## **Opis problemu**

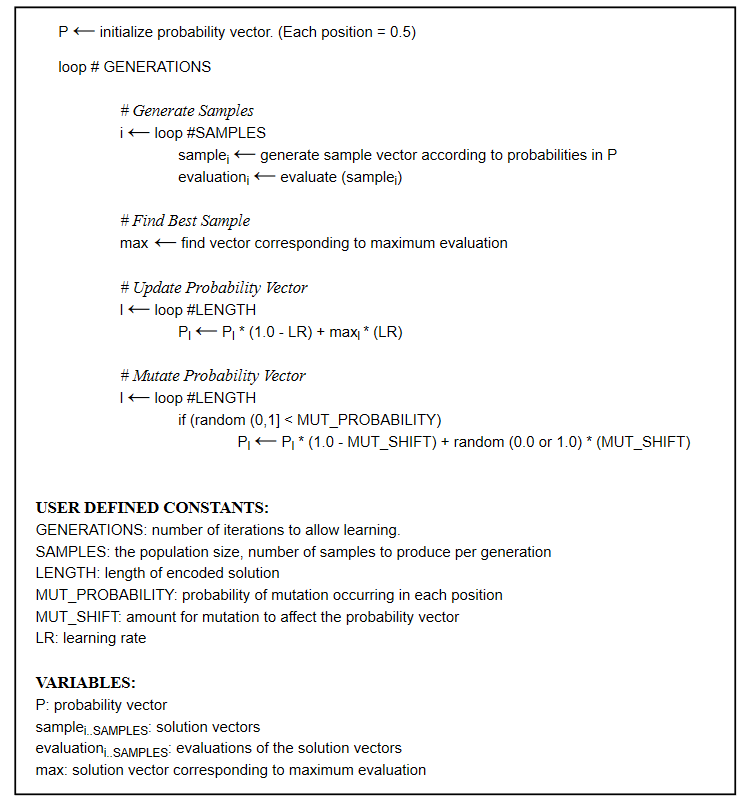
Problem plecakowy (Knapsack problem) jest problemem optymalizacyjnym polegającym na maksymalizacji sumarycznej wartości przedmiotów w plecaku przy jednoczesnym uwzględnieniu wielkości plecaka (przedmioty muszą mieścić się w plecaku). Każdy przedmiot ma zdefiniowaną wagę i wartość. Dla losowo generowanych problemów dane są nieskorelowane, co znaczy, że nie można określić zależności pomiędzy wartością, a wagą przedmiotów. W tym przypadku algorytm szybciej się uczy ponieważ różnice pomiędzy wartościami, a wagami są większe i łatwiej jest znaleźć bardzo wartościowe i bardzo tanie przedmioty. Natomiast dla danych skorelowanych wartość przedmiotu jest zależna od jego wagi, jest mniejsza różnica w relacji wagi do wartości poszczególnych przedmiotów. Zazwyczaj w tym przypadku wartość stanowi funkcję wagi, a rozwiązania są cięższe do znalezienia.

## **Cel projektu**

Celem projektu jest rozwiązanie problemu plecakowego dla danych skorelowanych i nieskorelowanych poprzez implementację algorytmów PBIL i Boltzmann/Tsetlin, porównanie algorytmów oraz przeprowadzenie analizy statystycznej wyników.

## **Algorytm PBIL**

Algorytm PBIL (Population-Based Incremental Learning) jest optymalizacyjnym algorytmem genetycznym, w którym ewolucja następuje głównie dla genotypu całej populacji niż dla jednostki. Często określany jest prostszą wersją algorytmu ewolucyjnego, jednak w wielu przypadkach osiąga lepsze wyniki niż standardowy algorytm ewolucyjny. W poniższej ramce znajduje się pseudokod algorytmu PBIL opisany przez A Baluja:

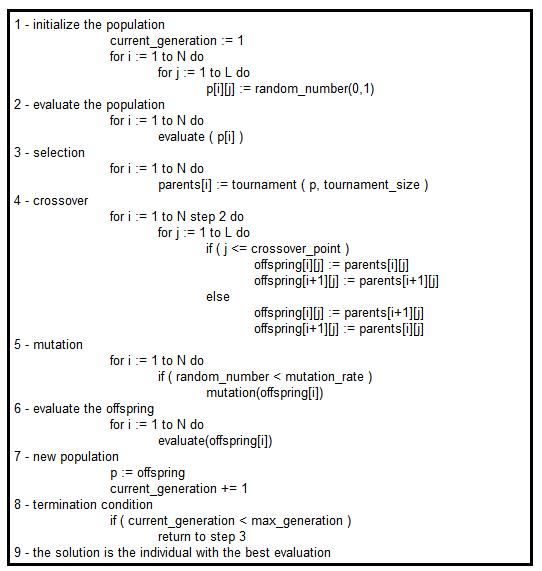


Po inicjalizacji wektora prawdopodobieństwa występuje pętla generacji, w której na początku każdej iteracji generowane są próbki, które następnie są oceniane, żeby wybrać najlepszą, następnie aktualizowany jest wektor prawdopodobieństwa po czym dochodzi do jego mutacji.

## **Klasyczny Algorytm Genetyczny**

Algorytmy genetyczne opierają się na ewolucyjnym procesie doboru naturalnego. Chodzi o to, że tylko najlepiej przystosowane osobniki w populacji zostaną wybrane do rozmnażania się i przenoszenia potomstwa do następnego pokolenia. Podobnie algorytm próbuje ulepszyć swoją populację rozwiązań z każdą iteracją.

Zanim przejdziemy do algorytmu, musimy wstępnie zdefiniować dwa ważne komponenty. Najpierw musimy zdecydować, jak przedstawić rozwiązania. Niektóre ogólne sposoby na zrobienie tego mogą obejmować użycie bitów w ciągu lub użycie listy identyfikatorów. Zmieni się to w zależności od problemu. W przypadku każdego problemu będą jednak dostępne różne opcje. Następnie potrzebna jest odpowiednia funkcja sprawności. To też będzie inne dla każdego problemu. Funkcja sprawności powinna być w stanie ocenić, jak blisko jest rozwiązanie do rozwiązania optymalnego. Ważne jest również, aby wziąć pod uwagę efektywność funkcji sprawności, ponieważ będzie ona wykonywana na każdej osobie podczas każdej z iteracji.



## **Porównanie algorytmów**

Zaimplementowane algorytmy PBIL oraz Klasyczny Algorytm Genetyczny porównamy czasowo i pod kątem (przykładowo) otrzymanej średniej wartości funkcji przystosowania(fitness) dla kilkuset uruchomień i różnych wartości zmiennych:

* N – długości plecaka,
* R – zakresu współczynników,

## **Wykorzystane technologie**

Wybrany język programowania: Python

**Wykorzystany zbior danych**

<http://artemisa.unicauca.edu.co/~johnyortega/instances_01_KP/>

**Algorytm genetyczny - uruchomienie i instrukcja dla użytkownika**

W terminale trzeba wpisać następujące komendy:

- cd sciezka\_do\_pliku\_z\_projektem

- python genetic.py **nazwa\_pliku\_z\_problemem** 100 ts 1c 0.7 0.01 200 g

(*mogą to być np.: problem-1.cnf, knapPI\_1\_100\_1000\_1)*

- powyższa komenda wymaga podania następujących parametrów:

1. Nazwa pliku zawierającego problem

2. Liczba osobników w populacji(rozmiar plecaka)

3. Metoda selekcji: rs (pozycja), ts (turniejowa), bs (Boltzmanna)

4. Metoda krzyżowania: 1c (1-punktowa), uc (jednolita)

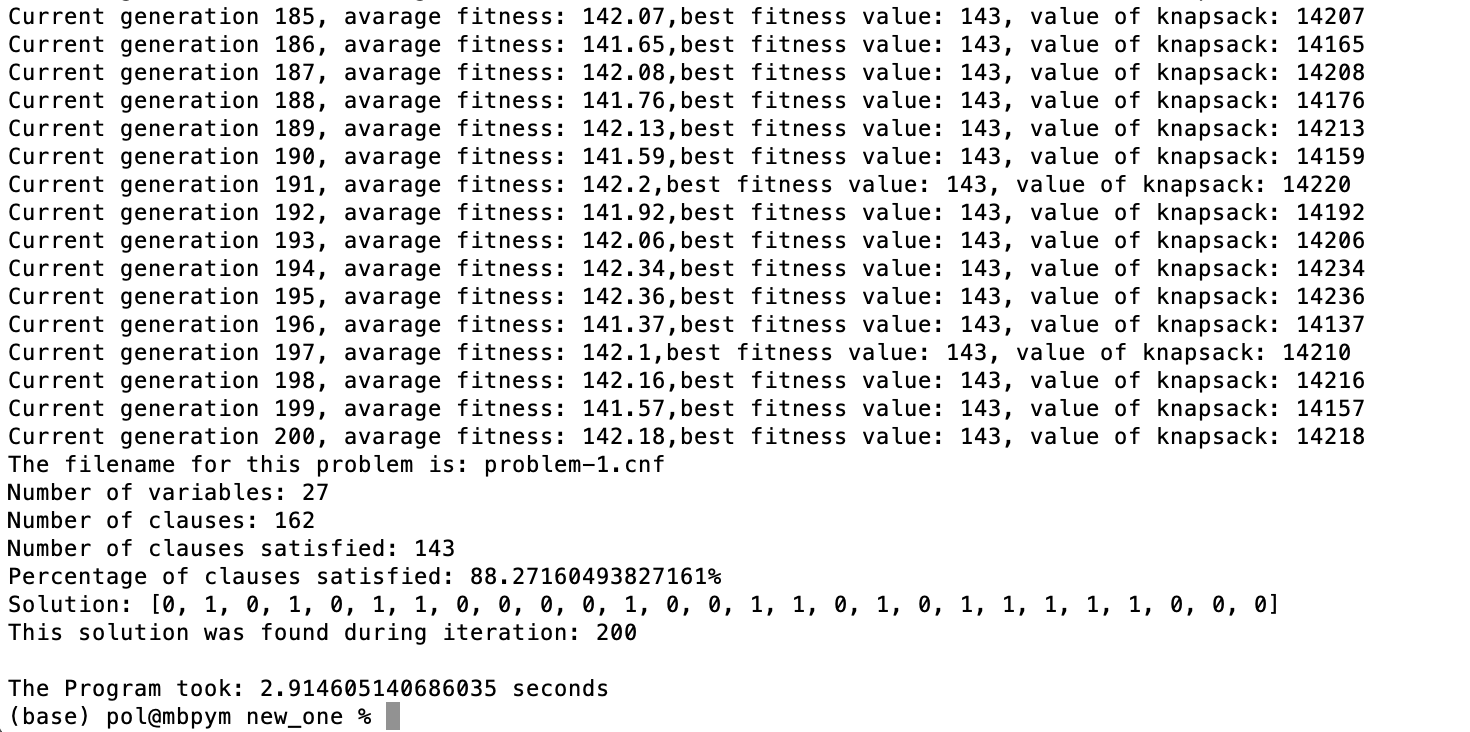
5. Prawdopodobieństwo krzyżowania

6. Prawdopodobieństwo mutacji

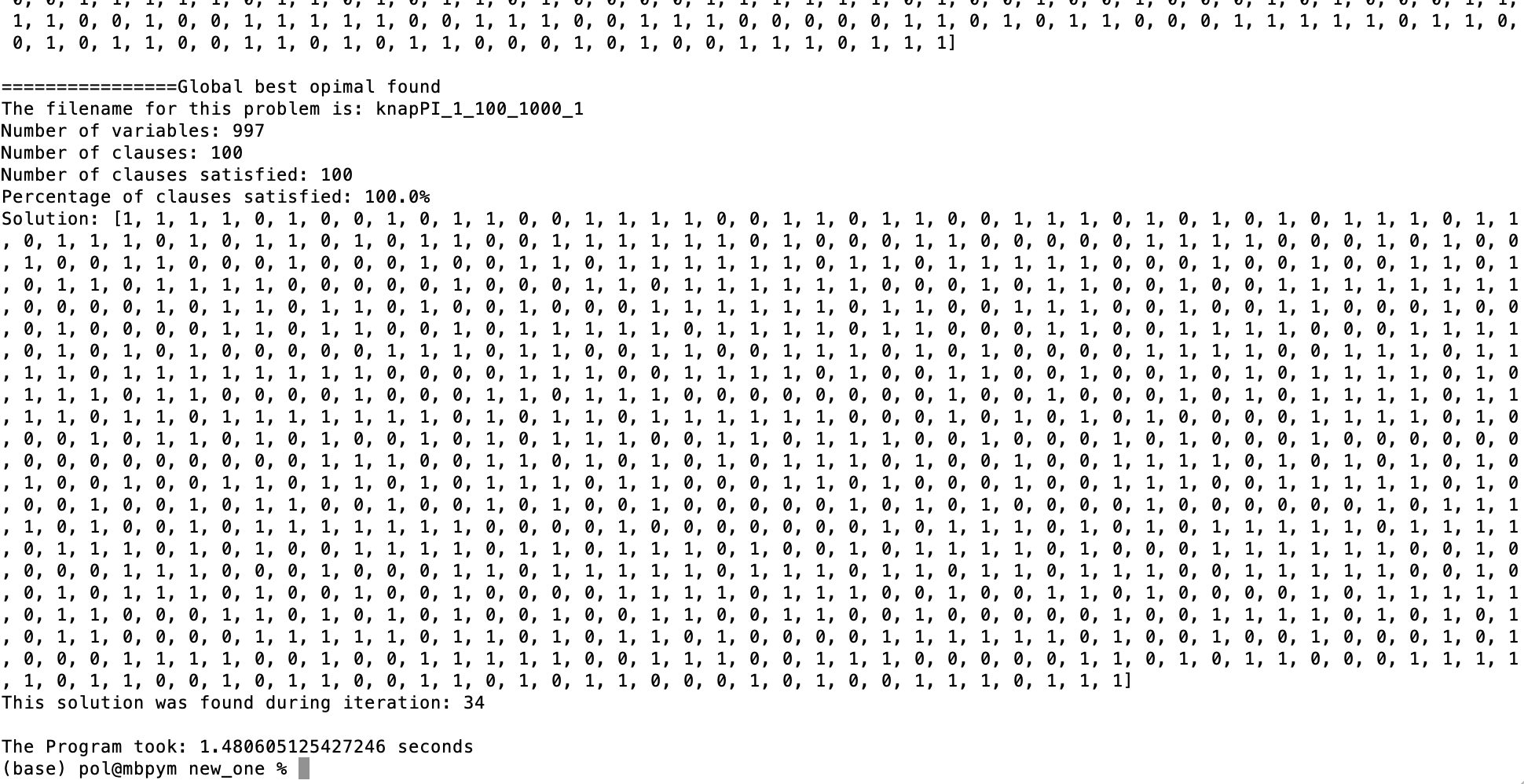
7. Liczba pokoleń do uruchomienia

8. Wybór algorytmu: g (genetyczny)

Cały przebieg pozwiązywania problemu plecakowego jest wypisywany do konsoli użytkownika.



Po rozwiązaniu problemu plecakowego użytkownik zostanie poinformowany o tym w kosoli:

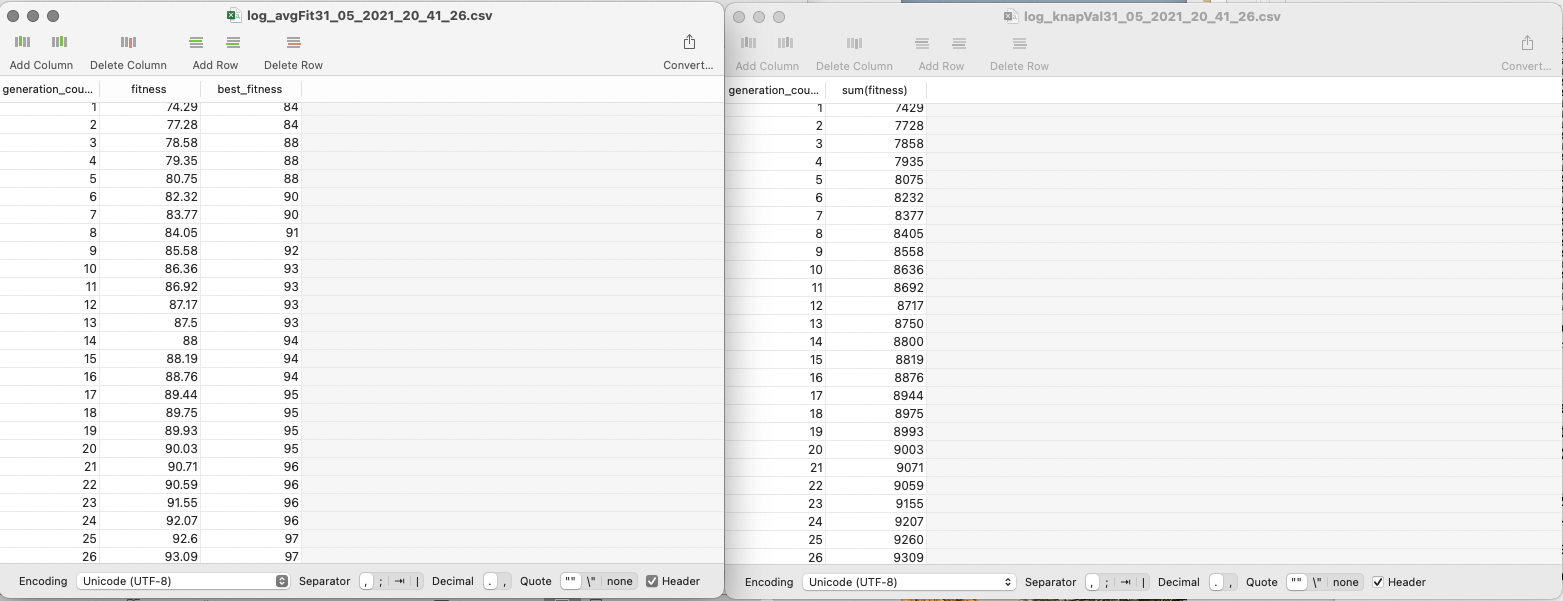


Wyniki z logami działania algorytmu są zapisywane do plików o nazwie:

- log\_avgFit\_Data\_Czas.csv

- log\_knapVal\_Data\_Czas.csv

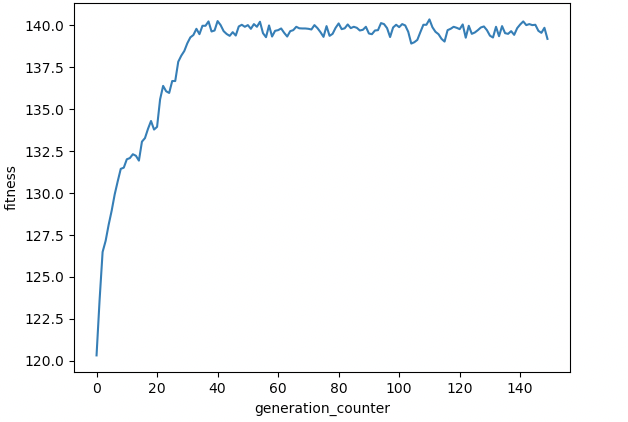
Pliki mają postać tablicy zawierającej numery iteracji i wartości funkcji:



Następnie użytkownik może wyświetlić wykresy dla rozwiązanego problemu plecakowego za pomocą wygenerowanych logów, wpisując następujące komendy:

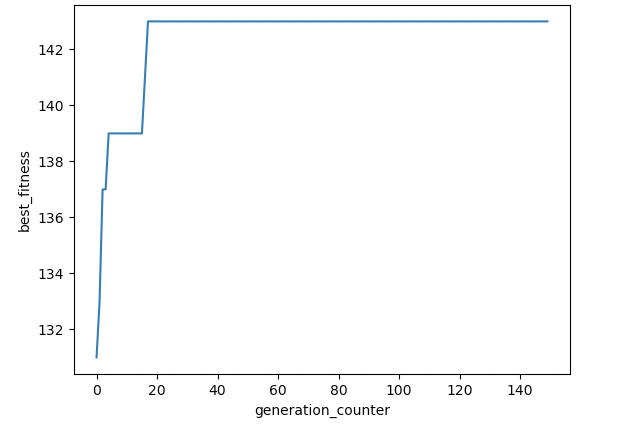
- python plot\_a.py **nazwa\_pliku\_log\_avgFit**

(wykres liniowy najlepszej uśrednionej wartość funkcji fitness dla każdej generacji)



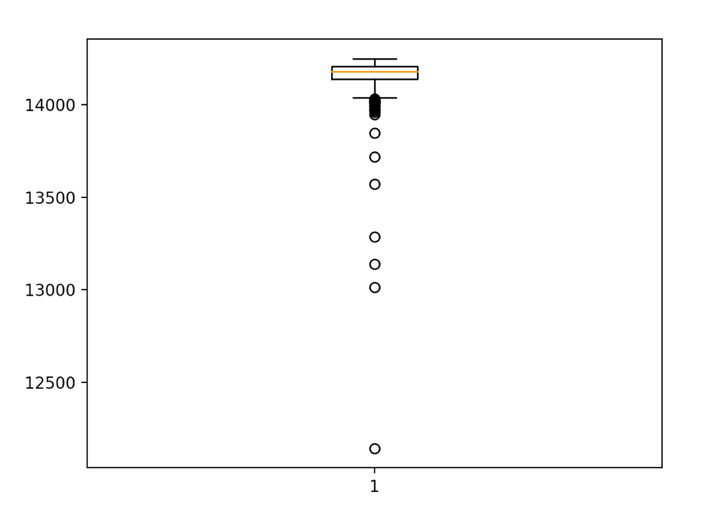
- python plot\_a2.py **nazwa\_pliku\_log\_avgFit**

(wykres liniowy uśrednionej wartości funkcji fitness dla każdej generacji)



- python plot\_b.py **nazwa\_pliku\_log\_knapVal**

(wykres pudełkowy rozkładu wartości plecaka dla wszystkich generacji)

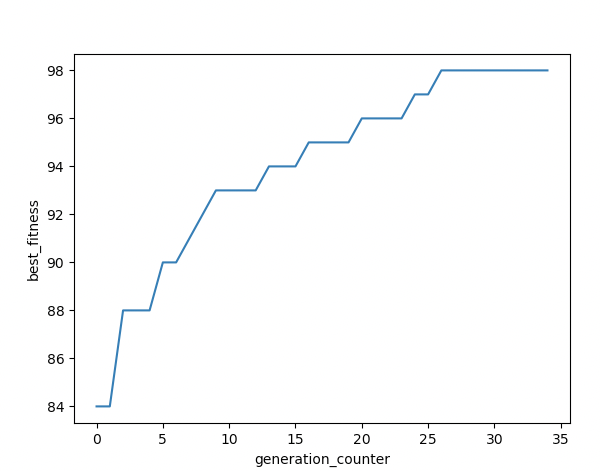
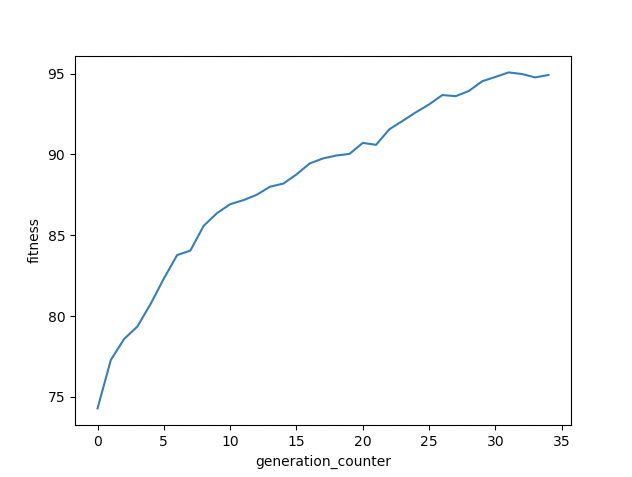


**Wykresy dla istniejących problemów (zbiorów danych)**

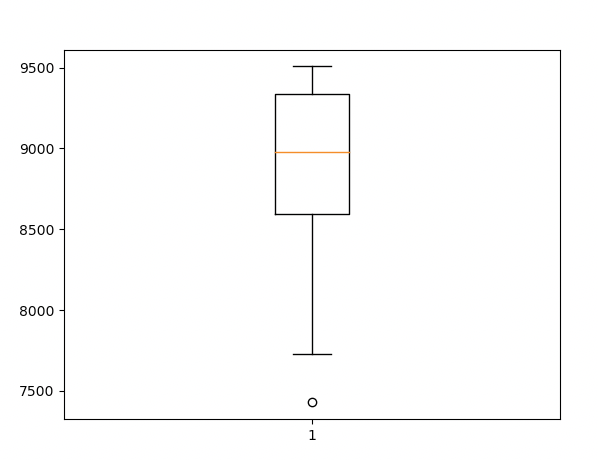
Wykresy liniowe wartości funkcji fitness i najlepszej uśrednionej wartość funkcji fitness dla algorytmu Genetic dla max. 200 iteracji oraz wykresy pudełkowe rozkładu wartości plecaka dla następujących problemów:

**knapPI\_1\_100\_1000\_1**

(ilość elementów w zbiorze danych: 100, największa wartość elementu ze zbioru: 997)

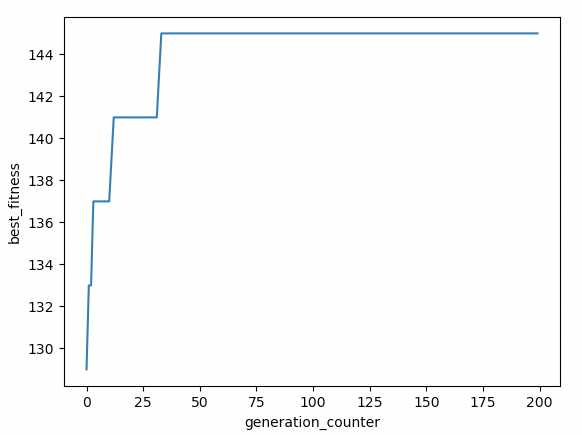
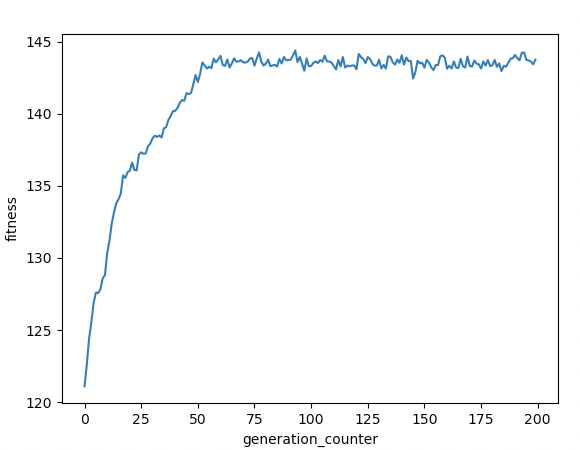


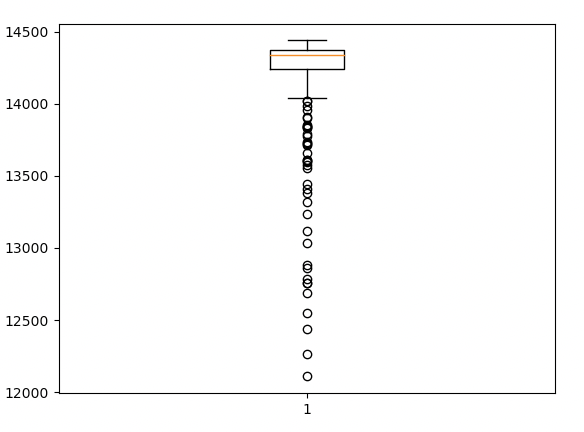
Rozwiązanie dla danego problemu znaleziono w 35-ej iteracji.



**problem-1.cnf**

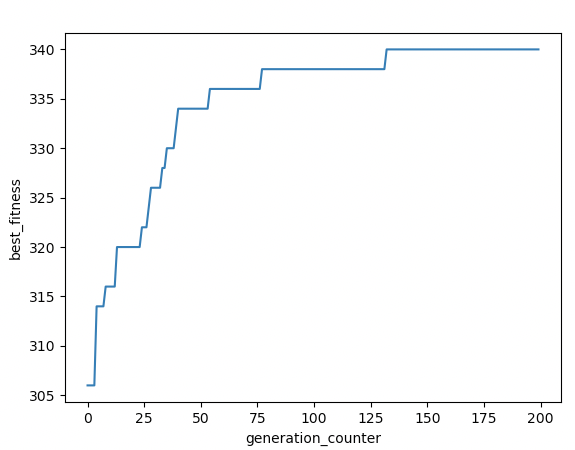
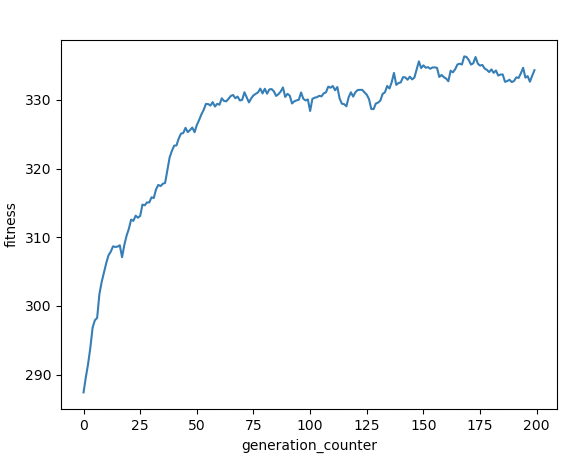
(ilość elementów w zbiorze danych: 162, największa wartość elementów ze zbioru: 27)

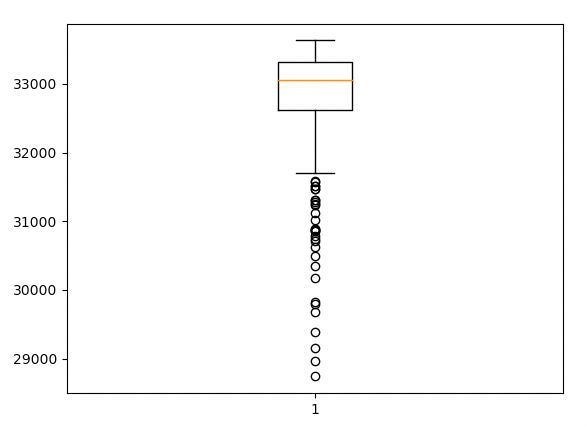




**problem-2.cnf**

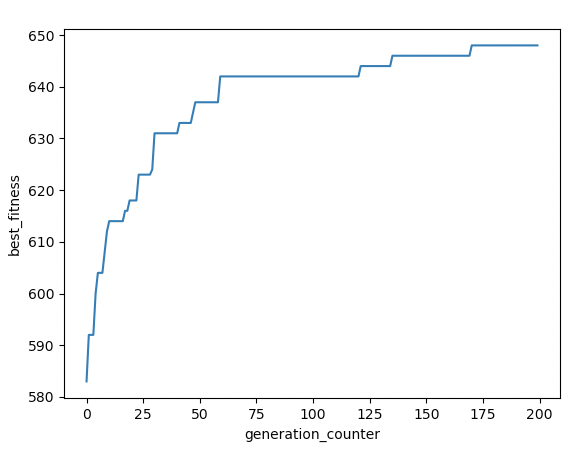
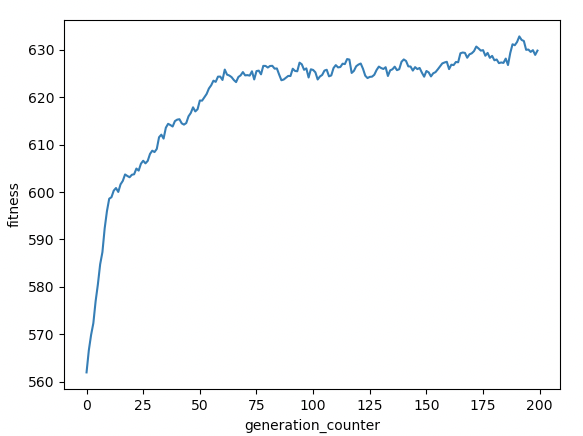
(ilość elementów w zbiorze danych: 384, największa wartość elementów ze zbioru: 64)

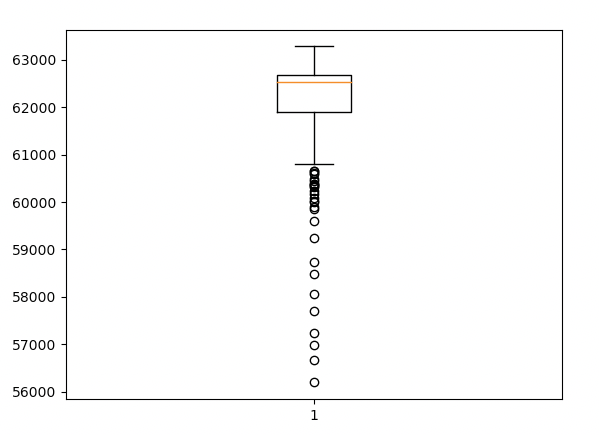




**problem-3.cnf**

(ilość elementów w zbiorze danych: 750, największa wartość elementów ze zbioru: 125)





**Testy**

Czasowe porównanie rozwiązywania problemów.

**Klasyczny algorytm genetyczny**

|  |  |  |
| --- | --- | --- |
| Nazwa problemu | Ilość iteracji | Czas |
| knapPI\_1\_100\_1000\_1 | 45(rozwiązanie znaleziono) | 1.82 |
| problem-1.cnf | 200 | 2.87 |
| problem-2.cnf | 200 | 6.76 |
| problem-3.cnf | 200 | 12.75 |

**PBIL**

|  |  |  |
| --- | --- | --- |
| Nazwa problemu | Ilość iteracji | Czas |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Wnioski**

Podczas przeprowadzenia testów rozwiązywania problemu plecakowego i jego oceny, udało się porównać dokładność i szybkość **klasycznego algorytmu genetycznego** do **PBIL**. Testując cztery problemy plecakowe o różnej wielkości, porównano algorytmy zarówno w zadaniach łatwych, jak i trudnych. Oba algorytmy znalazły optymalne rozwiązanie dla najłatwiejszego problemu(knapPI\_1\_100\_1000\_1), jednak klasyczny algorytm genetyczny stał się dokładniejszy, gdy problemy stawały się coraz trudniejsze (wykresy dla problemów problem-1, problem-2, problem-3). Rozwiązywanie każdego problemu przez klasyczny algorytm genetyczny zajmowało prawie dwa razy więcej czasu, więc okazało się, że jest znacznie mniej efektywny czasowo. Dlatego można stwierdzić, że klasyczny algorytm genetyczny jest wolniejszy, ale też dokładniejszy.

## **Literatura**

[1] Adaptive Algorithms for solving the Knapsack Problem, Sharif Samir Hamed, Marco A. Wiering, 2019

[2] Population-Based Incremental Learning:A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning, Shumeet Baluja, 1994