

# **Metody metaheurystyczne**

## **Projekt 2**

### **Temat:**

“Rozwiązanie problemu marszrutyzacji przy pomocy algorytmu przeszukiwania z tabu”

Prowadzący: **prof. UEK dr hab. Grażyna Paliwoda-Pękosz**

### **Wykonali:**

Konrad Adamik - ZZISS2-2311IS

Weronika Mirek - ZZISS2-2312IS

## **Spis treści**

Definicja problemu optymalizacyjnego	3
Opis zastosowanego algorytmu	4
Wyniki	8
Wnioski	10
Netografia	11

## 1. Definicja problemu optymalizacyjnego

Rozważanym problemem jest problem marszrutyzacji (VRP - *Vehicle Routing Problem*), który polega na znalezieniu najbardziej optymalnego zestawu tras przewozowych dla pewnej określonej ilości środków transportu, które mają obsłużyć określoną ilość klientów. W standardowym problemie każdy klient powinien być obsłużony tylko przez jeden pojazd, a każdy pojazd musi wrócić do punktu początkowego po wizycie wszystkich przypisanych klientów.

Najbardziej powszechną modyfikacją jest problem, w którym istnieje jeden główny magazyn z pewną ilością pojazdów oraz określona liczba miast, do których te pojazdy muszą dotrzeć, tak aby łączna trasa przebyta przez pojazdy była najkrótsza.

Kryteria optymalizacyjne powyższego problemu to całkowity koszt transportu, który może być wyrażony w takich jednostkach jak:

- Odległość - najmniejsza łączna trasa przebyta przez wszystkie pojazdy
- Czas - najkrótszy łączny czas potrzebny na przebycie wszystkich tras,
- Cena - najmniejsza łączna cena (zależna od określonych przez użytkownika czynników) potrzebna na przebycie wszystkich tras.

Istnieje także wiele odmian problemu VRP, które pod uwagę biorą jeden lub więcej ograniczających parametrów tak jak pojemność samochodów, wiele magazynów startowych czy możliwość wcześniejszego powrotu do magazynu.

Opisywany w projekcie problem to odmiana VRP zwana CVRP (*Capacitated Vehicle Routing Problem*), dla którego kryterium optymalizacyjnym jest znalezienie najmniejszej łącznej trasy dla wszystkich pojazdów biorąc pod uwagę ograniczenia związane z ich pojemnością. Każdy klient w rozważanym problemie posiada pewne zapotrzebowanie. Pojazdy muszą wyruszyć z magazynu, odwiedzając swój zestaw klientów (klienci nie mogą się powtarzać), nie przekraczając swojej pojemności a na końcu wrócić do punktu startowego.

## 2. Opis zastosowanego algorytmu


Zastosowanym przez nas algorytmem do rozwiązania powyższego problemu jest przeszukiwanie z tabu. Jest to metoda metaheurystyczna wykorzystująca metody lokalnego sąsiedzkiego wyszukiwania.

Lokalne sąsiedzkie wyszukiwanie polega na iteracyjnym przeszukiwaniu sąsiedztwa obecnego rozwiązania (wybrania nieznacznie różnego, np poprzez zamianę dwóch miast, ścieżek pomiędzy miastami itp), sprawdzając kolejne rozwiązanie i wybiera je tylko gdy jest lepsze od obecnego. Wyszukiwanie lokalne ma jednak tendencje do blokowania się w rejonach rozwiązania które jest minimum lokalnym (dla problemu minimalizacji) podczas gdy istnieje lepsze rozwiązanie.

Przeszukiwanie z tabu wprowadza dodatkową listę Tabu (słowo pochodzi od języka tongańskiego, oznacza rzeczy, których nie można dotknąć bo są święte), która przechowuje ruchy zakazane, czyli takie, których przeszukiwanie lokalne nie może użyć by znaleźć kolejne rozwiązanie. Dodatkowo przeszukiwanie z Tabu pozwala na wybranie gorszego rozwiązania, w celu zniwelowania problemu utknięcia w lokalnym minimum.

Algorytm zaimplementowany przez nas jest otwarty na modyfikacje. Przy pomocy pliku konfiguracyjnego (*config.txt*) użytkownik może dowolnie zmieniać poniższe wartości:

- Liczba iteracji (*maximumIterations*)
- Rozmiar listy tabu (*tabuListSize*)
- Kadencja (*cadence*)
- Seed dla generatora liczb losowych (*randomSeedGenerator*)



1	5000
2	100
3	30
4	1
5	

**Rys 1.:** Przykładowa treść pliku konfiguracyjnego

Na wejście programu wchodzi również plik z danymi dotyczącymi rozwiązywanego problemu w formacie:

- Liczba pojazdów
- Pojemność pojazdów
- Nazwa miasta,zapotrzebowanie,szerokość geograficzna, długość geograficzna (w formacie DD)

1	5
2	1000
3	Kraków,0,50.06835396394607,19.9440330916948
4	Białystok,500,53.1321003522936,23.168303572712947
5	Bielsko-Biała,50,49.80540744426564,19.05579816246941
6	Chrzanów,400,50.134474537802156,19.398192427817595
7	Gdańsk,200,54.37256547735844,18.623777381853735
8	Gdynia,100,54.52381045432751,18.51172083217754
9	Gliwice,40,50.29118914781811,18.671219651039902
10	Gromnik,200,49.8378202552782,20.96195423629714
11	Katowice,300,50.25773311928509,19.035992148086056
12	Kielce,30,50.865632803041485,20.628814616278376
13	Krosno,60,49.682198558407634,21.765661220534504
14	Krynica,50,49.421428083650596,20.95931413164279
15	Lublin,50,51.24586910634751,22.567275823696246
16	Łódź,160,51.75906799854293,19.453798701517833
17	Malbork,100,54.03560211260634,19.037285666387753
18	Nowy Targ,120,49.47755139275044,20.03156676253084
19	Olsztyn,300,53.778291358093455,20.480558664351374
20	Poznań,100,52.406342511599135,16.92404287751222
21	Puławny,200,51.416185106219984,21.971161831771095
22	Radom,100,51.40240674858944,21.147495996421405
23	Rzeszów,60,50.04201175345502,21.999504944386256
24	Sandomierz,200,50.682292211704585,21.750335235531214
25	Szczecin,150,53.428416231851784,14.552400399580305
26	Szczucin,60,50.309358120975546,21.074894422100936
27	Szklarska Poręba,50,50.82809799417874,15.52517480336706
28	Tarnów,70,50.01235981232834,20.985014481632366
29	Warszawa,200,52.229601025984486,21.01304239806023
30	Wieliczka,90,49.98705452367493,20.064306223857024
31	Wrocław,40,51.10778914939742,17.036104535056676
32	Zakopane,200,49.275918049029045,19.9712679861164
33	Zamość,300,50.72278381441436,23.25177534443028
34	

Rys 2.: Przykładowa treść pliku

Po odczytaniu wszystkich potrzebnych danych z plików, program rozpoczyna swoje działanie od wygenerowania pierwszego rozwiązania. Robi to zgodnie z poniższymi krokami:

1. Zainicjalizuj trasę dla każdego pojazdu dodając magazyn jako punkt startowy.
2. Posortuj listę klientów po zapotrzebowaniu od największego do najmniejszego.
3. Dopóki lista klientów nie jest pusta, powtarzaj:

- a. Na podstawie seeda pomieszaj elementy listy klientów.
  - b. Dla każdego pojazdu jeśli lista klientów nie jest pusta i jeśli możliwe jest dodanie klienta (sprawdź, czy dodanie nie spowoduje przekroczenia pojemności pojazdu) to dodaj klienta na pierwszej pozycji listy klientów.
4. Do każdego pojazdu dodaj magazyn na końcu jego trasy.
5. Ustaw znalezione rozwiązanie na najlepsze znane rozwiązanie.
6. Oblicz całkowitą sumę dystansów dla znalezionej konfiguracji i ustaw na najlepszą znaną sumę.

Przykładowe pierwsze losowe rozwiązanie (dla konfiguracji 1000000, 5, 3, 834658):

**Trasa 1:**

0:Kraków -> 8:Katowice -> 7:Gromnik -> 15:Nowy Targ -> 19:Radom -> 20:Rzeszów -> 6:Gliwice -> 0:Kraków

**Trasa 2:**

0:Kraków -> 3:Chrzanów -> 26:Warszawa -> 13:Łódź -> 25:Tarnów -> 2:Bielsko-Biała -> 24:Szklarska Poręba -> 28:Wrocław -> 0:Kraków

**Trasa 3:**

0:Kraków -> 1:Białystok -> 4:Gdańsk -> 22:Szczecin -> 17:Poznań -> 0:Kraków

**Trasa 4:**

0:Kraków -> 30:Zamość -> 21:Sandomierz -> 5:Gdynia -> 27:Wieliczka -> 10:Krosno -> 12:Lublin -> 9:Kielce -> 0:Kraków

**Trasa 5:**

0:Kraków -> 16:Olsztyn -> 18:Puławy -> 29:Zakopane -> 14:Malbork -> 23:Szczucin -> 11:Krynica -> 0:Kraków

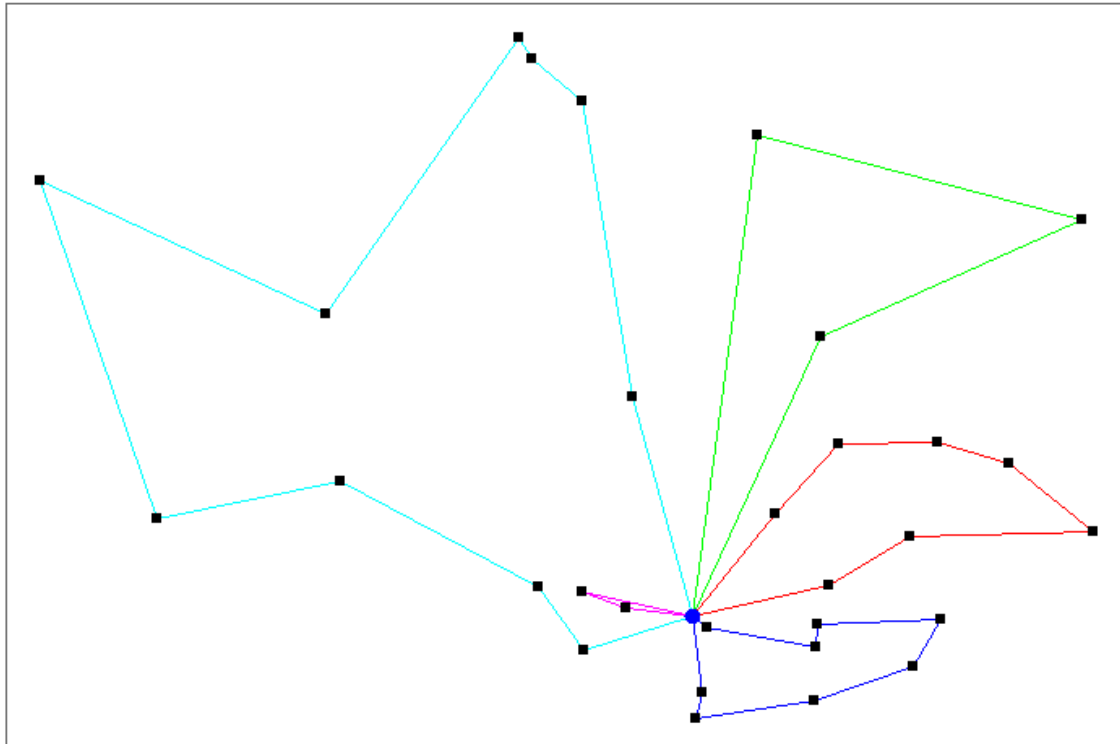
Następnie właściwy algorytm może rozpocząć swoje działanie. Warunkiem końcowym głównej pętli jest maksymalna liczba iteracji podana w pliku konfiguracyjnym razem z rozmiarem listy tabu oraz kadencją, czyli liczbą iteracji przez którą element będzie przechowywany na liście. Na liście tabu będzie przechowywany element *Swap*, czyli losowa para miast, które zostaną zamienione ze sobą miejscami. Główna pętla działa według poniższych kroków:

1. Na podstawie seeda pomieszaj listę zawierającą indeksy klientów (z pominięciem magazynu) oraz zapisz parę wylosowanych klientów do zmiennej typu *Swap*.
2. Sprawdź, czy wylosowany *Swap* nie znajduje się na liście tabu, jeśli tak, wróć do punktu 1.
3. Znajdź, w których pojazdach znajdują się wylosowane miasta i zapisz je do zmiennych.
4. Sprawdź czy wylosowany *Swap* nie spowoduje, że w zapisanych pojazdach pojemność zostanie przekroczona. Jeśli tak, wróć do punktu 1.

5. Zamień wylosowane miasta w trasach odpowiednich pojazdów.
6. Dodaj znaną zamianę (parę miast - *Swap*) na listę tabu jeśli rozmiar listy na to pozwala.
7. Dla każdego elementu na liście tabu zdekrementuj kadencję, sprawdź czy któryś z elementów nie ma kadencji równej 0, jeśli tak to usuń go z listy.
8. Wylicz sumę dystansów znalezionej trasy.
9. Jeśli ta suma jest lepsza od najlepszej znanej sumy, zapisz ją jako nową najlepszą znaną sumę, zapisz także znalezione rozwiązanie jako najlepsze znane rozwiązanie.

### 3. Wyniki

Rozwiązywany problem posiadał 30 miast, każdy z własnym zapotrzebowaniem, oraz 5 pojazdów, każdy o pojemności 1000. Poszukiwana droga dąży do tego aby zminimalizować sumę dróg potrzebną na dostarczenie zapotrzebowania do każdego miasta, tak by nie przekroczyć pojemności pojazdu. Na początek eksperymentowaliśmy z programem HeuristicLab, najlepsze znalezione przez nas rozwiązanie to łączna długość tras wynosząca 3876,16 km, jej wizualizacja przedstawia się następująco:



Rys 3.: Rozwiązanie z programu HeuristicLab

#### Trasa

1:

Kraków -> Nowy Targ -> Zakopane -> Krynica -> Krosno -> Rzeszów -> Tarnów -> Gromnik -> Wieliczka -> Kraków

#### Trasa 2:

Kraków -> Warszawa -> Białystok -> Olsztyn -> Kraków

#### Trasa 3:

Kraków -> Szczucin -> Sandomierz -> Zamość -> Lublin -> Puławy -> Radom -> Kielce -> Kraków

#### Trasa 4:

Kraków -> Łódź -> Malbork -> Gdańsk -> Gdynia -> Poznań -> Szczecin -> Szklarska Poręba -> Wrocław -> Gliwice -> Bielsko-Biała -> Kraków

#### Trasa 5:

Kraków -> Katowice -> Chrzanów -> Kraków



Używając naszego algorytmu, przeprowadziliśmy wielokrotne testy zmieniając parametry konfiguracyjne. Dla następująco losowo wybranej trasy o łącznej długości 8553.64 km:

**Trasa**

**1:**

0:Kraków -> 1:Białystok -> 7:Gromnik -> 22:Szczecin -> 14:Malbork -> 9:Kielce -> 0:Kraków

**Trasa 2:**

0:Kraków -> 8:Katowice -> 4:Gdańsk -> 5:Gdynia -> 19:Radom -> 10:Krosno -> 6:Gliwice -> 0:Kraków

**Trasa 3:**

0:Kraków -> 16:Olsztyn -> 21:Sandomierz -> 15:Nowy Targ -> 27:Wieliczka -> 20:Rzeszów -> 11:Krynica -> 0:Kraków

**Trasa 4:**

0:Kraków -> 30:Zamość -> 18:Puławy -> 29:Zakopane -> 17:Poznań -> 23:Szczucin -> 12:Lublin -> 28:Wrocław -> 0:Kraków

**Trasa 5:**

0:Kraków -> 3:Chrzanów -> 26:Warszawa -> 13:Łódź -> 25:Tarnów -> 2:Bielsko-Biała -> 24:Szkłarska Poręba -> 0:Kraków

Udało się osiągnąć następujący najlepszy wynik o łącznej długości trasy 5429,58 km:

**Trasa**

**1:**

0:Kraków -> 5:Gdynia -> 30:Zamość -> 13:Łódź -> 3:Chrzanów -> 19:Radom -> 0:Kraków

**Trasa 2:**

0:Kraków -> 27:Wieliczka -> 4:Gdańsk -> 24:Szkłarska Poręba -> 23:Szczucin -> 20:Rzeszów -> 2:Bielsko-Biała -> 0:Kraków

**Trasa 3:**

0:Kraków -> 9:Kielce -> 15:Nowy Targ -> 22:Szczecin -> 10:Krosno -> 1:Białystok -> 8:Katowice -> 0:Kraków

**Trasa 4:**

0:Kraków -> 25:Tarnów -> 11:Krynica -> 17:Poznań -> 26:Warszawa -> 14:Malbork -> 12:Lublin -> 28:Wrocław -> 0:Kraków

**Trasa 5:**

0:Kraków -> 16:Olsztyn -> 29:Zakopane -> 21:Sandomierz -> 6:Gliwice -> 7:Gromnik -> 18:Puławy -> 0:Kraków

## 4. Wnioski

Implementacja algorytmu przeszukiwania z tabu jest stosunkowo prosta i nie sprawiała większych kłopotów, problem stanowiło jednak zrozumienie jak zastosować ten algorytm do rozwiązania problemu marszrutyzacji. Algorytm ten jest jednak bardzo wydajny, przeprowadzenie jednego przeszukiwania zarówno w przypadku HeuristicLab jak i naszego algorytmu wynosi około 50 milisekund.

Napisany przez nas algorytm potrafi znaleźć rozwiązanie znacznie lepsze od rozwiązania wygenerowanego losowo. W przypadku, w którym znaleźliśmy najlepsze rozwiązanie, algorytm znalazł trasę lepszą o ponad 3000 km, co daje zdecydowaną poprawę kryterium minimalizacji. Należy jednak zauważyć, że daleko mu jednak jeszcze do rozwiązania optymalnego znalezione przez HeuristicLab. Kandydatem do poprawy, tak aby poprawić optymalność całego algorytmu, jest metoda wybierająca kolejnych sąsiadów. Potencjalnym usprawnieniem byłoby rozbudowanie tej metody o możliwość przeczucania miast do innych tras bez zamiany. Algorytm mógłby wtedy przeszukać znacznie większą ilość potencjalnie lepszych tras.

## 5. Netografia

1. [https://www.szkolnictwo.pl/szukaj,Problem\\_marszrutyzacji](https://www.szkolnictwo.pl/szukaj,Problem_marszrutyzacji) (Dostęp 07.01.2021)
2. [https://pl.qaz.wiki/wiki/Tabu\\_search](https://pl.qaz.wiki/wiki/Tabu_search) (Dostęp 07.01.2021)
3. [https://pl.qaz.wiki/wiki/Local\\_search\\_\(optimization\)](https://pl.qaz.wiki/wiki/Local_search_(optimization)) (Dostęp 08.01.2021)
4. <https://cs.pwr.edu.pl/zielinski/lectures/om/localsearch.pdf> (Dostęp 08.01.2021)
5. <https://nowosad.github.io/ahod/12-tabu-search.html#1> (Dostęp 08.01.2021)
6. <http://www.cs.put.poznan.pl/mkomosinski/lectures/optimization/TS.pdf> (Dostęp 07.01.2021)