Heurystyki zachłanne

Autorzy sprawozdania

Filip Marciniak 148148, Szymon Pasternak 148146

Wstęp

W ramach zadania należało przeprowadzić badania dotyczące trzech algorytmów:

1. Heurystyka najbliższego sąsiada (nearest neighbor)
2. Metoda rozbudowy cyklu (greedy cycle)
3. Heurystyka zachłanna oparta na żalu (regret heuristics)

na przykładzie zmodyfikowanego problemu 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`[1]. Odległości pomiędzy wierzchołkami obliczone zostały jako odległości euklidesowe oraz zaokrąglone matematycznie.

Kod programu

Kod programu dostępny jest w repozytorium: https://github.com/Johnybonny/IMO

Opis algorytmów

Wszystkie z podanych poniżej algorytmów akceptują na wejściu macierz odległości pomiędzy danymi wierzchołkami. Zmienne użyte w każdym z algorytmów:

- taken - wektor wartości oznaczających czy wierzchołek na danej pozycji należy do któregoś z cykli,
- cyclesPoints - wektor zawierający cykle, cyclesPoints[i] - wektor wierzchołków w i-tym cyklu,

Algorytm Nearest Neighbor

```
dopóki jakakolwiek wartość z taken jest false
{
  dla każdego cyklu w cyclesPoints
  {
    dla każdego wierzchołka w cyklu
    {
      dla każdego wolnego punkt z taken
      {
        oblicz odległość pomiędzy wierzchołkiem a punktem.
        jeżeli odległość jest najmniejsza z dotychczasowych odległości, to
zapamiętaj ten wierzchołek oraz punkt.
      }
    }
    oblicz koszty wstawienia nowego punktu przed oraz po wierzchołku należącym do
```

```
cyklu.
    wstaw nowy punkt w miejsce w cyklu, które spowoduje mniejszy wzrost długości
cyklu.
    oznacz punkt jako wykorzystany w taken.
  }
}
wierzchołki należące do każdego z cykli znajdują się w cyclesPoints.
```

Algorytm Greedy Cycle

```
dopóki jakakolwiek wartość z taken jest false
{
  dla każdego cyklu w cyclesPoints
  {
    dla każdego wierzchołka w cyklu
    {
      dla każdego wolnego punkt z taken
      {
        oblicz koszt dodania tego punktu pomiędzy rozpatrywanym wierzchołkiem a
następnym.
        jeżeli koszt jest najmniejszy z dotychczasowych kosztów, to zapamiętaj ten
wierzchołek oraz punkt.
      }
    }
    wstaw nowy punkt do cyklu za zapamiętanym wierzchołkiem.
    oznacz punkt jako wykorzystany w taken.
  }
}
wierzchołki należące do każdego z cykli znajdują się w cyclesPoints.
```

Algorytm 2-regret

```
dopóki jakakolwiek wartość z taken jest false
{
  dla każdego cyklu w cyclesPoints
  {
    dla każdego wierzchołka w cyklu
    {
      dla każdego wolnego punkt z taken
      {
          oblicz koszt dodania tego punktu pomiędzy rozpatrywanym wierzchołkiem a
następnym i zapisz go do wektora kosztów.
      }
      oblicz żal dla wartości z wektora kosztów.
      pomniejsz go o najmniejszy koszt pomnożony przez wagę.
      jeżeli otrzymana wartość jest większa niż dotychczasowe, to zapamiętaj ten
wierzchołek oraz punkt, dla którego ta wartość została osiągnięta.
    }
    wstaw nowy punkt do cyklu za zapamiętanym wierzchołkiem.
```

```
      oznacz punkt jako wykorzystany w taken.
    }
  }
  wierzchołki należące do każdego z cykli znajdują się w cyclesPoints.
```

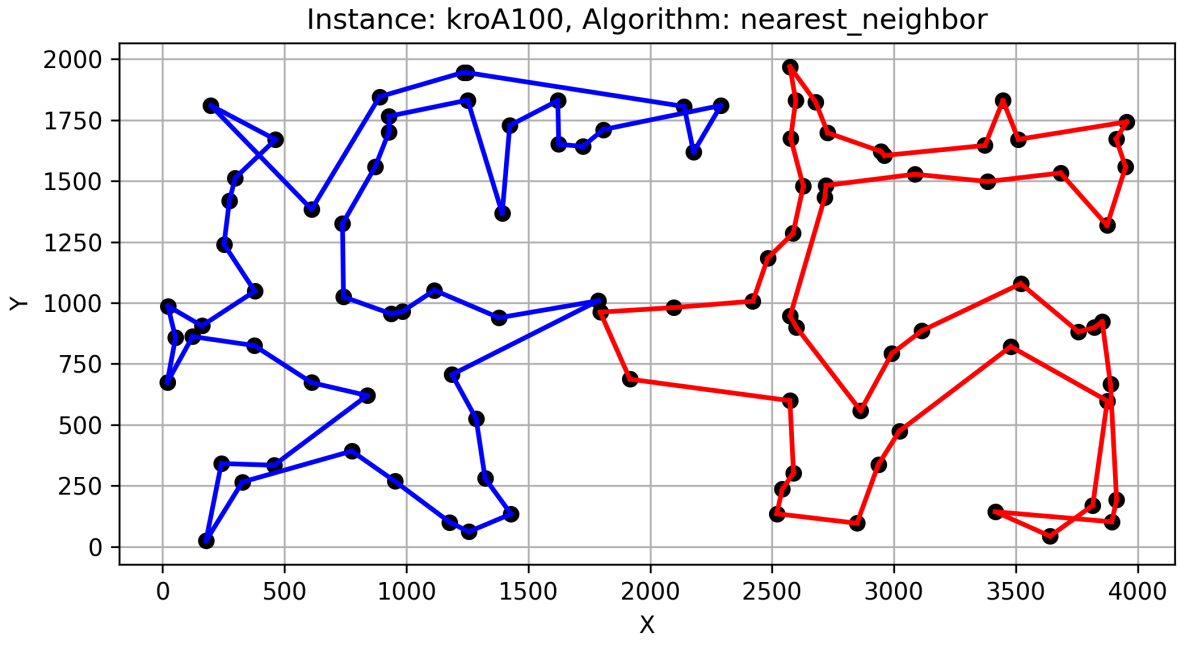
Wyniki

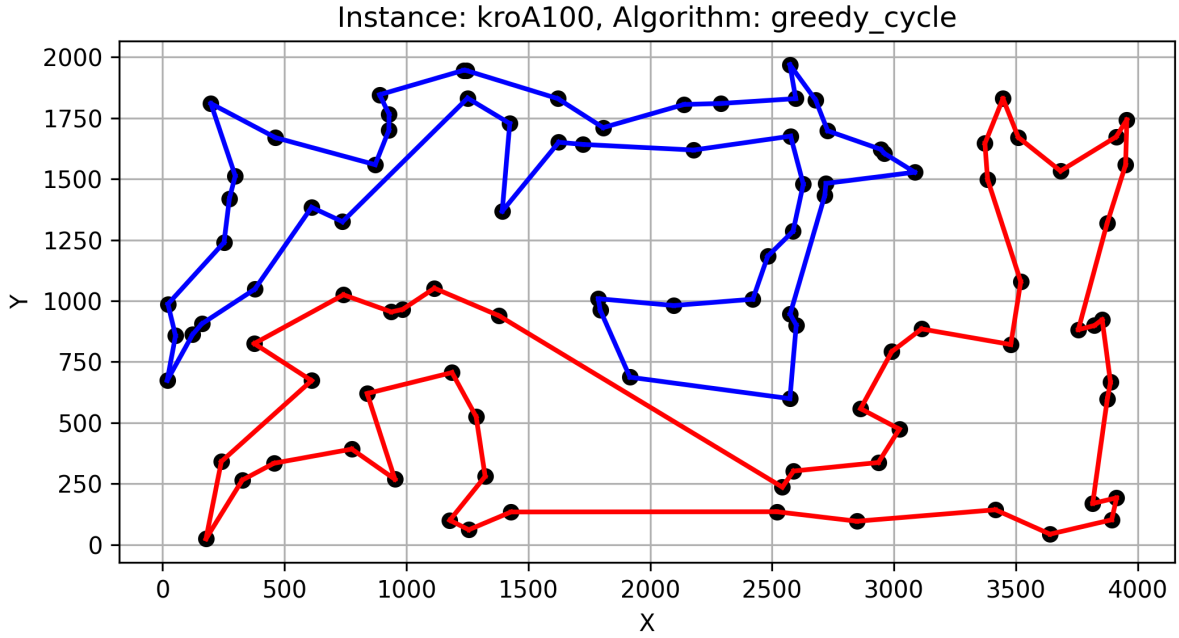
Każdy algorytm uruchomiony został 100 razy - dla każdego wierzchołka, który wyznaczany był jako startowy. W tabeli przedstawione zostały wyniki działania programu. Algorytm korzystający z techniki 2-regret uruchomiony został dla różnych parametrów wagi - od 0 do 1 z krokiem co 0.1. W tabeli przedstawione zostały wyniki dla wagi równej 0.6.

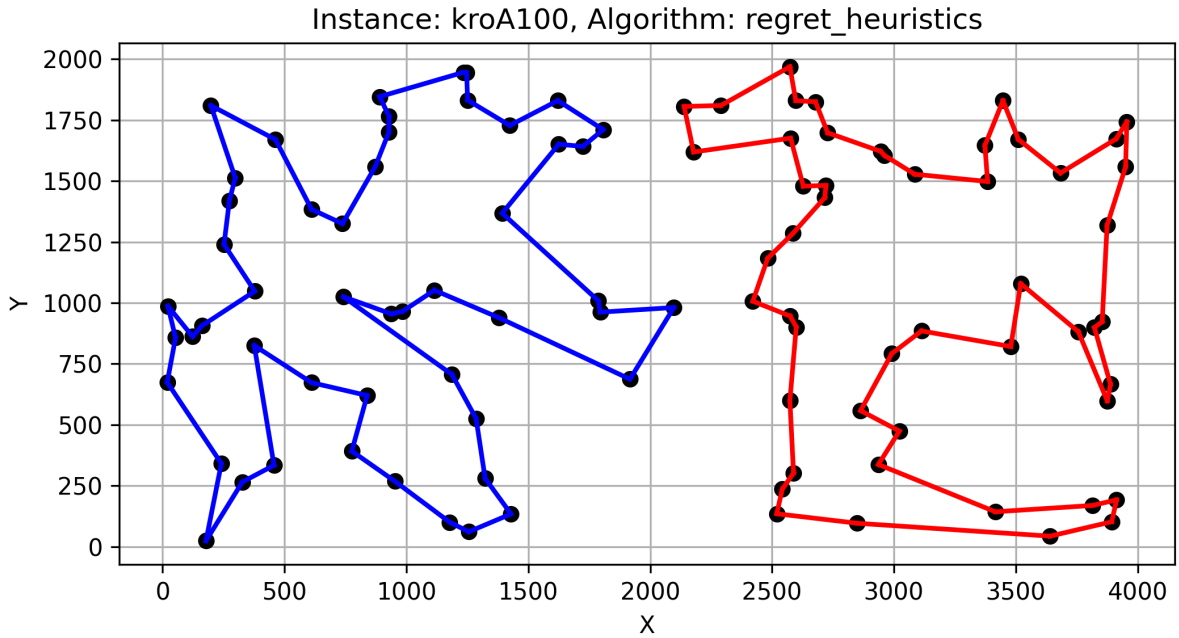
	kroA100			kroaB		
Algorytm	min	mean	max	min	mean	max
Nearest neighbor	26519	30826	33902	26939	29309	31523
Greedy cycle	26304	28708	29980	27180	28538	30197
2-Regret (waga 0.6)	22992	25870	30183	23876	27049	31185

Poniżej umieszczone zostały wizualizacje najlepszych z uzyskanych wyników (w tabeli wartości min):

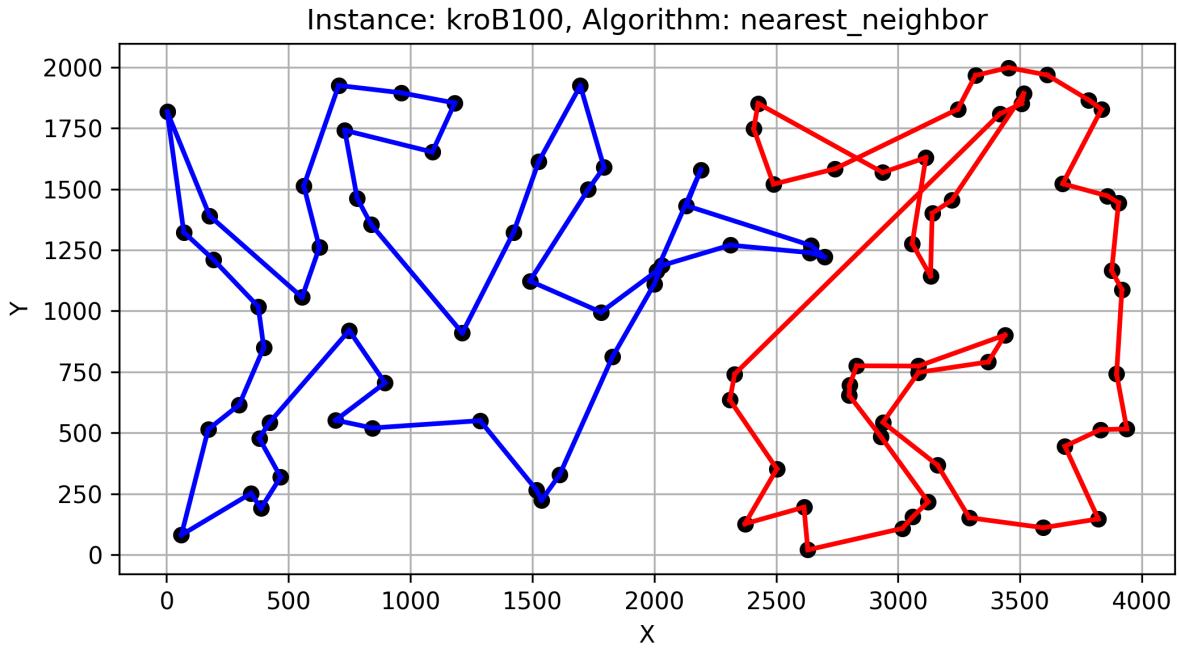
kroA100.tsp

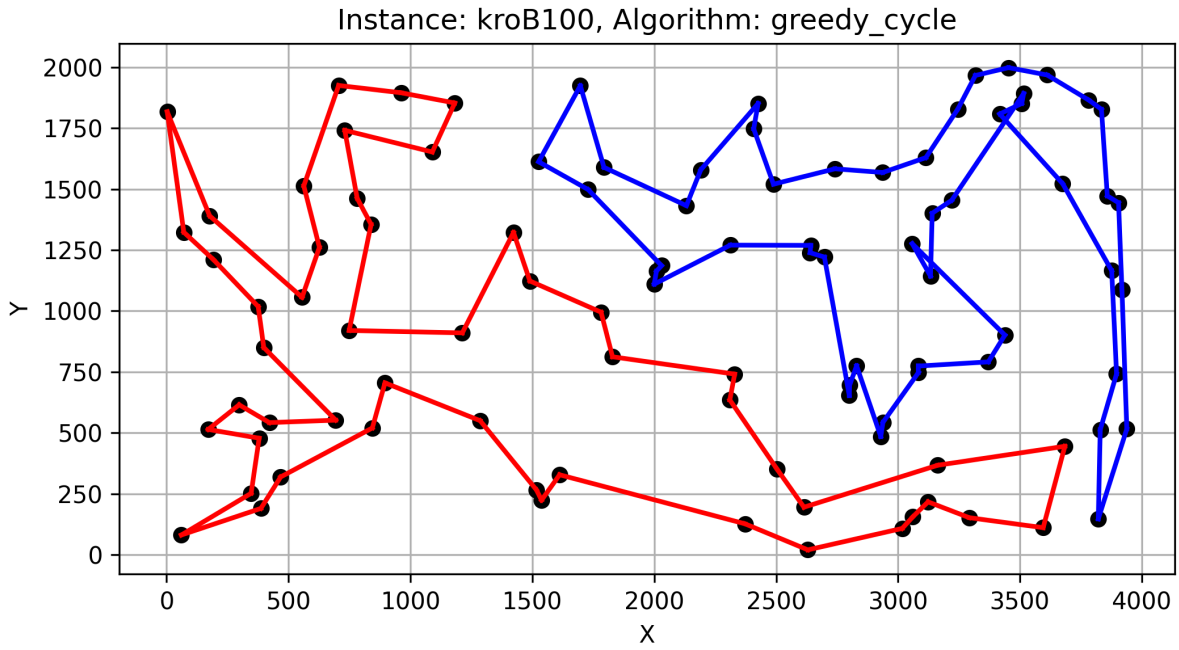


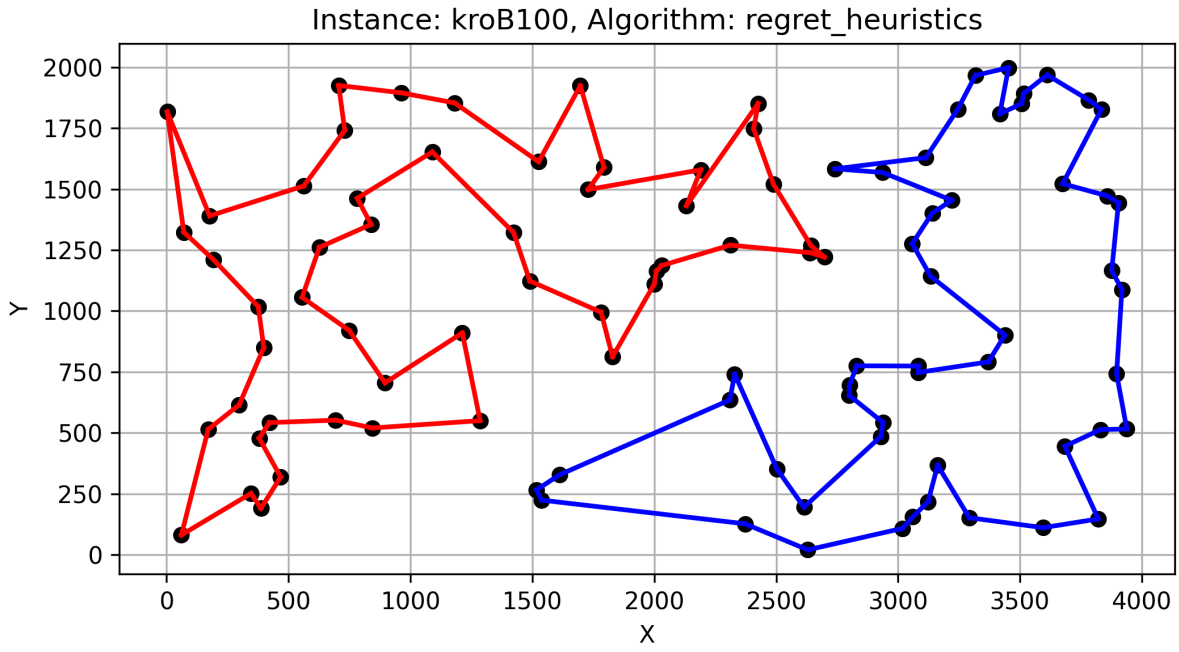

Instance: kroA100, Algorithm: greedy_cycle
Y
X
0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000
2500
3000
3500
4000


Instance: kroA100, Algorithm: regret_heuristics
Y
X
0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000
2500
3000
3500
4000

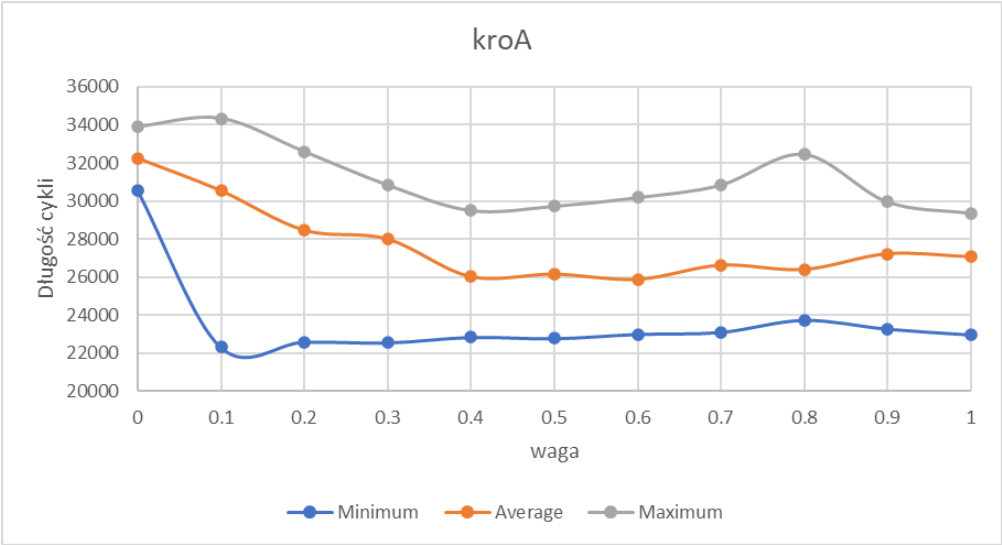
kroB100.tsp

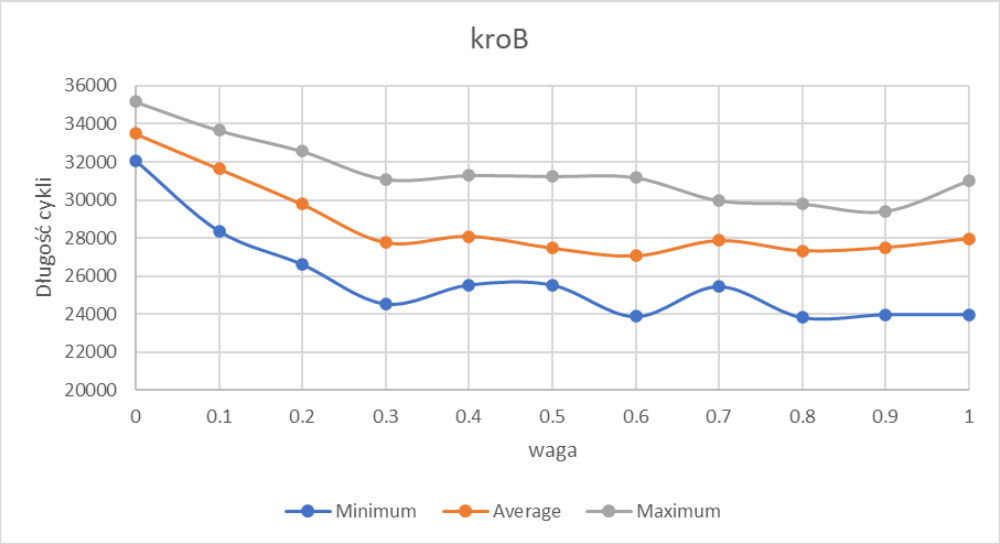

Instance: kroB100, Algorithm: nearest_neighbor
Y
X
0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000
2500
3000
3500
4000


Instance: kroB100, Algorithm: greedy_cycle
Y
X
0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000
2500
3000
3500
4000



Wykresy zależności długości cykli od wagi dla algorytmu opartego na heurystyce 2-żal





Wnioski

1. Najlepszym algorytmem dla obu instancji okazał się algorytm korzystający z heurystyki 2-żalu. Osiągnął o wiele lepsze wyniki od pozostałych jeżeli chodzi o wartości minimalne oraz średnie. Warto jednak zauważyć, że przy nieprzychylnym wyborze wierzchołków startowych algorytm ten radzi sobie podobnie jak greedy cycle. Można to jednak skorygować dzięki odpowiedniemu dobraniu wartości wagi.
2. Drugim algorytmem jest metoda rozbudowy cyklu. Radzi sobie ona lepiej niż algorytm oparty o najbliższych sąsiadów, co szczególnie widać na wartości średniej dla instancji kroA.
3. Najgorszym algorytmem jest metoda najbliższego sąsiada. Warto jednak zauważyć, że jest to najprostsza z metod i wymaga najmniej obliczeń do uzyskania wyniku.
4. Dla algorytmu opartego na 2-żalu niezbędne jest uwzględnienie wagi. W rozpatrywanym problemie brak wagi lub waga równa 0 nie prowadzi do uzyskania satysfakcjonujących wyników. Wynika to z natury rozpatrywanego problemu - algorytm korzystający z 2-żalu ma za zadanie "patrzeć w przyszłość", co jest bardzo utrudnione jeżeli budujemy dwa cykle równocześnie. W wyniku otrzymujemy dwa podobne do siebie cykle, które mają duże wartości długości. Po wprowadzeniu wagi dany cykl nie rozpatruje bardzo odległych wierzchołków co pozwala mu na budowanie rozwiązania bardziej lokalnego i tym samym nie przeszkadza drugiemu cyklowi.

Bibliografia

[1] https://github.com/mastqe/tsplib