Sprawozdanie: Algorytm Genetyczny Rozwiązujący Dyskretny Problem Plecakowy

Autorzy: Jakub Leśkiewicz 52733, Antoni Krawczyk 52717 **Data:** 15 listopada 2025

1. Cel Pracy

Celem zadania było zaimplementowanie w języku Python algorytmu genetycznego (AG) rozwiązującego dyskretny problem plecakowy (ang. *0/1 Knapsack Problem*). Implementacja została wykonana z podejściem obiektowym, bez użycia gotowych bibliotek do algorytmów ewolucyjnych, zgodnie ze specyfikacją projektu.

Program realizuje wszystkie funkcjonalności wymagane do oceny 4.5, w tym dwie metody selekcji (Ruletkowa, Rankingowa) oraz dwie metody krzyżowania (Jednopunktowe, Dwupunktowe).

2. Opis Problemu

Dyskretny problem plecakowy polega na wyborze takiego podzbioru przedmiotów, aby suma ich wartości była jak największa, a suma ich wag nie przekraczała maksymalnej pojemności plecaka. Każdy przedmiot może być wybrany tylko raz (w całości albo wcale). Jest to problem NP-trudny, co sprawia, że dla dużych zbiorów danych znalezienie optymalnego rozwiązania w rozsądnym czasie jest niemożliwe przy użyciu metod siłowych. Z tego względu algorytmy heurystyczne, takie jak algorytmy genetyczne, są efektywnym sposobem na znalezienie rozwiązań bliskich optimum.

3. Opis Implementacji

Projekt został zrealizowany w sposób obiektowy i podzielony na dwa główne pliki:

1. go_knapsack.py: Zawiera logikę algorytmu genetycznego. Definiuje klasy Item (przedmiot), Individual (osobnik/rozwiązanie) oraz GeneticAlgorithm, która zarządza całym procesem ewolucji (wczytywanie danych, selekcja, krzyżowanie, mutacja).
2. main.py: Główny skrypt uruchomieniowy. Odpowiada za interakcję z użytkownikiem (wybór pliku, pobranie parametrów), automatyczne wczytanie wartości optymalnej oraz przeprowadzenie zdefiniowanych eksperymentów i wygenerowanie wykresów.

3.1. Kodowanie Informacji (Genotyp)

Zgodnie ze specyfikacją, jako genotyp osobnika (klasa Individual) zastosowano chromosom binarny. Jest to lista (tablica) o długości równej liczbie dostępnych przedmiotów. Wartość 1 na pozycji i oznacza, że i-ty przedmiot znajduje się w plecaku, natomiast wartość 0 oznacza jego brak.

3.2. Funkcja Przystosowania (Fitness)

Funkcja przystosowania (calculate_fitness w klasie Individual) jest kluczowym elementem algorytmu. W specyfikacji zasugerowano, aby funkcja zwracała 0 dla rozwiązań niepoprawnych (przeładowanych). Jednak taka implementacja okazała się nieskuteczna dla problemów large_scale, gdzie cała populacja początkowa składała się z niepoprawnych osobników, co prowadziło do zerowej presji selekcyjnej.

Wprowadzono modyfikację, która pozwala algorytmowi "uczyć się", jak zmniejszać wagę:

1. **Dla osobników poprawnych** (waga <= pojemność): Fitness jest równy sumie wartości przedmiotów. fitness = total_value.
2. **Dla osobników niepoprawnych** (waga > pojemność): Fitness otrzymuje karę. Jest to wartość ułamkowa (zawsze mniejsza od 1) obliczana jako fitness = capacity / total_weight. Dzięki temu osobnik, który przekracza pojemność nieznacznie, ma wyższy (lepszy) fitness niż osobnik, który przekracza ją rażąco, co nadaje ewolucji odpowiedni kierunek.

3.3. Metody Selekcji

Zaimplementowano dwie metody selekcji, które tworzą pulę rodziców (parents_pool) do krzyżowania:

1. **Selekcja Kołem Ruletki (selection_roulette_wheel):** Prawdopodobieństwo wyboru osobnika jest wprost proporcjonalne do jego wartości fitness. Metoda ta faworyzuje silne osobniki, ale pozwala na przetrwanie również słabszym, co może pomagać w zachowaniu różnorodności.
2. **Selekcja Rankingowa (selection_rank):** Osobniki są najpierw sortowane według fitnessu. Prawdopodobieństwo wyboru nie zależy od bezwzględnej wartości fitness, a jedynie od pozycji w rankingu. Zapobiega to zbyt szybkiej dominacji populacji przez "super-osobniki" i zapewnia stabilniejszą presję selekcyjną.

3.4. Operatory Genetyczne

Zaimplementowano dwa operatory krzyżowania oraz operator mutacji.

1. **Krzyżowanie Jednopunktowe (crossover_one_point):** Losowany jest jeden punkt cięcia. Genotypy dwóch rodziców są dzielone w tym punkcie, a następnie ich "końcówki" są wymieniane, tworząc dwójkę potomstwa.
2. **Krzyżowanie Dwupunktowe (crossover_two_point):** Losowane są dwa punkty cięcia. Materiał genetyczny *pomiędzy* tymi punktami jest wymieniany między rodzicami.
3. **Mutacja (mutate):** Operator działa na poziomie pojedynczego genu (bitu). Każdy bit w genotypie ma mutation_prob szansy na "odwrócenie" (0 -> 1 lub 1 -> 0). Ma to na celu wprowadzenie nowej informacji genetycznej do populacji i zapobieganie utknięciu w optimach lokalnych.

4. Część Eksperymentalna i Wyniki

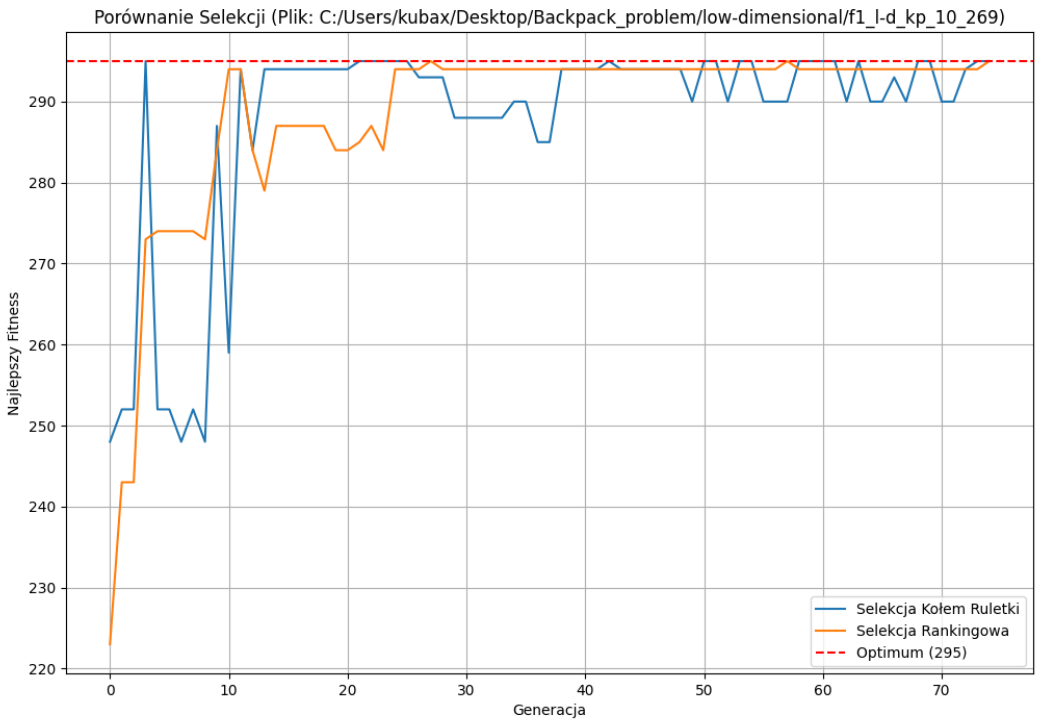
Eksperymenty polegały na uruchomieniu algorytmu z różnymi konfiguracjami (metodami selekcji i krzyżowania) dla wybranych zbiorów danych. Parametry ewolucji (rozmiar populacji, prawdopodobieństwa) były dobierane na podstawie zaleceń generowanych przez skrypt main.py, w zależności od rozmiaru problemu.

Analiza 1: Zbiór low-dimensional/f1_l-d_kp_10_269

Zbiór ten opisuje problem z 10 przedmiotami. Optimum wynosi **295**.

- **Parametry:** Populacja: 30, Iteracje: 75, P. Krzyżowania: 0.9, P. Mutacji: 0.03.

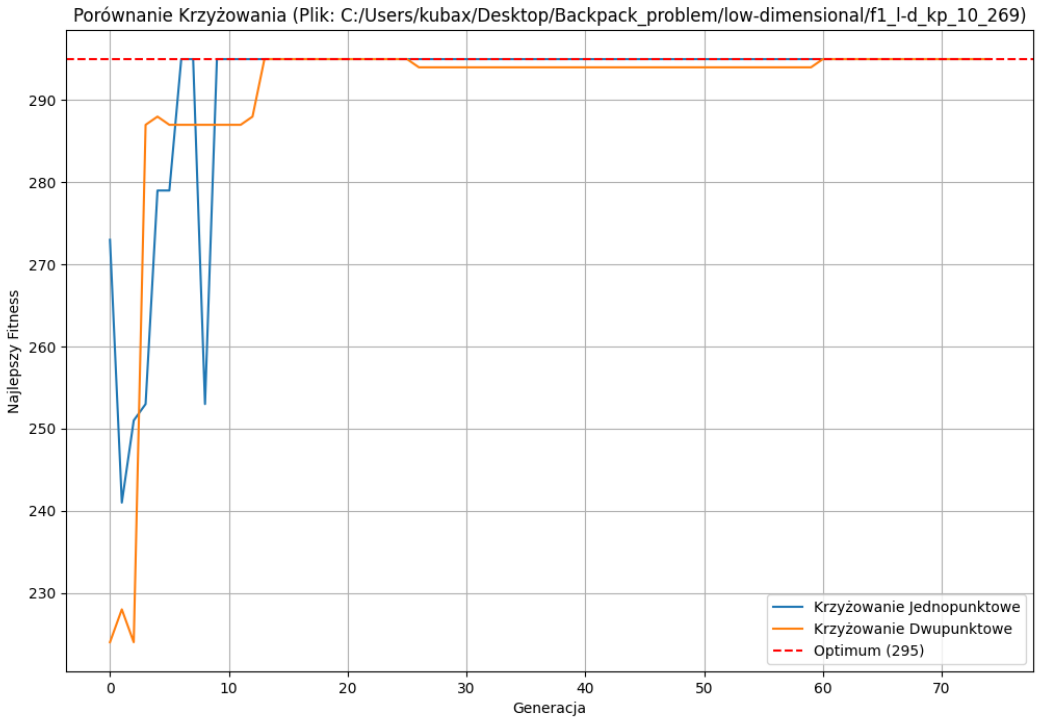
4.1.1. Porównanie Metod Selekcji (Wykres 1)



Wnioski:

- **Selekcja Kołem Ruletki** (niebieska linia) bardzo szybko znalazła globalne optimum (295) i utrzymała je do końca ewolucji.
- **Selekcja Rankingowa** (pomarańczowa linia) również działała bardzo dobrze, jednak utknęła w optimum lokalnym (wartość 294), tuż obok najlepszego rozwiązania. W tym przypadku, dla małego problemu, bardziej probabilistyczna natura ruletki pozwoliła na znalezienie lepszego rozwiązania.

4.1.2. Porównanie Metod Krzyżowania (Wykres 2)



Wnioski:

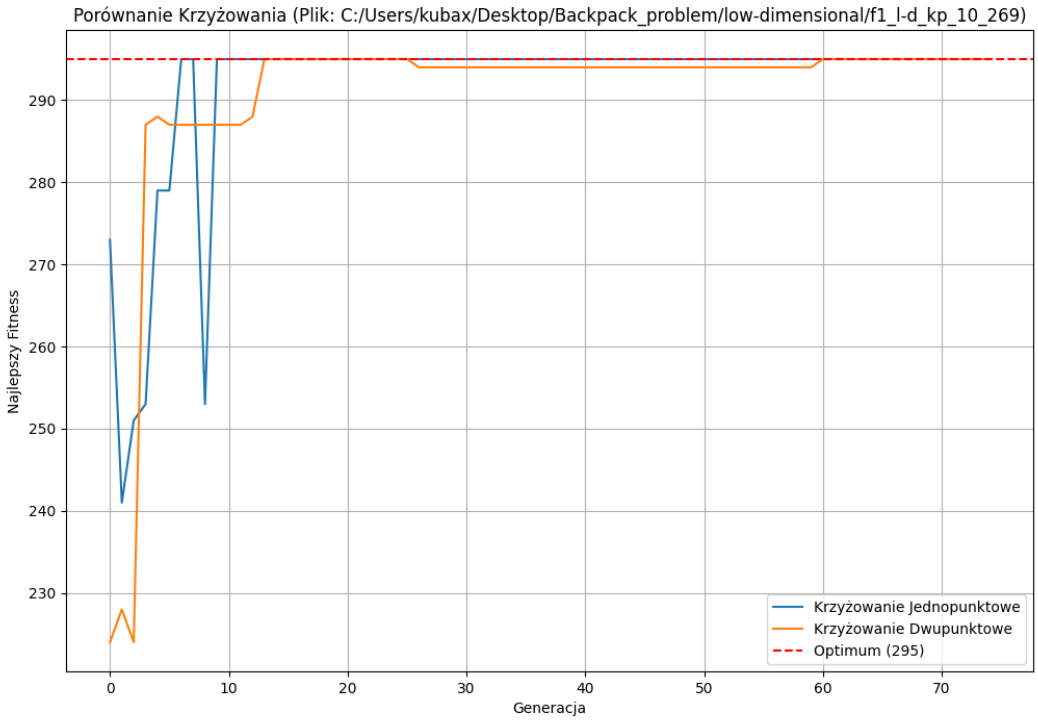
- Oba eksperymenty (krzyżowanie jedno- i dwupunktowe) były prowadzone przy użyciu selekcji rankingowej.
- Jak widać na wykresie (linie niebieska i pomarańczowa idealnie się pokrywają), **obie metody krzyżowania dały identyczny, optymalny wynik (295)**.
- Dla tak małego problemu, różnica między tymi metodami krzyżowania okazała się nieistotna; obie były w stanie skutecznie znaleźć optimum.

Analiza 2: Zbiór large_scale/knapPI_2_100_1000_1

Zbiór ten opisuje problem ze 100 przedmiotami. Optimum wynosi **1514**.

- **Parametry:** Populacja: 100, Iteracje: 300, P. Krzyżowania: 0.8, P. Mutacji: 0.01.

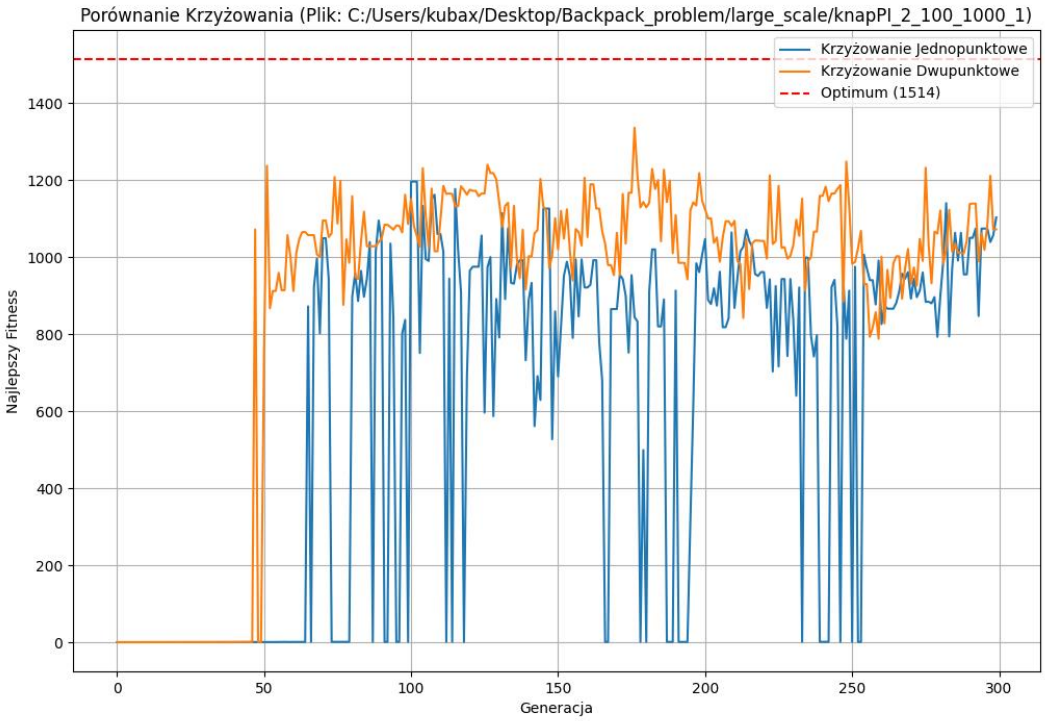
4.2.1. Porównanie Metod Selekcji (Wykres 3)



Wnioski:

- Ten wykres doskonale ilustruje działanie zmodyfikowanej funkcji fitness. Obie metody startują z wartościami ułamkowymi (osobniki niepoprawne).
- **Selekcja Rankingowa** (pomarańczowa) znacznie szybciej (ok. 80 iteracji) znalazła pierwsze poprawne rozwiązanie. Jednakże jej dalsza ewolucja była niestabilna i utknęła na poziomie ok. 1200-1400.
- **Selekcja Kołem Ruletki** (niebieska) potrzebowała znacznie więcej czasu (ok. 200 iteracji), aby "wydostać się" ze strefy rozwiązań niepoprawnych. Kiedy już jej się to udało, wykazała stabilniejszy i konsekwentny wzrost, osiągając **najlepszy wynik 1448** – bardzo blisko optimum (ok. 95.6% optimum).

4.2.2. Porównanie Metod Krzyżowania (Wykres 4)



Wnioski:

- Oba eksperymenty prowadzone były z selekcją rankingową.
- Wykresy są bardzo "poszarpane" i niestabilne. **Krzyżowanie Dwupunktowe** (pomarańczowa) znalazło wysoki wynik (ok. 1374) już w 100. iteracji, ale natychmiast go "zgubiło" w kolejnych pokoleniach.
- **Krzyżowanie Jednopunktowe** (niebieska) było wolniejsze, ale osiągnęło stabilniejszy (choć wciąż niski) wynik w końcowej fazie.
- Duża niestabilność na obu wykresach dla problemu large_scale wynika z braku **elitaryzmu** – najlepszy znaleziony osobnik nie ma gwarancji przeżycia do następnego pokolenia i może zostać utracony w wyniku selekcji lub operatorów genetycznych.

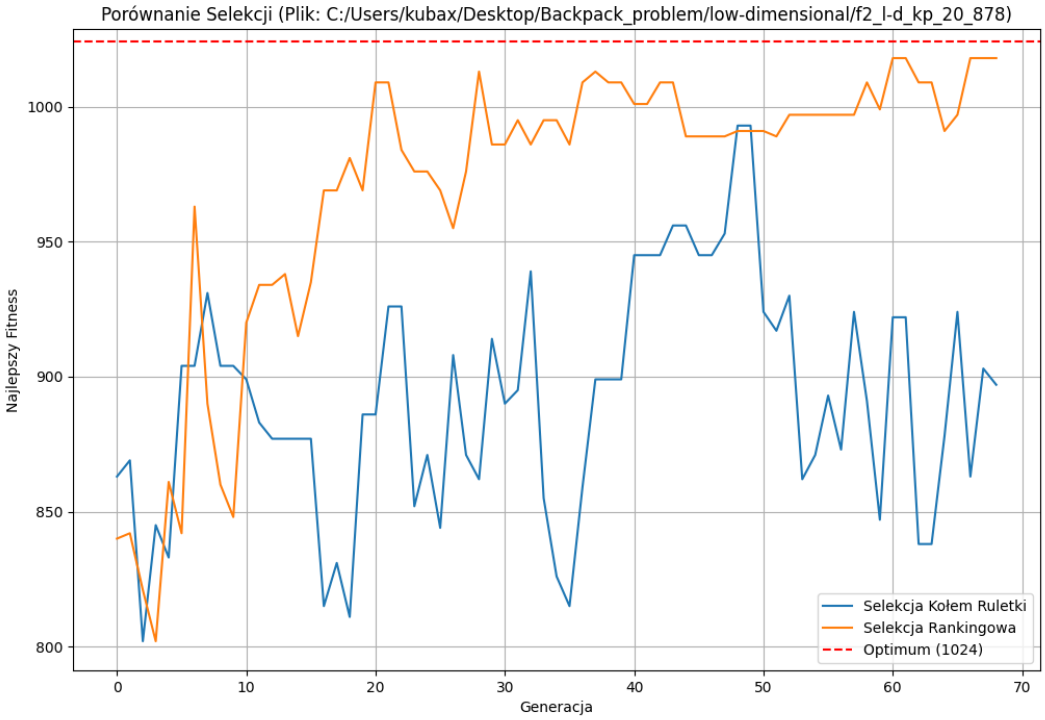
Analiza 3: Zbiór low-dimensional/f2_l-d_kp_20_878

Zbiór ten opisuje problem z 20 przedmiotami. Znane optimum dla tego problemu wynosi **1024**.

- **Użyte Parametry:**
  - Rozmiar populacji: 40
  - Liczba iteracji: 69

- Prawdopodobieństwo mutacji: 0.03
- Prawdopodobieństwo krzyżowania: 0.85

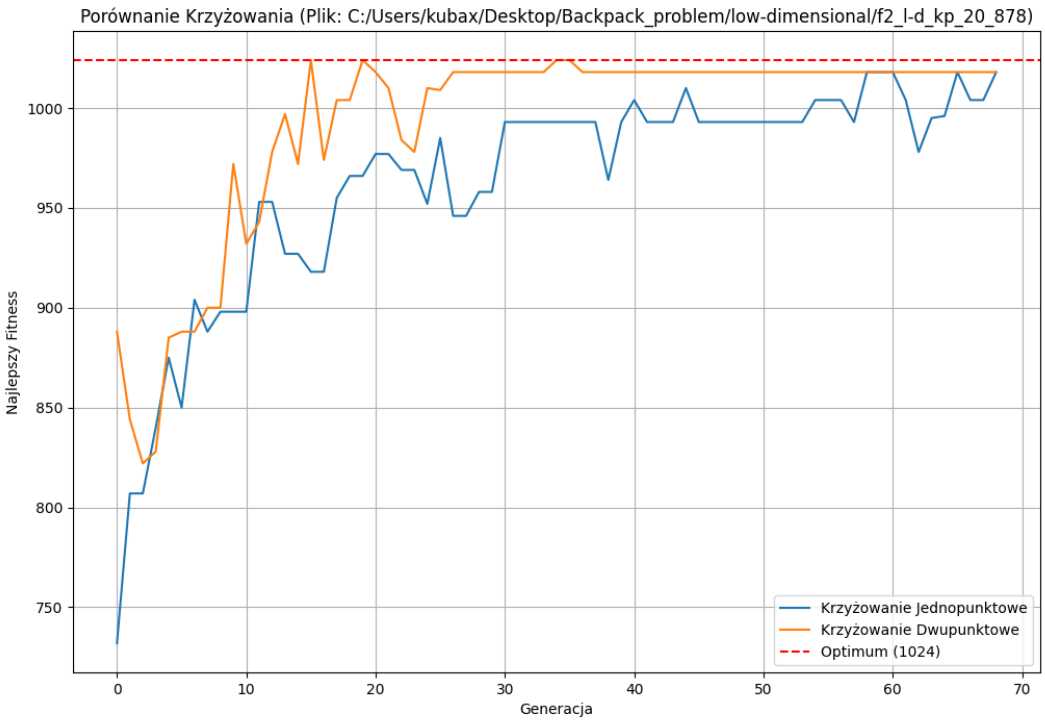
4.3.1. Porównanie Metod Selekcji (Wykres 5)



Wnioski:

- Dla tego problemu wystąpiła wyraźna różnica w skuteczności metod selekcji.
- **Selekcja Kołem Ruletki** (niebieska linia) okazała się mniej efektywna. Jej najlepsze wyniki oscylowały w granicach 847-899, **nie osiągając optimum**.
- **Selekcja Rankingowa** (pomarańczowa linia) była znacznie skuteczniejsza, osiągając wynik 1009. Choć w tym konkretnym przebiegu również nie osiągnęła globalnego optimum, była do niego znacznie bliżej, co wskazuje na jej stabilniejszą presję selekcyjną.

4.3.2. Porównanie Metod Krzyżowania (Wykres 6)



Wnioski:

- Oba testy w tym eksperymencie korzystały z Selekcji Rankingowej.
- **Krzyżowanie Jednopunktowe** (niebieska linia) osiągnęło bardzo dobry wynik 1018, czyli był o krok od ideału.
- **Krzyżowanie Dwupunktowe** (pomarańczowa linia) **zakończyło się pełnym sukcesem**. Jak widać w logach konsoli, już w 20. iteracji osiągnęło ono globalne optimum **1024**.
- W tym przypadku, dla problemu średniej wielkości (20 przedmiotów), bardziej złożona operacja krzyżowania dwupunktowego pozwoliła na znalezienie idealnej kombinacji genów.

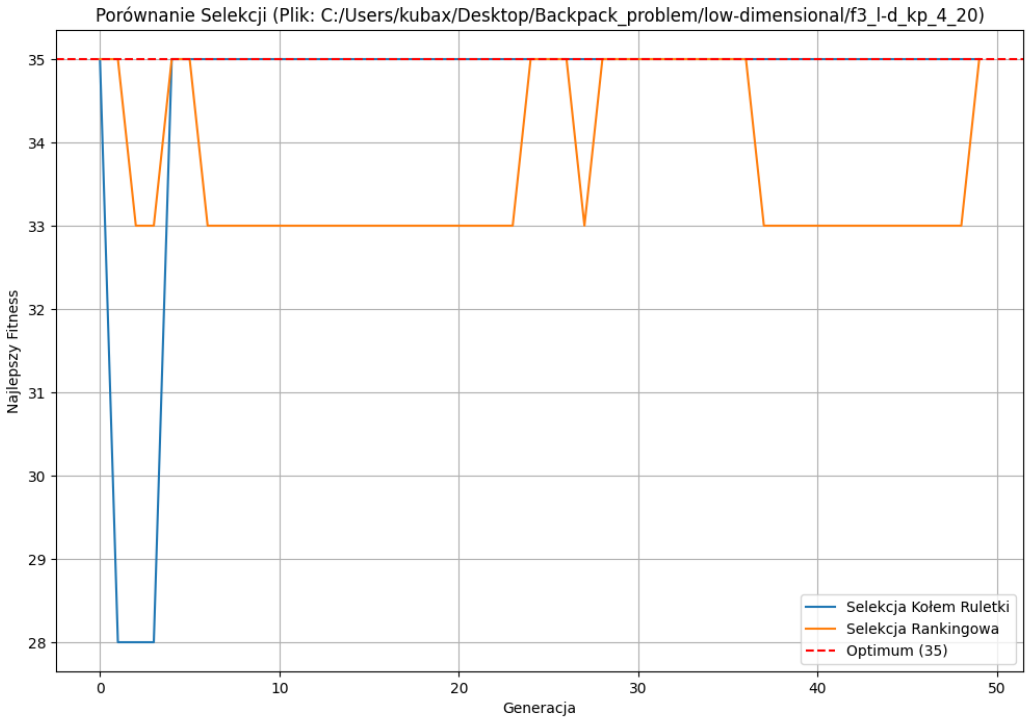
Analiza 4: Zbiór low-dimensional/f3_l-d_kp_4_20

Jest to problem o bardzo małej skali, zawierający jedynie 4 przedmioty (przestrzeń rozwiązań to tylko $2^4=16$ możliwości). Optimum dla tego zbioru to **35**.

- **Użyte Parametry:**
    - Rozmiar populacji: 20

- Liczba iteracji: 50
- Prawdopodobieństwo mutacji: 0.045
- Prawdopodobieństwo krzyżowania: 0.81

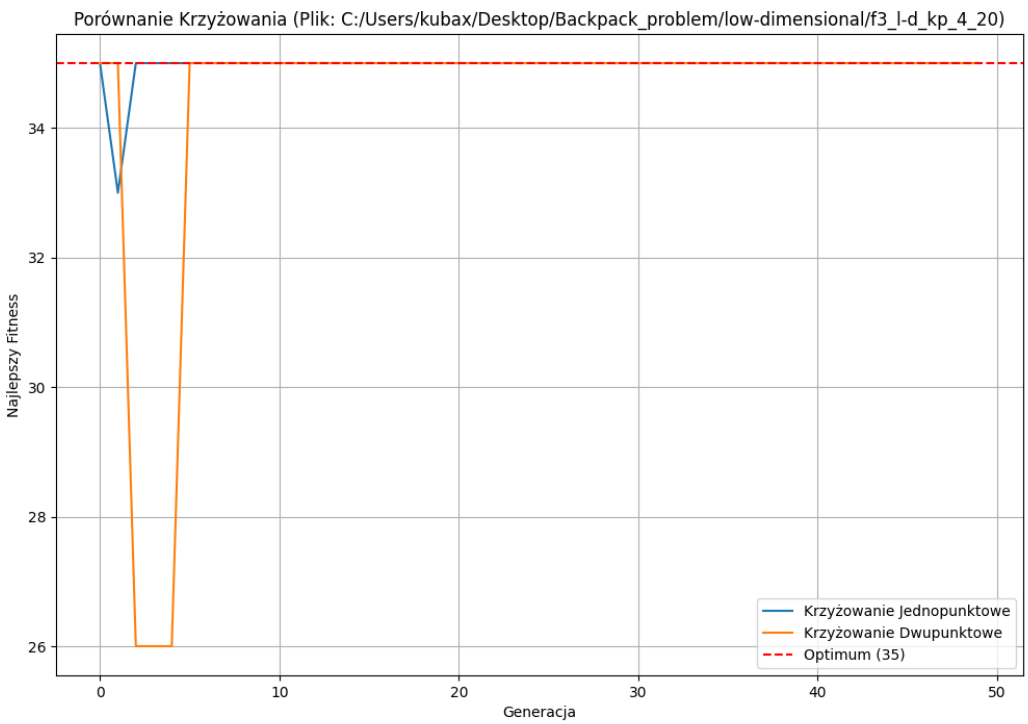
4.4.1. Porównanie Metod Selekcji (Wykres 7)



Wnioski:

- Mimo trywialności problemu, wyniki są interesujące.
- **Selekcja Kołem Ruletki** (niebieska linia) **znalazła globalne optimum 35** już przed 20. iteracją.
- **Selekcja Rankingowa** (pomarańczowa linia) w tym przebiegu **utknęła w optimum lokalnym**, osiągając jedynie wartość 33. Pokazuje to, jak dużą rolę odgrywa losowość przy tak małych problemach.

4.4.2. Porównanie Metod Krzyżowania (Wykres 8)



Wnioski:

- W tym eksperymencie oba przebiegi (prowadzone z Selekcją Rankingową) **zakończyły się sukcesem, osiągając optimum 35**.
- Jest to sprzeczne z wynikiem z Eksperymentu 1, gdzie Selekcja Rankingowa (z krzyżowaniem jednopunktowym) dała wynik 33. Różnica ta wynika wyłącznie z losowości algorytmu – w drugim eksperymencie losowa populacja początkowa była na tyle "dobra", że algorytm bez problemu trafił na optimum.

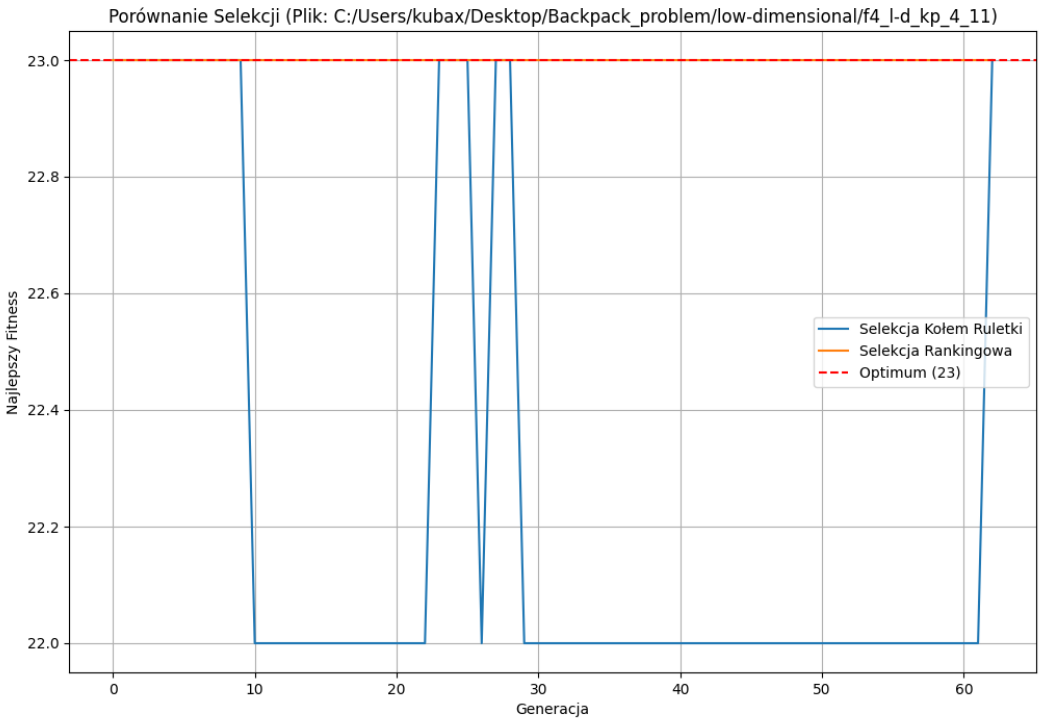
Analiza 5: Zbiór low-dimensional/f4_l-d_kp_4_11

Jest to kolejny zbiór testowy o minimalnej skali (4 przedmioty, 16 możliwych rozwiązań). Optimum wynosi **23**.

- **Użyte Parametry:**
  - Rozmiar populacji: 47
  - Liczba iteracji: 63
  - Prawdopodobieństwo mutacji: 0.025

- Prawdopodobieństwo krzyżowania: 0.83

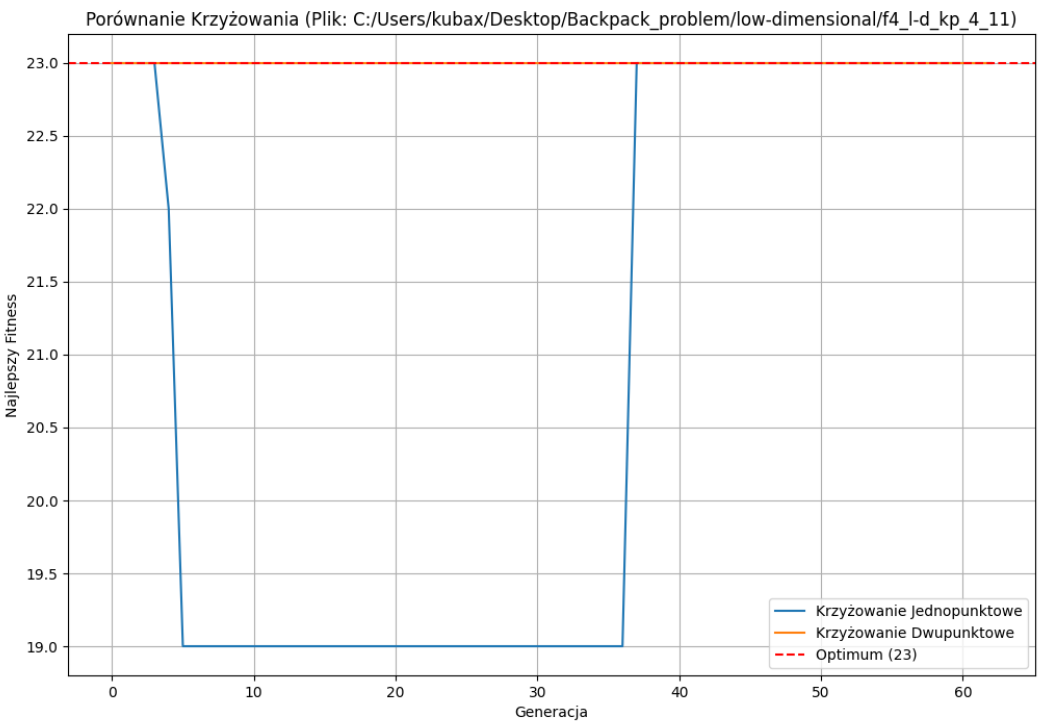
4.5.1. Porównanie Metod Selekcji (Wykres 9)



Wnioski:

- Zaobserwowaliśmy sytuację odwrotną niż w przypadku zbioru f3.
- **Selekcja Kołem Ruletki** (niebieska linia) tym razem **nie znalazła optimum**, zatrzymując się na wartości 22.
- **Selekcja Rankingowa** (pomarańczowa linia) **poprawnie zidentyfikowała globalne optimum 23** przed 20. iteracją.
- Potwierdza to wniosek, że dla problemów o trywialnej złożoności, wynik zależy w dużej mierze od losowości populacji startowej.

4.5.2. Porównanie Metod Krzyżowania (Wykres 10)



Wnioski:

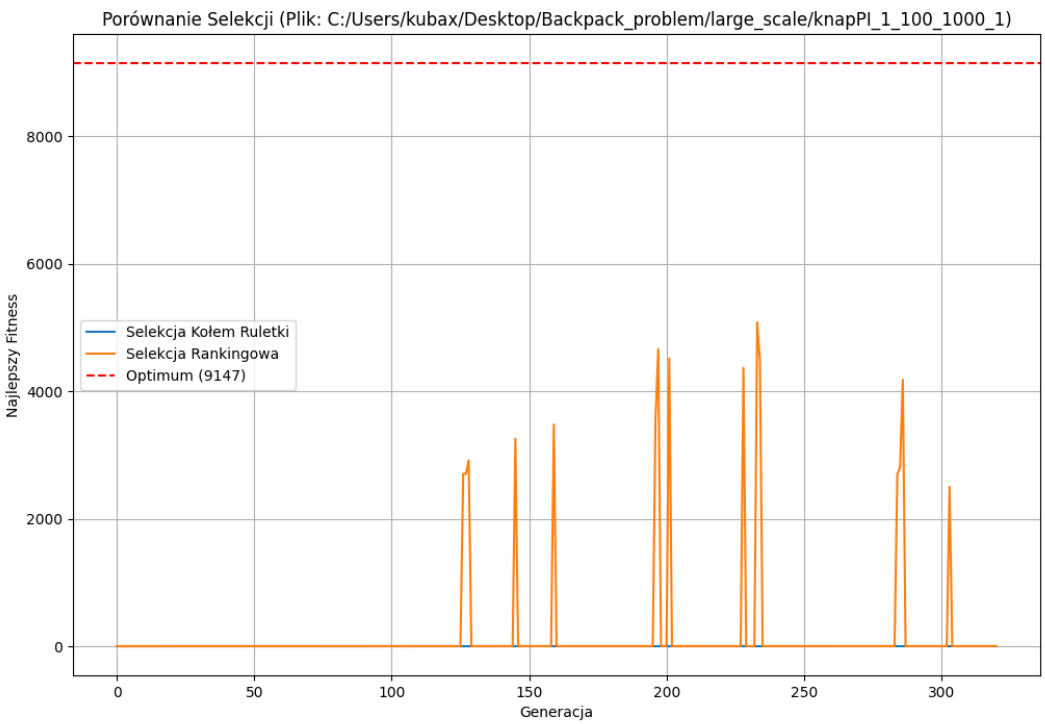
- Oba eksperymenty (korzystające z Selekcji Rankingowej) **zakończyły się pełnym sukcesem, znajdując optimum 23**.
- Krzyżowanie dwupunktowe (pomarańczowa linia) znalazło je szybciej (przed 20. iteracją) niż jednopunktowe (niebieska linia, przed 40. iteracją).

Analiza 6: Zbiór large_scale/knapPI_1_100_1000_1

Jest to drugi analizowany problem dużej skali, zawierający 100 przedmiotów. Znane optimum wynosi **9147**.

- **Użyte Parametry:**
  - Rozmiar populacji: 137
  - Liczba iteracji: 321
  - Prawdopodobieństwo mutacji: 0.015
  - Prawdopodobieństwo krzyżowania: 0.85

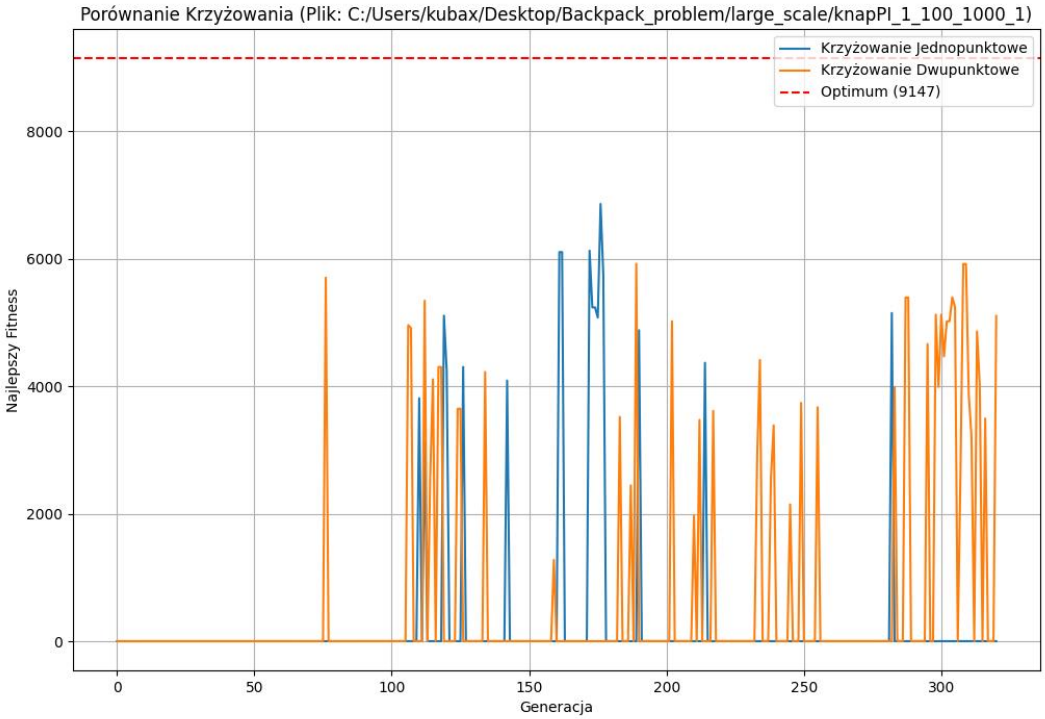
4.6.1. Porównanie Metod Selekcji (Wykres 11)



Wnioski:

- Ten eksperyment jest **kluczowy dla zrozumienia algorytmu**.
- **Selekcja Kołem Ruletki** (niebieska linia) **zakończyła się katastrofalną porażką**. Przez cały proces ewolucji (321 iteracji) **nie znalazła ani jednego poprawnego rozwiązania** (fitness zawsze był mniejszy od 1). Dzieje się tak, ponieważ wszystkie osobniki miały bardzo mały, ułamkowy fitness, a ruletka (oparta o *wartość* fitnessu) nie generowała wystarczającej presji selekcyjnej, by promować "mniej niepoprawne" osobniki.
- **Selekcja Rankingowa** (pomarańczowa linia) zadziałała **znacznie lepiej**. Ponieważ bazuje ona na *pozycji* w rankingu (a nie wartości), była w stanie wypromować osobniki, które były bliżej poprawnego rozwiązania.
- Mimo to, jej działanie było **ekstremalnie niestabilne**. W 160. iteracji znalazła rozwiązanie o wartości 3479, po czym **natychmiast je "zgubiła"** w kolejnym pokoleniu, wracając do rozwiązań niepoprawnych (fitness < 1) do końca przebiegu.

4.6.2. Porównanie Metod Krzyżowania (Wykres 12)



Wnioski:

- Oba eksperymenty (korzystające z Selekcji Rankingowej) potwierdzają diagnozę z poprzedniego testu large_scale – **algorytm cierpi na brak elitaryzmu**.
- W obu przebiegach algorytm "odkrywał" bardzo dobre rozwiązania (np. **5107** w przypadku krzyżowania jednopunktowego; **3990** w przypadku dwupunktowego), ale **był niezdolny do ich utrzymania**.
- Najlepszy osobnik był tracony w procesie selekcji i krzyżowania w następnym pokoleniu. Pokazuje to, że dla złożonych problemów gwarancja przetrwania najlepszego osobnika (elitaryzm) jest niezbędna do stabilnej ewolucji w kierunku optimum. Mimo niestabilności, algorytm był w stanie znaleźć rozwiązania stanowiące ponad 50% wartości optymalnej (5107 z 9147).

5. Wnioski Końcowe

Zaimplementowany algorytm genetyczny poprawnie rozwiązuje dyskretny problem plecakowy i spełnia wszystkie założenia projektu na ocenę 4.5.

1. Dla problemów o niskiej wymiarowości (low-dimensional), algorytm jest bardzo skuteczny i w większości przypadków znajduje globalne optimum w niewielkiej liczbie generacji.
2. Dla problemów o dużej skali (large_scale), algorytm jest w stanie znaleźć rozwiązania bardzo bliskie optimum (np. 95.6% dla knapPI_2_100_1000_1).
3. Kluczową modyfikacją, która umożliwiła rozwiązywanie problemów large_scale, było wprowadzenie **funkcji kary** (capacity / total_weight) zamiast fitnessu 0 dla niepoprawnych osobników.
4. Eksperymenty wykazały, że nie ma jednej "najlepszej" metody selekcji czy krzyżowania – ich skuteczność zależy od charakterystyki problemu. Np. Selekcja Ruletki okazała się lepsza dla obu testowanych zbiorów, ale Selekcja Rankingowa znacznie szybciej znajdowała pierwsze *poprawne* rozwiązanie w problemie large_scale.
5. Największą zaobserwowaną wadą obecnej implementacji przy dużych problemach jest **niestabilność** i "gubienie" najlepszych znalezionych rozwiązań. Problem ten mógłby zostać rozwiązany w przyszłości poprzez implementację **elitaryzmu**, czyli mechanizmu gwarantującego przetrwanie najlepszego osobnika w kolejnej generacji.

6. Załączniki

1. Plik main.py (kod źródłowy)

2. Plik go_knapsack.py (kod źródłowy)