|  |  |  |
| --- | --- | --- |
| **Wykonali:** Wiktoria Szweda  Dmytro Protsailo  Łukasz Osiewicz | **Prowadzący:** prof. dr hab. Inż.  Andrzej Jaszkiewicz | Wydział Informatyki i Telekomunikacji PP - YouTubeWydział Informatyki i Telekomunikacji Politechniki Poznańskiej **Politechnika Poznańska**  **Wydział Informatyki i Telekomunikacji** |
| **Temat:**  **Heurystyki konstrukcyjne** | |
| **Data wykonania ćwiczenia:**  4.03.2024  **Data złożenia sprawozdania:**  18.03.2024 |  |

# Opis zadania:

Problemem rozpatrywanym w ramach zadania jest zmodyfikowany problem komiwojażera. Operując na zbiorach wierzchołków należy utworzyć dwa oddzielne cykle Hamiltona (każdy zawierający 50% danych wierzchołków) minimalizując przy tym funkcję celu będącą sumą odległości euklidesowych wszystkich wierzchołków między sobą:

Celem zadania jest zaimplementowanie algorytmów zachłannych (najbliższego sąsiada i rozbudowy cyklu) i algorytmu opartego na żalu (żal = 2) oraz porównanie wyników zwróconych w wyniku ich działania. Każdy algorytm uruchamiany jest 100 razy dla każdej instancji, na podstawie tych wartości wyliczono średnią oraz wyciągnięto najlepszą oraz najgorszą iterację.

# Algorytmy:

W bieżącym rozdziale przedstawiono jest pseudokod dla wszystkich zaimplementowanych algorytmów:

## Metody Ogólne

Funkcja calculateRoundedDistance(wierszcholek1 , wierszcholek2):

Dystans =

Zwróć zaokrąglony do liczb całych dystans

Funkcja calculateDistanceMatrix(tablica wierszcholków):

distanceMatrix = Tablica dwuwymiarowa liczb całkowitych

Dla każdego wierzchołka w tablicy wierzchołków

Dla każdego sąsiada wybranego wierzchołka

dystans między wierzchołkami = calculateRoundedDistance(dwa wierzchołka odpowiadające bieżącemu kroku pętli)

distanceMatrix = dystans od wierzchołka do sąsiada

distanceMatrix = dystans od sąsiada do wierszcholka

zwróc distanceMatrix

## Heurystyka najbliższego sąsiada:

Funkcja chooseRandomFurthestVertices(numVertices, distanceMatrix)

seed generatora liczb przypadkowych oparty na czasie

generator liczb przypadkowych w zakresie od 0 do numVertices-1

startVertex1 = indeks 1 wierszcholka startowego wygenerowany za pomocy generatora

distances = tablica dystansów do każdego sąsiada startVertex1

Dla każdego indeksu wierszcholka z numVertices

wpisz do distances odpowiedni dystans z distanceMatrix

maxDistance = maksymalna odleglość

startVertex2 = indeks 2 wierszcholka startowego

Dla każdego indeksu wierszcholka z numVertices

Gdy indeks nie jest startVertex1 i dystans dla wierszcholka wybrany z distances mniejszy od maxDistance

maxDistance = dystans dla wierszcholka wybrany z distances

startVertex2 = indeks wierszcholka

zwróć pare startVertex1 i startVertex2

Funkcja greedyExpansion(tablica ze scieżką tour, distanceMatrix, availableVertices):

bestInsertPosition = najlepsze miejsce wstawienia nowego wierzchołku (indeks sąsiada)

minInsertCost = najmniejszy znaleziony koszt wstawienia

bestInsertedVertex = indeks najlepszego wierszcholku dla wstawienia

Dla każdego indeksu w availableVertices

Dla każdego indeksu w tour

insertCost = 0 koszt wstawienia wybranego wierszcholka na wybraną pozycje w tour

Gdy indeks z tour nie jest pierwszym

insertCost += dystans od wierszcholka z tour-1 do wierzchołka wybranego z availableVertices

Gdy indeks z tour nie jest ostatnim

insertCost += dystans od wierzchołka wybranego z availableVertices

do wierszcholka z tour

Gdy indeks z tour nie jest pierwszym i indeks z tour nie jest ostatnim

insertCost -= dystans między wierzchołkami gdzie był wstawiony wierzchołek z availableVertices

Gdy insertCost < minInsertCost

minInsertCost = insertCost

bestInsertPosition = wybrany indeks z tour

bestInsertedVertex = wybrany indeks availableVertice

dodaj bestInsertedVertex do tour

usuń bestInsertedVertex z bestInsertedVertex

Funkcja greedyAlgorithmRandomStart(distanceMatrix, wymagany rozmiar sciezki tourSize)

ilość wierzchołków w zestawie numVertices = rozmiar distanceMatrix

para wierszcholków startingVertices = chooseRandomFurthestVertices(numVertices , distanceMatrix)

pierwsza sciezka tour1 = startingVertice.first

druga sciezka tour2 = startingVertice. second

para ścieżek tours = tour1 i tour2

tablica availableVertices o rozmiarze numVertices

Dla każdego indeksu w numVertices

availableVertices[] = indeks

usun startingVertice.first z availableVertices

usun startingVertice.second z availableVertices

Poki rozmiar sciezek z tours jest mniejsza od tourSize

greedyExpansion(tours.first, distanceMatrix, availableVertices)

greedyExpansion(tours.second, distanceMatrix, availableVertices)

zwróć pare tours.first I tours. second

## Metoda rozbudowy cyklu:

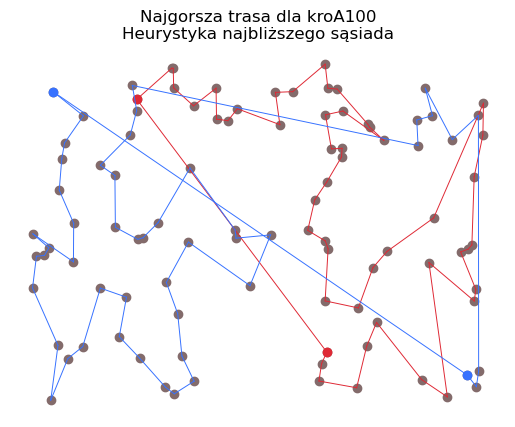
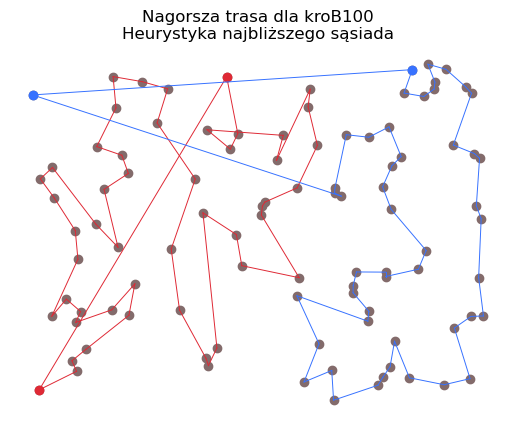
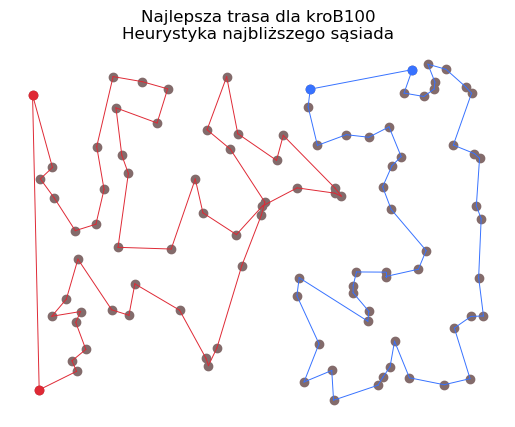
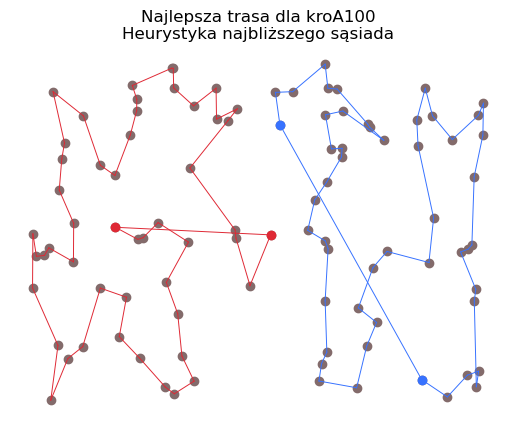
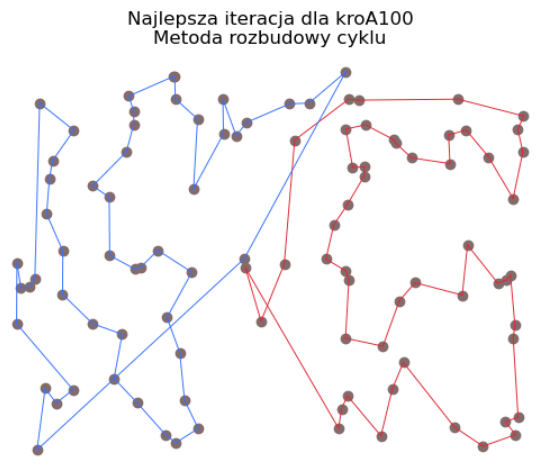
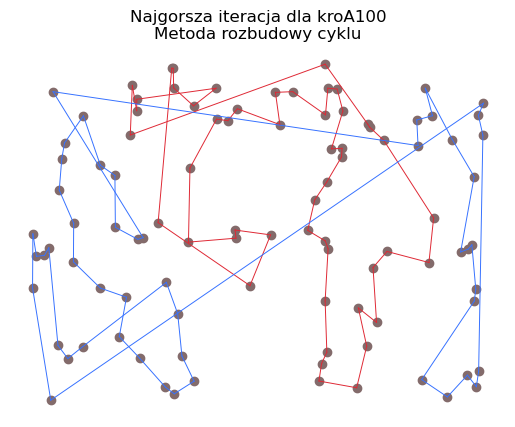
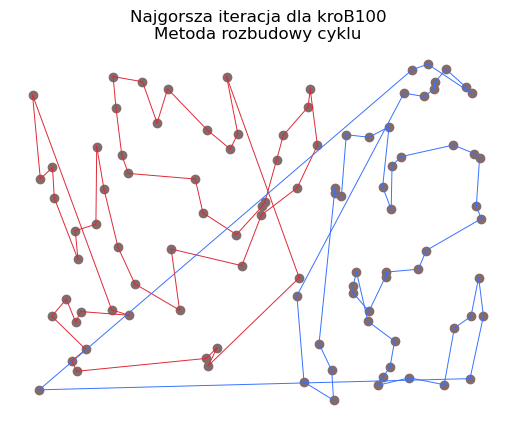
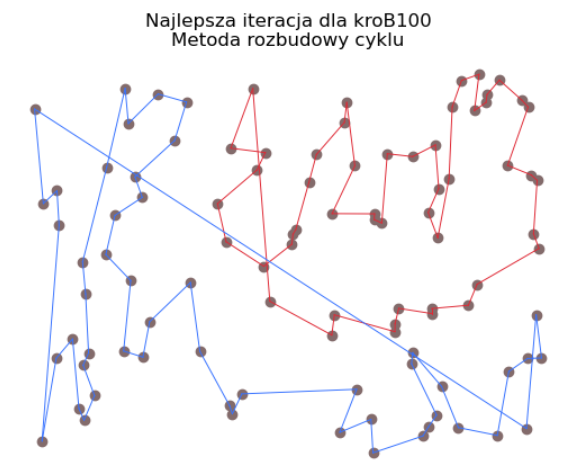
## Heurystyka zachłanna oparta na żalu:

# Wyniki:

|  |  |  |
| --- | --- | --- |
|  | ***kroA100.tsp*** | ***kroB100.tsp*** |
| **Heurystyka najbliższego sąsiada** | 28631 (23660 – 32193) | 28983(25281 - 31036) |
| **Metoda rozbudowy cyklu** | 30770 (26338 – 34557) | 32215 (29225 – 36490) |
| **Heurystyka zachłanna oparta na żalu** |  |  |

# Wizualizacja otrzymanych wyników:

Otrzymane ścieżki w wyniku działania algorytmów prezentują się następująco:

* Dla heurystyki najbliższego sąsiada:
* Dla metody rozbudowy cyklu:

# Wnioski:

# Kod programu: