** pochodzące z biblioteki **TSPLib**. Dla każdej z instancji zbadano:

- różne rozwiązania początkowe: losowe oraz uzyskane heurystyką 2-żal,
- różne rodzaje sąsiedztwa: zamianę wierzchołków oraz krawędzi,
- różne rodzaje przeszukiwania: zachłanne oraz strome.

Łącznie uzyskano więc 8 różnych wyników dla każdej z instancji. Dodatkowo przeprowadzono również badanie losowego błądzenia, które polega na wykonywaniu losowych ruchów. Czas trwania takiego błądzenia został ustalony jako średnio naj wolniejsza z wersji lokalnego przeszukiwania.

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. Jako algorytm 2-żal użyty został algorytm zaimplementowany w ramach pierwszego sprawozdania. Każdy z algorytmów uruchomiony został 100 razy. Dla algorytmu 2-żal w każdej ze 100 iteracji wybierany był po kolei każdy wierzchołek początkowy jako start pierwszego cyklu i najdalszy od niego wierzchołek jako start drugiego cyklu. Dla algorytmu, w którym rozwiązanie startowe jest losowe w każdej ze 100 iteracji rozwiązanie początkowe było na nowo losowane.

W liście możliwych ruchów znajdują się ruchy oznaczające wymianę wierzchołków pomiędzy cyklami oraz w zależności od rozpatrywanego rodzaju sąsiedztwa:

- ruchy oznaczające wymianę wierzchołków w ramach jednego cyklu
- ruchy oznaczające wymianę krawędzi w cyklu

W poniższych pseudokodach wykorzystana została notacja:

- `distances[a][b]` - oznacza odległość pomiędzy punktami **a** i **b**

Algorytm delty funkcji celu

- Wymiana wierzchołków A i B pomiędzy dwoma cyklami

```
ustaw deltę na 0.  
odejmij od delty distances[wierzchołek przed A][A] oraz distances[A][wierzchołek po A].  
odejmij od delty distances[wierzchołek przed B][B] oraz distances[B][wierzchołek po B].  
dodaj do delty distances[wierzchołek przed A][B] oraz distances[wierzchołek po A][B].  
dodaj do delty distances[wierzchołek przed B][A] oraz distances[wierzchołek po B][A].  
zwróć deltę.
```

- Zamiana miejscami wierzchołków A i B w cyklu

```
ustaw deltę na 0.  
odejmij od delty distances[wierzchołek przed A][A] oraz distances[A][wierzchołek po A].  
odejmij od delty distances[wierzchołek przed B][B] oraz distances[B][wierzchołek po B].  
dodaj do delty distances[wierzchołek przed A][B] oraz distances[B][wierzchołek po A].  
dodaj do delty distances[wierzchołek przed B][A] oraz distances[A][wierzchołek po B].  
jeżeli wierzchołki A i B sąsiadują ze sobą  
{  
    dodaj do delty 2*distances[A][B].  
}  
zwróć deltę.
```

- Zamiana miejscami krawędzi A->(wierzchołek po A) i B->(wierzchołek po B)

```
ustaw deltę na 0.  
odejmij od delty distances[A][wierzchołek po A] oraz distances[B][wierzchołek po B].  
dodaj do delty distances[A][B] oraz distances[wierzchołek po A][wierzchołek po B].  
zwróć deltę.
```

Algorytm lokalnego błędzenia

```
wygeneruj rozwiązanie startowe i listę możliwych ruchów.  
oblicz łączną długość rozwiązania startowego.  
dopóki nie skończy się czas przeznaczony na działanie algorytmu  
{  
    wylosuj ruch z listy możliwych.  
    oblicz deltę korzystając z odpowiedniego algorytmu.  
    wykonaj ruch.  
    dodaj do łącznej długości obliczoną deltę.
```

```
    jeżeli łączna długość jest najmniejsza z dotychczasowych to zapamiętaj ją.  
}  
zwróć zapamiętaną łączną długość.
```

Algorytm zachłannego lokalnego przeszukiwania

```
wygeneruj rozwiązanie startowe i listę możliwych ruchów.  
powtarzaj  
{  
    przetasuj listę możliwych ruchów.  
    dla każdego ruchu w liście  
    {  
        oblicz deltę korzystając z odpowiedniego algorytmu.  
        jeżeli delta jest ujemna to wykonaj ruch i zakończ pętlę.  
    }  
    jeżeli nie została znaleziona żadna ujemna delta to zakończ pętlę.  
}  
zwróć rozwiązanie.
```

Algorytm stromego lokalnego przeszukiwania

```
wygeneruj rozwiązanie startowe i listę możliwych ruchów.  
powtarzaj  
{  
    dla każdego ruchu w liście dotyczącego wymiany wierzchołków pomiędzy cyklami  
    {  
        oblicz deltę korzystając z odpowiedniego algorytmu.  
        jeżeli delta jest najmniejsza z dotychczasowych to zapamiętaj ją oraz ruch.  
    }  
    jeżeli używane ruchy wewnętrztrasowe to wymiana wierzchołków  
    {  
        dla każdego ruchu w liście dotyczącego wymiany wierzchołków w cyklu  
        {  
            oblicz deltę korzystając z odpowiedniego algorytmu.  
            jeżeli delta jest najmniejsza z dotychczasowych to zapamiętaj ją oraz ruch.  
        }  
    }  
    jeżeli używane ruchy wewnętrztrasowe to wymiana krawędzi  
    {  
        dla każdego ruchu w liście dotyczącego wymiany krawędzi w cyklu  
        {  
            oblicz deltę korzystając z odpowiedniego algorytmu.  
            jeżeli delta jest najmniejsza z dotychczasowych to zapamiętaj ją oraz ruch.  
        }  
    }  
    wykonaj zapamiętany ruch.  
    jeżeli nie została znaleziona żadna ujemna delta to zakończ pętlę.
```

}

zwrć rozwiązanie.

Wyniki

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

			kroA100			kroB100		
Rozwiązanie początkowe	Przeszukiwanie	Sąsiedztwo	min	mean	max	min	mean	max
Losowe	Brak (rozwiązanie startowe)		154430	170325	190820	144358	166180	184229
	Losowe błądzenie		132402	141202	147487	128926	139112	143824
	Zachłanne	Wierzchołki	36281	44947	56245	36570	44582	52943
		Krawędzie	26286	29476	33116	27387	30127	33795
	Strome	Wierzchołki	37023	46350	56882	39659	46747	56728
		Krawędzie	26370	29293	32858	27001	29924	32460
Regret	Brak (rozwiązanie startowe)		22922	25870	30183	23876	27094	31185
	Losowe błądzenie		24008	30714	41296	24017	30634	40911
	Zachłanne	Wierzchołki	22746	25112	28644	23453	26103	30859
		Krawędzie	22141	24603	29245	23412	25615	28790
	Strome	Wierzchołki	22746	25111	28673	23453	26115	30859
		Krawędzie	22093	24325	28776	23412	25448	28790

W tabeli poniżej przedstawione zostały czasy wykonania poszczególnych algorytmów w milisekundach.

			kroA100			kroB100		
			czas [ms]					
Rozwiązanie początkowe	Przeszukiwanie	Sąsiedztwo	min	średnia	max	min	średnia	max
Losowe	Zachłanne	Wierzchołki	264.53	343.83	516.78	254.95	333.27	438.70

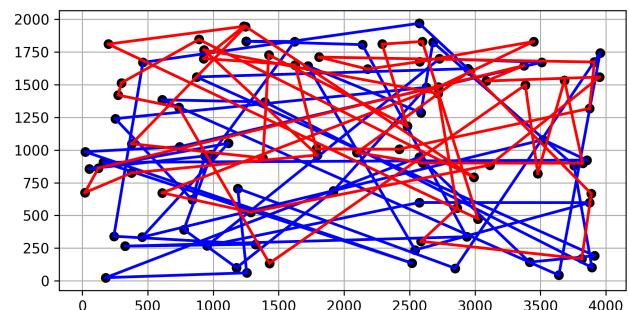
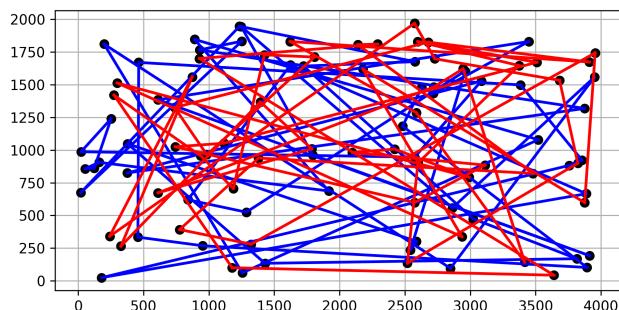
			kroA100			kroB100		
		Krawędzie	285.80	336.63	421.87	256.338	331.508	391.19
	Strome	Wierzchołki	68.68	97.81	133.38	78.32	99.04	136.48
		Krawędzie	64.20	78.66	110.12	64.89	80.663	110.76
2-żal	Zachłanne	Wierzchołki	2.44	11.17	40.31	2.37	11.72	30.74
		Krawędzie	2.49	15.27	46.66	5.56	17.78	39.18
	Strome	Wierzchołki	1.61	7.40	19.35	1.62	7.72	17.55
		Krawędzie	1.41	8.66	18.31	3.52	10.33	21.34

Poniżej umieszczone zostały wizualizacje najlepszych z uzyskanych wyników:

Wizualizacja kroA100.tsp

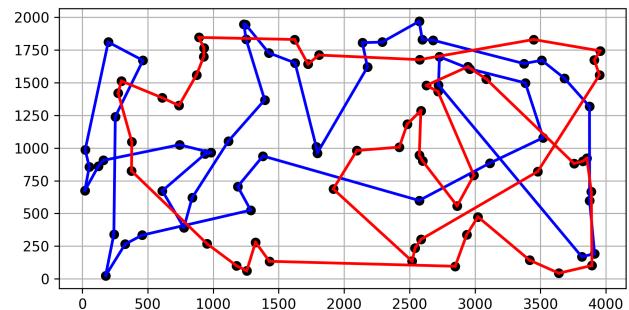
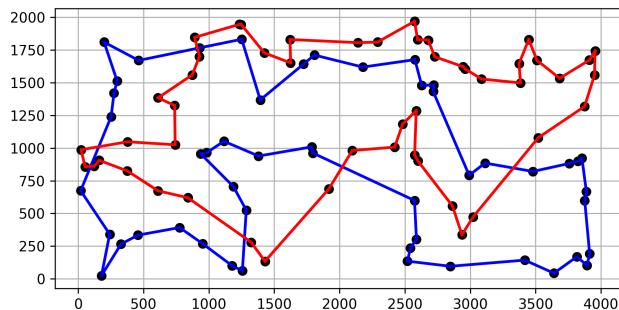
Losowe

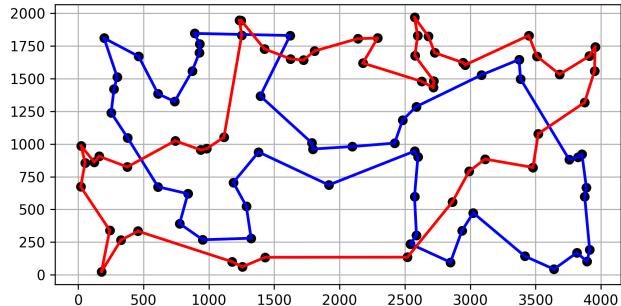
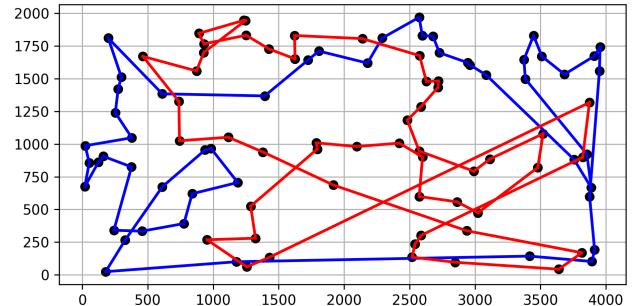
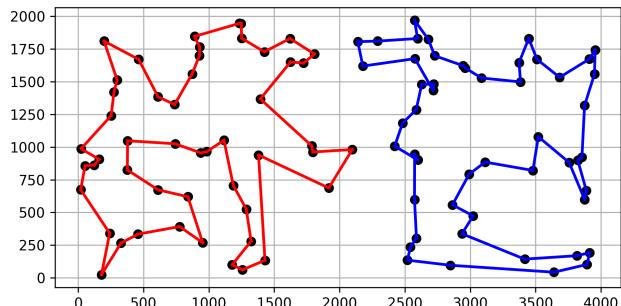
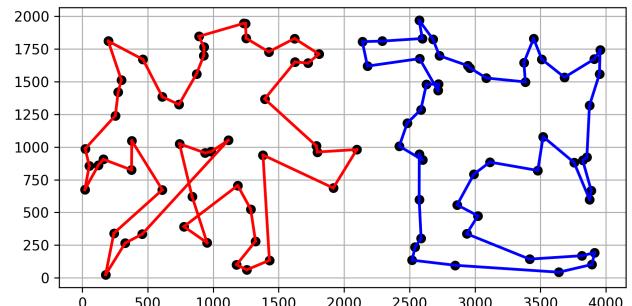
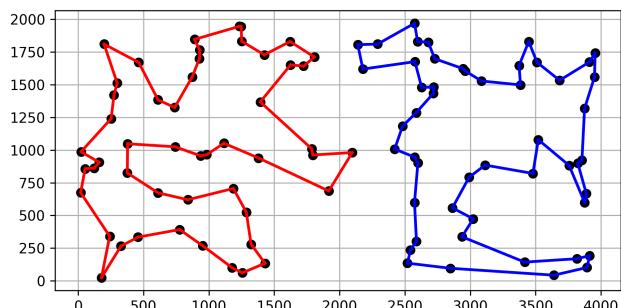
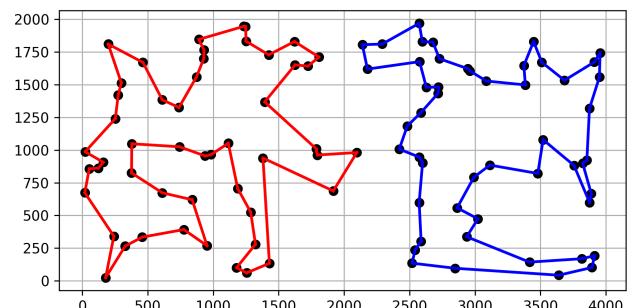
Losowe z losowym błędzeniem

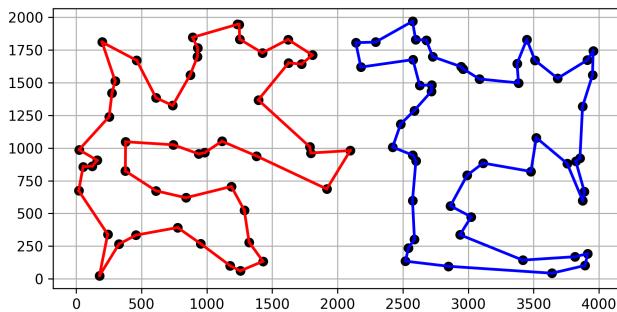
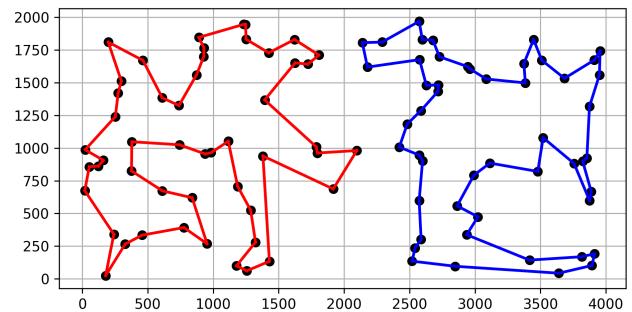
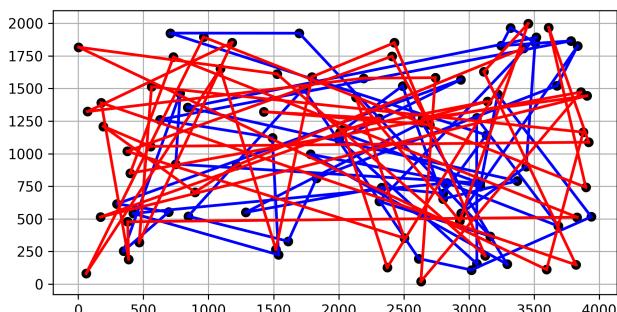
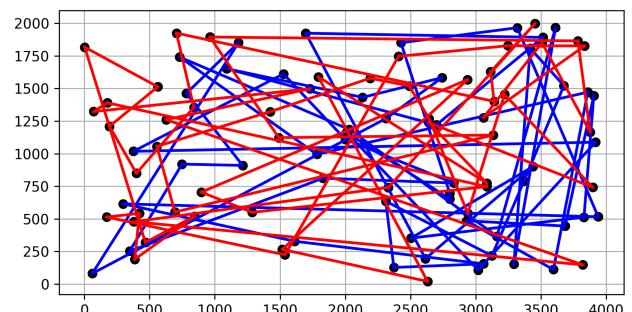
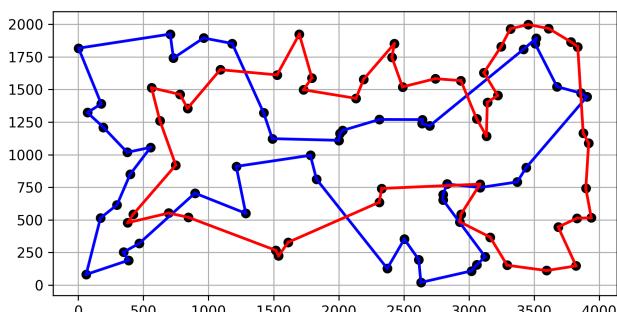
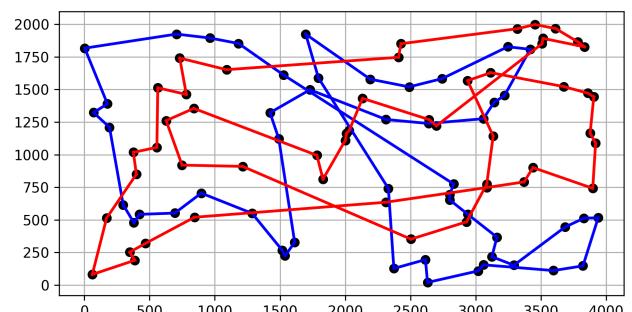


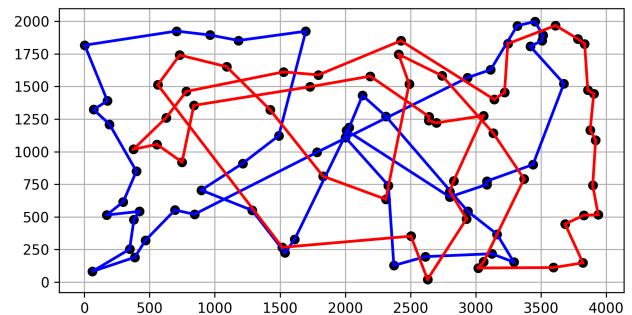
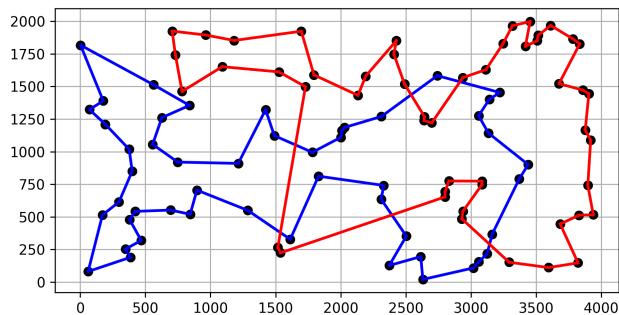
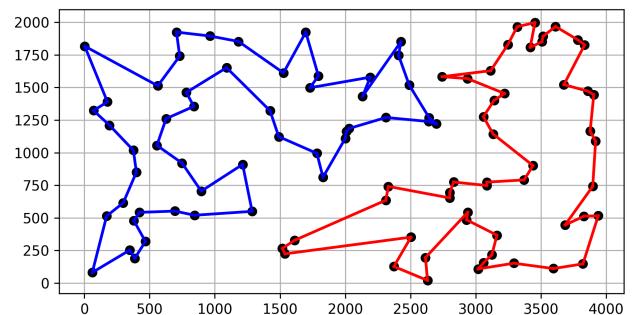
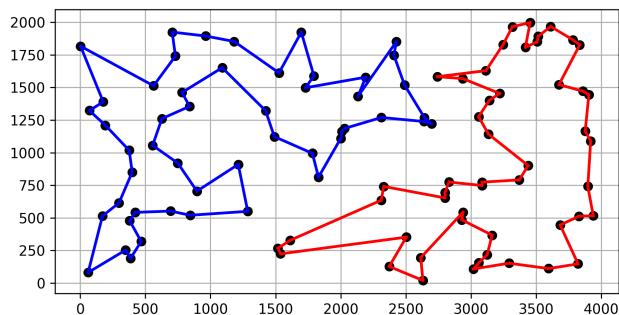
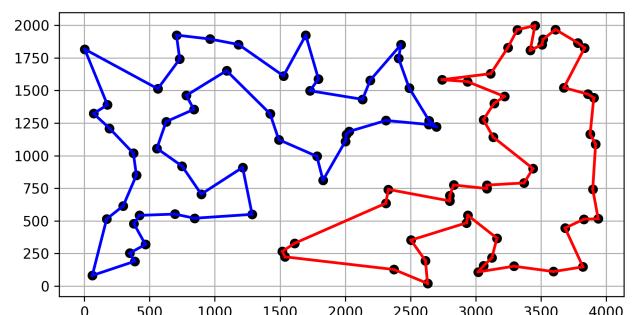
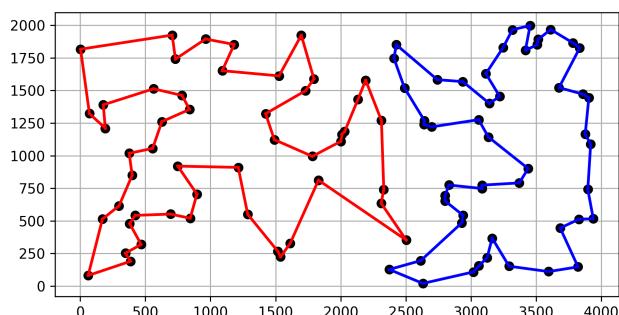
Losowe, zachłanne, krawędzie

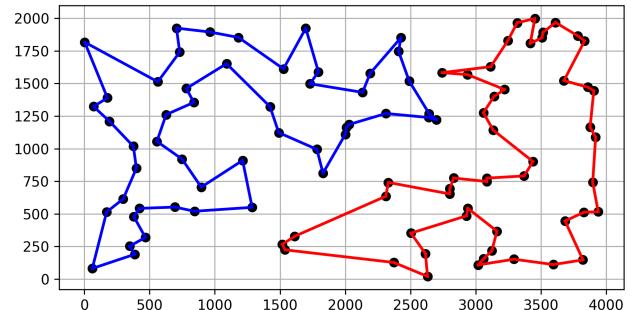
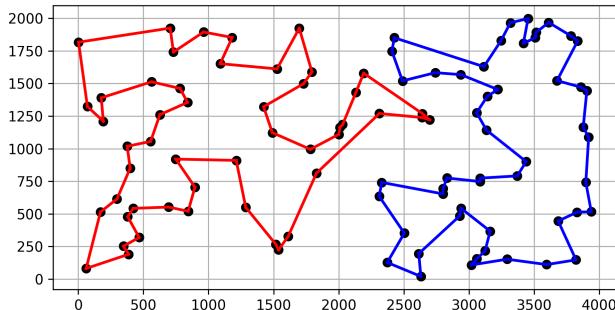
Losowe, zachłanne, wierzchołki



Łosowe, strome, krawędzie**Łosowe, strome, wierzchołki****2-żal****2-żal z losowym błędzeniem****2-żal, zachłanne, krawędzie****2-żal, zachłanne, wierzchołki**

2-żal, strome, krawędzie**2-żal, strome, wierzchołki****Wizualizacja kroB100.tsp****Losowe****Losowe z losowym błędzeniem****Losowe, zachłanne, krawędzie****Losowe, zachłanne, wierzchołki****Losowe, strome, krawędzie****Losowe, strome, wierzchołki**

Losowe, strome, krawędzie**Losowe, strome, wierzchołki****2-żal****2-żal z losowym błędzeniem****2-żal, zachłanne, krawędzie****2-żal, zachłanne, wierzchołki**

2-żal, strome, krawędzie**2-żal, strome, wierzchołki**

Wnioski

1. Lokalne przeszukiwanie jest efektywną metodą poprawy rozwiązania bazowego. W każdym przypadku udało się poprawić rozwiązanie wygenerowane jako startowe. Dla rozwiązań losowych jest to spora poprawa - w średnim przypadku o 130 tysięcy jednostek (z 170 tysięcy do 40 tysięcy), a dla heurystyki 2-żalu o 1000 jednostek (z 26 tysięcy do 25 tysięcy).
2. W rozpatrywanym problemie lepiej wypada sąsiedztwo definiowane jako wymiana dwóch krawędzi. Jest to prawdziwe dla każdego zbadanego przypadku.
3. Algorytm stromego lokalnego przeszukiwania radzi sobie lepiej niż algorytm zachłannego lokalnego przeszukiwania w przypadku średnim dla zamiany krawędzi. Dla sąsiedztwa, w którym możliwa jest zamiana wierzchołków sytuacja nie jest tak klarowna - dla startowych rozwiązań losowych algorytm zachłanny radzi sobie lepiej i jest to różnica około 1500 jednostek, natomiast dla rozwiązań startowych wygenerowanych heurystyką 2-żal wyniki są do siebie bardzo zbliżone, a często nawet identyczne.
4. Czasy wykonania algorytmów są w sporej mierze uzależnione od rozwiązania startowego. Dla rozwiązań losowych istnieje dużo ruchów, które zdolne są je poprawić, dlatego przetwarzanie trwa o wiele dłużej niż dla rozwiązania startowego 2-żal, gdzie jest mało ruchów, które można wykonać i są one w stanie poprawić ogólny wynik.
5. Różnica w czasie działania pomiędzy algorytmem zachłannym, a stromym wynika z tego, że w przeszukiwaniu zachłannym rozwiązanie poprawiane jest małymi krokami - mimo tego, że nie szukamy lokalnie najlepszej poprawy, tylko akceptujemy pierwszą, którą znajdziemy, to wykonujemy tych popraw bardzo dużo, a przez to często zmieniamy cykle. W przeszukiwaniu stromym podejmujemy lokalnie najlepsze ruchy, więc mimo tego że musimy policzyć dla każdego deltę, to wykonujemy jedynie jeden ruch, który mocno poprawia rozwiązanie.