

POLITECHNIKA POZNAŃSKA

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

INSTYTUT INFORMATYKI

ZAKŁAD INTELIGENTNYCH SYSTEMÓW WSPOMAGANIA  
DECYZJI



LOKALNE PRZESZUKIWANIE

INTELIGENTNE METODY OPTYMALIZACJI

PAWEŁ CHUMSKI, 144392

PAWEŁ.CHUMSKI@STUDENT.PUT.POZNAN.PL

MICHAŁ CIESIELSKI, 145325

MICHAŁ.P.CIESIELSKI@STUDENT.PUT.POZNAN.PL

PROWADZĄCY:

PROF. DR HAB. INŻ. ANDRZEJ JASZKIEWICZ

ANDRZEJ.JASZKIEWICZ@CS.PUT.POZNAN.PL

02-04-2023

## Spis treści

|   |           |
|---|-----------|
| <b>Wstęp</b>  | <b>3</b>  |
| <b>1 Algorytmy lokalnego przeszukiwania</b>             | <b>3</b>  |
| 1.1 Wykorzystanie delty funkcji celu                    | 3         |
| 1.2 Algorytm w wersji zachłannej                        | 3         |
| 1    1 Opis algorytmu                                   | 3         |
| 1    2 Pseudokod  | 4         |
| 1.3 Algorytm w wersji stromej                           | 4         |
| 1    1 Opis algorytmu                                   | 4         |
| 1    2 Pseudokod  | 5         |
| 1.4 Algorytm losowego błędzenia                         | 5         |
| 1    1 Opis algorytmu                                   | 5         |
| 1    2 Pseudokod  | 6         |
| <b>2 Wizualizacje najlepszych rozwiązań</b>             | <b>6</b>  |
| 2.1 kroA100   | 6         |
| 2.2 kroB100   | 9         |
| <b>3 Porównanie wyników eksperymentu obliczeniowego</b> | <b>11</b> |
| <b>Podsumowanie</b>                                     | <b>12</b> |
| Wnioski   | 12        |
| Kod programu  | 12        |

## WSTĘP

Celem zadania była implementacja algorytmów lokalnego przeszukiwania w celu rozwiązania i poprawy zmodyfikowanego problemu komiwojażera. Algorytmy te startują z wybranego rozwiązania początkowego, a następnie przeglądają przestrzeń możliwych rozwiązań dostępnych w danym kroku, akceptując rozwiązania przynoszące poprawę.

Pod uwagę wzięto algorytmy lokalnego przeszukiwania w wersji stromej (steepest) i zachłannej (greedy). W obu wersjach wykorzystano dwa rodzaje ruchów - międzytrasowe oraz wewnątrztrasowe. Dla ruchów wewnątrztrasowych wyróżniamy dwa rodzaje zmiany aktualnego rozwiązania - poprzez wymianę dwóch wierzchołków lub dwóch krawędzi. Bazowe rozwiązania zostały wygenerowane na podstawie losowego doboru wierzchołków oraz rozwiązania najlepszą heurystyką z poprzednich zajęć (ważony 2-żal). Jako punkt odniesienia do łącznie testowanych 8 kombinacji wykorzystano algorytm losowego błędzenia, który w każdej iteracji wykonuje losowo wybrany ruch (niezależnie od jego oceny) i zwraca najlepsze znalezione w ten sposób rozwiązanie. Algorytm ten działa w takim samym czasie jak średnio najwolniejsza z wersji lokalnego przeszukiwania.

Eksperymenty zostały przeprowadzone na instancjach kroA100.tsp oraz kroB100.tsp z biblioteki TSPLib.

## ALGORYTMY LOKALNEGO PRZESZUKIWANIA

### 1.1 WYKORZYSTANIE DELTY FUNKCJI CELU

Do badania jakości proponowanego rozwiązania wykorzystano algorytmy obliczające deltę funkcji celu w trzech wariantach: wymiana wierzchołków między cyklami, wymiana wierzchołków wewnątrz cyklu, wymiana krawędzi wewnątrz cyklu. Każdy z nich, w zależności od badanego sąsiedztwa, liczy różnicę wyników po rozwiązaniu i przed rozwiązaniem. Implementacja delty jest dość prosta, różni się nieznacznie w poszczególnych przypadkach. Jest niezbędna do wykrywania poprawy w rozpatrywanych algorytmach lokalnego przeszukiwania.

### 1.2 ALGORYTM W WERSJI ZACHŁANNEJ

#### 1 OPIS ALGORYTMU

Implementacja opiera się na sprawdzeniu każdej możliwej zmiany wierzchołków pomiędzy dwoma cyklami, do momentu znalezienia ruchu, dla którego delta jest dodatnia ujemna - wtedy nowe rozwiązanie posiada krótszą długość ścieżek. Proces ulepszania jest przerywany w momencie, kiedy zostanie znalezione rozwiązanie polepszające długość cykli. Przy wybieraniu rozwiązań została wykorzystana funkcja shuffle z modułu random w języku Python. Zwraca ona w naszym przypadku listę kandydatów na rozwiązanie polepszające w losowej kolejności.

## 2 PSEUDOKOD

- Wymiana wierzchołków między cyklami w wersji zachłannej:

```
1 Znajdź wszystkie możliwe kombinacje par z obu cykli do wymiany i poprawy bieżącego
  rozwiązania;
2 Wymieszaj kolejność wygenerowanych kombinacji (metoda shuffle );
3 for (i, j) in candidates do
4   | while Nowe rozwiązanie jest gorsze (delta > 0) do
5   |   | sprawdź kolejną parę do wymiany i znajdź nową wartość delta;
6   | end
7 end
8 Popraw bieżące rozwiązanie wymieniając odpowiednie wierzchołki w parze dla obu cykli;
9 return delta;
```

- Wymiana wewnątrz cyklu w wersji zachłannej:

```
1 Wymieszaj kolejność wierzchołków dla obu cykli (metoda shuffle );
2 for cycle in cycles do
3   | Znajdź wszystkie możliwe kombinacje par z cyklu do wymiany i poprawy bieżącego
    | rozwiązania;
4   | Wymieszaj kolejność wygenerowanych kombinacji (metoda shuffle );
5   | for (i, j) in candidates do
6   |   | while Nowe rozwiązanie jest gorsze (delta > 0) do
7   |   |   | sprawdź kolejną parę do wymiany i znajdź nową wartość delta
8   |   |   | end
9   |   | end
10 end
11 Popraw bieżące rozwiązanie wymieniając odpowiednie wierzchołki w parze dla cyklu;
12 return delta;
```

- Główna funkcja przeszukująca:

```
1 while True do
2   | wymieszaj listę dostępnych ruchów (wewnątrztrasowe i międzytrasowe);
3   | wywołuj funkcję wymiany wierzchołków między cyklami do momentu poprawy ( $\text{delta} \leq 0$ );
4   | wywołuj funkcję wymiany wewnątrz cyklu (wierzchołków lub krawędzi) do momentu
    | poprawy ( $\text{delta} \leq 0$ );
5 end
6 return cykle z zastosowanym poprawionym rozwiązaniem;
```

### 1.3 ALGORYTM W WERSJI STROMEJ

#### 1 OPIS ALGORYTMU

Algorytm lokalnego przeszukania w wersji stromej polega na wybieraniu w każdym kroku ruchu, który maksymalnie poprawia jakość rozwiązania. Implementacja opiera się na porównywaniu długości cykli przed i po wykonaniu ruchu, a następnie wyborze najlepszego ruchu. Proces ulepszania jest kontynuowany, dopóki istnieją ruchy, które poprawiają jakość rozwiązania. Przy wyborze kolejnego ruchu, rozważane są wszystkie możliwe pary wierzchołków, a następnie wybierana jest ta para, dla której ruch maksymalnie skraca długość cykli.

## 2 PSEUDOKOD

- Wymiana wierzchołków między cyklami w wersji stromej:

```
1 Znajdź wszystkie możliwe kombinacje par z obu cykli do wymiany i poprawy bieżącego
rozwiązania;
2 Zainicjalizuj pustą listę delta_scores;
3 for (i, j) in candidates do
4     | znajdź wartość delta dla pary (i, j) zawierającej wierzchołki z obu cykli i dodaj ją do listy
    | delta_scores;
5 end
6 return najmniejszą wartość z listy delta_scores;
```

- Wymiana wewnątrz cyklu w wersji stromej:

```
1 Zainicjalizuj pustą listę delta_scores;
2 for cycle in cycles do
3     | for (i, j) in inside_candidates do
4         | znajdź wartość delta dla bieżącej kombinacji wierzchołków i aktualnego cyklu;
5         | dodaj tę wartość do listy delta_scores;
6     | end
7 end
8 return najmniejszą wartość z listy delta_scores;
```

- Główna funkcja przeszukująca:

```
1 while True do
2     | Zainicjalizuj pustą listę delta_scores;
3     | for move in moves do
4         | znajdź wartość delta dla bieżącego ruchu (wewnątrztrasowy - wymiana wierzchołków lub
        | krawędzi, międzytrasowy);
5         | dodaj tę wartość do listy delta_scores;
6     | end
7     | zastosuj rozwiązanie z minimalną wartością delta z listy delta_scores;
8 end
9 return cykle z zastosowanym poprawionym rozwiązaniem;
```

### 1.4 ALGORYTM LOSOWEGO BŁĄDZENIA

#### 1 OPIS ALGORYTMU

Algorytm losowego błędzenia polega na losowym wyborze początkowej permutacji wierzchołków, a następnie wykonuje określoną liczbę iteracji, w każdej zmieniając losowo wybrane pary wierzchołków. Nie ma znaczenia, czy taki ruch poprawia jakość rozwiązania, czy też nie. Jeśli podczas przeszukiwania zostanie znalezione rozwiązanie, które skraca długość cyklu, algorytm zwróci ten wynik. Ostatecznie algorytm zwróci najlepsze rozwiązanie znalezione w procesie losowego przeszukiwania. Algorytm ten działa w takim samym czasie jak średnio najwolniejsza z wersji lokalnego przeszukiwania na danej instancji.

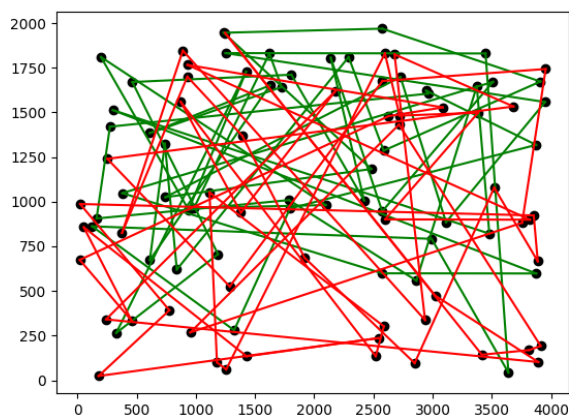
## 2 PSEUDOKOD

- Algorytm losowego błędzenia:

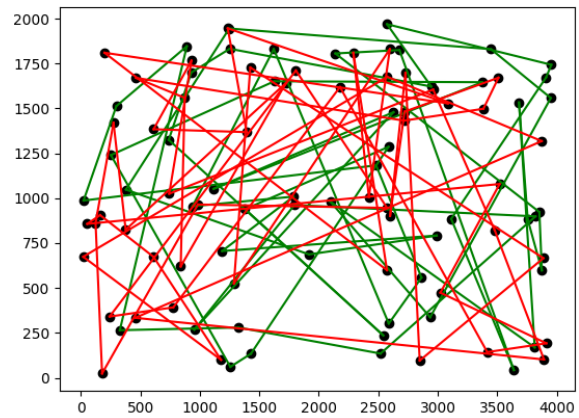
```
1 przypisz do zmiennej best_solution aktualny układ cykli, do zmiennej best_score aktualną
  długość tras;
2 while total_time < time_limit do
3   wybierz losowo jeden z dostępnych ruchów: międzytrasowy, wewnątrztrasowy z wymianą
     wierzchołków, wewnątrztrasowy z wymianą krawędzi;
4   wykonaj wylosowany ruch;
5   przypisz nową długość tras do zmiennej new_score;
6   if new_score < best_score then
7     przypisz aktualny układ cykli do zmiennej best_solution;
8     przypisz nową długość tras do zmiennej best_score;
9   end
10 end
11 return best_solution;
```

## WIZUALIZACJE NAJLEPSZYCH ROZWIĄZAŃ

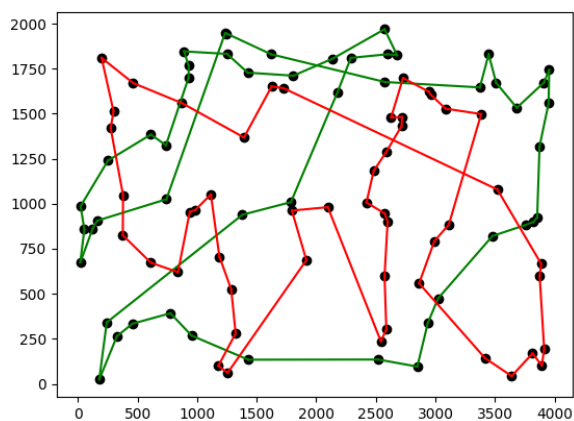
### 2.1 KROA100



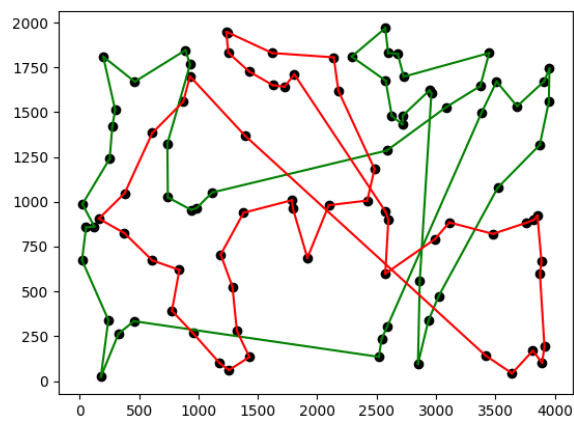
Rys. 1. Random solution, None



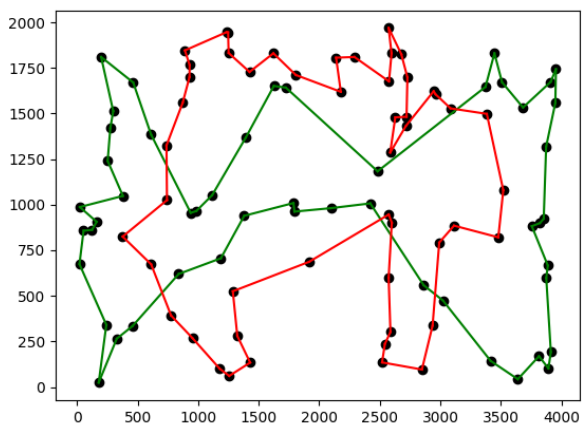
Rys. 2. Random solution, Random Search



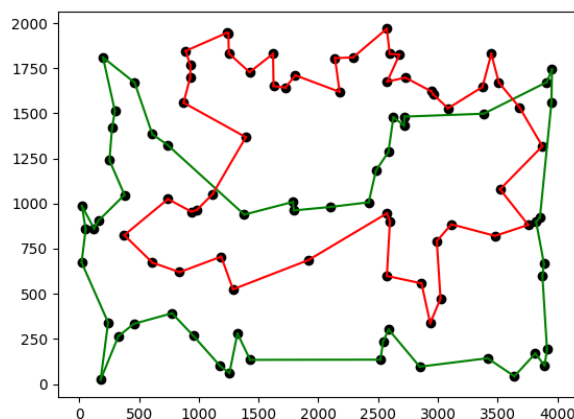
Rys. 3. Random solution, Greedy Search, Vertices



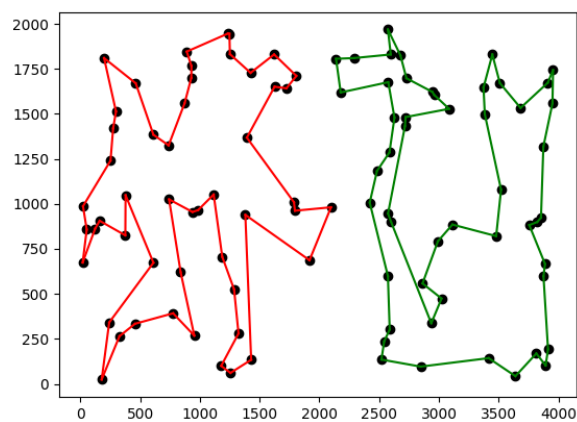
Rys. 4. Random solution, SteepestSearch, Vertices



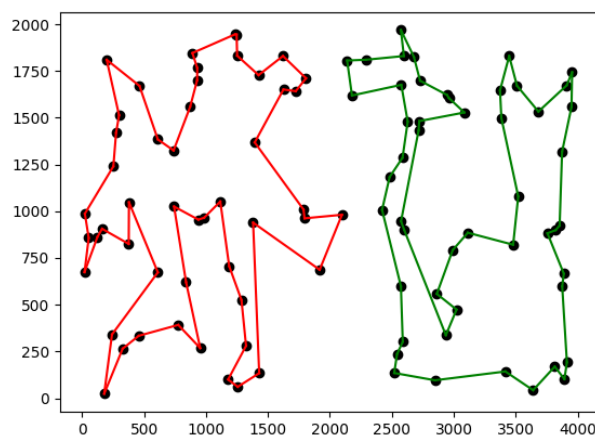
Rys. 5. Random solution, Greedy Search, Edges



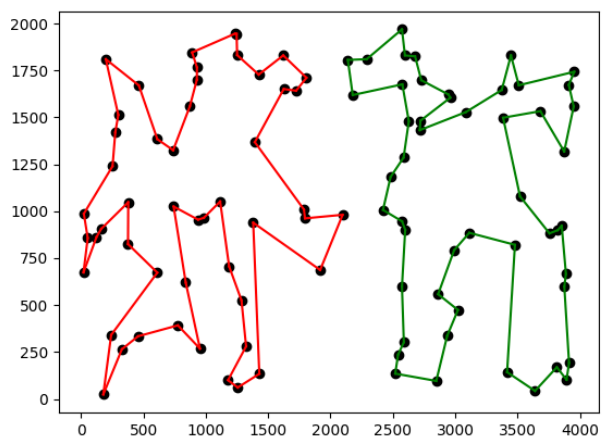
Rys. 6. Random solution, SteepestSearch, Edges



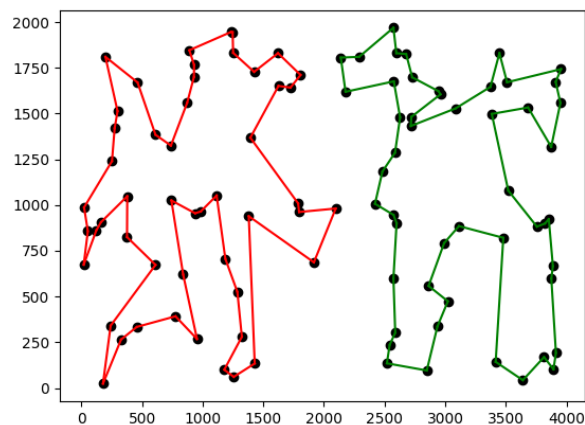
Rys. 7. Regret heuristic, None



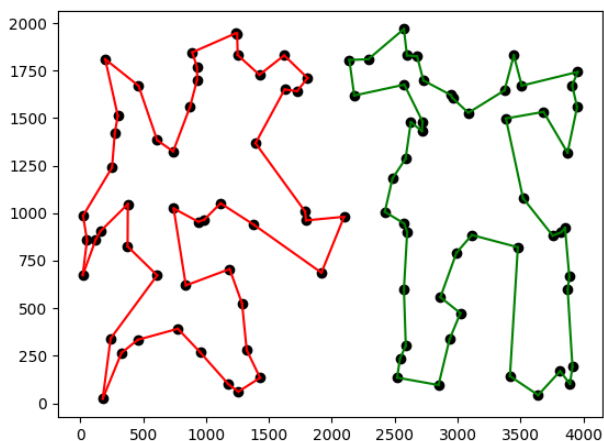
Rys. 8. Regret heuristic, Random Search



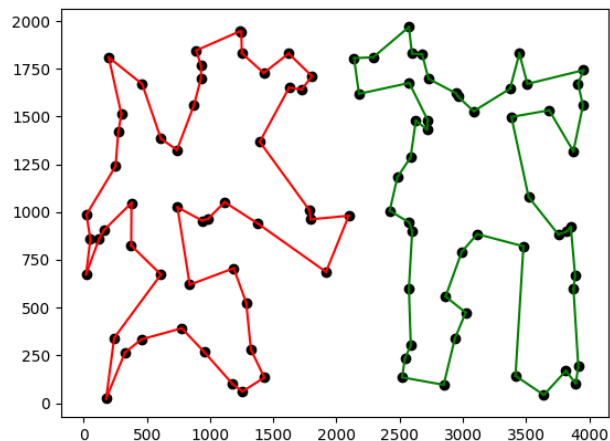
Rys. 9. Regret heuristic, Greedy Search, Vertices



Rys. 10. Regret heuristic, SteepestSearch, Vertices



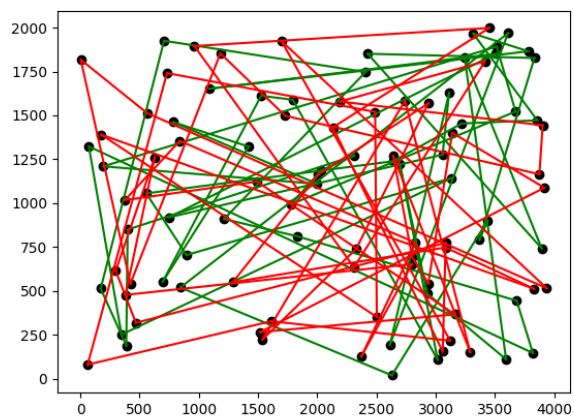
Rys. 11. Regret heuristic, Greedy Search, Edges



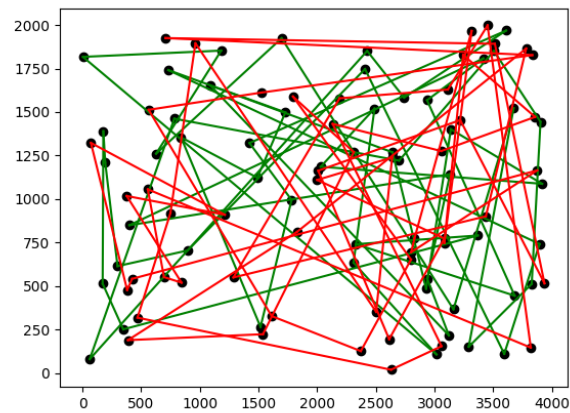
Rys. 12. Regret heuristic, SteepestSearch, Edges



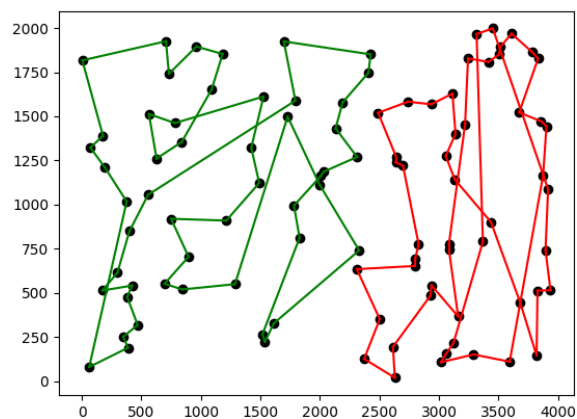
## 2.2 KROB100



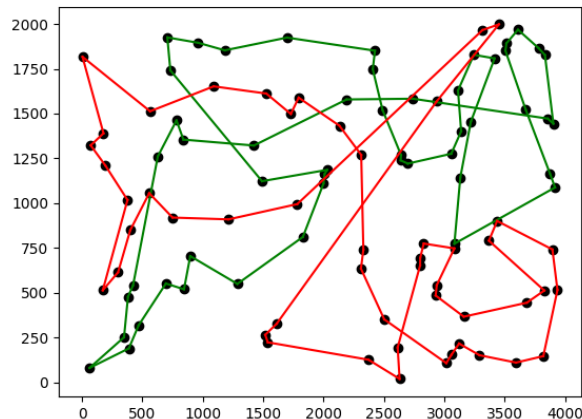
Rys. 13. Random solution, None



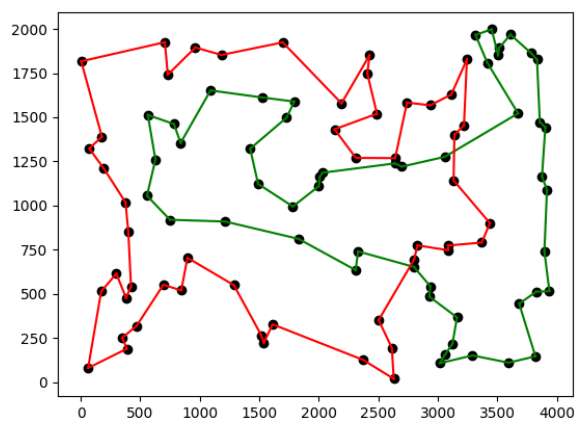
Rys. 14. Random solution, Random Search



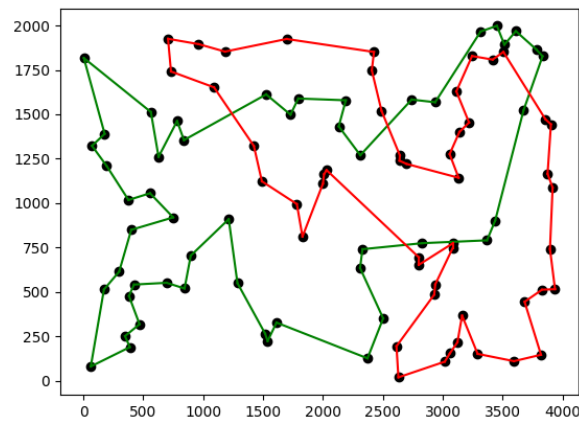
Rys. 15. Random solution, Greedy Search, Vertices



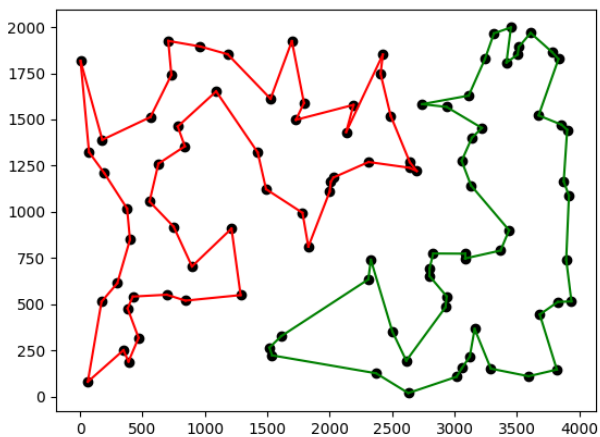
Rys. 16. Random solution, SteepestSearch, Vertices



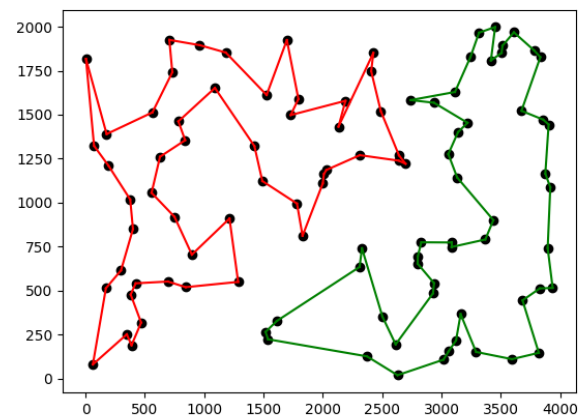
Rys. 17. Random solution, Greedy Search, Edges



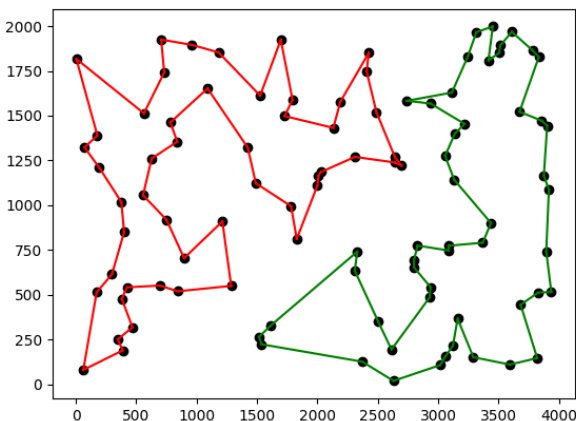
Rys. 18. Random solution, SteepestSearch, Edges



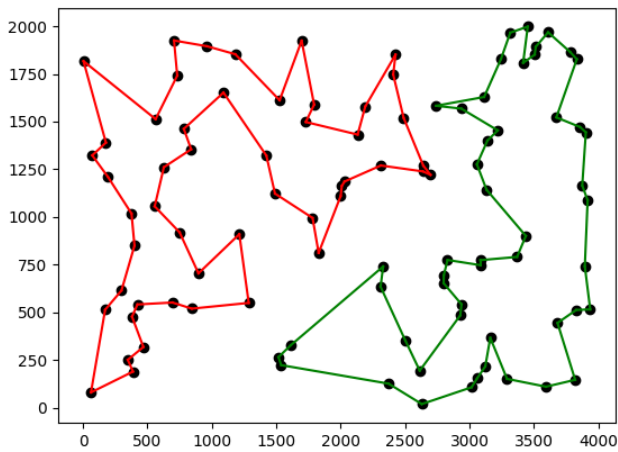
Rys. 19. Regret heuristic, None



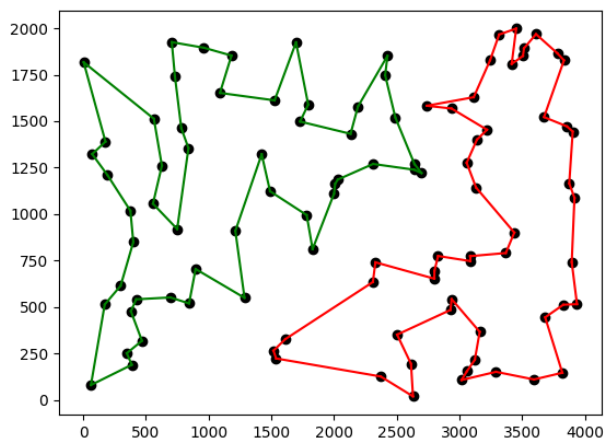
Rys. 20. Regret heuristic, Random Search



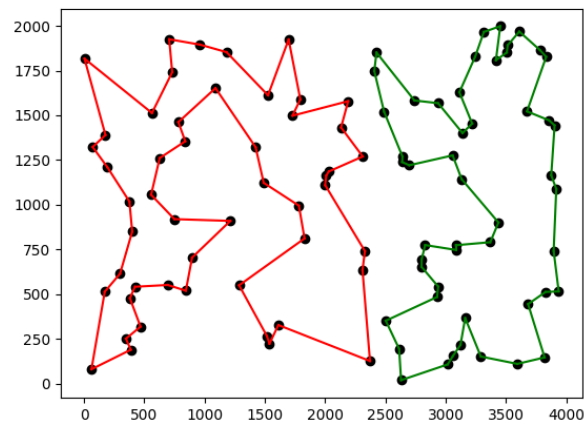
Rys. 21. Regret heuristic, Greedy Search, Vertices



Rys. 22. Regret heuristic, SteepestSearch, Vertices



Rys. 23. Regret heuristic, Greedy Search, Edges



Rys. 24. Regret heuristic, SteepestSearch, Edges

## PORÓWNANIE WYNIKÓW EKSPERYMENTU OBLICZENIOWEGO

Każdy algorytm był testowany 100 razy dla każdej instancji. Dla rozwiązań początkowych uzyskiwanych za pomocą 2-Żalu za każdym razem był wybierany po kolei inny wierzchołek startowy. Dla rozwiązań początkowych losowych, za każdym razem losowane było inne rozwiązanie losowe.

Tab. 1. Tabela zawierająca długości ścieżek dla poszczególnych algorytmów

| Instancja              |                       | kroA100.tsp |         |          | kroB100.tsp |         |          |
|------------------------|-----------------------|-------------|---------|----------|-------------|---------|----------|
| Rozwiązanie początkowe | Losowe przeszukiwanie | Minimum     | Średnia | Maksimum | Minimum     | Średnia | Maksimum |
| Losowe                 | Brak                  | 144450      | 170842  | 186209   | 144764      | 169081  | 188782   |
|                        | Losowe błędzenie      | 128924      | 143393  | 149901   | 128737      | 141674  | 147244   |
|                        | Zachłanny-wierzchołki | 32370       | 40465   | 50511    | 31930       | 40591   | 49689    |
|                        | Stromy-wierzchołki    | 32499       | 42875   | 50875    | 34818       | 42405   | 53311    |
|                        | Zachłanny-krawędzie   | 25584       | 27648   | 30896    | 25668       | 28225   | 30503    |
|                        | Stromy-krawędzie      | 22179       | 24911   | 28116    | 23361       | 25128   | 27361    |
| 2-Żal                  | Brak                  | 23729       | 26573   | 32441    | 23828       | 27561   | 29793    |
|                        | Losowe błędzenie      | 23729       | 26573   | 32441    | 23828       | 27561   | 29793    |
|                        | Zachłanny-wierzchołki | 23199       | 25395   | 30915    | 23450       | 25870   | 28172    |
|                        | Stromy-wierzchołki    | 23199       | 25357   | 30746    | 23450       | 25769   | 28254    |
|                        | Zachłanny-krawędzie   | 22179       | 25026   | 28389    | 23327       | 25324   | 27353    |
|                        | Stromy-krawędzie      | 22179       | 24911   | 28116    | 23361       | 25128   | 27361    |

Tab. 2. Tabela zawierająca czasy wykonania dla algorytmów przeszukiwania lokalnego

| Instancja              |                       | kroA100.tsp |             |              | kroB100.tsp |             |              |
|------------------------|-----------------------|-------------|-------------|--------------|-------------|-------------|--------------|
| Rozwiązanie początkowe | Losowe przeszukiwanie | Minimum [s] | Średnia [s] | Maksimum [s] | Minimum [s] | Średnia [s] | Maksimum [s] |
| Losowe                 | Zachłanny-wierzchołki | 0.668309    | 1.553914    | 2.388938     | 0.762002    | 1.553679    | 2.401998     |
|                        | Stromy-wierzchołki    | 1.950886    | 3.276431    | 4.330033     | 1.648003    | 3.452736    | 5.125548     |
|                        | Zachłanny-krawędzie   | 0.844566    | 1.482320    | 2.374819     | 0.801265    | 1.428275    | 2.038041     |
|                        | Stromy-krawędzie      | 1.230108    | 2.446090    | 3.202898     | 1.496318    | 2.440500    | 3.308036     |
| 2-Żal                  | Zachłanny-wierzchołki | 0.043996    | 0.179921    | 0.466757     | 0.045001    | 0.199190    | 0.477996     |
|                        | Stromy-wierzchołki    | 0.071001    | 0.299266    | 0.686514     | 0.098024    | 0.320772    | 0.706000     |
|                        | Zachłanny-krawędzie   | 0.052006    | 0.202597    | 0.703121     | 0.079001    | 0.246190    | 0.580997     |
|                        | Stromy-krawędzie      | 0.060052    | 0.304799    | 1.071845     | 0.124997    | 0.355708    | 0.629158     |

## PODSUMOWANIE

### WNIOSKI

- Dla wariantów wewnątrztrasowych zdecydowanie lepszy okazał się wariant wymiany krawędzi niż wierzchołków. Nie istnieje sytuacja w tabeli, w której wariant wymiany wierzchołków otrzymałby lepszy wynik końcowy.
- Dla losowego rozwiązania startowego lepsze minimum osiąga algorytm zachłanny.
- Dla dobrego rozwiązania początkowego osiągniętego heurystyką 2-żalu osiągnane wyniki mają podobne wartości.
- W przypadku wielokrotnego uruchamiania danego algorytmu, lepszym algorytmem przeszukiwania lokalnego okazał się algorytm zachłanny.
- Zgodnie z założeniami, algorytm zachłanny osiąga najlepsze czasy w każdym przypadku.

### KOD PROGRAMU

<https://github.com/imo/lab2>