Inteligentne metody optymalizacji

Laboratorium nr 1

24 stycznia 2024

Prowadzący: prof. dr hab. inż. Andrzej Jaszkiewicz

Autorzy: **Krzysztof Martin** 141275
Bartosz Paliński 145224

Zajęcia poniedziałkowe, 11:45.

Oświadczam/y, że niniejsze sprawozdanie zostało przygotowane wyłącznie przez powyższych autora/ów, a wszystkie elementy pochodzące z innych źródeł zostały odpowiednio zaznaczone i są cytowane w bibliografii.

1 Opis zadania

Celem zajęć laboratoryjnych było zaimplementowanie oraz przetestowanie lokalnego przeszukiwania, rozwiązujące zmodyfikowany problem komiwojażera. Algorytmy te rozpoczynają się od wybranego rozwiązania początkowego, a następnie przeszukują sąsiedztwo dostępnych rozwiązań, akceptując tylko te, które przynoszą poprawę, dopóki jest to możliwe.

Warianty algorytmów lokalnego przeszukiwania, które zostały rozważone:

- algorytm lokalnego przeszukiwania w wersji stromej (steepest),
- algorytm lokalnego przeszukiwania w wersji zachłannej (greedy)

Dla obydwu algorytmów możliwe są 2 rodzaje ruchów:

- międzytrasowe,
- wewnątrztrasowe
  .

Ponadto dla ruchów wewnątrztrasowych możliwe są 2 rodzaje operacji:

- wymiana dwóch wierzchołków,
- wymiana dwóch krawędzi
  .

Efektywność algorytmów była oceniana na podstawie instancji kroaA100 i kroB100 z biblioteki TSPLib.

Dodatkowo, jako punkt odniesienia wykorzystano algorytm losowego błądzenia, 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.

2 Algorytmy

2.1 Delta funckji celu

Aby zbadać jakość proponowanej zamiany, obliczano deltę funckji celu. Funkcją celu w naszym przypadku jest sumaryczna długość obu cykli, delta funckji celu obliczana jest w następujący sposób:

długość wstawianych krawędzi - długość usuwanych krawędzi

Zatem delta ta będzie **minimalizowana**

2.2 Lokalne przeszukiwanie w wersji zachłannej (greedy)

2.2.1 Generatory par wierzchołków oraz krawędzi

Aby zapewnić losowe przeglądanie ruchów, generowane są losowe permutacje dostępnych par wierzchołków/krawędzi dla dwóch rodzajów wymian.

Kandydaci na wymianę wierzchołków między cyklami:

1: Przetasuj losowo zbiór wierzchołków należących do pierwszego cyklu.
2: Przetasuj losowo zbiór wierzchołków należących do drugiego cyklu.
3: Wykonaj iloczyn kartezjański obu zbiorów, operacja ta definuje kolejność przeglądania par wierzchołków.

Kandydaci na wymianę wierzchołków/krawędzi w ramach jednego cyklu:

1: Przetasuj losowo zbiór wierzchołków/krawędzi należących do danego cyklu.
2: Wygeneruj zbiór kombinacji par wierzchołków/ krawędzi, operacja ta definuje kolejność przeglądania par wierzchołków.

2.2.2 Wymiana wierzchołków między cyklami

1: **Dla każdej** każdej losowo wygenerowanej pary wierzchołków:
2: Oblicz deltę dla proponowanje zamiany
3: **Jeżeli** delta funckji celu jest mniejsza niż zero:
4: Zaaplikuj zamianę wierzchołków między cyklami do obecnego rozwiązania
5: Zwróć informację o tym, czy udało się polepszyć funkcję celu

2.2.3 Wymiana wierzchołków wewnątrz cyklu

1: **Dla każdej** każdej losowo wygenerowanej pary wierzchołków:
2: Oblicz deltę dla proponowanje zamiany
3: **Jeżeli** delta funckji celu jest mniejsza niż zero:
4: Zaaplikuj zamianę wierzchołków w ramach danego cyklu do obecnego rozwiązania
5: Zwróć informację o tym, czy udało się polepszyć funkcję celu

2.2.4 Wymiana krawędzi wewnątrz cyklu

1: **Dla każdej** każdej losowo wygenerowanej pary krawędzi:
2: Oblicz deltę dla proponowanje zamiany
3: **Jeżeli** delta funckji celu jest mniejsza niż zero:
4: Zaaplikuj zamianę krawędzi w ramach danego cyklu do obecnego rozwiązania
5: Zwróć informację o tym, czy udało się polepszyć funkcję celu

2.2.5 Funkcja przeszukująca

Dla pierwszego rodzaju sąsiedztwa:

1: **Powtarzaj:**
2: Wygeneruj losowe pary wierzchołków
3: Wykonaj funkcję wymiany wierzchołków między cyklami
4: Wykonaj funkcję wymiany wierzchołków wewnątrz cyklu
5: **Jeżeli** obie operacje nie poprawiają funckji celu
6: Zakończ wykonywanie

Dla drugiego rodzaju sąsiedztwa:

1: **Powtarzaj:**
2: Wygeneruj losowe pary wierzchołków
3: Wykonaj funkcję wymiany wierzchołków między cyklami
4: Wykonaj funkcję wymiany krawędzi wewnątrz cyklu
5: **Jeżeli** obie operacje nie poprawiają funckji celu

6: Zakończ wykonywanie

2.3 Lokalne przeszukiwanie w wersji stromej (steepest)

Dla pierwszego rodzaju sąsiedztwa:

1: **Powtarzaj:**
2: Znajdź minilaną wartość delty funckji celu wśród wszystkich międzycyklowych wymian par wierzchołków.
3: Znajdź minilaną wartość delty funckji celu wśród wszystkich wewnątrzcyklowych wymian par wierzchołków.
4: Wybierz minimalną wartość z kroku: **2,3**
5: **Jeżeli** nie poprawiono funckji celu
6: Zakończ wykonywanie
7: Zaaplikuj wyznaczoną zamianę wierzchołków dla której delta funckji celu jest najmniejsza

Dla drugiego rodzaju sąsiedztwa:

1: **Powtarzaj:**
2: Znajdź minilaną wartość delty funckji celu wśród wszystkich międzycyklowych wymian par wierzchołków.
3: Znajdź minilaną wartość delty funckji celu wśród wszystkich wewnątrzcyklowych wymian par krawędzi.
4: Wybierz minimalną wartość z kroku: **2,3**
5: **Jeżeli** nie poprawiono funckji celu
6: Zakończ wykonywanie
7: Zaaplikuj wyznaczoną zamianę wierzchołków lub krawędzi dla której delta funckji celu jest najmniejsza

2.4 Algorytm losowego błądzenia

Jako punkt odniesienia zaimplementowano również algorytm losowego błądzenia, 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 na danej instancji

1: **Dopóki** nie upłynął maksymalny czas wykonywania
2: Wybierz losową z akcji: wymiana wierzchołków między cyklami, wymiana wierzchołków wewnątrz cyklu, wymiana krawędzi wewnątrz cyklu
3: Wylosuj parę wierzchołków lub krawędzi
4: Aplikuj wybraną akcję dla wybranej pary wierzchołków/krawędzi
5: **Jeżeli** sumaryczna długość cykli jest mniejsza od obecnie minimlanej
6: Ustaw tą wartość jako nową wartość minimalną
7: Zwróć rozwiązanie z najmniejszą sumaryczną długością cykli

3 Wyniki eksperymentu obliczeniowego

Każdy algorytm został przetestowany 100 razy dla każdej instancji, za każdym razem wybierając inny wierzchołek startowy cyklu dla heurystyki 2-regret oraz generując inne losowe rozwiązanie początkowe. Średnie, najlepsze oraz najgorsze wyniki są przedstawione w poniższej tabeli.

Tabela 1: Losowe rozwiązanie początkowe

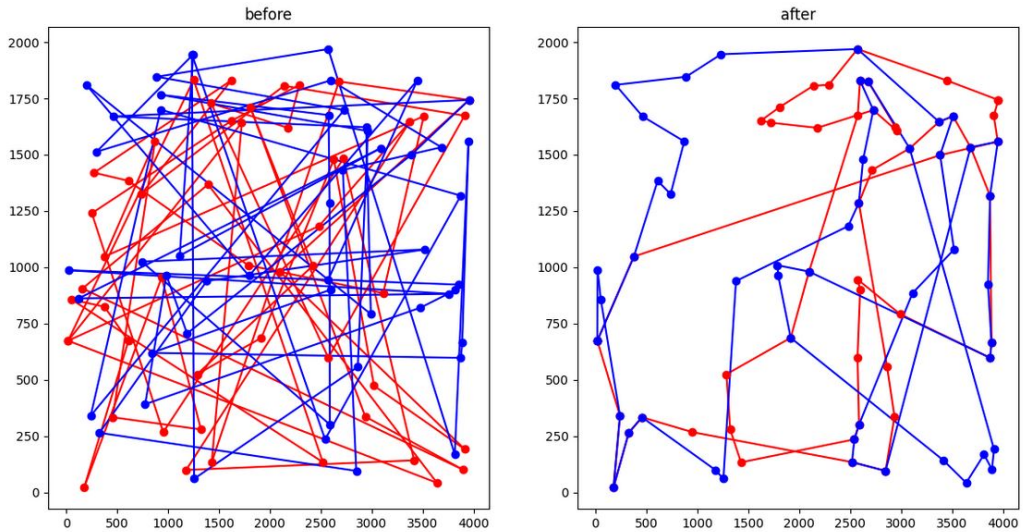
Lokalne przeszukanie	Instancja kroA100	Instancja kroB100
None	169784 (151478 - 192138)	174754 (154594 - 196338)
Greedy Vertex	42461 (30906 - 54013) - 30,1 ms	41856 (33267 - 51873) - 29 ms
Greedy Edge	28521 (25806 - 31210) - 11,7 ms	28900 (26871 - 31403) - 11 ms
Steepest Vertex	43077 (34302 - 51939) - 137,5 ms	43703 (35443 - 52759) - 138,8 ms
Steepest Edge	**28045** (**24815** - 32337) - 67,6 ms	**28633** (**26257** - 32183) - 66,5 ms
Random walk	147732 (137219 - 154586) - 138,8 ms	146086 (133811 - 151841) - 138,8 ms

Tabela 2: Rozwiązanie początkowe - 2-regret

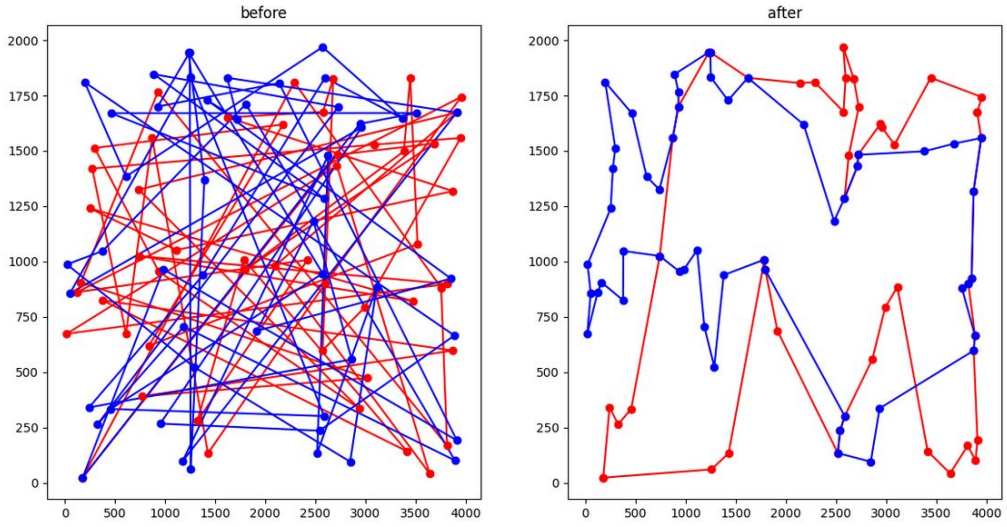
Lokalne przeszukanie	Instancja kroA100	Instancja kroB100
None	27587 (22499 - 32374)	28242 (24691 - 30467)
Greedy Vertex	25963 (22314 - 30219) - 9,5 ms	26571 (23784 - 29807) - 9,3 ms
Greedy Edge	25464 (**22249** - 28995) - 5,7 ms	26233 (**23154** - 29807) - 5,4 ms
Steepest Vertex	25924 (22314 - 29226) - 11,8 ms	26168 (24167 - 29525) - 15,4 ms
Steepest Edge	**25447** (22275 - 28365) - 8,6 ms	**26154** (23647 - 29525) - 9,5 ms

4 Wizualizacje najlepszych rozwiązań

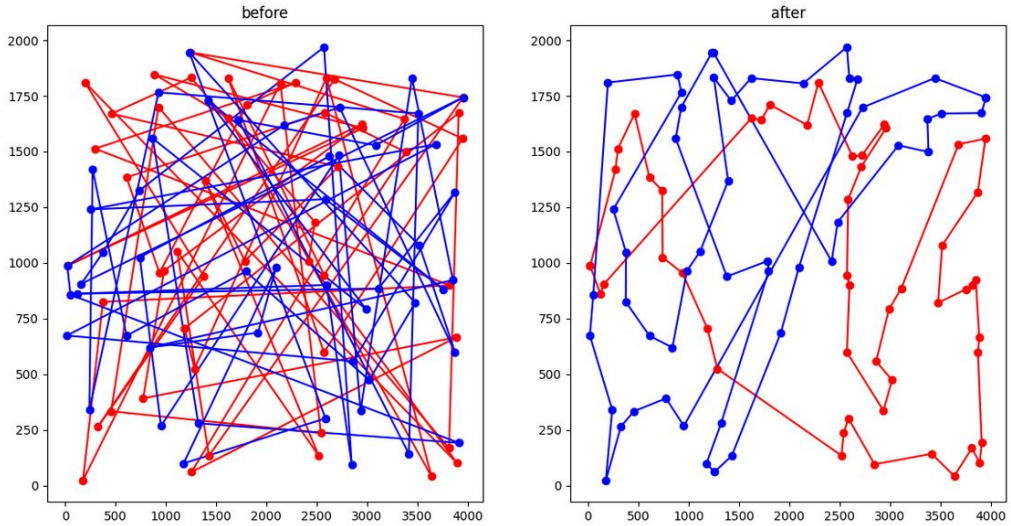
4.1 Losowe rozwiązanie początkowe - kroaA100



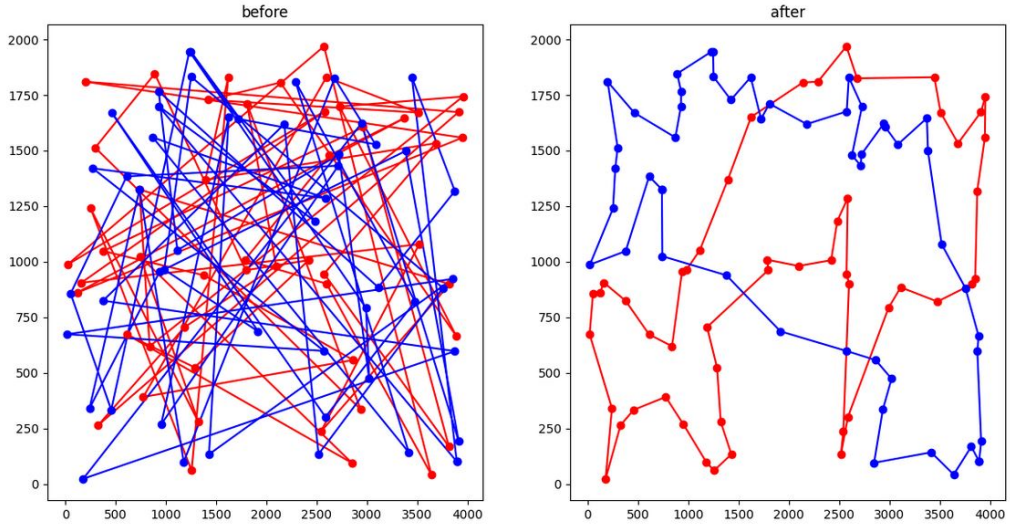
Rysunek 1: Greedy Vertex



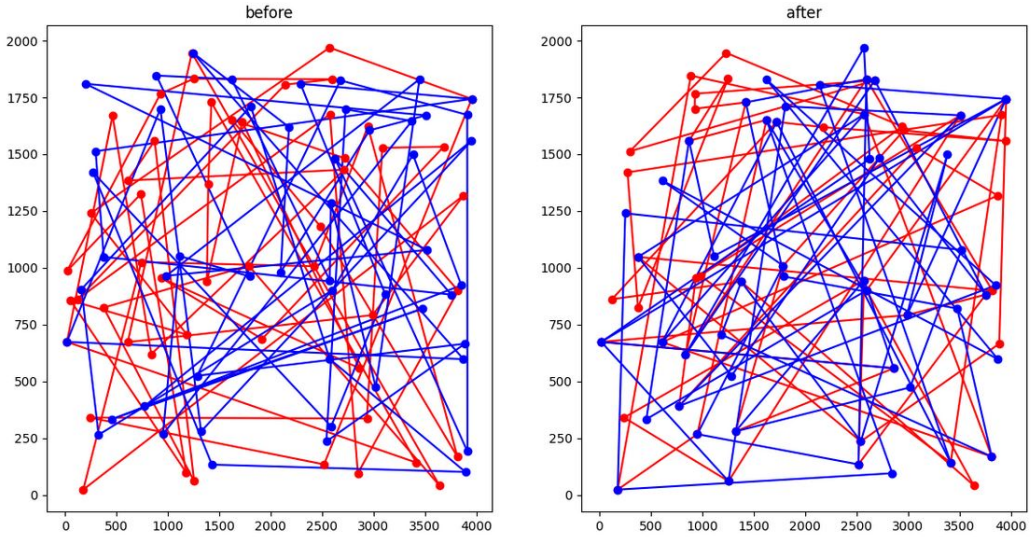
Rysunek 2: Greedy Edge



Rysunek 3: Steepest Vertex

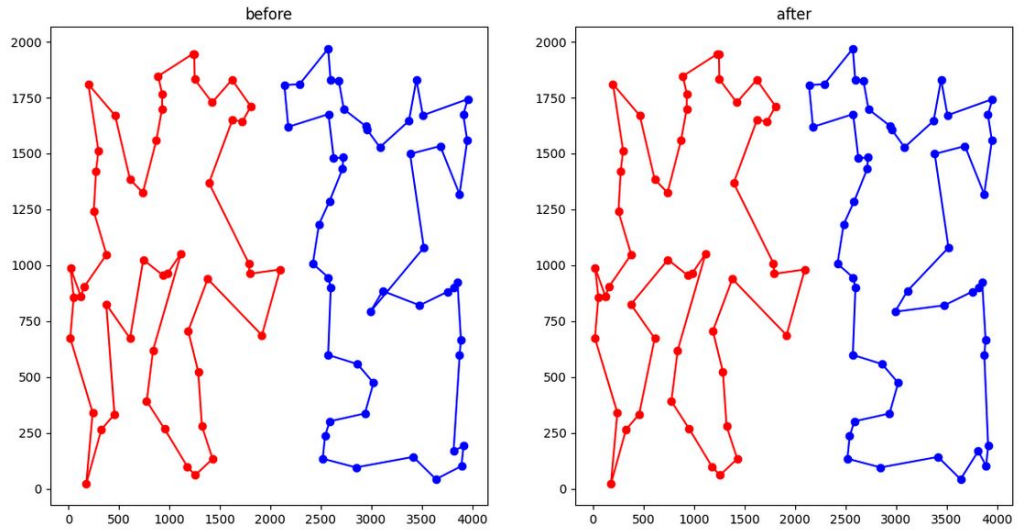


Rysunek 4: Steepest Edge

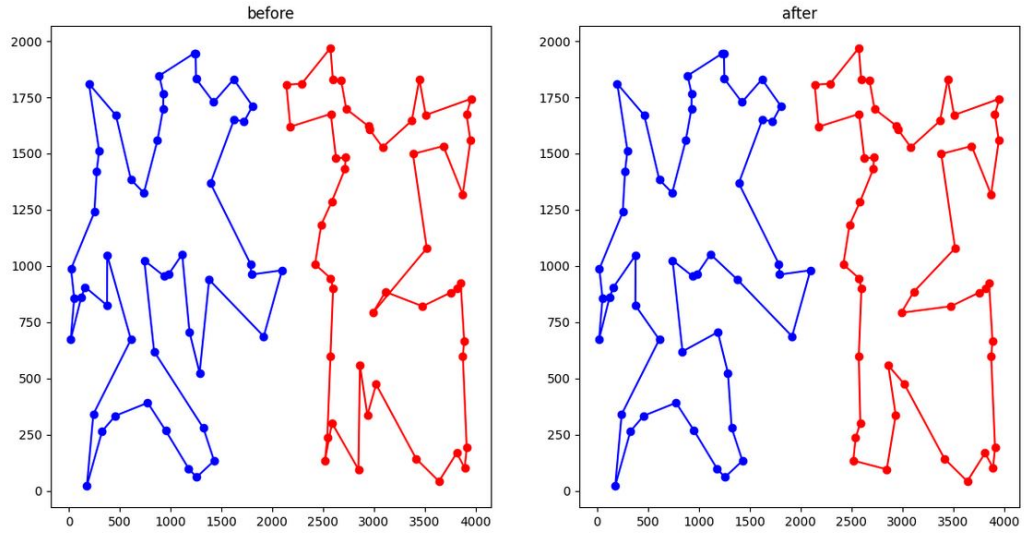


Rysunek 5: Random walk

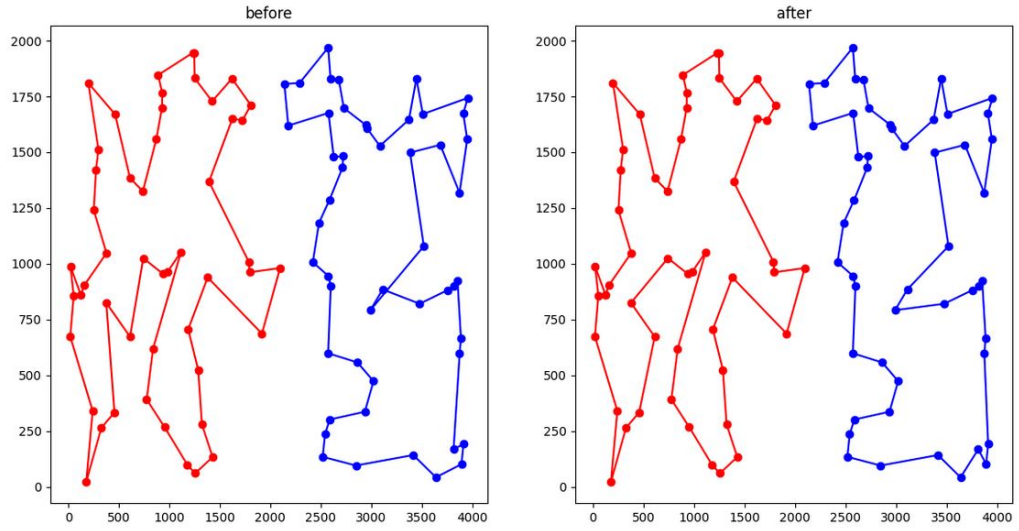
4.2 Rozwiązanie początkowe 2-regret - kroaA100



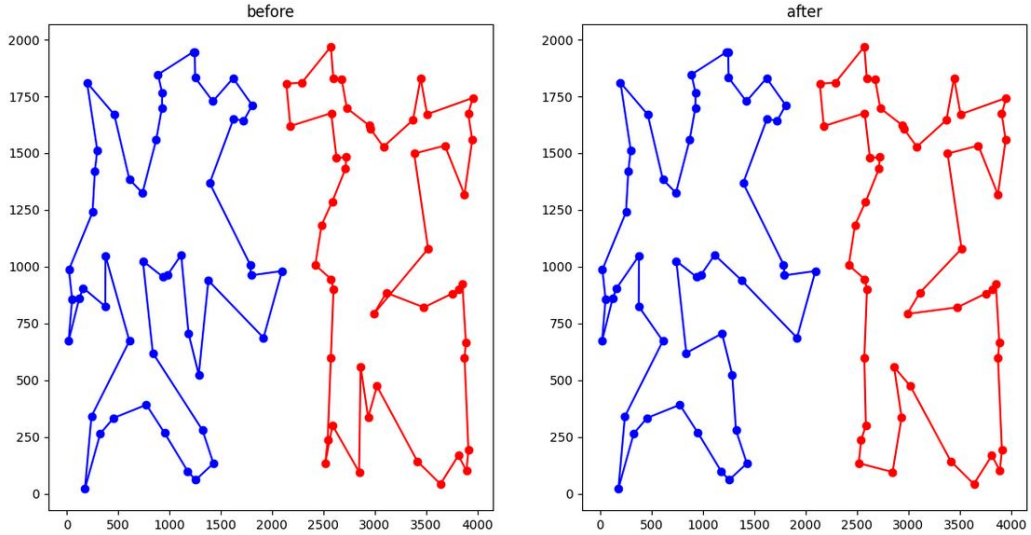
Rysunek 6: Greedy Vertex



Rysunek 7: Greedy Edge

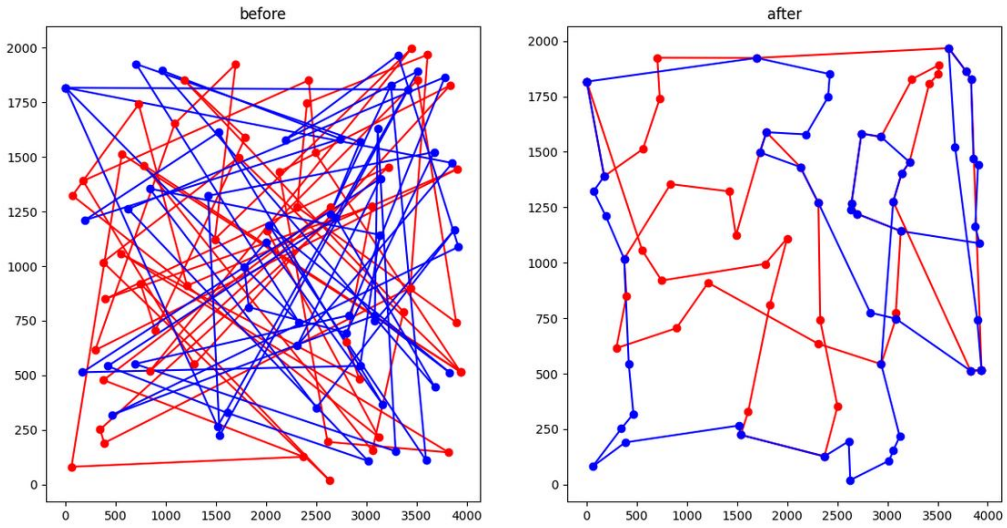


Rysunek 8: Steepest Vertex

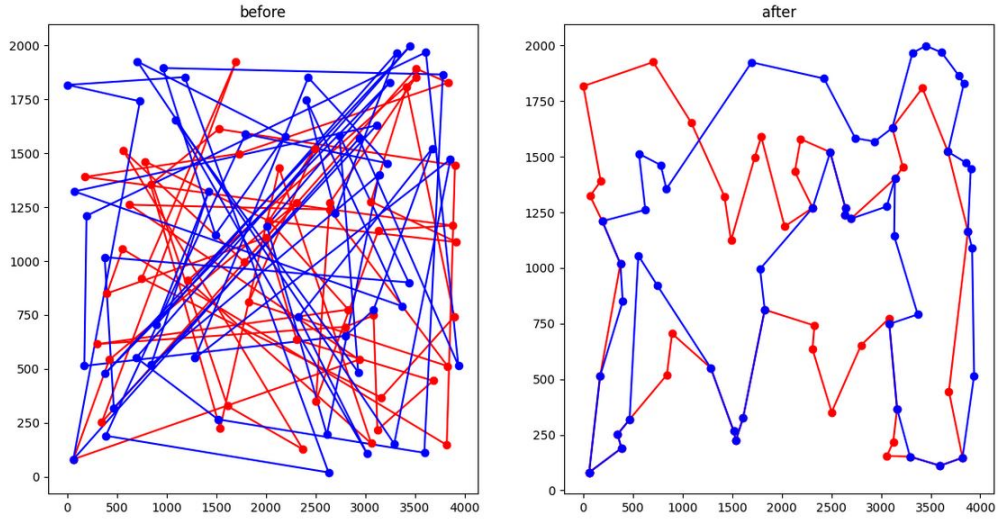


Rysunek 9: Steepest Edge

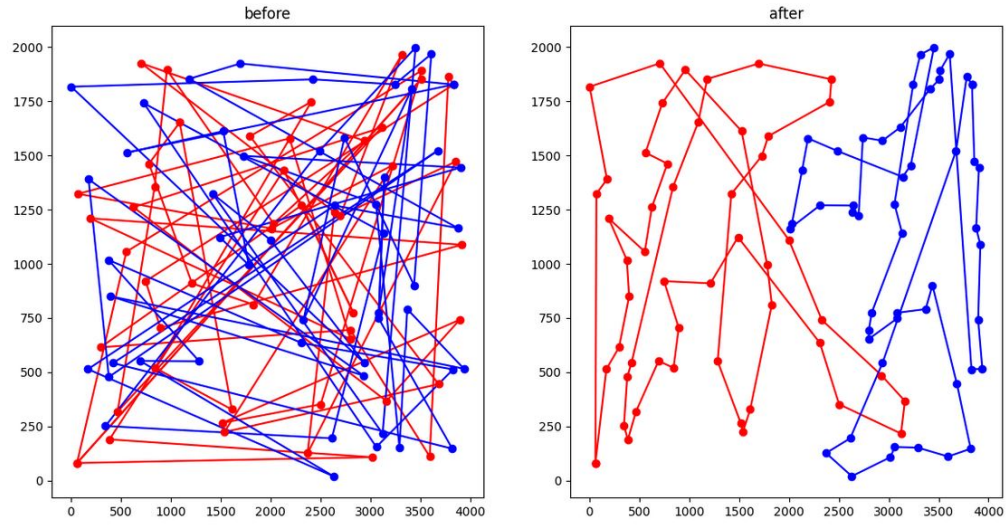
4.3 Losowe rozwiązanie początkowe - kroaB100



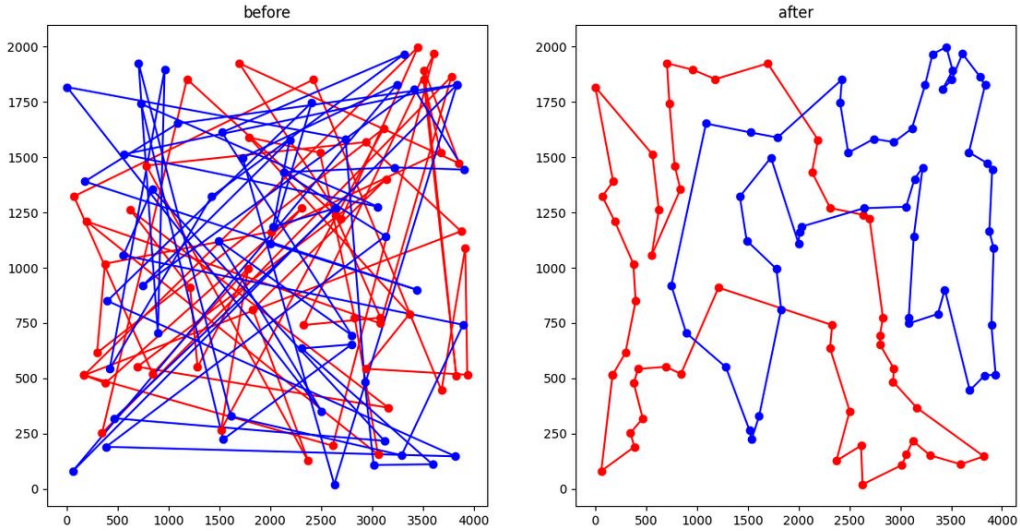
Rysunek 10: Greedy Vertex



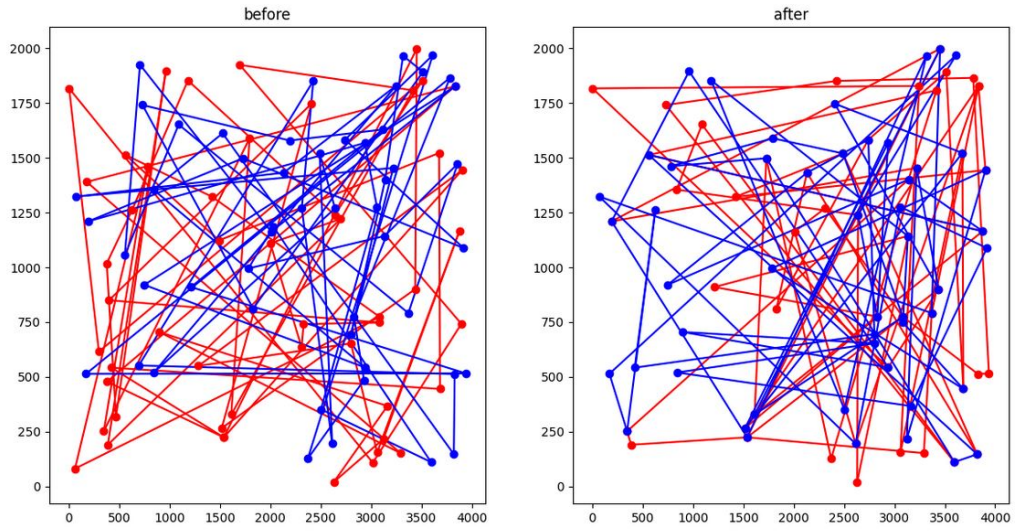
Rysunek 11: Greedy Edge



Rysunek 12: Steepest Vertex

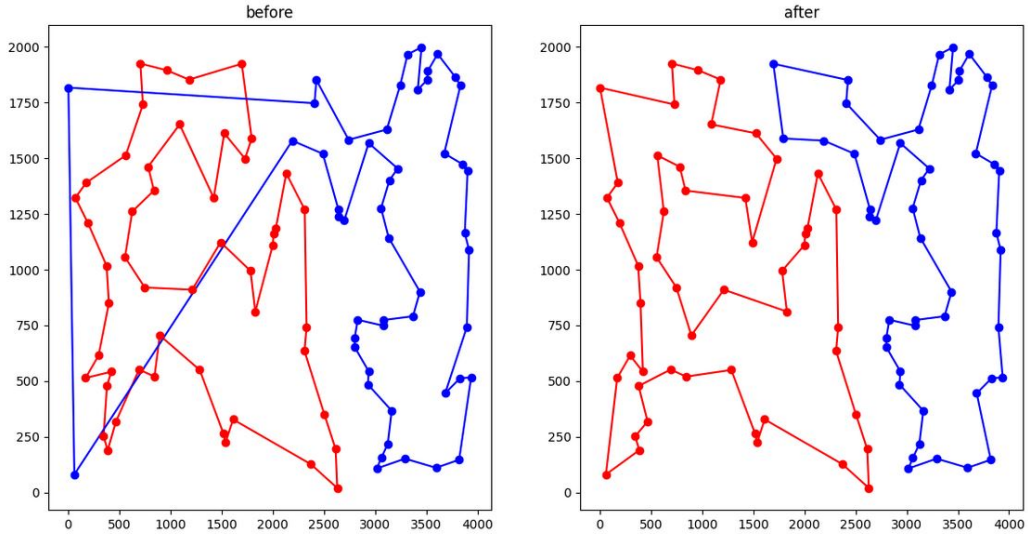


Rysunek 13: Steepest Edge

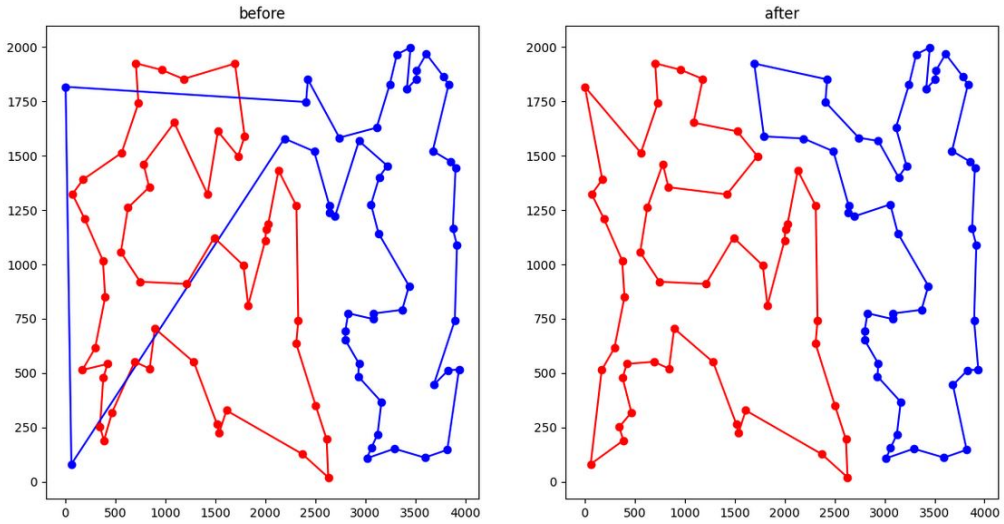


Rysunek 14: Random walk

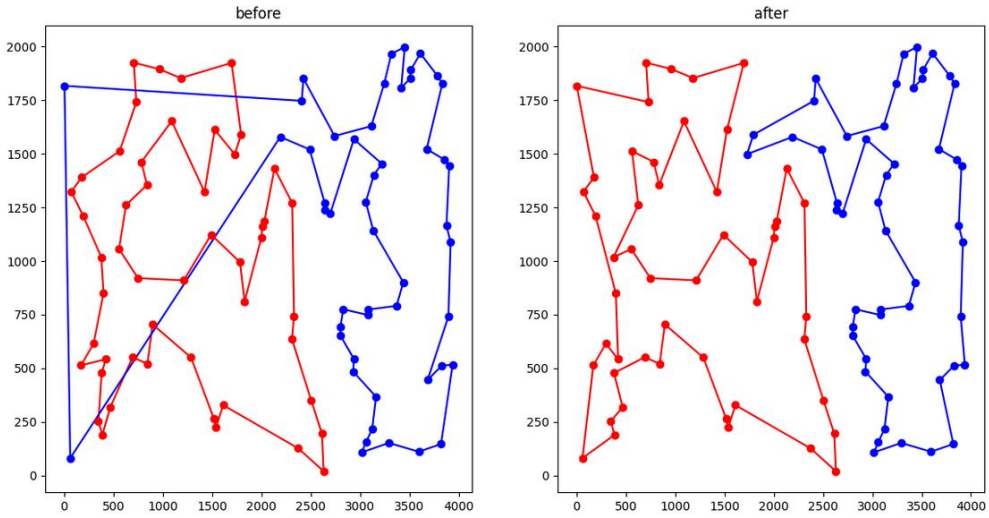
4.4 Rozwiązanie początkowe 2-regret - kroaB100



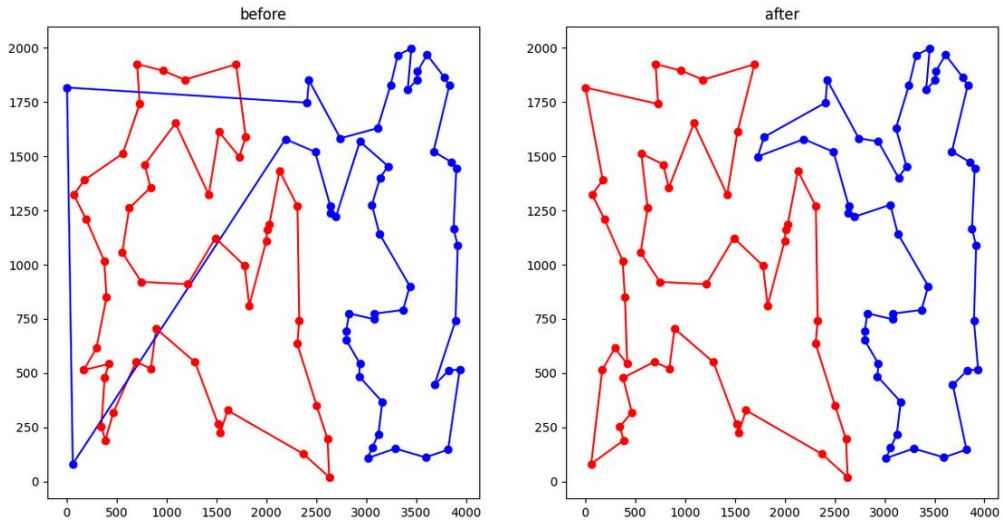
Rysunek 15: Greedy Vertex



Rysunek 16: Greedy Edge



Rysunek 17: Steepest Vertex



Rysunek 18: Steepest Edge

5 Wnioski

- Algorytm zachłanny w pojedyńczym uruchomieniu średnio osiąga gorsze wyniki od wersji stromej, jednak w wyniku wielokrotnego uruchomienia jest wstanie znaleźć wyniki lepsze niż wersja storma w tym samym czasie.
- Algorytmy zachłanny wykonuje się szybciej niż wersja stroma. Szczególnie różnicę tą widać dla losowego rozwiązania początkowego, dla którego wykonuje się nawet 5 razy szybciej.
- Algorytmy loklanego przeszukania dla losowego rozwiązania początkowego wykonują się dłużej, niż dla początkowego rozwiązania opartego na heurystyce 2-żalu. Wynika to z faktu, że dla losowego rozwiązania istnieje więcej możliwości do optymalizacji, niż w porównaniu do wstępnie zoptymalizowanego.
- Algorytm losowego błądzenia zwraca wyniki niewiele lepsze od losowo wygenerowanego rozwiąznia.
- Dla lokalnego przeszukiwania, rozpoczynającego się od wyniku heurystyki 2-żal, użycie heurystyki zbudowanej na najlepszym wierzchołku początkowym nie zawsze daje najlepszy rezultat. Heurystyka zbudowana na potencjalnie gorszym wierzchołku początkowym, może być zdolna do lepszej optymalizacji poprzez lokalne przeszukanie i osiągnąć najlepszy rezultat.

6 Kod programu

Kod źródłowy dostępny on-line: https://github.com/barosz0/IMO/tree/main/lab2