Multi-objective Traveling Salesman Problem

Kamil Michalak

1 Opis projektu

1.1 Cel projektu

Celem projektu jest implementacja różnych algorytmów rozwiązujących *wielokryterialny problem komiwojażera* (*multi-objective TSP*), przetestowanie ich i porównanie efektywności.

1.2 Definicja problemu

Dane: N miast i p kosztów $c_{i,j}^k$ $(k = 1, ..., p)$ podróży z miasta i do j.

Cel: Znaleźć permutację ρ (złożoną z N miast), która minimalizuje funkcję

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

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

1.3 Wykorzystane dane

W testach wykorzystano instancje problemu z biblioteki TSPLIB (`http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`). Zaimplementowane algorytmy przetestowano na instancjach

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

Optymalne rozwiązania dla 1-kryterialnych instancji:

- *kro100A* – 21285.4
- *kro100B* – brak danych
- *kro100C* – 20750.7
- *kro100D* – 21294.2

1.4 Dane techniczne

- Język programowania: Python + kompilator Numba do przyspieszenia działania kodu
- System: Ubuntu 20.04
- Procesor: AMD Ryzen 7 4800H
- 32 GB pamięci RAM

2 NSGA-II

2.1 Szczegóły algorytmu

Operatory krzyżowania

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

Wybór rodziców do krzyżowania - Tournament selection

Losujemy K osobników z populacji i sortujemy je według rangi i crowding distance. Wybieramy najlepszego z prawdopodobieństwem p, drugiego z prawdopodobieństwem $p \cdot (1-p)$, trzeciego – $p \cdot (1-p)^2$ itd.

Operatory mutacji

- Reverse sequence mutation – odwrócenie losowego wycinka permutacji
- Swap mutation – zamiana 2 losowych elementów permutacji

2.2 Testy

Przetestowano algorytm na danych (*kroA100, kroB100*) i (*kroC100, kroD100*) dla każdej możliwej kombinacji krzyżowania i mutacji z wymienionych w poprzedniej sekcji.

- rozmiar populacji: 200
- liczba iteracji: 20000
- w selekcji $K = 20$, $p = 0.9$

Czas wykonania każdego z testów wyniósł ok. 8 minut.

krzyżowanie	mutacja	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

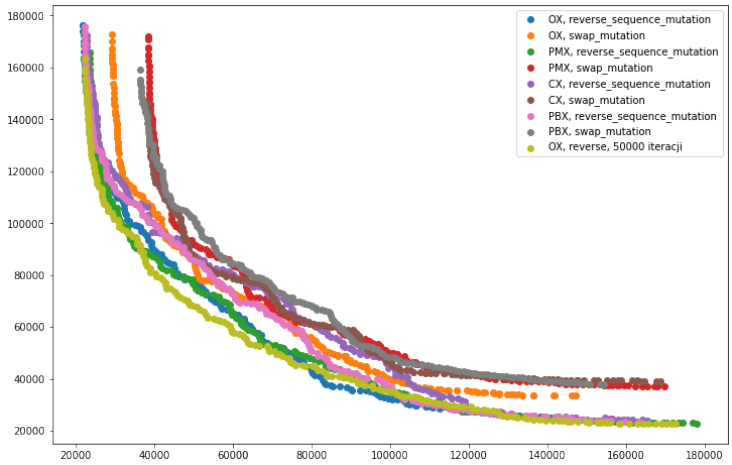
Tabela 1: Najmniejsze wyniki znalezione przez algorytm NSGA-II, dla 20000 iteracji

Wyniki

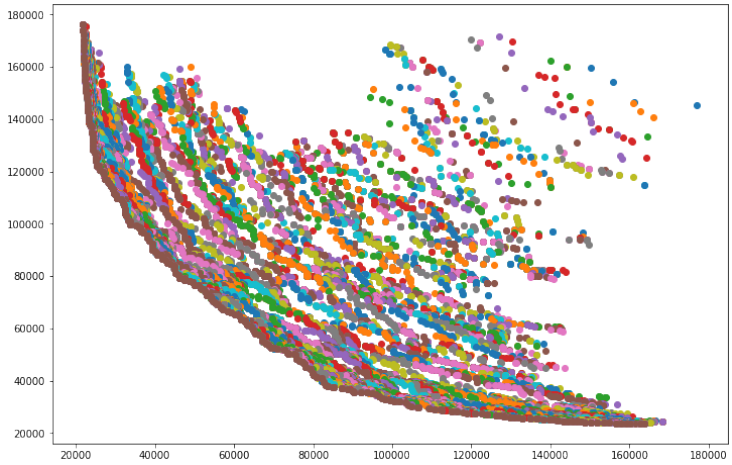
- Na rys. 3 przedstawiono wykresy minimalnych, maksymalnych i średnich długości cykli w populacji, dla kolejnych iteracji algorytmu
- Na rys. 1 przedstawiono wyniki najlepszych rozwiązań dla wszystkich wariantów algorytmu dla 20000 iteracji, oraz jednego (OX + reverse sequence) dla 50000
- Na rys 2 przedstawiono niezdominowane osobniki dla kolejnych (nie wszystkich) iteracji algorytmu w dla krzyżowania order-1 i mutacji reverse sequence
- W tabeli 1 przedstawiono długości najkrótszych znalezionych cykli (wyniki w jednym wierszu pochodzą z **różnych** rozwiązań)

Obserwacje

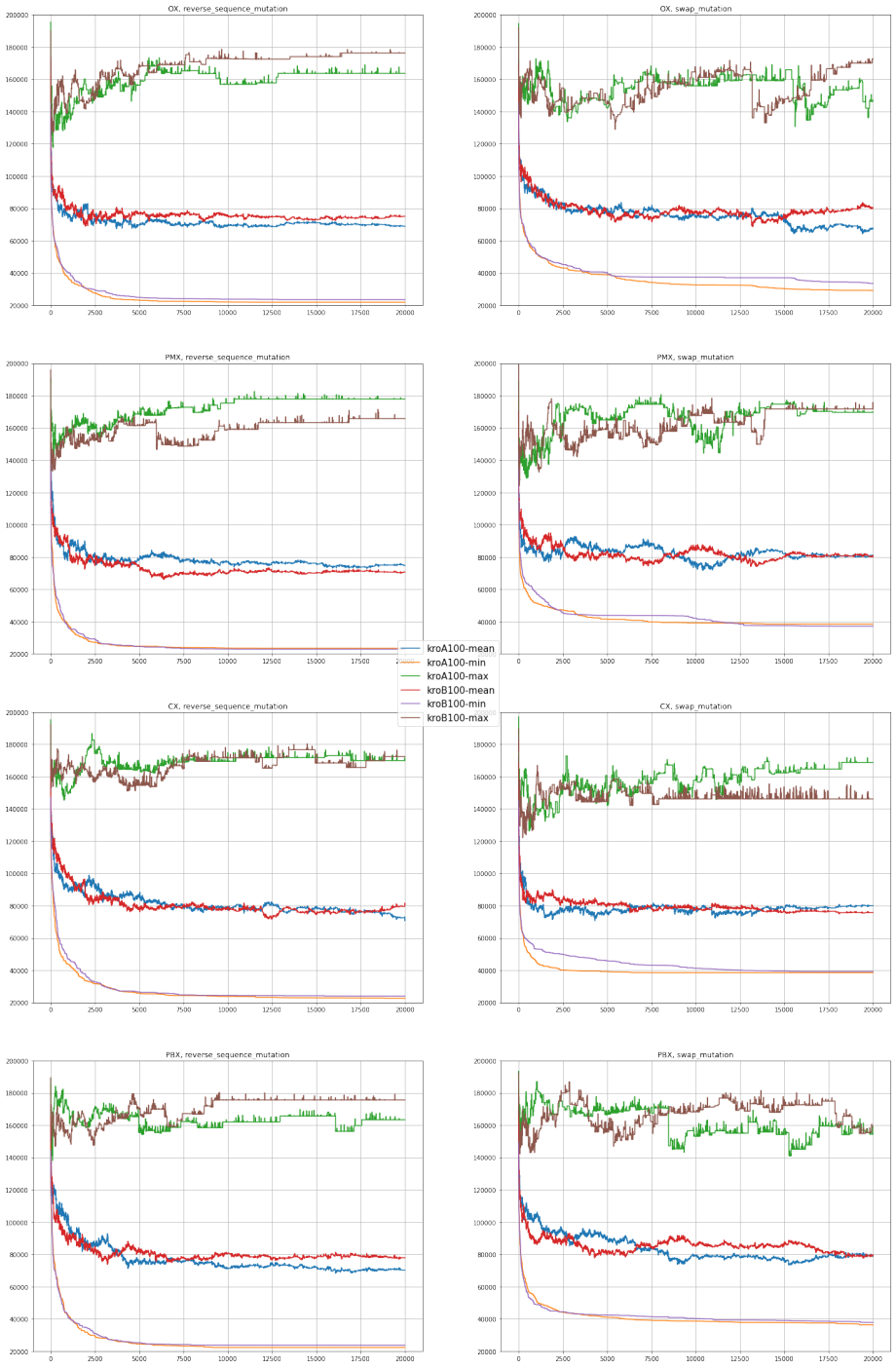
- Algorytm osiągnął dobre wyniki, w najlepszych wariantach zbliżył się do optymalnych rozwiązań
- Mutacja *reverse sequence* daje istotnie lepsze wyniki od mutacji *swap*
- Różne operatory krzyżowania dają dosyć zbliżone wyniki dla tego samego operatora mutacji.
- Uruchomienie algorytmu dla większej liczby iteracji poprawiło bardziej zbalansowane wyniki, nie poprawiło wyników skrajnych



Rysunek 1: Najlepsze (niezdominowane) rozwiązania dla wszystkich wariantów NSGA-II, $(kroA100, kroB100)$



Rysunek 2: Niezdominowane osobniki w kolejnych iteracjach algorytmu NSGA-II, $((kroA100, kroB100)$, OX, RSM)



Rysunek 3: Minimalne, maksymalne i średnie wartości wyników populacji dla kolejnych iteracji NSGA-II, $(kroA100, kroB100)$

3 Multiple objective genetic local search (MOGLS)

3.1 Algorytm

1. Utwórz losową populację P, każdego z osobników zoptymalizuj za pomocą przeszukiwania lokalnego i losowej funkcji wagi u
2. W każdej iteracji:
   - 2.1 Wylosuj funkcję wagi u
   - 2.2 Wybierz K najlepszych osobników według funcji u, tworząc tymczasową populację TP
   - 2.3 Z populacji TP wylosuj 2 osobniki x_1 i x_2. Utwórz z nich potomka x_3 i zoptymalizuj go używając przeszukiwania lokalnego i funkcji u, tworząc x'_3
   - 2.4 Jeśli x'_3 jest lepszy od najgorszego osobnika z P (według funkcji u), dodaj go do P

Funkcja wagi

Funkcja wagi u przyporządkowuje każdemu z kryterium wagę w_i, dla i od 1 do liczby kryteriów. Suma wag jest zawsze równa 1. Wynik osobnika x, dla funkcji u to.

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

gdzie C_i to i-te kryterium

Algorytm losowania funkcji.

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

gdzie J-liczba kryteriów, $rand()$-losowa liczba z przedziału $(0, 1)$

Przeszukiwanie lokalne

```
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;
}
```

Źródło: Wikipedia

3.2 Testy

Implementację MOGLS uruchomiono dla instancji ($kroA100, kroB100$), dla każdego z 4 operatorów krzyżowania.

- liczba osobników w początkowej populacji P: 50
- liczba iteracji: 200
- rozmiar populacji TP: 15

Czas wykonania testów wyniósł ok. 10 minut dla operatorów OX i PMX, oraz ok. 20 minut dla operatorów CX i PBX.

Wyniki

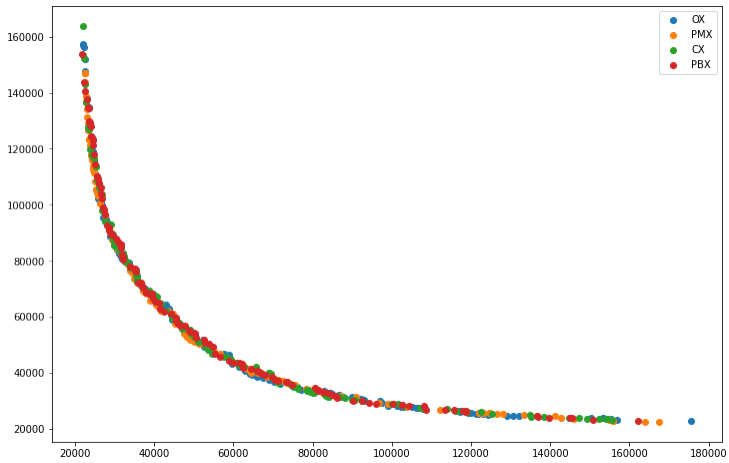
- W tabeli 2 przedstawiono długości najkrótszych znalezionych cykli (wyniki w jednym wierszu pochodzą z **różnych** rozwiązań)
- Na rys. 4 przedstawiono wyniki najlepszych osobników dla każdego z wariantów algorytmu.

Obserwacje

- Algorytm MOGLS, dzięki przeszukiwaniu lokalnemu potrzebował o wiele mniej iteracji, żeby osiągnąć wyniki podobne do NSGA-II, ale czas wykonania każdej iteracji był wielokrotnie dłuższy. Test wariantu MOGLS wykonał się w podobnym czasie do testu wariantu NSGA-II
- Różnice pomiędzy wynikami dla różnych wariantów MOGLS praktycznie nie występują, co sugeruje, że to przeszukiwanie odpowiada przede wszystkim za efektywność algorytmu.

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

Tabela 2: Najmniejsze wyniki znalezione przez algorytm MOGLS



Rysunek 4: Najlepsze (niezdominowane) rozwiązania dla wszystkich wariantów MOGLS

- Wybór operatora krzyżowania ma wpływ na czas działania algorytmu. Gorszy potomek będzie się dłużej optymalizował przeszukiwaniem lokalnym.

4 Podsumowanie

Zaimplementowane algorytmy osiągnęły dobre wyniki w testach. Rozwiązania znalezione NSGA-II w wersji z mutacją RSM i MOGLS są tylko nieznacznie gorsze od rozwiązań optymalnych. Wyniki obu algorytmów są zbliżone, na podstawie przeprowadzonych testów nie da się jednoznacznie stwierdzić, który algorytm jest lepszy. Czasy działania NSGA-II i szybszych wersji MOGLS również są podobne. Należałoby przeprowadzić testy dla innych instancji problemu. Warto również zaimplementować w przyszłości inne algorytmy i porównać je z opisanymi w tym raporcie.