Multi-objective Traveling Salesman Problem

Kamil Michalak

1 Project Description

1.1 Project Goal

The goal of the project is to implement various algorithms that solve the *multi-objective Traveling Salesman Problem* (*multi-objective TSP*), test their efficiency and compare their performance.

1.2 Problem Definition

Input: N cities and p costs $c^k_{i,j}$ $(k = 1, ..., p)$ of traveling from city i to city j.

Goal: Find a permutation ρ (composed of N cities) that minimizes the function

$$z(\rho) = (z_1(\rho), ..., z_p(\rho))$$

$$z_k(\rho) = \sum_{i=1}^{N} c^k_{\rho(i),\rho(i+1)} + c^k_{\rho(N),\rho(1)}$$

1.3 Data Used

In the tests, problem instances from the TSPLIB library (`http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`) were used. The implemented algorithms were tested on the following instances:

- $(kroA100, kroB100)$
- $(kroC100, kroD100)$

Optimal solutions for 1-objective instances:

- *kro100A* – 21285.4
- *kro100B* – no data available
- *kro100C* – 20750.7
- *kro100D* – 21294.2

1.4 Technical Data

- Programming Language: Python + Numba compiler to accelerate code execution
- System: Ubuntu 20.04
- Processor: AMD Ryzen 7 4800H
- 32 GB RAM

2 NSGA-II

2.1 Algorithm Details

Crossover Operators

- Order-1 crossover (OX)
- Partial-mapped crossover (PMX)
- Cycle crossover (CX)
- Position-based crossover (PBX)

Parent Selection - Tournament selection

We draw K individuals from the population and sort them by rank and crowding distance. We choose the best with probability p, the second with probability $p \cdot (1 - p)$, the third – $p \cdot (1 - p)^2$, and so on.

Mutation operators

- Reverse sequence mutation – reversing a random slice of a permutation
- Swap mutation – swapping 2 random elements of a permutation

2.2 Tests

The algorithm was tested on (*kroA100, kroB100*) and (*kroC100, kroD100*) data for each possible combination of crossover and mutation listed in the previous section.

- population size: 200
- number of iterations: 20000
- in selection K =20, p =0.9

Each test took approximately 8 minutes to perform.

crossover	mutation	kroA100	kroB100	kroC100	kroD100
OX	reverse	21876.7	23585.6	22466.7	22268.9
PBX	reverse	22307.4	23703.0	22832.0	22322.9
CX	reverse	22676.6	23979.2	22017.9	22877.8
PMX	reverse	23463.1	22963.7	22376.5	21664.3
OX	swap	29143.2	33533.1	30177.9	28891.3
PBX	swap	36349.1	37808.1	32084.5	32744.9
CX	swap	38494.1	38494.1	35818.7	35331.7
PMX	swap	38532.6	37113.2	35377.8	37344.4

Tabela 1: Najmniejsze wyniki znalezione przez algorytm NSGA-II, dla 20000 iteracji

Results

- Fig. **??** presents graphs of minimum, maximum and average cycle lengths in the population for successive iterations of the algorithm
- Fig. **??** shows the results of the best solutions for all variants of the algorithm for 20000 iterations, and one (OX + reverse sequence) for 50000
- Fig. **??** shows non-dominated individuals for successive (not all) iterations of the algorithm for order-1 crossover and reverse sequence mutation
- The table 1 shows the lengths of the shortest cycles found (results in one row come from **different** solutions)

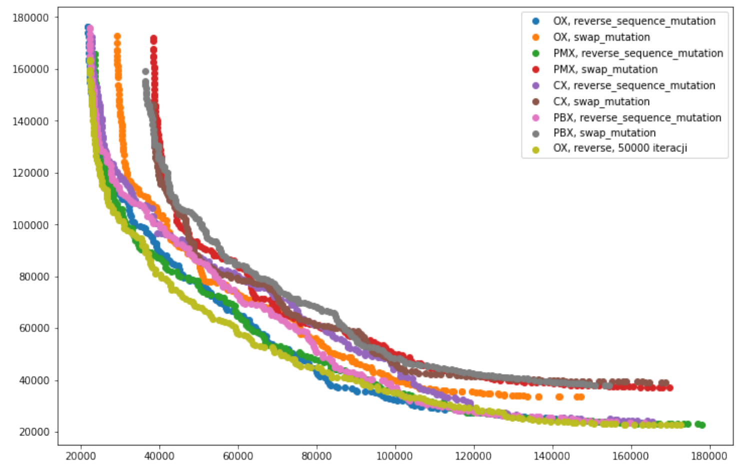
Observations

- The algorithm achieved good results, approaching optimal solutions in the best variants
- The *reverse sequence* mutation performs significantly better than the *swap* mutation
- Different crossover operators give quite similar results for the same mutation operator.
- Running the algorithm for more iterations improved more balanced results, did not improve extreme results

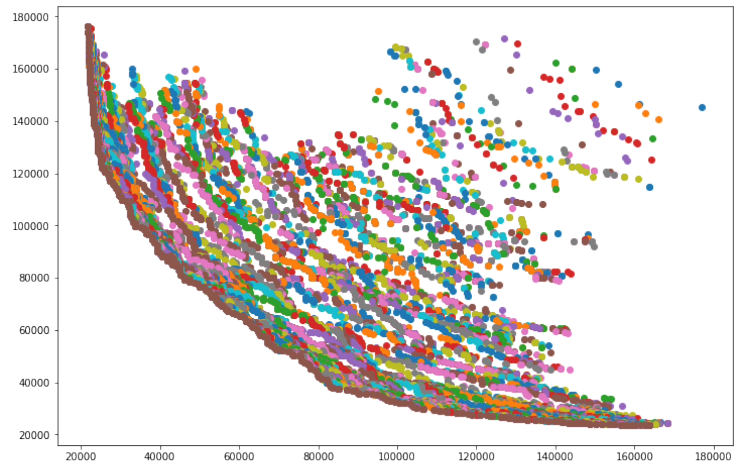
3 Multiple objective genetic local search (MOGLS)

3.1 Algorithm

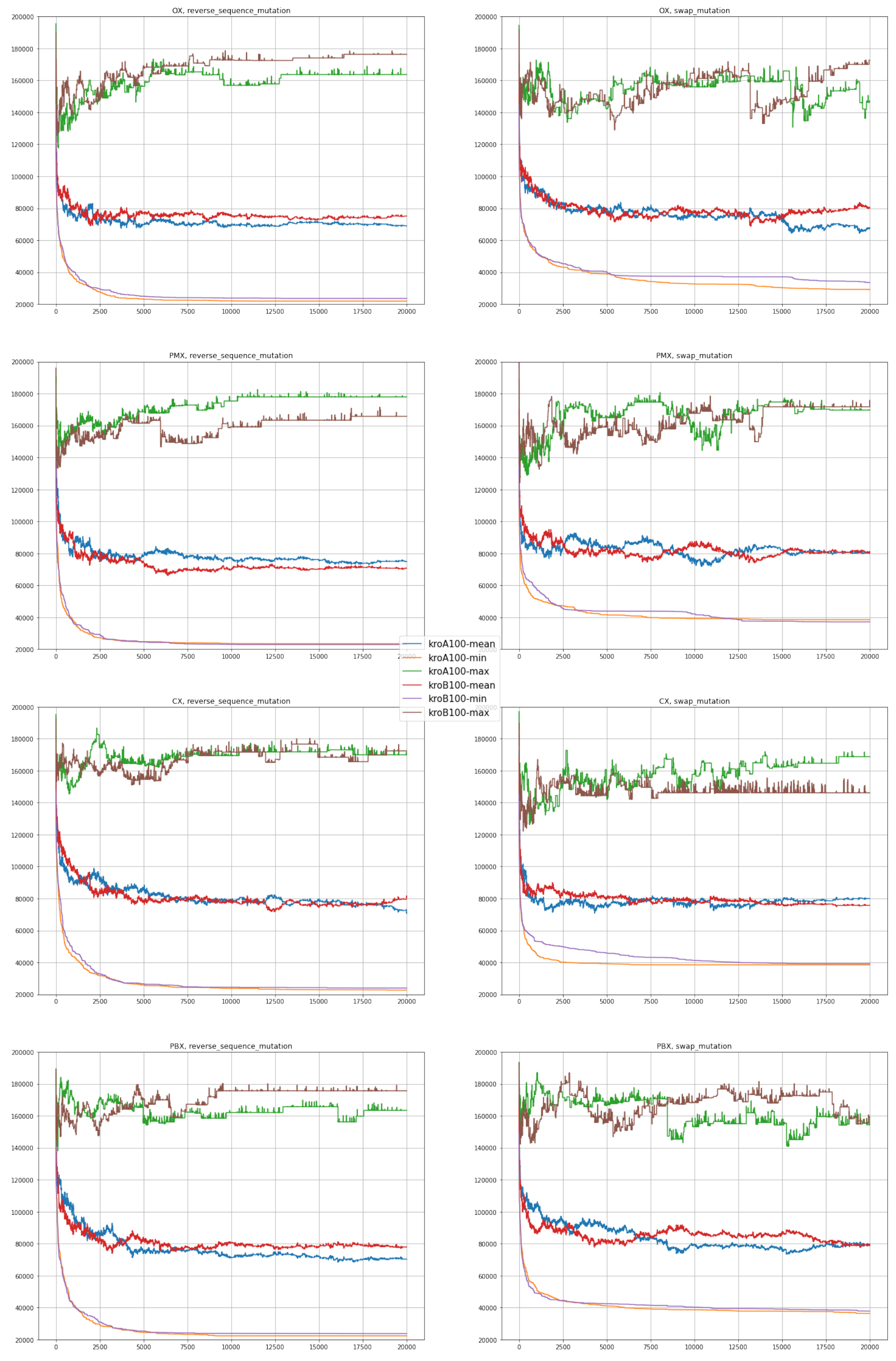
1. Create a random P population, optimize each individual using a local search and a random u weight function



Rysunek 1: Best (non-dominated) solutions for all NSGA-II variants, $(kroA100, kroB100)$



Rysunek 2: Non-dominated individuals in successive iterations of the NSGA-II algorithm, ($(kroA100, kroB100)$, OX, RSM)



Rysunek 3: Minimum, maximum, and mean population score values for successive NSGA-II iterations, $(kroA100, kroB100)$

2. In each iteration:

   2.1 Pick random weight function u

   2.2 Select K of the best individuals by the u function, creating a temporary TP population

   2.3 From the TP population, draw 2 individuals x_1 and x_2. Make them a descendant of x_3 and optimize it using local search and u to create x'_3

   2.4 If x'_3 is better than the worst person in P (according to the u function), add it to P

weight function

The u weight function assigns w_i to each criterion, for i from 1 to the number of criteria. The sum of the weights is always equal to 1. The result of the individual x for the function u is.

$$\sum_i w_i \cdot distance(x, C_i)$$

where C_i is ith criterion

Feature randomization algorithm.

- $w_1 = 1 - \sqrt[J-1]{rand()}$
- $w_i = (1 - \sum_{j=1}^{i-1} w_j)(1 - \sqrt[J-1-j]{rand()})$
- $w_J = (1 - \sum_{j=1}^{J-1} w_j)$

where J-number of criteria, $rand()$-random number from $(0, 1)$ range

Local Search

```
repeat until no improvement is made {
    best_distance = calculateTotalDistance(existing_route)
    start_again:
    for (i = 0; i <= N; i++) {
        for (k = i + 1; k <= N; k++) {
            new_route = 2optSwap(existing_route, i, k)
            new_distance = calculateTotalDistance(new_route)
            if (new_distance < best_distance) {
                existing_route = new_route
                best_distance = new_distance
                goto start_again
            }
        }
    }
}
```

```
procedure 2optSwap(route, i, k) {
    1. take route[0] to route[i-1] and add them in order to new_route
    2. take route[i] to route[k] and add them in reverse order to new_route
    3. take route[k+1] to end and add them in order to new_route
    return new_route;
}
```

Source: Wikipedia

3.2 Tests

The MOGLS implementation was launched for the $(kroA100, kroB100)$ instance, for each of the 4 crossover operators.

- number of individuals in the initial P population: 50
- number of iterations: 200
- TP population size: 15

The testing time was approximately 10 minutes for OX and PMX operators, and approximately 20 minutes for CX and PBX operators.

Results

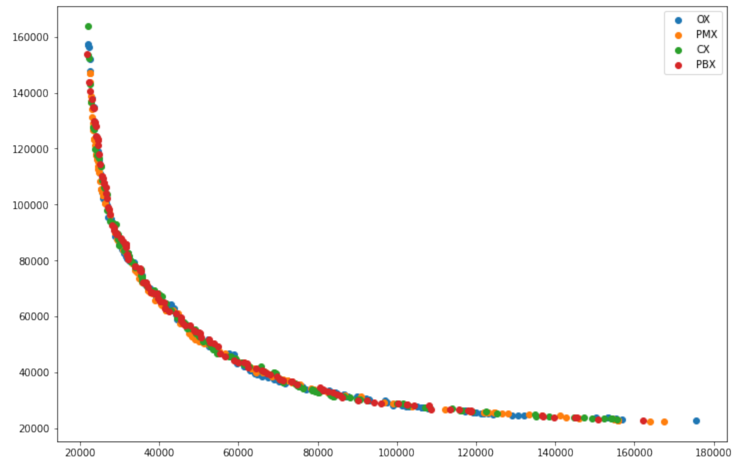
- The table 2 shows the lengths of the shortest cycles found (results in one row come from **different** solutions)
- Fig. **??** shows the results of the best individuals for each variant of the algorithm.

Observations

- The MOGLS algorithm, thanks to the local search, needed much less iterations to achieve results similar to NSGA-II, but the execution time of each iteration was many times longer. The MOGLS variant test performed in a similar time to the NSGA-II variant test
- Differences between the results for different variants of MOGLS are practically non-existent, which suggests that this search is primarily responsible for the efficiency of the algorithm.
- The choice of the crossover operator affects the running time of the algorithm. The inferior descendant will take longer to optimize with local search.

crossover	kroA100	kroB100
OX	22046.3	22949.7
PBX	21863.7	22910.7
CX	21991.2	23288.3
PMX	22454.7	22490.0

Tabela 2: Best scores found by the MOGLS algorithm



Rysunek 4: Best (non-dominated) solutions for all MOGLS variants

4 Summary

The implemented algorithms achieved good test results. The solutions found for NSGA-II in the version with the RSM and MOGLS mutations are only slightly worse than the optimal solutions. The results of both algorithms are similar, based on the tests it is impossible to clearly determine which algorithm is better. The runtimes of the NSGA-II and the faster MOGLS versions are also similar. Testing for other instances of the problem would need to be done. It is also worth implementing other algorithms in the future and comparing them with those described in this report.