Radosław Matuszczyk
Inżynieria i Analiza Danych, AGH
Inteligencja obliczeniowa w analizie danych

Implementacja algorytmu genetycznego dla problemu NP-trudnego

Znalezienie dokładnego rozwiązania niektórych problemów optymalizacyjnych jest trudne obliczeniowo i wymaga czasu, który znacząco rośnie wraz z wielkością danych wejściowych. Problemy, dla których nie możemy znaleźć dokładnego rozwiązania w czasie wielomianowym określamy mianem problemów NP-trudnych. Jednym z podejść do rozwiązywania problemów NP-trudnych jest stosowanie algorytmów heurystycznych, które pozwalają na znalezienie przybliżonego rozwiązania w rozsądnym czasie. Przykładem takich algorytmów są algorytmy genetyczne.

Algorytm genetyczny może posłużyć do rozwiązania wybranego problemu NP-trudnego, np. problemu komiwojażera. Problem komiwojażera polega na znalezieniu najkrótszej drogi pomiędzy N miastami, która zaczyna się i kończy w tym samym mieście, a każde miasto jest odwiedzone dokładnie raz. Jego złożoność obliczeniowa wynosi $N!$, dlatego możemy znaleźć rozwiązanie jedynie dla małych instancji.

Algorytm genetyczny działa na populacji chromosomów, gdzie każdy chromosom reprezentuje potencjalne rozwiązanie. W przypadku problemu komiwojażera, chromosom może być interpretowany jako lista miast, które określają kolejność odwiedzin. Następnie, w każdej iteracji algorytmu, populacja jest poddawana operacjom selekcji, krzyżowania i mutacji, które prowadzą do generowania nowych potomków potencjalnie lepiej przystosowanych do rozwiązania problemu niż ich rodzice.

W implementacji algorytmu genetycznego dla problemu komiwojażera, użyłem dwóch selekcji – rankingowej i ruletkowej. Selekcja rankingowa polega na sortowaniu osobników w populacji według ich przystosowania (długości trasy), a następnie wybieraniu najlepszych osobników na podstawie pozycji w rankingu. Selekcja ruletkowa polega z kolei na losowym wyborze osobników z populacji z prawdopodobieństwem proporcjonalnym do ich przystosowania. W obu selekcjach prawdopodobieństwo wybrania osobnika do rozrodu jest równe od 0 do 1 i jest tym większe im lepsze jest przystosowanie danego osobnika.

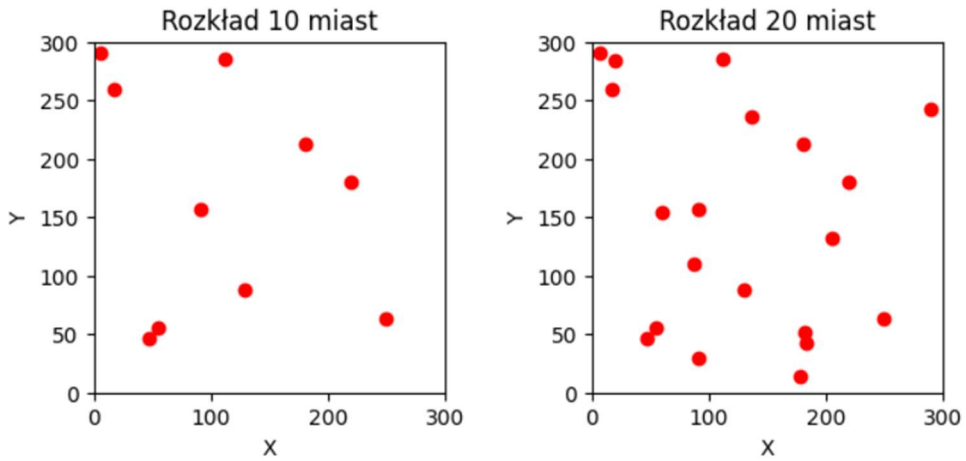
Po selekcji dwóch osobników następowało krzyżowanie ich genotypów w celu generacji potomków, którzy zachowują częściową kolejność miast z obu rodziców. Aby zapewnić różnorodność populacji i zmniejszyć ryzyko utknięcia w minimum lokalnym, zastosowałem również operator mutacji, który zamieniał dwa losowo wybrane miasta w chromosomie.

Po każdej iteracji algorytmu, oceniałem populację na podstawie funkcji celu, która mierzyła sumaryczną długość trasy. Następnie, wybierałem najlepszego osobnika, czyli chromosom o najniższej wartości funkcji celu, i porównywałem ją z najlepszym rozwiązaniem znalezionym do tej pory. Jeśli wartość funkcji celu dla najlepszego osobnika była lepsza niż dla dotychczas najlepszego rozwiązania, to aktualizowałem najlepsze rozwiązanie. Nowe pokolenie stawało się nową populacją, przy czym, aby nie stracić najlepszego dotychczas znalezionego rozwiązania kilku najlepszych rodziców przeżywało.

Algorytm przewiduje nastawy poniższych parametrów (w nawiasie podano wartości domyślne):

- liczba miast,
- rozmiar mapy,
- liczebność populacji,
- prawdopodobieństwo krzyżowania (0.99),
- prawdopodobieństwo mutacji (0.05),
- metoda selekcji (rankingowa),
- ziarno generatora do losowania miast lub współrzędne miast,
- współczynnik elitaryzmu (0.02).

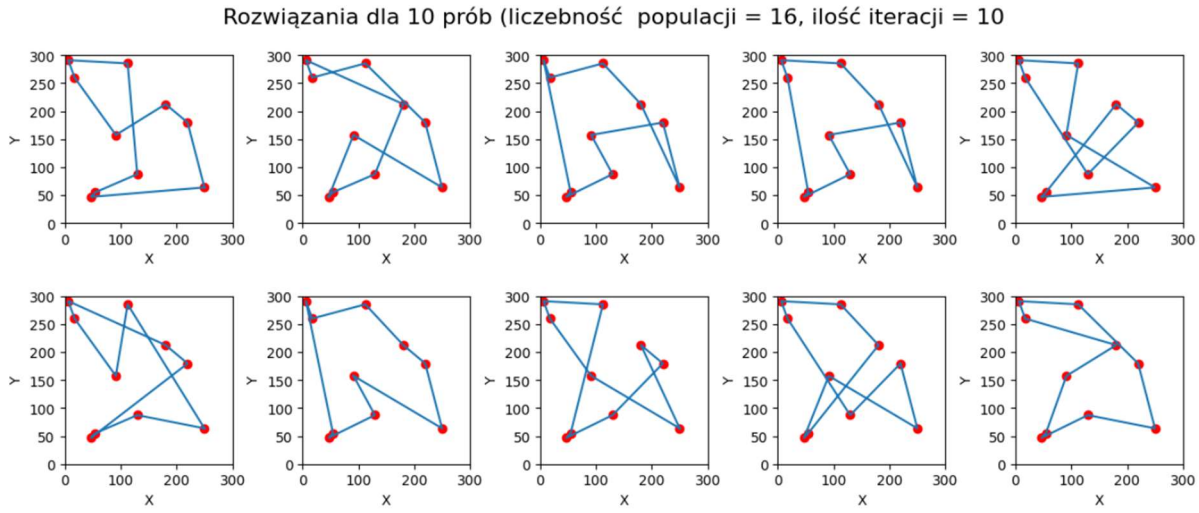
Na wykresie 1. przedstawiono dwa przykładowe rozkłady miast dla $N \in \{10, 20\}$, na których będziemy testować wyniki.



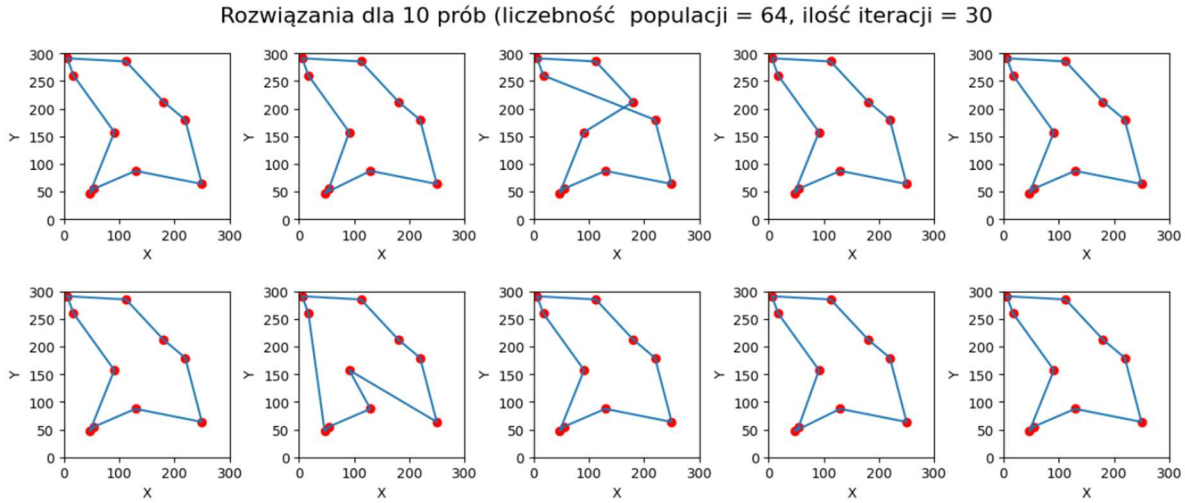
Wykres 1. Przykładowe rozkłady miast

Z powodu losowej natury algorytmu genetycznego każda próba rozwiązania problemu komiwojażera może kończyć różnym skutkiem. Nie mamy także możliwości oceny jakości uzyskanego wyniku. Możemy jedynie badać rozwiązania uzyskane w wyniku wielokrotnego uruchamiania naszej implementacji.

Wykres 2. oraz wykres 3. obrazują niestabilność uzyskiwanych rozwiązań. Ustawiając lepsze parametry metody zwiększamy szansę uzyskania dokładniejszego wyniku. Nie mamy natomiast żadnej gwarancji otrzymania prawidłowego wyniku.

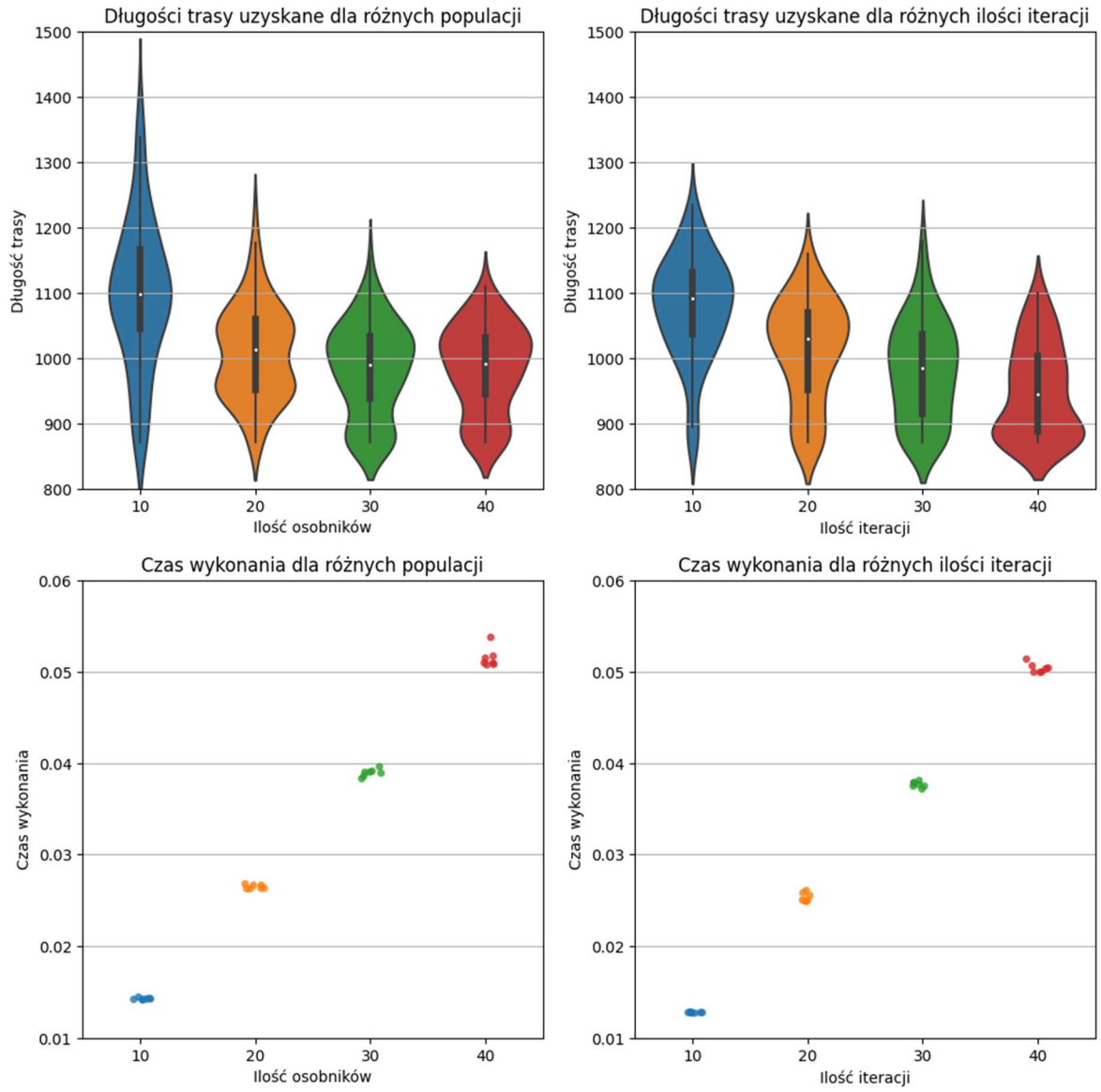


Wykres 2.

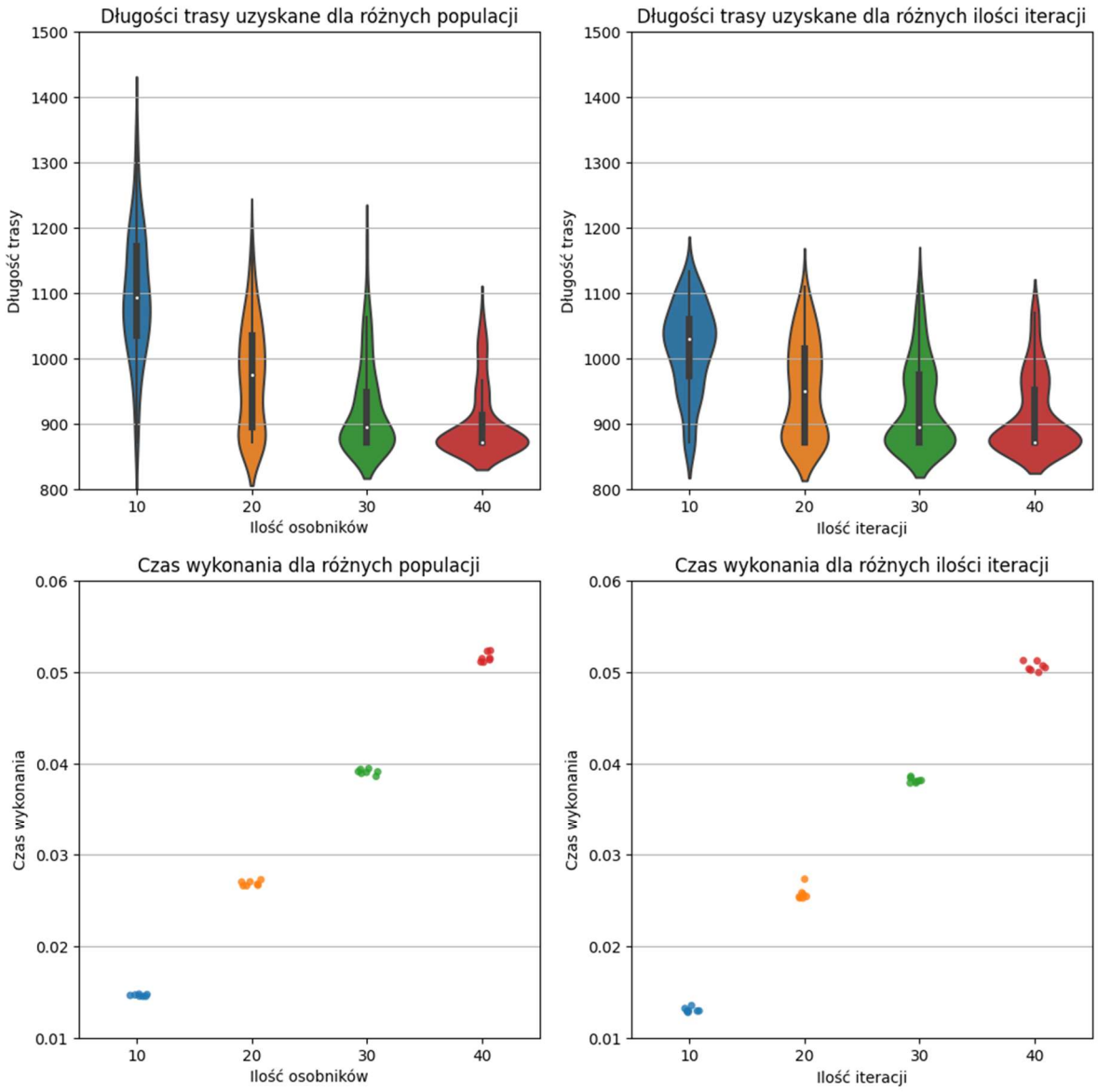


Wykres 3.

Możemy zaobserwować, że zwiększenie liczebności populacji i ilości iteracji spowodowało ustabilizowanie wyników. Sprawdźmy jak zmienia się długość podróży i czas wykonania algorytmu w zależności od liczebności populacji, ilości iteracji i metody selekcji dla 10 miast. Badając liczebność populacji ustawiłem ilość iteracji na 25. Podobnie badając ilość iteracji ustawiłem liczebność populacji na 25. Wyniki prezentują wykresy 4. i 5.



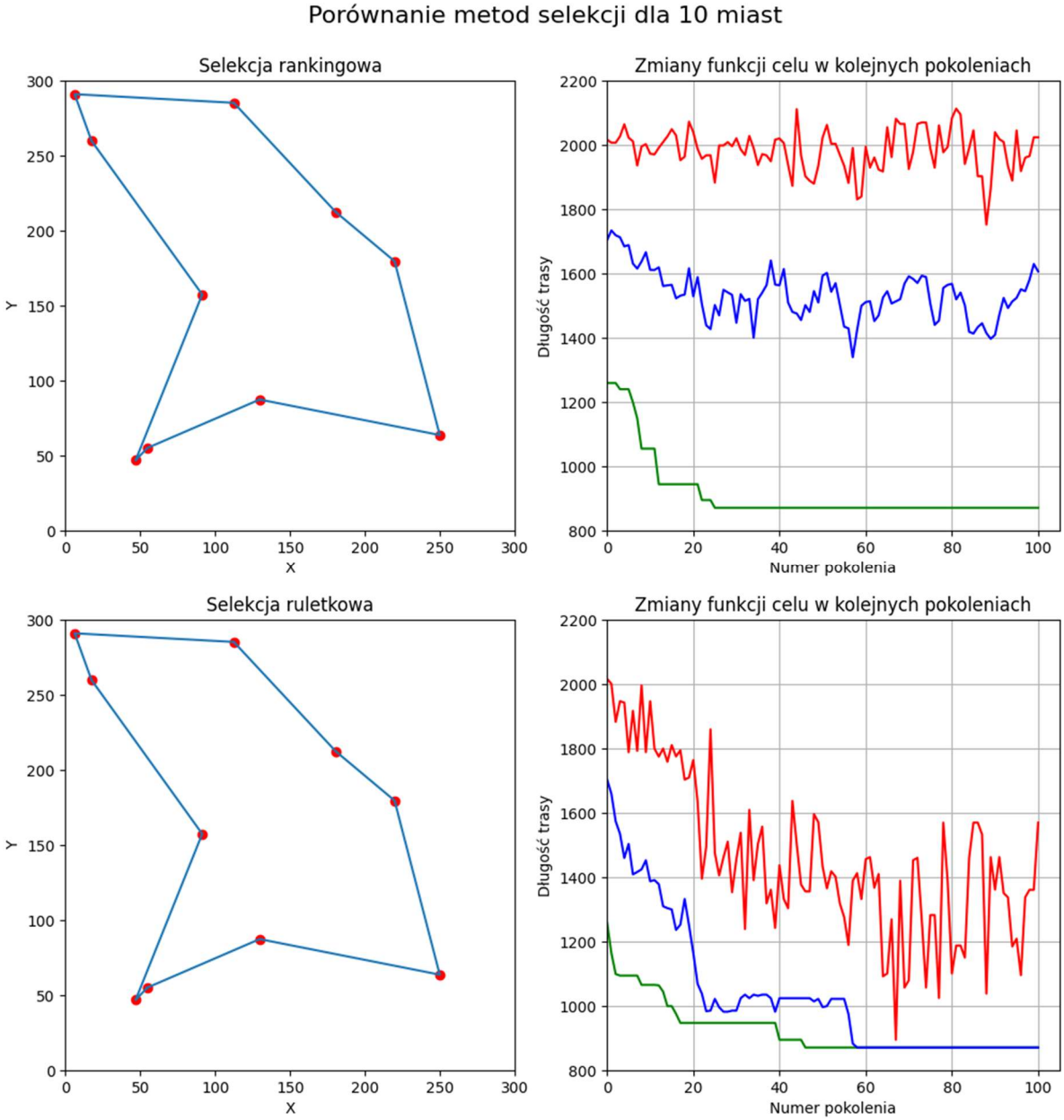
Wykres 4. Wpływ parametrów na jakość i czas rozwiązania dla metody rankingowej



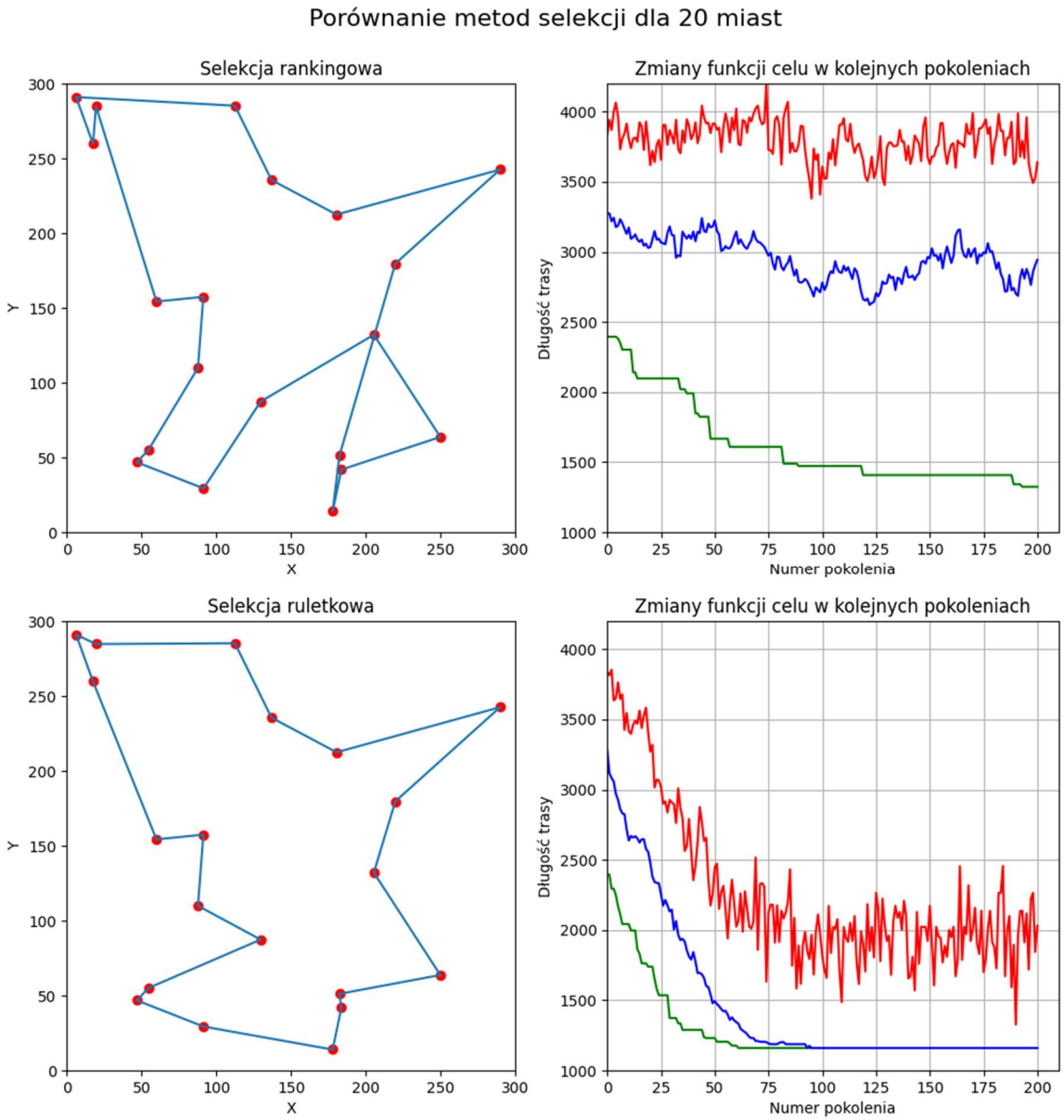
Wykres 5. Wpływ parametrów na jakość i czas rozwiązania dla metody ruletkowej

W przypadku obu metod czas wykonania algorytmu zwiększa się proporcjonalnie wraz ze wzrostem liczebności populacji oraz ilości iteracji, natomiast poprawienie rezultatów widoczne jest bardziej w przypadku wzrostu liczebności populacji. Wnioskujemy zatem, że w celu poszukiwania lepszego rozwiązania, bardziej opłaca się zwiększać liczebność populacji.

Czas wykonania algorytmu dla obu metod jest identyczny, jednakże selekcja ruletkowa daje lepsze rezultaty od selekcji rankingowej. Porównajmy dokładniej dwie zaimplementowane metody. Pomiarów dokonamy dla 10 miast i populacji 64 osobników oraz 20 miast i 128 osobników.


Porównanie metod selekcji dla 10 miast
Selekcja rankingowa
Zmiany funkcji celu w kolejnych pokoleniach
Selekcja ruletkowa
Zmiany funkcji celu w kolejnych pokoleniach
X
Y
Długość trasy
Numer pokolenia

Wykres 6.



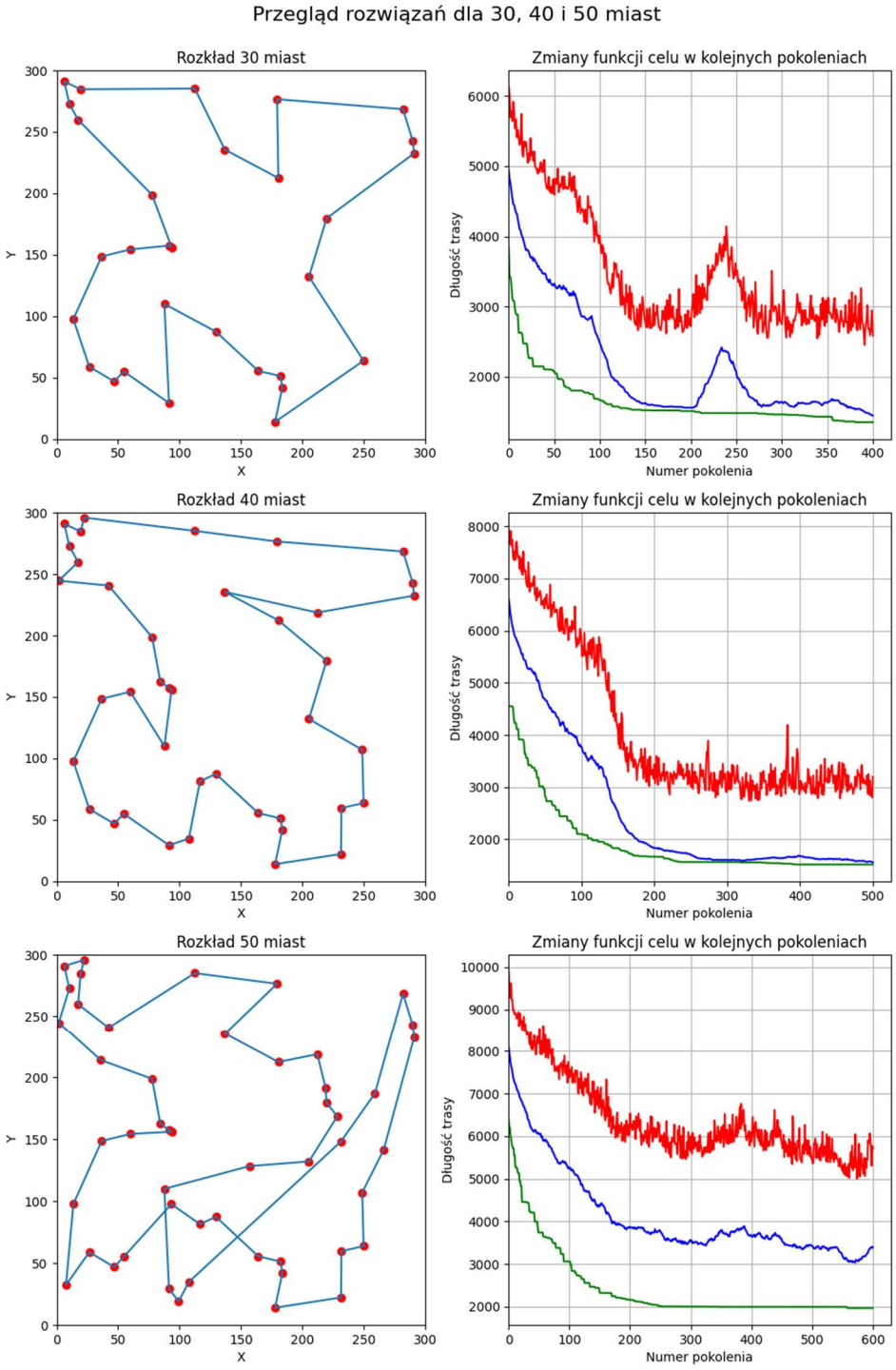
Wykres 7.

Metoda ruletkowa zdaje się dawać lepsze rezultaty. Osiąga szybszą konwergencję, w wyniku czego daje lepsze rezultaty dla tej samej ilości iteracji. Pamiętajmy, że nie jest to regułą – mimo zazwyczaj lepszych wyników selekcji ruletkowej, może zdarzyć się, że uzyskamy lepszy rezultat dla metody rankingowej.

Uzbrojeni w wiedzę o dostrajanych parametrów spróbujmy znaleźć rozwiązanie dla 30, 40 i 50 miast. Jako metodę selekcji wybrałem metodę ruletkową. Liczbę miast, liczebność populacji i ilość iteracji ustawiłem zgodnie z Tabelą 1. Dla pozostałych parametrów zostawiłem wartości domyślne.

liczba miast	liczebność populacji	ilość iteracji
30	1024	400
40	2048	500
50	4096	600

Tabela 1.



Wykres 8.

Rozwiązania dla 20 i 30 miast wizualnie wydają się być dobre, nie mamy natomiast żadnej obiektywnej miary poprawności, która niepodważalnie osądziłaby o skuteczności algorytmu. W przeciwieństwie do powyższych, rozwiązanie dla 50 miast mogłoby być lepsze – występuje krzyżowanie trasy. Algorytm utknął w lokalnym minimum, z którego nie był w stanie się wydostać przez paręset iteracji. Aby zwiększyć szansę uzyskania lepszego rezultatu należałoby zwiększyć liczebność populacji. Czas poszukiwania ostatniego rozwiązania wyniósł około 11 min, co jest fenomenalnym wynikiem w porównaniu z czasem, który byłby potrzebny w celu poszukiwania idealnego rozwiązania.

Podsumowując, algorytmy genetycznie dobrze sprawdzają się przy znajdywaniu przybliżonego rozwiązania problemu NP-trudnego, w szczególności umożliwiają znalezienie rozwiązania problemu, dla którego przeszukiwanie całej przestrzeni rozwiązań jest niemożliwe z uwagi na długi czas wykonywania. Problem komiwojażera, którego złożoność obliczeniowa wynosi *N!* może zostać przybliżony za pomocą liniowego algorytmu genetycznego. Należy mieć na uwadze, że nie mamy gwarancji, że uzyskane rozwiązanie jest najlepszym możliwym rozwiązaniem, a także, że dobranie lepszych parametrów zawsze dostarczy lepszy rezultat.