Metaheurystyki — zadanie 3

Algorytm genetyczny

GRUPA 3 — piątek 10:15

Bartosz Kołaciński
251554

Nikodem Nowak
251598

27 listopada 2025

Użyte technologie	Python 3.13
Użyte biblioteki	random, functools, time, numpy, pandas, matplotlib.pyplot

Spis treści

1 Opis zasad działania algorytmu **3**
1.1 Opis problemu plecakowego 3
1.2 Opis algorytmu genetycznego 3
1.3 Opis implementacji rozwiązania 3
1.4 Instrukcja uruchomienia programu 4

2 Eksperymenty i wyniki **5**
2.1 Analiza wpływu parametrów algorytmu 5
2.1.1 Wpływ wielkości populacji 5
2.1.2 Wpływ prawdopodobieństwa krzyżowania 5
2.1.3 Wpływ prawdopodobieństwa mutacji 6
2.2 Porównanie strategii selekcji 7
2.2.1 Selekcja Ruletkowa 8
2.2.2 Selekcja Turniejowa 10

3 Wnioski **12**

1 Opis zasad działania algorytmu

1.1 Opis problemu plecakowego

Problem plecakowy (ang. *Knapsack Problem*) to jeden z klasycznych problemów optymalizacji kombinatorycznej. Dany jest zbiór przedmiotów, z których każdy ma określoną wagę oraz wartość. Celem jest wybór takiego podzbioru przedmiotów, aby suma ich wartości była jak największa, jednocześnie nie przekraczając żądanej maksymalnej wagi plecaka.

1.2 Opis algorytmu genetycznego

Algorytm genetyczny to metaheurystyka inspirowana biologicznym procesem ewolucji naturalnej. Działa on na populacji osobników, z których każdy reprezentuje potencjalne rozwiązanie problemu. Proces optymalizacji przebiega w cyklach zwanych epokami (lub generacjami), w których następują:

- **Selekcja**: Wybór najlepiej przystosowanych osobników do reprodukcji.
- **Krzyżowanie**: Łączenie cech dwóch osobników (rodziców) w celu stworzenia potomstwa.
- **Mutacja**: Losowa zmiana w kodzie genetycznym potomstwa, wprowadzająca różnorodność.
- **Ocena (Fitness)**: Obliczenie wartości funkcji przystosowania dla każdego osobnika.

Algorytm dąży do znalezienia osobnika o najwyższej wartości funkcji przystosowania.

1.3 Opis implementacji rozwiązania

Logika algorytmu genetycznego została zaimplementowana w klasie `GeneticAlgorithm`. Rozwiązanie (osobnik) jest reprezentowane jako lista liczb całkowitych (0 lub 1), gdzie 1 oznacza zabranie przedmiotu, a 0 jego pozostawienie. Długość listy odpowiada liczbie wszystkich dostępnych przedmiotów.

```
    class GeneticAlgorithm:
        MAX_WEIGHT = 6404180

        def __init__(self, selection_strategy, cross_strategy,
                     mutation_strategy, items: list[dict]):
            self.selection_strategy = selection_strategy
            self.cross_strategy = cross_strategy
            self.mutation_strategy = mutation_strategy
            self.items = items
            self.weights = self.items["Waga"].values
            self.values = self.items["Wartosc"].values

```

Kod 1: Inicjalizacja klasy GeneticAlgorithm

Funkcja przystosowania (`calculate_fitness_indiv`) oblicza sumaryczną wartość przedmiotów w plecaku. Jeśli waga przedmiotów przekracza `MAX_WEIGHT`, funkcja zwraca 0, co eliminuje niepoprawne rozwiązania z procesu ewolucji.

```
    @cache
    def calculate_fitness_indiv(self, individual) -> float:
        total_weight = sum(
            gene * weight for gene, weight in zip(individual, self.weights)
        )
        total_value = sum(gene * value for gene, value in
                          zip(individual, self.values))

        if total_weight > self.MAX_WEIGHT:
            return 0.0
        else:
            return int(total_value)

```

Kod 2: Obliczanie funkcji przystosowania

Główna pętla algorytmu (`run`) wykonuje zadaną liczbę iteracji. W każdej iteracji następuje ocena populacji, wybór rodziców, krzyżowanie i mutacja. Algorytm śledzi najlepsze znalezione rozwiązanie.

Zaimplementowano dwie strategie selekcji:

- **Turniejowa** (`TournamentSelection`): Losuje grupę osobników i wybiera najlepszego z nich.
- **Ruletkowa** (`RouletteWheelSelection`): Prawdopodobieństwo wyboru osobnika jest proporcjonalne do jego funkcji przystosowania.

Zaimplementowano dwie strategie krzyżowania:

- **Jednopunktowe** (`OnePointCrossover`): Losuje punkt podziału i wymienia fragmenty rodziców.
- **Dwupunktowe** (`TwoPointCrossover`): Losuje dwa punkty i wymienia fragment pomiędzy nimi.

Jako strategię mutacji zastosowano `BitFlipMutation`, która z zadanym prawdopodobieństwem odwraca bit (0 na 1 lub 1 na 0) w genotypie osobnika.

1.4 Instrukcja uruchomienia programu

Program uruchamia się poprzez wywołanie pliku `main.py`, np. komendą:

```
python main.py
```

Program jest interaktywny i prosi użytkownika o:

1. Wybór strategii selekcji (Turniejowa/Ruletkowa).
2. Wybór strategii krzyżowania (Jednopunktowe/Dwupunktowe).
3. Podanie parametrów: wielkość populacji, liczba iteracji, prawdopodobieństwo krzyżowania, prawdopodobieństwo mutacji.

Po zakończeniu działania wyświetlane są: najlepsze znalezione rozwiązanie (wartość, waga, lista przedmiotów) oraz statystyki przebiegu.

W celu wygenerowania wszystkich eksperymentów opisanych w dalszej części sprawozdania, należy uruchomić skrypt:

```
python run_experiments.py
```

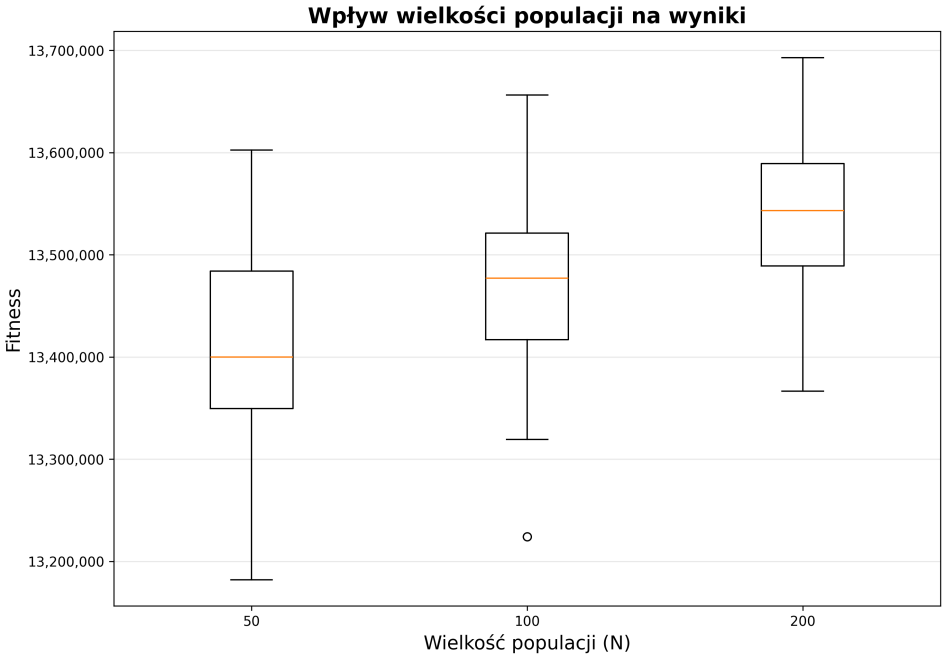
2 Eksperymenty i wyniki

Przeprowadzono serię eksperymentów w celu zbadania wpływu parametrów algorytmu na jakość znajdowanych rozwiązań. Dla każdej konfiguracji algorytm był uruchamiany wielokrotnie w celu uśrednienia wyników.

2.1 Analiza wpływu parametrów algorytmu

2.1.1 Wpływ wielkości populacji

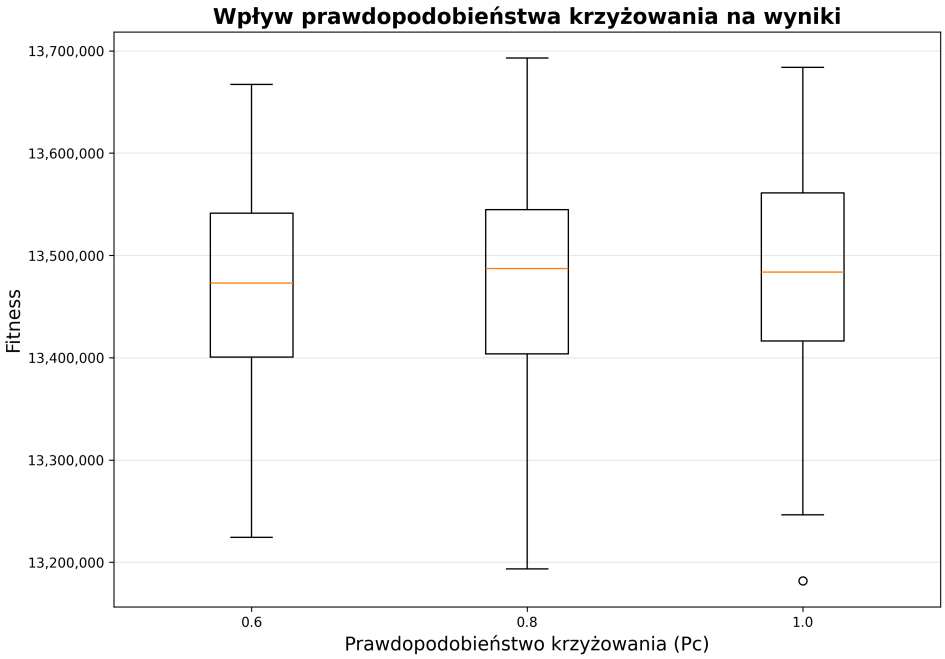
Wielkość populacji jest kluczowym parametrem wpływającym na różnorodność genetyczną. Z przeprowadzonych testów (Rys. 1) wynika, że większa populacja zazwyczaj prowadzi do lepszych wyników końcowych (wyższa wartość funkcji przystosowania), ponieważ algorytm przeszukuje większy obszar przestrzeni rozwiązań. Wiąże się to jednak z dłuższym czasem obliczeń.



Rysunek 1: Wpływ wielkości populacji na jakość rozwiązania.

2.1.2 Wpływ prawdopodobieństwa krzyżowania

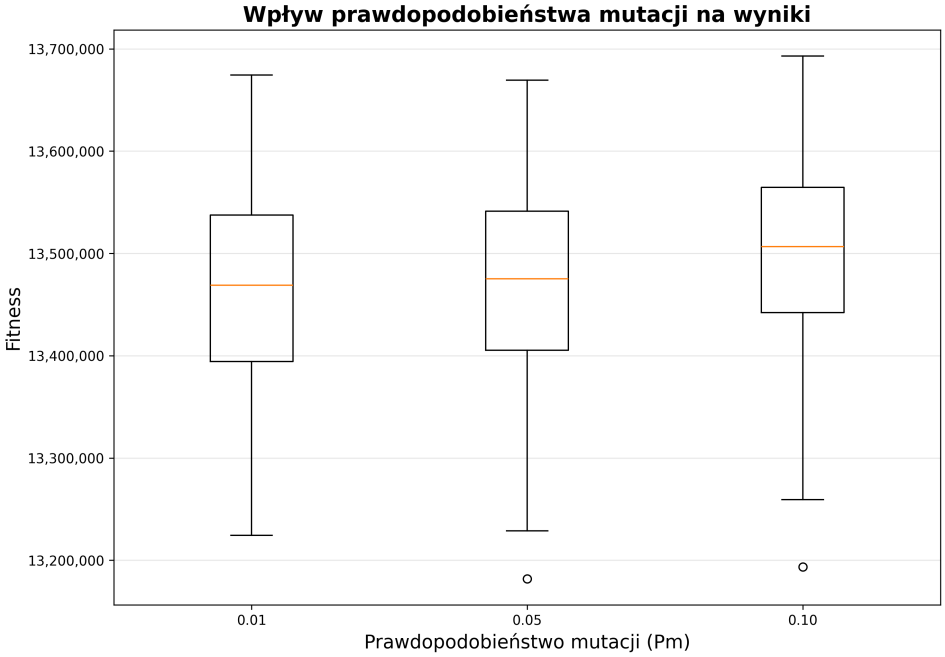
Krzyżowanie pozwala na łączenie dobrych cech rodziców. Wykres (Rys. 2) pokazuje zależność między prawdopodobieństwem krzyżowania a jakością wyniku. W naszym przypadku różnica w wynikach dla różnych wartości prawdopodobieństwa krzyżowania jest niewielka, nie można mówić o żadnej zależności.



Rysunek 2: Wpływ prawdopodobieństwa krzyżowania na jakość rozwiązania.

2.1.3 Wpływ prawdopodobieństwa mutacji

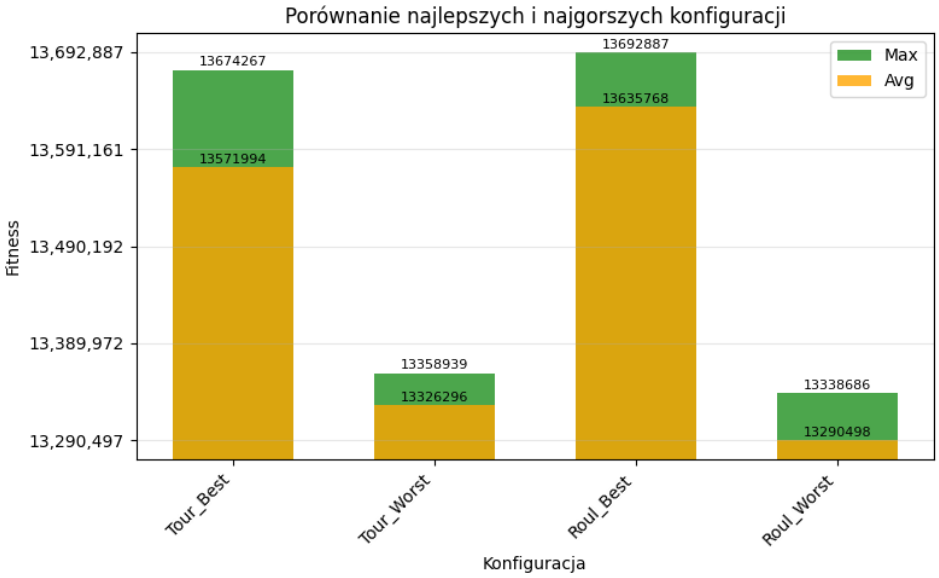
Mutacja wprowadza nowe geny do populacji, zapobiegając przedwczesnej zbieżności. Eksperymenty (Rys. 3) wskazują, że prawdopodobieństwo mutacji nie ma dużego wpływu na jakość rozwiązania. Najlepsze wartości osiągnięto dla prawdopodobieństwa mutacji 0.1, jednak różnice są minimalne.



Rysunek 3: Wpływ prawdopodobieństwa mutacji na jakość rozwiązania.

2.2 Porównanie strategii selekcji

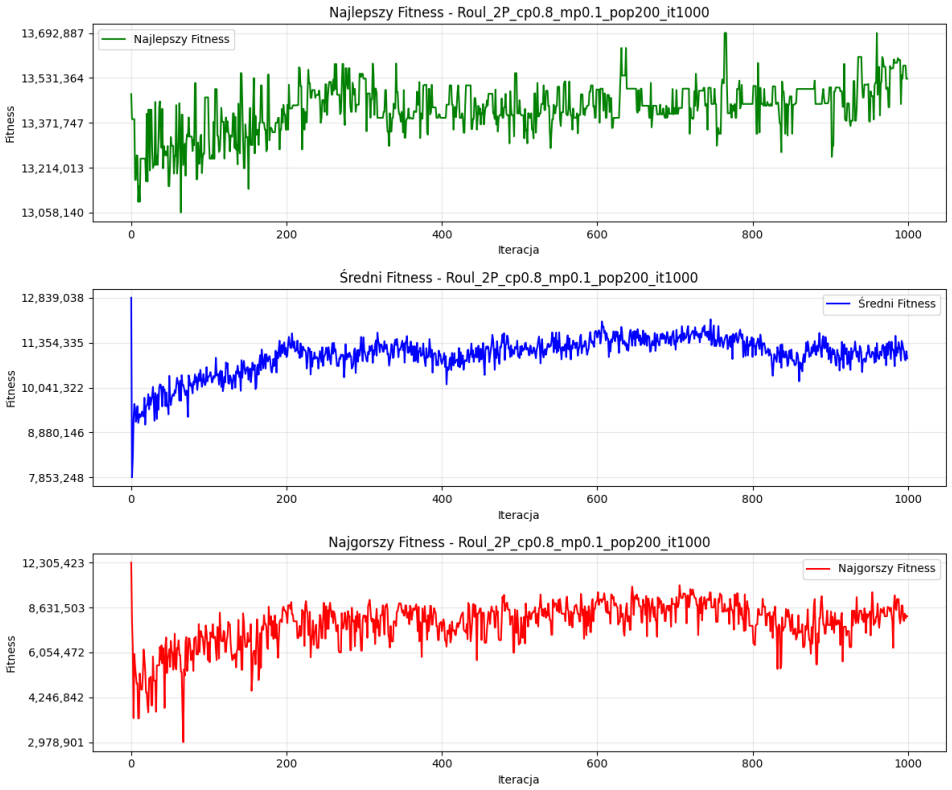
Porównano skuteczność dwóch strategii selekcji: ruletkowej oraz turniejowej. Rysunek 4 przedstawia zestawienie wyników (najlepszych i najgorszych z serii uruchomień) dla obu metod.



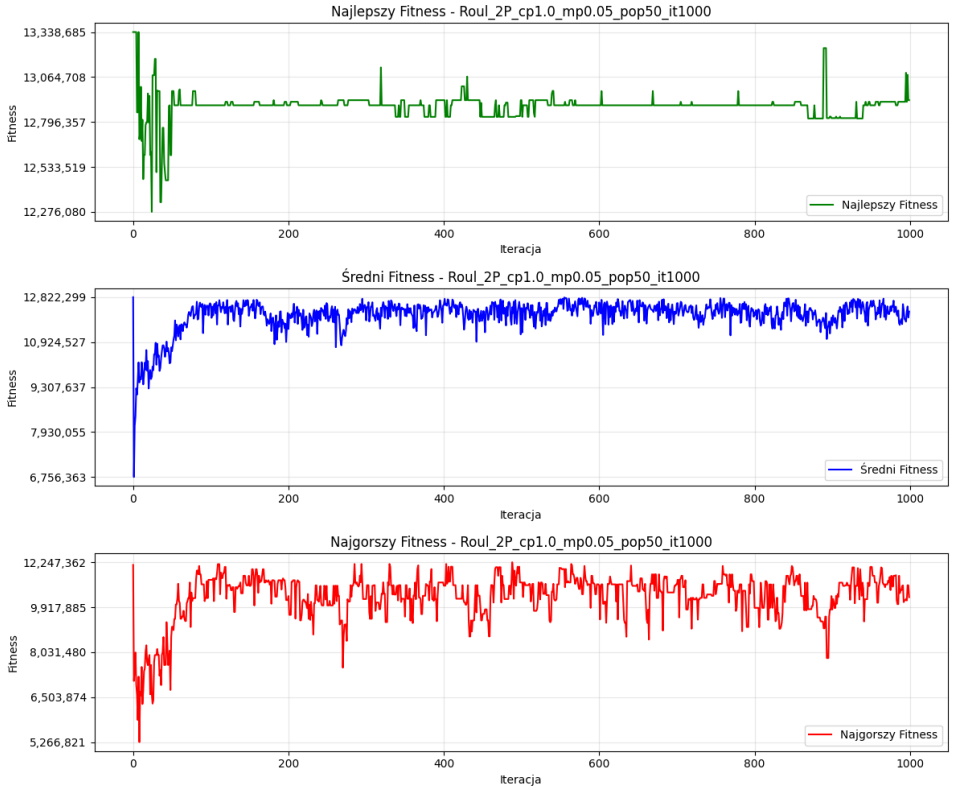
Rysunek 4: Porównanie wyników dla najgorszego i najlepszego uruchomienia dla badanych strategii selekcji.

Poniższe wykresy przedstawiają przebieg procesu optymalizacji w czasie (względem iteracji) dla obu strategii. Można zauważyć, jak zmieniały się: najlepsza wartość funkcji przystosowania (Best), średnia (Avg) oraz najgorsza (Worst) w populacji.

2.2.1 Selekcja Ruletkowa

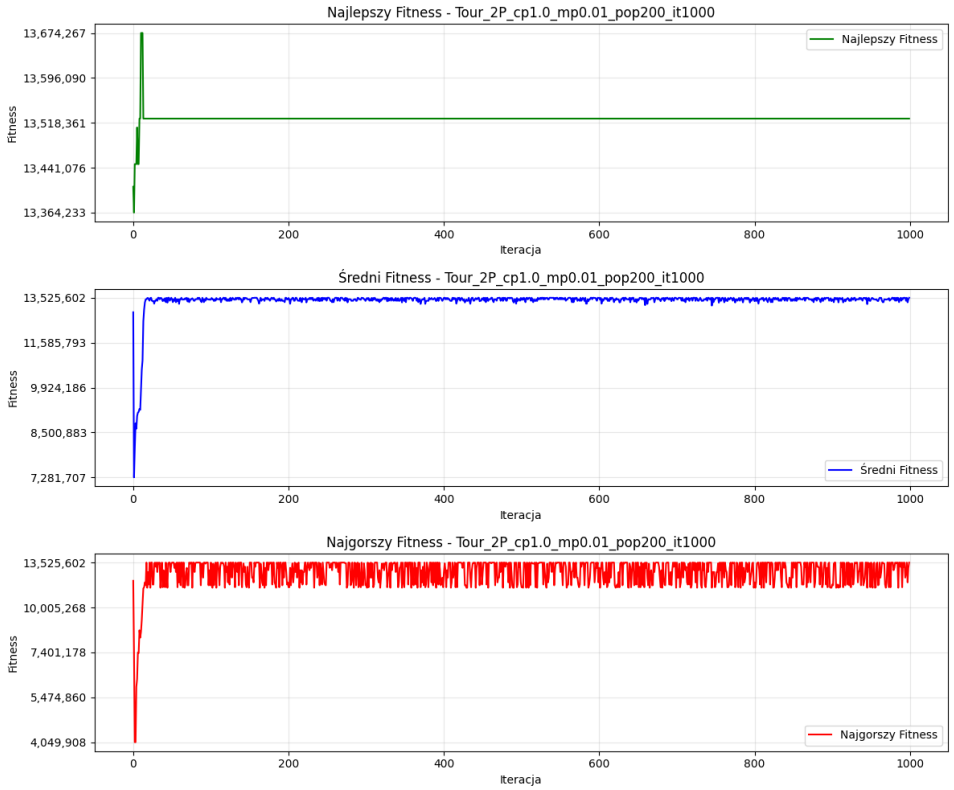


Rysunek 5: Przebieg optymalizacji dla selekcji ruletkowej (najlepszy uzyskany wynik).

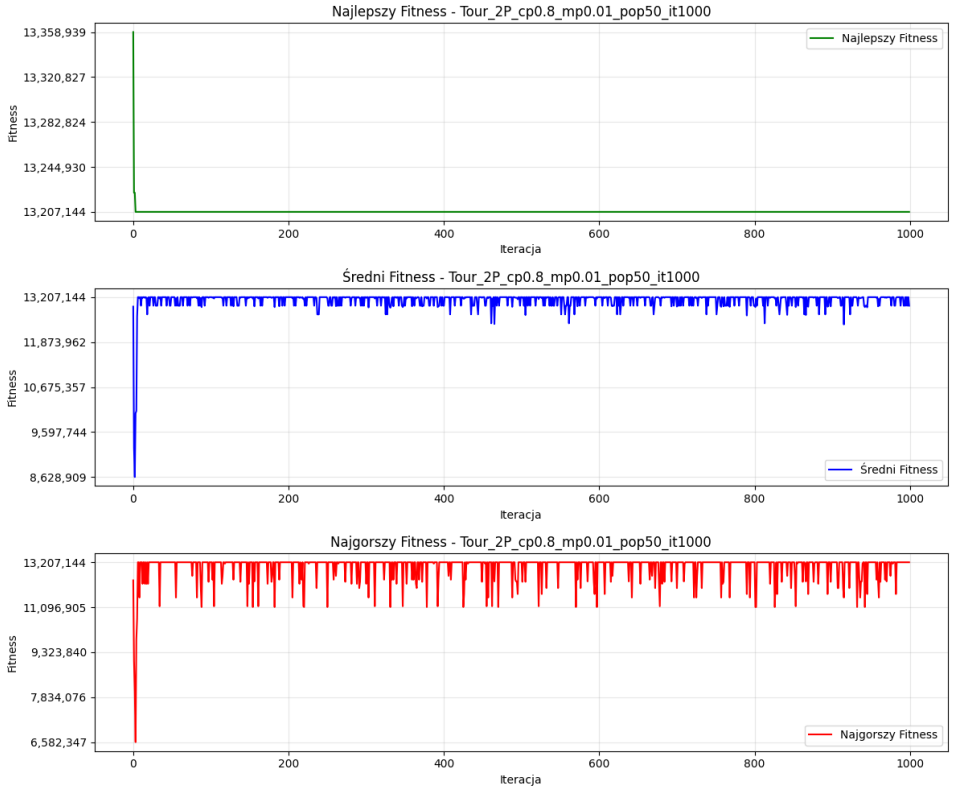


Rysunek 6: Przebieg optymalizacji dla selekcji ruletkowej (najgorszy uzyskany wynik).

2.2.2 Selekcja Turniejowa



Rysunek 7: Przebieg optymalizacji dla selekcji turniejowej (najlepszy uzyskany wynik).



Rysunek 8: Przebieg optymalizacji dla selekcji turniejowej (najgorszy uzyskany wynik).

3 Wnioski

Na podstawie przeprowadzonych eksperymentów sformułowano następujące wnioski:

- **Skuteczność algorytmu**: Algorytm genetyczny skutecznie znajduje rozwiązania problemu plecakowego o wysokiej wartości, zbliżone do optimum.
- **Wpływ populacji**: Większa populacja zwiększa szansę na znalezienie lepszego rozwiązania, ale wydłuża czas obliczeń. Należy dobrać kompromisową wartość (np. 100-200 osobników).
- **Parametry ewolucyjne**: Prawdopodobieństwo krzyżowania i mutacji miały stosunkowo niewielki wpływ na jakość końcowego rozwiązania w badanym zakresie. Zaleca się jednak stosowanie umiarkowanych wartości w celu zachowania różnorodności genetycznej.
- **Strategia selekcji**: Obie strategie (ruletkowa i turniejowa) pozwalają osiągnąć dobre wyniki. Selekcja turniejowa często powoduje szybszą zbieżność, ale też szybszą utratę różnorodności (utknięcia w lokalnym optimum). Selekcja ruletkowa daje większą szansę słabszym osobnikom, co podtrzymuje różnorodność.
- **Zbieżność**: Wykresy przebiegu pokazują, że średnia wartość przystosowania w populacji rośnie wraz z kolejnymi iteracjami, a następnie się stabilizuje, co potwierdza prawidłowe działanie mechanizmów ewolucyjnych.