Politechnika Poznańska

Wydział Informatyki i Telekomunikacji

Instytut Informatyki

Zakład Inteligentnych Systemów Wspomagania Decyzji

Heurystyki konstrukcyjne

Inteligentne metody optymalizacji

Paweł Chumski, 144392
pawel.chumski@student.put.poznan.pl

Michał Ciesielski, 145325
michal.p.ciesielski@student.put.poznan.pl

Prowadzący:
prof. dr hab. inż. Andrzej Jaszkiewicz
andrzej.jaszkiewicz@cs.put.poznan.pl

18-03-2023

Spis treści

Wstęp

Celem zadania była implementacja i porównanie trzech heurystyk w celu rozwiązania zmodyfikowanego problemu komiwojażera. Na podstawie danych współrzędnych wierzchołków wejściowych należało utworzyć, a następnie minimalizować dwie rozłączne zamknięte ścieżki (cykle), każdą zawierającą 50% wierzchołków.

Badane heurystyki to:
- Heurystyka najbliższego sąsiada (nearest neighbour)
- Metoda rozbudowy cyklu (greedy cycle)
- Heurystyka oparta na 2-żalu (regret heuristic)

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

Heurystyka najbliższego sąsiada (nearest neighbour)

1.1 Opis algorytmu

Algorytm najbliższego sąsiada polega na wybraniu losowego punktu startowego, a następnie iteracyjnym wybieraniu najbliższego nieodwiedzonego punktu i dodawaniu go do trasy. Proces jest powtarzany aż do odwiedzenia wszystkich punktów.

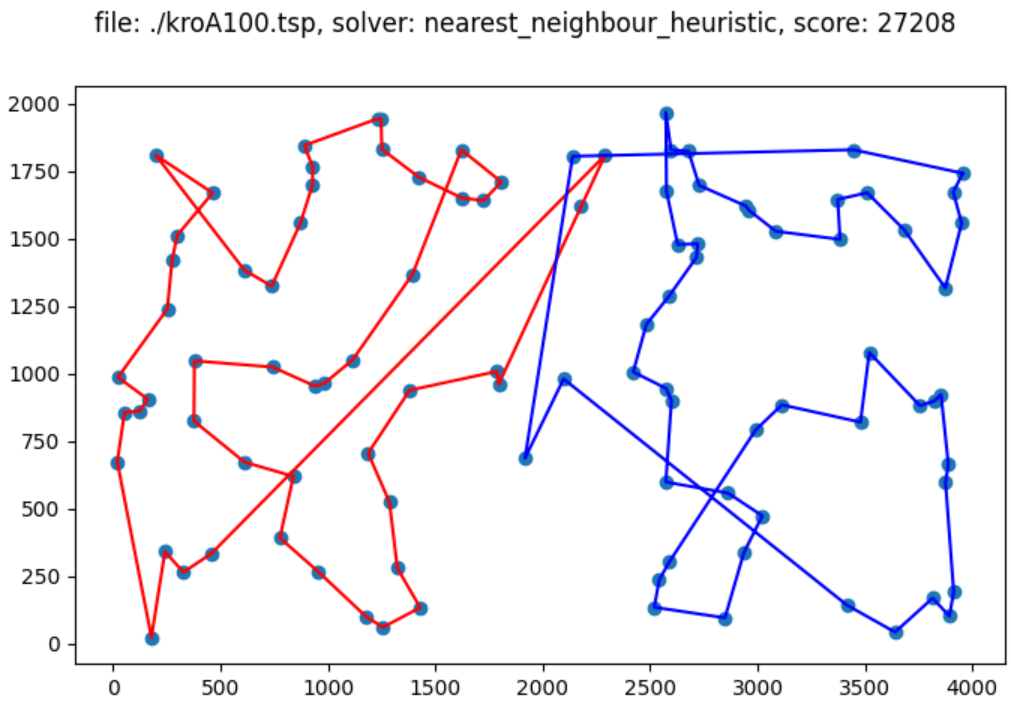
1.2 Pseudokod

```
Znajdź najbardziej oddalony wierzchołek od bieżącego, od którego zacznie się budowa drugiego cyklu;
Inicjalizuj listę remaining_nodes zawierająca potencjalnych kandydatów do wydłużenia ścieżki;
Inicjalizuj cykle, pierwszy zawierający wierzchołek wybrany, drugi najbardziej od niego oddalony;
while len(remaining_nodes) > 0 do
    for cycle in cycles.keys() do
        Wydłuż każdy cykl o najbliższego sąsiada dla ostatnio dodanego wierzchołka;
        Usuń dodany wierzchołek z listy remaining_nodes;
    end
end
return słownik zawierający listy wierzchołków dla obu cykli;
```

Algorithm 1: Heurystyka najbliższego sąsiada (nearest neighbour)

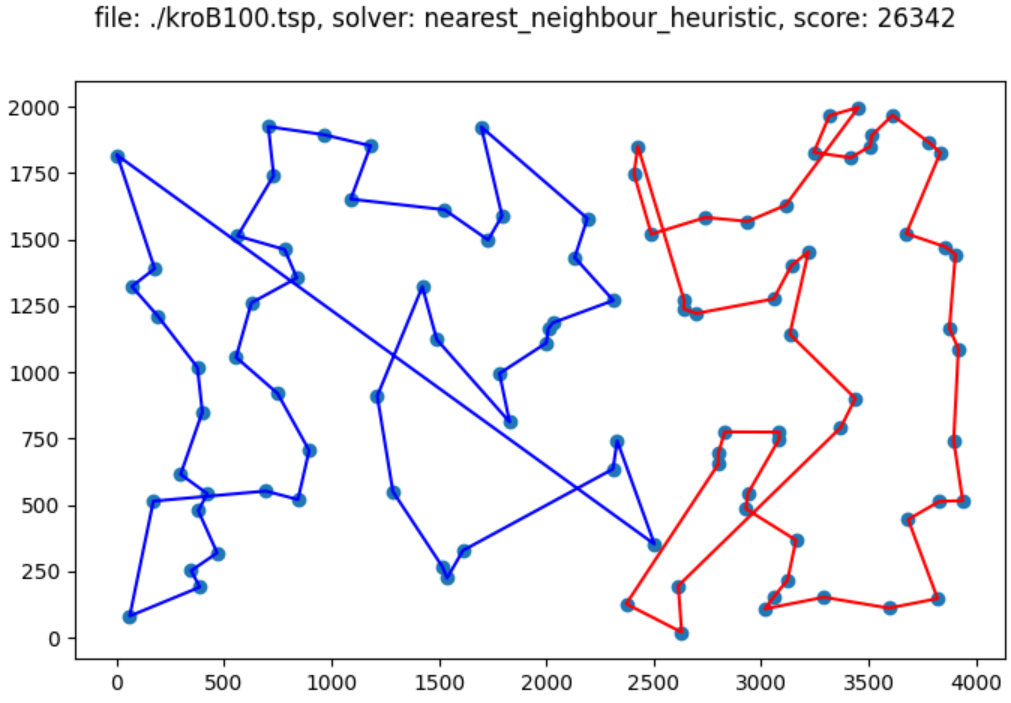
1.3 Wizualizacje

- Wynik dla kroA100:



Rys. 1. Obraz najlepszych ścieżek uzyskany dla heurystyki najbliższego sąsiada dla danych kroA100.

- Wynik dla kroB100:



Rys. 2. Obraz najlepszych ścieżek uzyskany dla heurystyki najbliższego sąsiada dla danych kroB100.

Metoda rozbudowy cyklu (greedy cycle)

2.1 Opis algorytmu

Algorytm Greedy Cycle Heuristics to metoda znajdowania rozwiązania problemu komiwojażera poprzez iteracyjne łączenie cykli o najmniejszej wadze. W każdej iteracji wybierany jest losowy nieodwiedzony punkt startowy, a następnie budowany jest cykl poprzez wybieranie najbliższego nieodwiedzonego punktu. Po zakończeniu budowania cyklu, łączy się go z dotychczasowymi cyklami poprzez wybór cyklu o najmniejszej sumie odległości i połączenie z nim nowo utworzonego cyklu. Proces jest powtarzany aż do odwiedzenia wszystkich punktów i utworzenia jednego cyklu.

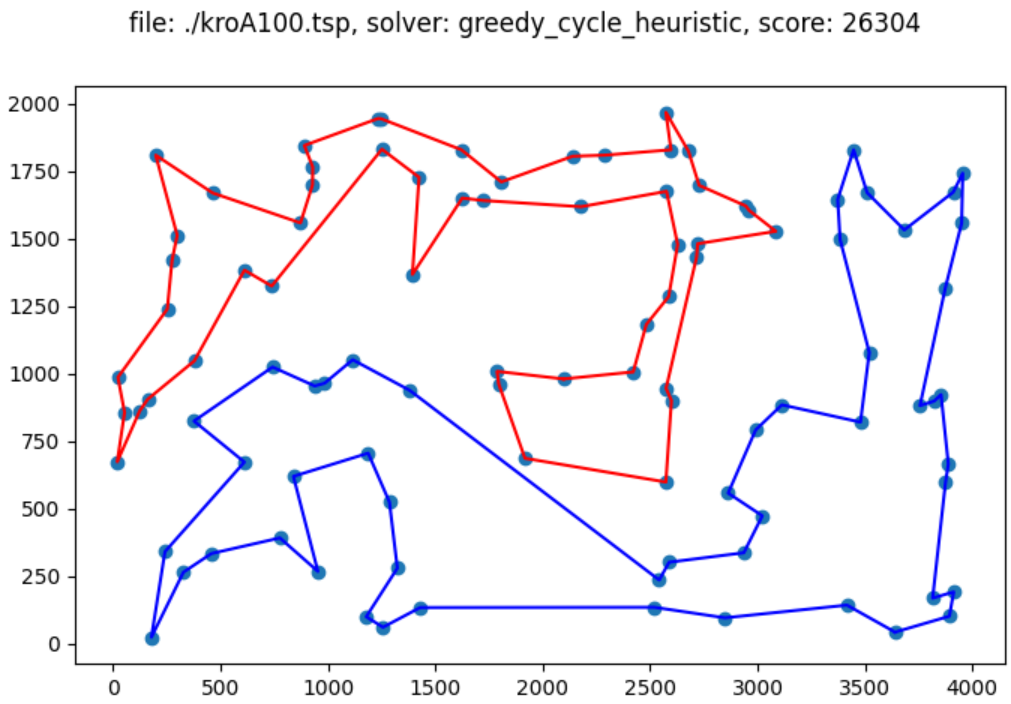
2.2 Pseudokod

```
Znajdź najbardziej oddalony wierzchołek od bieżącego, od którego zacznie się budowa drugiego cyklu;
Inicjalizuj listę remaining_nodes zawierająca potencjalnych kandydatów do wydłużenia ścieżki;
Inicjalizuj cykle, pierwszy zawierający wierzchołek wybrany, drugi najbardziej od niego oddalony;
while len(remaining_nodes) > 0 do
    for cycle in cycles.keys() do
        Znajdź wierzchołek z remaining_nodes, który minimalizuje różnicę wynikającą z wstawienia go w cykl na różne pozycje;
        Wstaw ten wierzchołek w cykl na najlepszą pozycję ;
        Usuń dodany wierzchołek z listy remaining_nodes;
    end
end
return słownik zawierający listy wierzchołków dla obu cykli;
```

Algorithm 2: Metoda rozbudowy cyklu (greedy cycle)

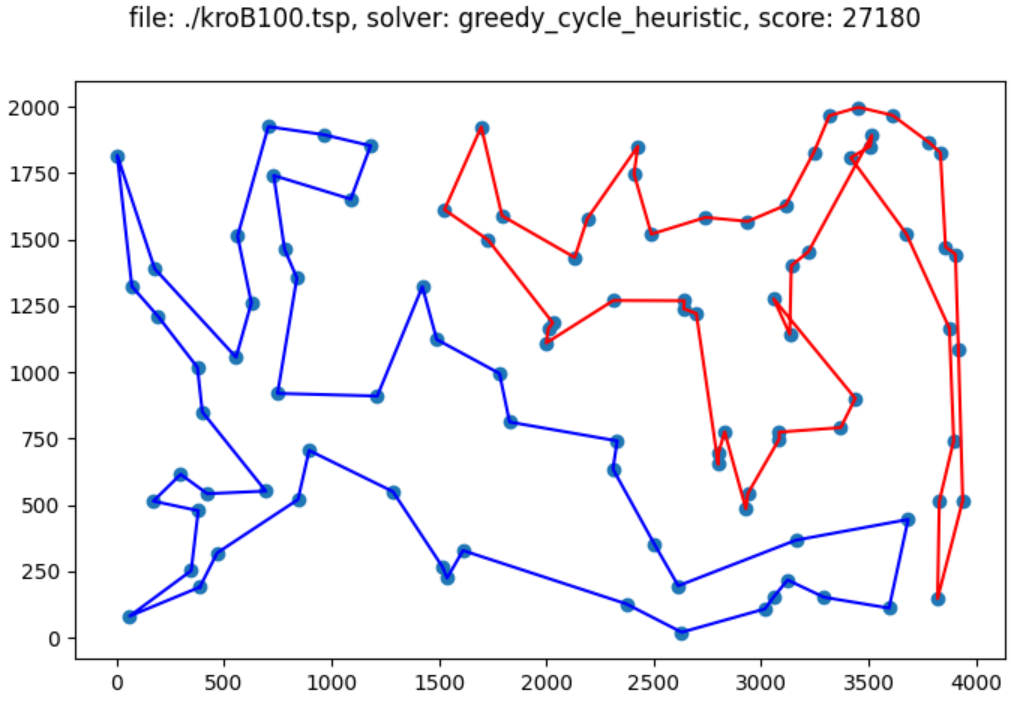
2.3 Wizualizacje

- Wynik dla kroA100:



Rys. 3. Obraz najlepszych ścieżek uzyskany dla heurystyki najbliższego sąsiada dla danych kroA100.

- Wynik dla kroB100:



Rys. 4. Obraz najlepszych ścieżek uzyskany dla heurystyki najbliższego sąsiada dla danych kroB100.

Heurystyka oparta na 2-żalu (regret heuristic)

3.1 Opis algorytmu

W przeciwieństwie do metody rozbudowy cyklu, wybór kolejnych punktów nie jest oparty na odległości, ale na tak zwanym żalu (2-żalu w tym przypadku), czyli wyboru maksymalnej różnicy odległości pomiędzy najlepszym wierzchołkiem do wstawienia zachłannie w danej chwili, a kolejnym możliwym z listy. Żal ten jest ważony wcześniej poprzez odjęcie od niego najlepszych możliwości zachłannych pomnożonych przez odpowiednią wagę.

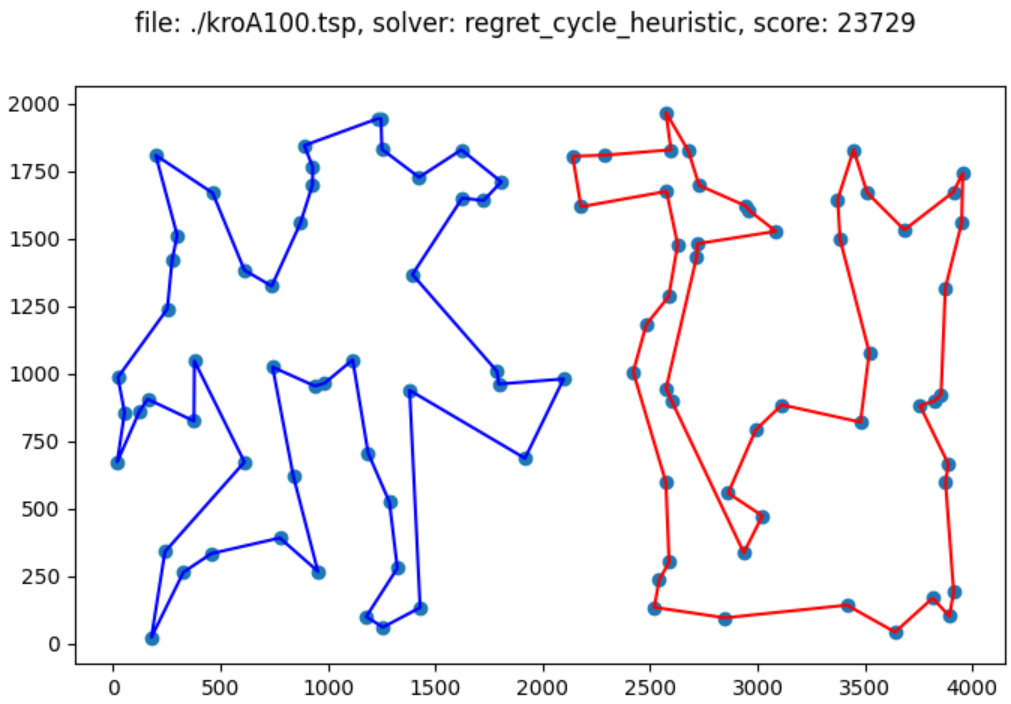
3.2 Pseudokod

```
1  Znajdź najbardziej oddalony wierzchołek od bieżącego, od którego zacznie się budowa drugiego
     cyklu;
2  Inicjalizuj listę remaining_nodes zawierająca potencjalnych kandydatów do wydłużenia ścieżki;
3  Inicjalizuj cykle, pierwszy zawierający wierzchołek wybrany, drugi najbardziej od niego
     oddalony;
4  while len(remaining_nodes) > 0 do
5  |  for cycle in cycles.keys() do
6  |  |  Dla każdego cyklu oblicz wartość punktacji dla każdego pozostałego wierzchołka,
   |  |    zgodnie z funkcją score_diff, która oblicza różnicę w koszcie między dodaniem nowego
   |  |    wierzchołka do cyklu a pozostawieniem cyklu bez wierzchołka, wykorzystując macierz
   |  |    odległości między wierzchołkami;
7  |  |  Wybierz wierzchołek, dla którego ważony 2-żal ma największą wartość ;
8  |  |  Usuń wierzchołek z listy pozostałych wierzchołków;
9  |  end
10 end
11 return słownik zawierający listy wierzchołków dla obu cykli;
```

Algorithm 3: Heurystyka oparta na 2-żalu (regret heuristic)

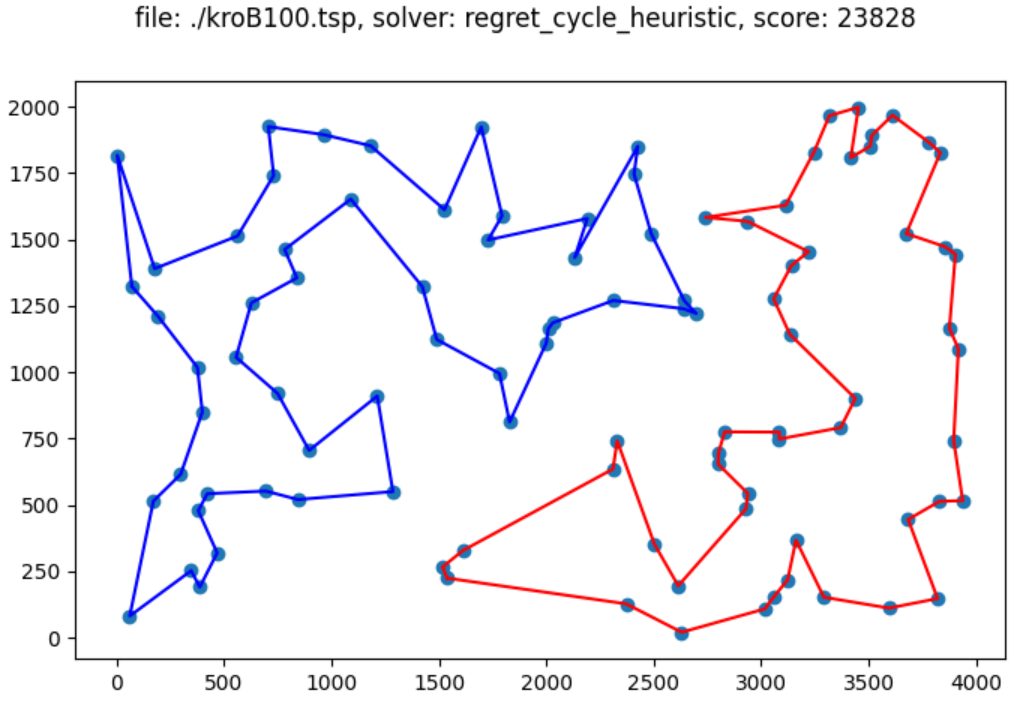
3.3 Wizualizacje

- Wynik dla kroA100:



Rys. 5. Obraz najlepszych ścieżek uzyskany dla heurystyki najbliższego sąsiada dla danych kroA100.

- Wynik dla kroB100:



Rys. 6. Obraz najlepszych ścieżek uzyskany dla heurystyki najbliższego sąsiada dla danych kroB100.

Podsumowanie

Porównanie wyników eksperymentu obliczeniowego

Algorithms	kroA100			kroB100		
	min	*mean*	*max*	*min*	*mean*	*max*
Nearest neighbour	27208	32496	35514	26342	32325	40264
Greedy cycle	26304	28708	29980	27180	28538	30197
Regret heuristic	23729	26573	32441	23828	27561	29793

Rys. 7. Wyniki eksperymentu

Wnioski

Pierwszy omawiany algorytm, czyli najbliższego sąsiada (nearest neighbour) nie zawsze daje optymalne rozwiązanie problemu komiwojażera, ale jest stosunkowo prosty i szybki w implementacji. Wadą tego algorytmu jest to, że wynik zależy od punktu startowego, co oznacza, że istnieją przypadki, w których algorytm może dać bardzo złe wyniki. Algorytm rozbudowy cyklu (greedy cycle) jest również stosunkowo prostym i szybkim sposobem rozwiązania problemów optymalizacyjnych. Jego działanie opiera się na kolejnym wyborze krawędzi, która pozwala na maksymalne zwiększenie wartości cyklu. Mimo że metoda ta nie gwarantuje znalezienia optymalnego rozwiązania, to często daje zadowalające wyniki w stosunkowo krótkim czasie. Jednym z jej wad jest jednak brak elastyczności i możliwość utknięcia w lokalnym minimum, co może prowadzić do niedoskonałych wyników. Daje lepsze wyniki niż pierwszy algorytm, zwłaszcza dla dużych grafów. Na przykładzie naszych danych widać że wyniki algorytmu rozbudowy cyklu są lepsze od algorytmu najbliższego sąsiada (oprócz wartości minimalnej dla kroB100). Zgodnie z oczekiwaniami, najlepsze wyniki, poza maksymalnym w kroA100, otrzymaliśmy dla algorytmu opartego na 2-żalu. Docelowo ten algorytm miał optymalizować problem, co zostało osiągnięte. W przeciwieństwie do algorytmu najbliższego sąsiada, wybór kolejnych punktów nie jest oparty jedynie na odległości, ale na tak zwanym żalu, czyli różnicy między odległością danego punktu, a dwoma najbliższymi punktami w trasy. Wybierany jest punkt, który ma największą wartość żalu, co powoduje, że algorytm jest bardziej złożony, ale często daje lepsze wynik, co widać w powyższej tabeli[7] (oprócz wspomnianej wyżej wartości maksymalnej dla kroA100). Anomalia związana z gorszym wynikiem dla najgorszego wariantu ścieżek w przypadku algorytmów opartego na 2-żalu dla instancji kroA100 wynika z potencjalnego „przeuczenia" wagi dobranej dla żalu. Z kolei anomalia gorszego wyniku dla algorytmu rozbudowy cyklu niż algorytmu najbliższego sąsiada wynika z gorzej dobranej heurystyki (obie bazują na algorytmach zachłannych) dla instancji kroB100.

Kod programu

https://github.com/imo/solution.py