Algorytmy Metaheurystyczne
Lista 3 - TSP - Algorytmy Genetyczne

Rafał Włodarczyk

02.06.2026

Spis treści

1 Model

Model TSP zakłada graf pełny $G = (V, E)$ z wagami.

2 Źródło dla danych

Dane do programu są wzięte ze strony internetowej:

`https://www.math.uwaterloo.ca/tsp/world/countries.html`

oraz są dostępne do pobrania w formacie `.tsp` za pomocą skryptu `get_data.sh`.

3 Idea Algorytmu Genetycznego

Algorytm Genetyczny zgodnie z opisem na liście składa się z pięciu kroków

1. Populacja Początkowa. Wykorzystałem permutację powstałą za pomocą MST.
2. Ewaluacja i Selekcja. Selekcja odbywa się za pomocą przyspieszonej metody turniejowej. Wybierane są dwa dowolne elementy populacji (część losowa), a następnie brany jest większy z nich (część turniejowa).
3. Krzyżowanie. Wykorzystane zostaną dwa algorytmy `PMX Crossover` oraz `OX Crossover`.
4. Mutacja. Odbywa się za pomocą wybrania dwóch indeksów x, y, a następnie odwrócenia podpermutacji - wykonanie inwersji.
5. Warunek Końca. Ustalony na sztywną liczbę generacji. Zwracamy najlepszy z wyników

Ponadto rozwinięty zostanie o dwa dodatkowe elementy

1. Memetyka - Przyspieszenie *dojrzewania* nowych elementów populacji. Zastosowany jest skrócony algorytm Tabu Search, z ustaloną niską maksymalną liczbą iteracji.
2. Wyspowość - Podzielenie populacji na mniejsze niezależne podpopulacje, które po ustalonej liczbie generacji wymieniają się informacjami za pomocą specjalnej funkcji.

4 Zadanie 1 - Metody krzyżowania

Wybrałem dwie metody krzyżowania:

1. PMX (Partially Mapped Crossover) - Goldberg / Lingle, 1985.
2. OX (Order Crossover) - Davis, 1985

4.1 Metoda PMX

Metoda PMX (Partially Mapped Crossover) to algorytm krzyżowania permutacji stworzony z myślą o TSP. Bierzemy dwie permutacje p_1, p_2, będziemy tworzyć dziecko c. Bierzemy dwa dowolne punkty $0 \leqslant x \leqslant y \leqslant n$, następnie dzielimy zbiór na trzy części: lewy (indeksy $0 \dots x-1$), środkowy (indeksy $x \dots y$), oraz prawy (indeksy $y+1 \dots n$).

1. Segment środkowy (x, y) zostaje powielony z p_1 do c
2. Znajdujemy konflikty - przenosimy elementy z permutacji p_2 do c, o ile już w c ich nie ma. Musimy znaleźć mapowania elementów $k_1, \dots, k_l$ do c spoza segmentu środkowego. Niech $k = k_i$ (dowolny brakujący element). Szukamy indeksu $i : p_2(i) = k$, znajdujemy wartość $m = p_1(i)$, a następnie szukamy dla jakiego j $p_2(j) = m$. Gdy $j \notin (x, y)$ możemy umieścić element $c(j) = k$. Osiągamy dzięki temu zestaw mapowania, który pozwala nam umieścić k w wolne miejsce c.

4.2 Metoda OX

Metoda OX (Order Crossover) to algorytm krzyżowania permutacji dla TSP. Bierzemy dwie permutacje p_1, p_2, będziemy tworzyć dziecko c. Bierzemy dwa dowolne punkty $0 \leqslant x \leqslant y \leqslant n$, następnie dzielimy zbiór na trzy części: lewy (indeksy $0 \dots x-1$), środkowy (indeksy $x \dots y$), oraz prawy (indeksy $y+1 \dots n$).

1. Segment środkowy (x, y) zostaje powielony z p_1 do c
2. Dokonujemy cyklicznego wypełnienia zaczynając od indeksu $i = y+1$, sprawdzamy czy element $p_2(i)$ jest już w c, jeśli jest to dodajemy indeks i przechodzimy dalej. Jeśli nie ma to wpisujemy $c(i) = p_2(i)$ i przesuwamy i na następne wolne miejsce.

4.3 Wyniki porównania metod PXM i OX

Dla zadanych danych OX jest lepszy od PMX. Zostanie zatem wykorzystany w zmodyfikowanych algorytmach.

5 Zadanie 2 - Algorytm Wyspowy

Algorytm wyspowy otrzymuje dodając dwa dodatkowe parametry

1. `icount` - Ilość wysp, na które zostanie podzielony zbiór populacji.
2. `interval` - Ilość iteracji po których zachodzi wymiana informacji pomiędzy wyspami. Niech I to będzie nasz interwał. Wtedy do wymiany informacji dojdzie w iteracjach $k \cdot I, k \in \mathbb{N}_{\geqslant 0}$

6 Zadanie 3 - Wykorzystanie OpenMP w celu przyspieszenia obliczeń

Wykorzystam bibliotekę OpenMP `<omp.h>` która pozwala na zrównoleglenie obliczeń - wykorzystuję rozdzielając część pętli na wiele rdzeni. Dzięki ustaleniu generatora liczb losowych per wątek mogę uzyskiwać różne wyniki symulując konsekwentne wykonywanie kolejnych iteracji algorytmów.

Wykorzystane elementy to m.in. zrównoleglenie podziału wysp oraz operacji niezależnych wysp oraz zrównoleglenie wykonania algorytmu memetycznego (tabu search).

6.1 Zrównoleglenie przy wyspach

`#pragma omp parallel for collapse(2)` - zrównoleglamy podwójne zagnieżdżenie pętli.

```
#pragma omp parallel for collapse(2)
for i in 1..n {
    for j in 1..m {

    }
}
```

Zostaje ewaluowany do pojedynczego egzemplarza, a następnie wykonuje każdą operacje równolegle

```
for (i,j) in (1..n, 1..m) {

}
```

6.2 Zrównoleglenie Tabu w wariancie Memetycznym

```
#pragma omp declare reduction(min_move : BestMove : \
    omp_out = (omp_in.delta < omp_out.delta) ? omp_in : omp_out) \
    initializer(omp_priv = {std::numeric_limits<int64_t>::max(), 0, 0, false})

#pragma omp parallel for reduction(min\_move\:local\_best) schedule(guided)
```

Każdy wątek dostaje kopie zmiennej `local_best`, która następnie przy łączeniu wątków zostaje poddana operacji redukcji `min_move` - która z dwóch lokalnych wartości wybiera mniejszy z nich (lepsze rozwiązanie).

7 Zadanie 4 - Algorytm Memetyczny

Moja realizacja algorytmu memetycznego opiera się o dodanie Tabu Search po mutacji dzieci poprzedniej populacji. Dzięki temu zdecydowanie szybciej (w obrębie jednej generacji) będziemy optymalizować do kolejnego minimum lokalnego po mutacji - czyli po potencjalnym wyskoczeniu z minimum lokalnego.

8 Porównanie Algorytmów z Listy 3

Stworzyłem cztery zestawy mające obrazować każdy z poszczególnych algorytmów:

1. `pmx` - Zestaw dla krzyżowania PMX
    - `population size` $= 20$

- `generations` = 100
- `icount` = 1 (brak niezależnych wysp)
- `interval` = 0 (nie dotyczy, jw.)
- `crossover` = `pmx`
- `enable mutation` = 1 (pozwalamy na mutacje)
- `enable ts` = 0 (wyłączamy memetyczny Tabu Search)

2. `ox` - Zestaw dla krzyżowania OX
   - `population size` = 20
   - `generations` = 100
   - `icount` = 1 (brak niezależnych wysp)
   - `interval` = 0 (nie dotyczy, jw.)
   - `crossover` = `ox`
   - `enable mutation` = 1 (pozwalamy na mutacje)
   - `enable ts` = 0 (wyłączamy memetyczny Tabu Search)
3. `memetic` - Algorytm Memetyczny
   - `population size` = 20
   - `generations` = 100
   - `icount` = 1 (brak niezależnych wysp)
   - `interval` = 0 (nie dotyczy, jw.)
   - `crossover` = `ox` (lepszy od `pmx`)
   - `enable mutation` = 1 (pozwalamy na mutacje)
   - `enable ts` = 1 (włączamy moduł Memetyczny - krótki Tabu Search)
4. `island` - Algorytm Wyspowy
   - `population size` = 20
   - `generations` = 100
   - `icount` = 10 (10 niezależnych wysp)
   - `interval` = 20 (wymiana informacji co 20 iteracji)
   - `crossover` = `ox` (lepszy od `pmx`)
   - `enable mutation` = 1 (pozwalamy na mutacje)
   - `enable ts` = 1 (włączamy moduł Memetyczny - krótki Tabu Search)

Tabela 1: Wyniki algorytmów Genetycznych

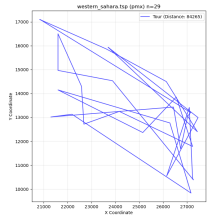
Algorithm	**W. Sahara** ($n = 29$)	**Djibouti** ($n = 38$)	**Qatar** ($n = 194$)	**Uruguay** ($n = 734$)	**Zimbabwe** ($n = 929$)	**Oman** ($n = 1979$)	**Canada** ($n = 4663$)	**Tanzania** ($n = 6117$)	**Egypt** ($n = 7146$)	**Ireland** ($n = 8246$)
pmx	84265	22795	85664	1412459	2161399	4731207	121144544	28291539	12016919	13762349
ox	83260	21899	63846	731047	961195	969790	11174641	2868516	957868	979856
memetic	27603	6656	9524	83679	100284	92449	1413280	445694	194054	241624
island	27750	6656	9869	84547	101064	92649	1420103	445778	193050	240879

Zarówno `pmx` oraz `ox` są przy małej liczbie generacji niezbyt interesujące, natomiast widać że `ox` jest lepszy. Dodanie przy tej samej liczbie generacji wysp oraz algorytmu memetycznego pozwala osiągnąć lepsze wyniki. Algorytm memetyczny w postaci skróconego Tabu Searcha pozwala na uzyskanie wyników optymalnych dla małych problemów, i w granicach $10\% - 20\%$ od wyniku optymalnego dla większych problemów.

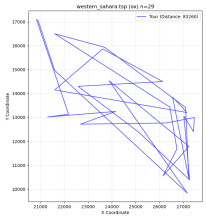
Algorytm Memetyczny w Oparciu o Tabu Search radzi sobie zdecydowanie najlepiej, co jest tożsame z wnioskami z poprzedniej listy - Tabu Search oraz jego elementy to dobre algorytmy Metaheurystyczne.

9 Wykresy najlepszych rozwiązań

9.1 Western Sahara



Western Sahara(pmx)

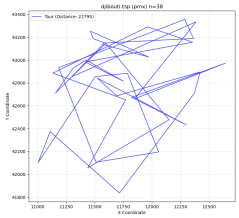


Western Sahara(ox)

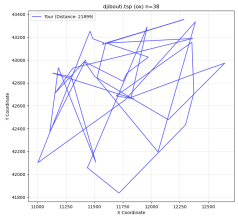
Western Sahara(memetic)

Western Sahara(island)

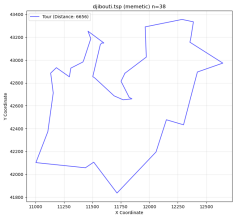
9.2 Djibouti



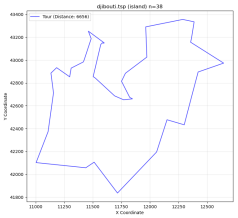
Djibouti(pmx)



Djibouti(ox)

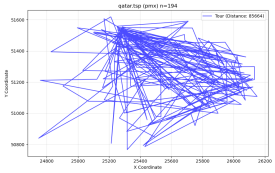


Djibouti(memetic)

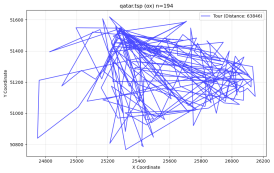


Djibouti(island)

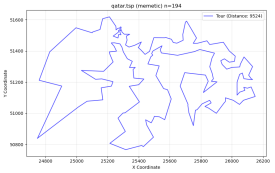
9.3 Qatar



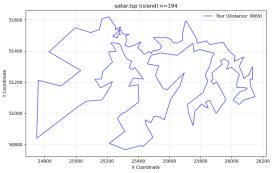
Qatar(pmx)



Qatar(ox)

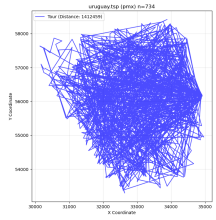


Qatar(memetic)

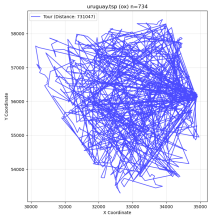


Qatar(island)

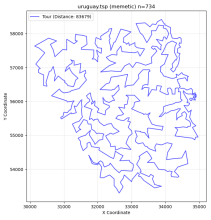
9.4 Uruguay



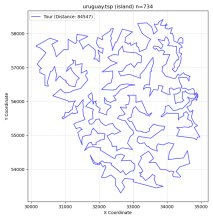
Uruguay(pmx)



Uruguay(ox)

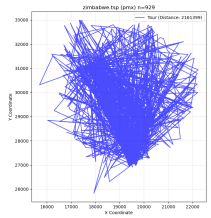


Uruguay(memetic)

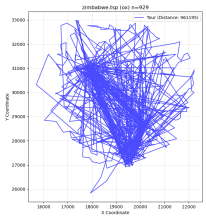


Uruguay(island)

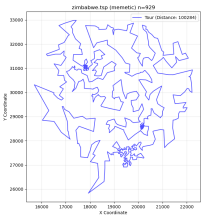
9.5 Zimbabwe



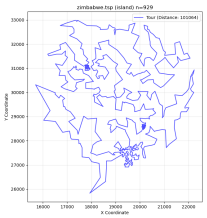
Zimbabwe(pmx)



Zimbabwe(ox)

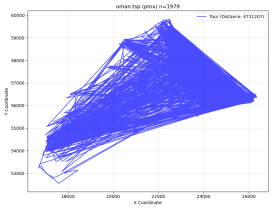


Zimbabwe(memetic)

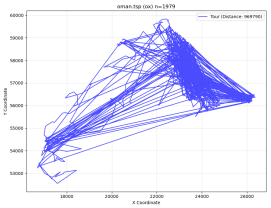


Zimbabwe(island)

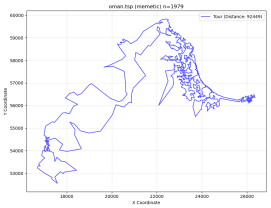
9.6 Oman



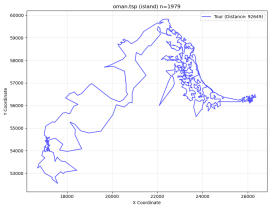
Oman(pmx)



Oman(ox)

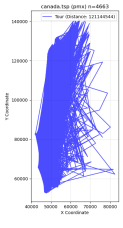


Oman(memetic)

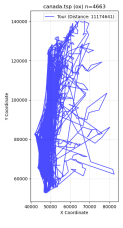


Oman(island)

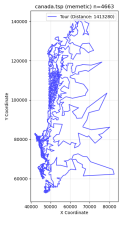
9.7 Canada



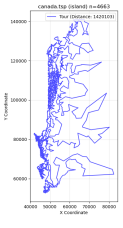
Canada(pmx)



Canada(ox)

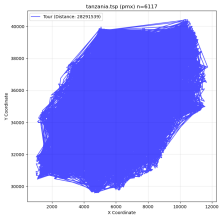


Canada(memetic)

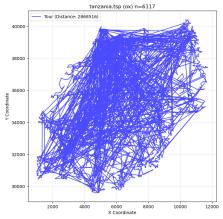


Canada(island)

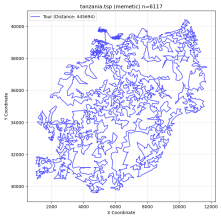
9.8 Tanzania



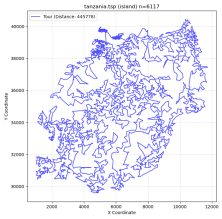
Tanzania(pmx)



Tanzania(ox)

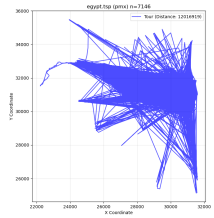


Tanzania(memetic)

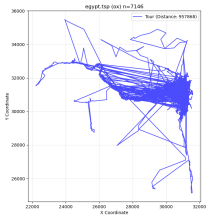


Tanzania(island)

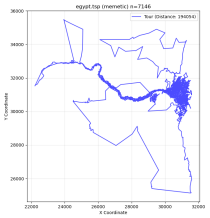
9.9 Egypt



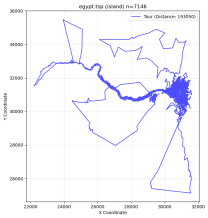
Egypt(pmx)



Egypt(ox)

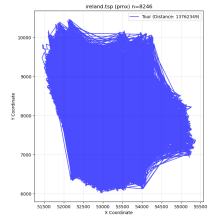


Egypt(memetic)

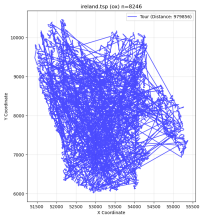


Egypt(island)

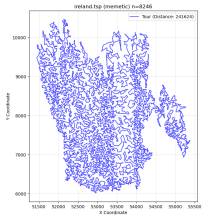
9.10 Ireland



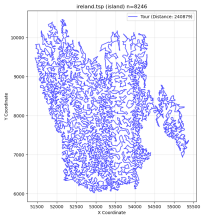
Ireland(pmx)



Ireland(ox)



Ireland(memetic)



Ireland(island)

10 Finalne zestawienie wszystkich algorytmów

10.1 Algorytmy zawarte w porównaniu

10.2 Zestawienie Wyników Numerycznych

11 Główne Wnioski z Ewolucji Algorytmów

1. **Wpływ startu** Znaczenie pozycji startowej jest kluczowe do otrzymania dobrego wyniku
2. **Wychodzenie z minimów lokalnych** Algorytmy powinny próbować wychodzić z minimów lokalnych, aby dążyć do lepszego rozwiązania.
3. **Podejście Genetyczne** Połączenie algorytmów typu Tabu Search z rozwiązaniami populacyjnymi pozwala osiągać zadowalające wyniki.

Tabela 2: Opis algorytmów metaheurystycznych

Algorytm	**Opis**	**Zachowanie**
Lista 1: Local Search + Rożne Sąsiedztwa		
L1: Full Inversion	Losowy start. Przeszukiwanie całego sąsiedztwa n^2 operacją `invert`.	Dokładna, lecz z wysoką złożonością - niepraktyczna dla dużych problemów.
L1: n-Losowych	Losowy Start. Przeszukiwanie n losowych sąsiadów w każdej iteracji.	Szybciej sprawdzamy mniej inwersji, tracąc dokładność.
L1: MST	Start z MST + heurystyka 2-OPT.	Znacznie lepszy od n losowych przy wynikach gorszych, lecz zbliżoną do pełnego sąsiedztw.
Lista 2: Wychodzenie z minimów lokalnych		
L2: Symulowane Wyżarzanie	Losowy Start. Dostrojony za pomocą T, α, e, t	Pozwala na wypadnięcie z minimum lokalnego, natomiast wraz z kolejnymi iteracjami zmniejszamy szansę na uzyskanie lepszego wyniku
L2: Tabu Search	Losowy Start. Local Search z listą Tabu	Znacznie lepszy od symulowanego wyżarzania, często wypada z minimum lokalnego uzyskując dobre wyniki
Lista 3: Algorytmy Genetyczne i Hybrydowe		
L3: Genetyczny PMX	Start Losowy. Wykorzystanie krzyżowania PMX	Szybki lecz niedokładny, gorszy od OX
L3: Genetyczny OX	Start Losowy. Wykorzystanie krzyżowania OX	Szybki lecz niedokładny, lepszy od PMX
L3: Memetyczny przez TS	Start z MST. Opatrzony skróconym Tabu Search	Wyniki zbliżone do optymalnych.
L3: Wyspowy + TS	Start z MST. Szybszy od memetycznego. Dobry i łatwy do zrównoleglenia dla dużych problemów.	Osiąga bliskie optimum wyniki dla każdego zestawu danych

Tabela 3: Porównanie długości tras dla trzech wyróżnionych zestawów

Algorytm *OPT*	**Djibouti** ($n = 38$) *6656*	**Zimbabwe** ($n = 929$) *95345*	**Ireland** ($n = 8246$) *206171*
Lista 1: Local Search + Rożne Sąsiedztwa			
L1: Full Inversion	7535	105188	**232691**
L1: n-Losowych	8034	163424	435965
L1: MST	5828	104581	233043
Lista 2: Wychodzenie z minimów lokalnych			
L2: Symulowane Wyżarzanie	7347	457583	5320736
L2: Tabu Search	**6656**	105303	255336
Lista 3: Algorytmy Genetyczne i Hybrydowe			
L3: Genetyczny PMX	22795	2161399	13762349
L3: Genetyczny OX	21899	961195	979856
L3: Memetyczny (OX + TS)	**6656**	**100284**	241624
L3: Wyspowy (OX + TS)	**6656**	101064	240879