Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Algorytmy ewolucyjne

Sprawozdanie z projektu nr 2

Karol Borowski

Warszawa, 2019

Spis treści

1. Wstęp

Projekt ma na celu rozwiązanie problemu plecakowego:

$$\max_x \sum_{i=1}^{n} p_i x_i$$

$$\sum_{i=1}^{n} w_i x_i <= W$$

gdzie: $p_i > 0$, $w_i > 0$, $x_i \in \{0, 1\}$, $W = 30\% \sum_{i=1}^{n} w_i$

korzystając z Global Optimization Toolbox (optimtool) znajdującego się w środowisku MATLAB. Wymagane było użycie algorytmu genetycznego, dlatego wykorzystałem funkcję `ga` z powyższego pakietu. Jest to funkcja minimalizująca, dlatego przekształciłem pierwotne założenie problemu plecakowego na

$$\min_x -\sum_{i=1}^{n} p_i x_i$$

Założenie dotyczące rozmiaru plecaka pozostało niezmienione. Algorytm testowałem dla 32 oraz 64 przedmiotów. Głównym celem projektu jest dobranie optymalnych parametrów algorytmu oraz metody selekcji.

2. Dobór optymalnych parametrów algorytmu

Aby algorytm działał zadowalająco, należy dobrać odpowiednie jego parametry jak i metodę selekcji. Za każdym uruchomieniem algorytmu otrzymujemy minimalnie różne wyniki, dlatego też przy każdej zmianie dowolnego parametru, uruchamiałem algorytm 20 razy i brałem średni wynik z otrzymanych iteracji. Kryterium, którym się kierowałem podczas doboru parametrów była wartość funkcji celu oraz czas wykonywania obliczeń.
Podczas testów sprawdziłem poniższe parametry:

— PopulationSize
— Selection Function
— CrossoverFraction
— EliteCount

Nie optymalizowałem kryterium zatrzymania algorytmu, ponieważ według mnie wartość domyślna działa prawidłowo.

2.1. Algorytm domyślny

Domyślny algorytm daje całkiem rozbieżne wyniki, widać to po różnicy między maksymalną, minimalną oraz średnią wartością funkcji celu przy wykonaniu 20 iteracji.

Wyniki dla $N = 32$ przedmiotów:

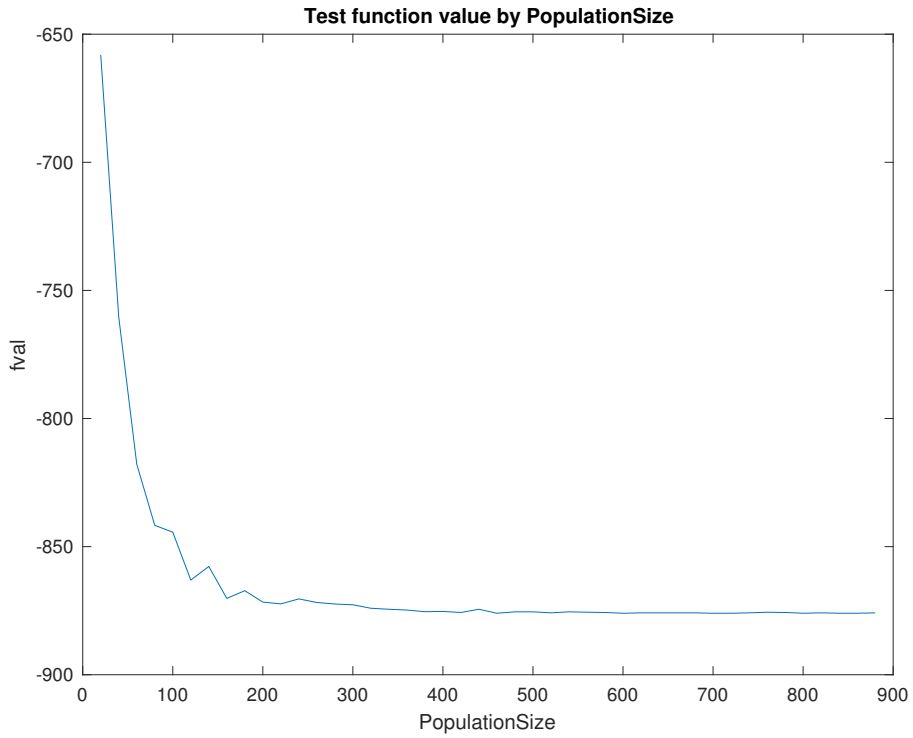
— Najlepszy wynik: -873
— Najgorszy wynik: -826
— Średni wynik: -856.3

Wyniki dla $N = 64$ przedmiotów:

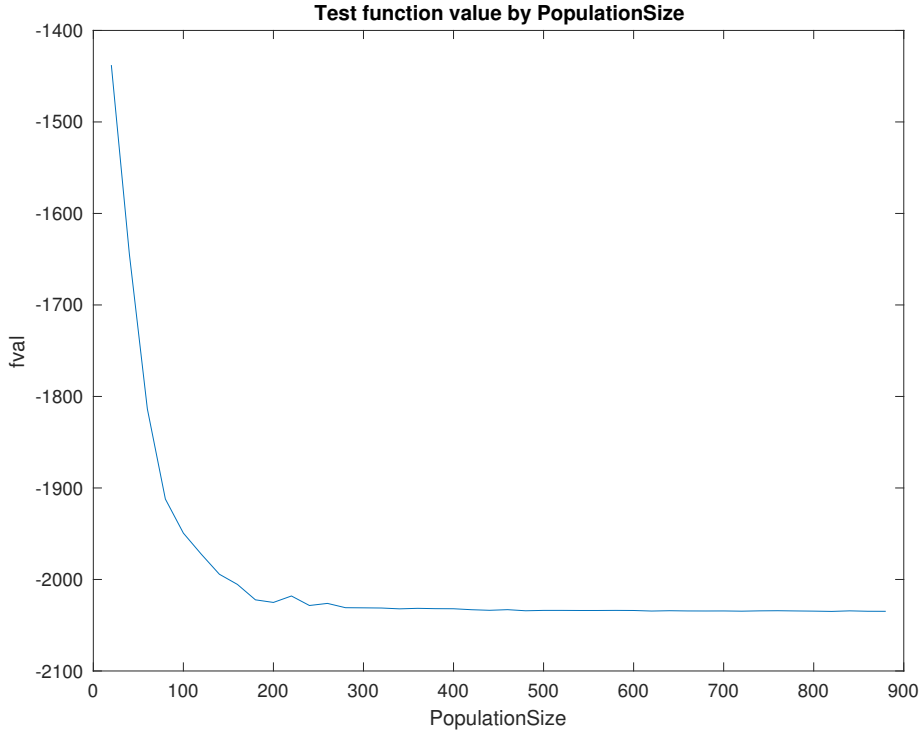
— Najlepszy wynik: -2033
— Najgorszy maksymalna: -1877
— Średni wynik: -1963.5

2.2. Rozmiar populacji

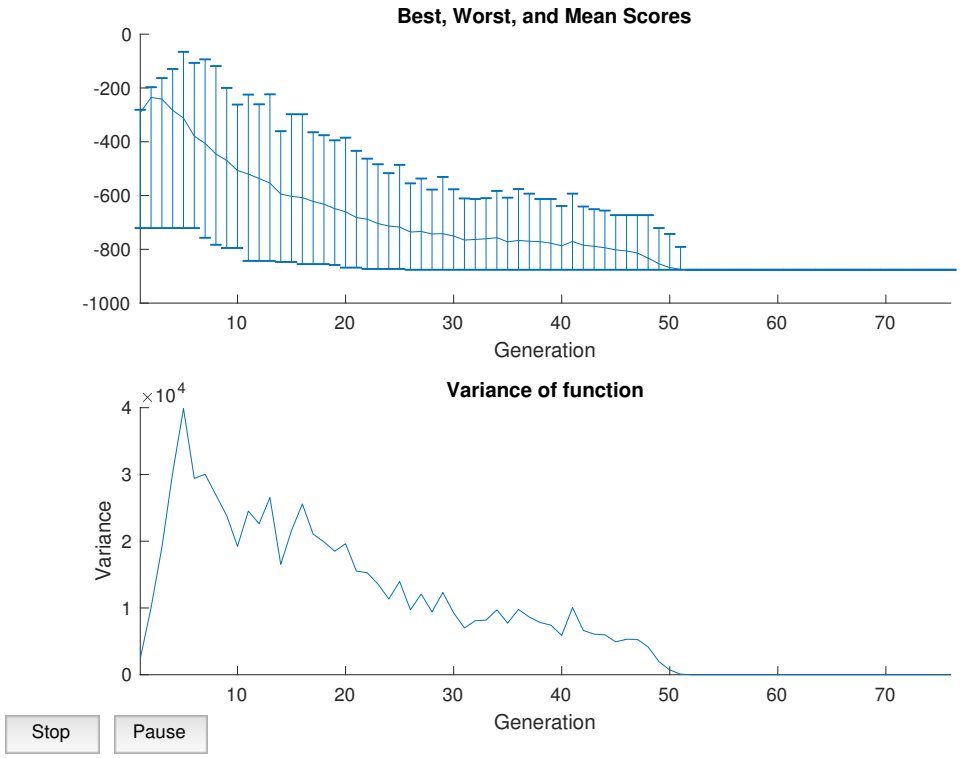
Pierwszym dobranym parametrem i zarazem najbardziej istotnym jest `PopulationSize`, czyli rozmiar populacji. Testowane wartości wynosiły od 20 do 880. Różnica między testowanymi wartościami była równa 20.
Wraz ze wzrostem liczebności populacji rośnie dokładność algorytmu, lecz również czas obliczeniowy. Jako optymalną wielkość populacji uznałem `PopulationSize` = 500 zarówno dla 32 jak i 64 przedmiotów. Taki rozmiar populacji daje wystarczająco dobre wyniki, a dalszy jej wzrost nie daje optymalnych zysków, ponieważ znacząco wydłuża się czas obliczeń.



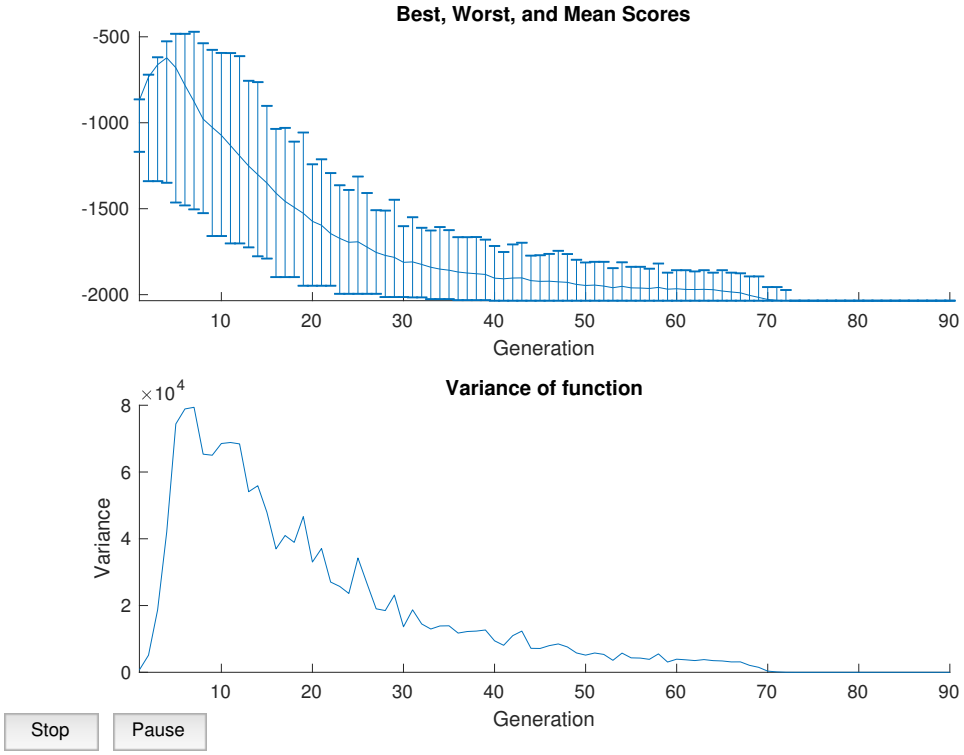
Rysunek 2.1. Zależność funkcji celu od wielkości populacji dla $n = 32$ przedmiotów.



Rysunek 2.2. Zależność funkcji celu od wielkości populacji dla $n = 64$ przedmiotów.



Rysunek 2.3. Uruchomienie algorytmu przy PopulationSize=880 dla $n = 32$ przedmiotów.



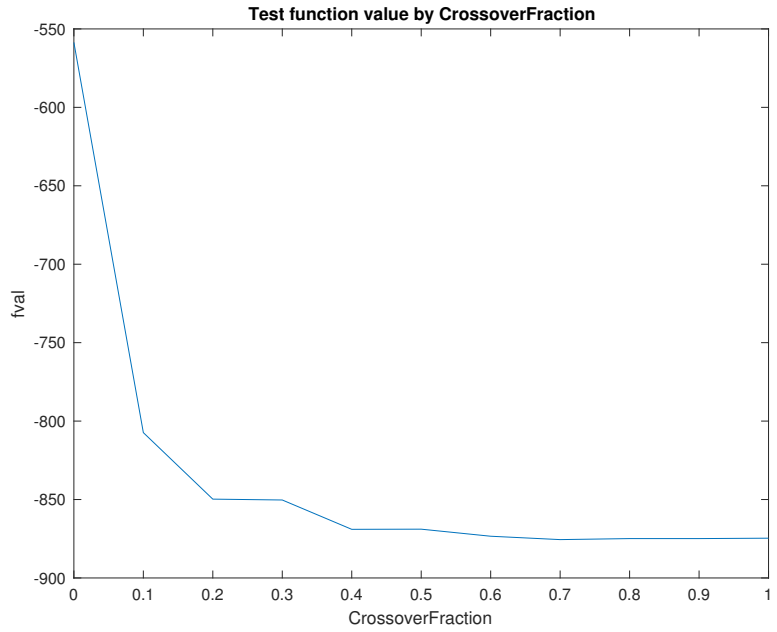
Rysunek 2.4. Uruchomienie algorytmu przy PopulationSize=880 dla $n = 32$ przedmiotów.

2.3. Selection

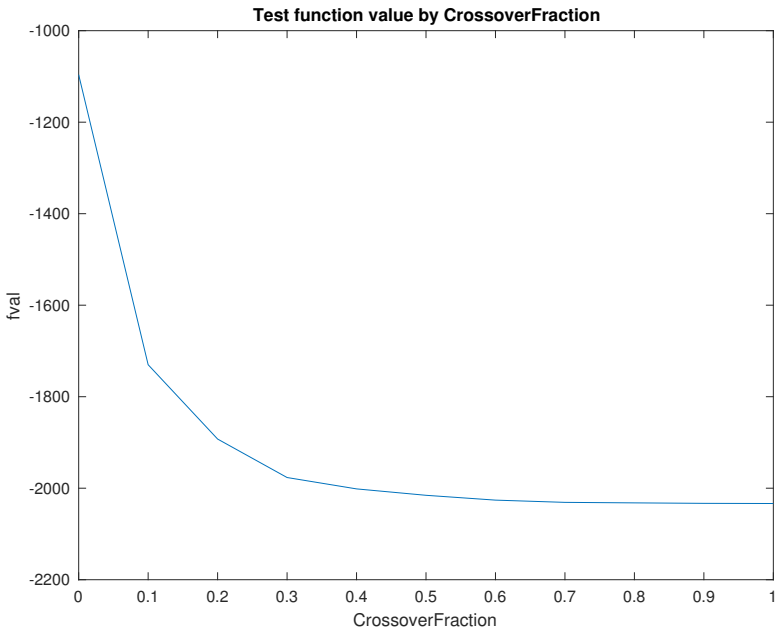
Próbowałem uruchomić algorytm z czterema różnymi metodami selekcji: `Stochastic uniform`, `Remainder`, `Uniform`, `Roulette`. Jedyną działającą metodą była `Stochastic uniform`, ponieważ reszta nie jest przystosowana do algorytmów z ograniczeniami liczbami całkowitymi.

2.4. CrossoverFraction

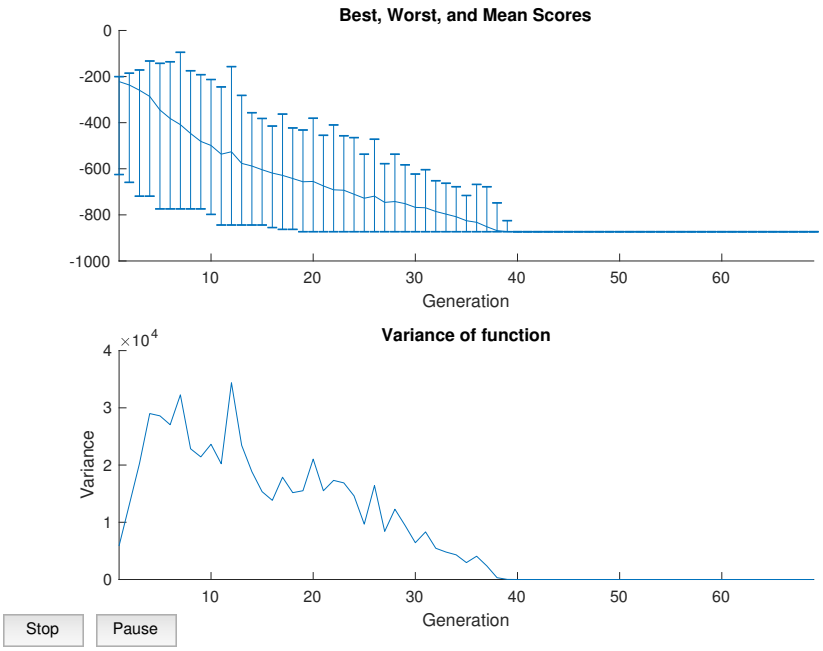
Współczynnik `CrossoverFraction` przyjmuje wartości od 0 do 1. Wraz ze wzrostem powyższego parametru maleje wartośc funkcji celu, więc za optymalną wartość przyjąłem `CrossoverFraction` = 1.



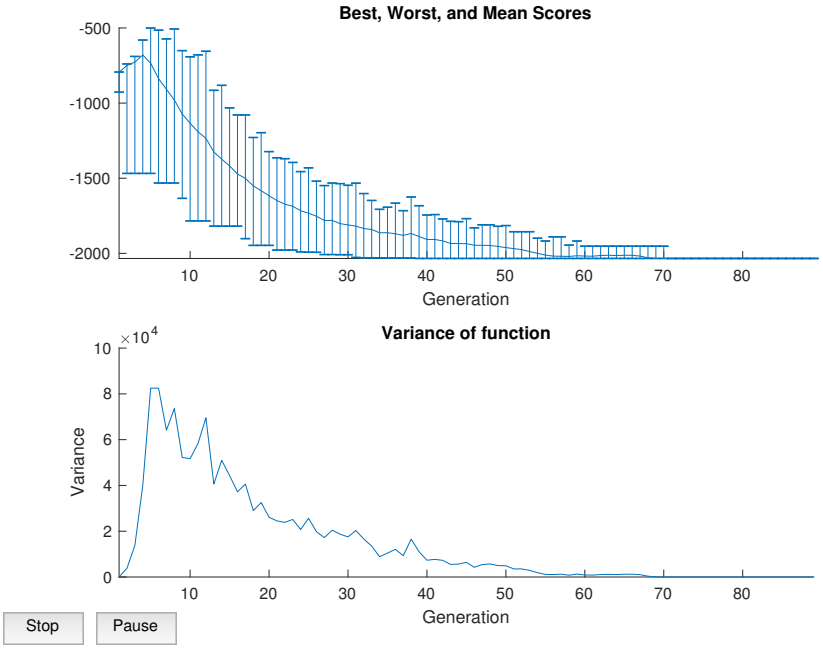
Rysunek 2.5. Zależność funkcji celu od CrossoverFraction dla $n = 32$ przedmiotów.



Rysunek 2.6. Zależność funkcji celu od CrossoverFraction dla $n = 64$ przedmiotów.



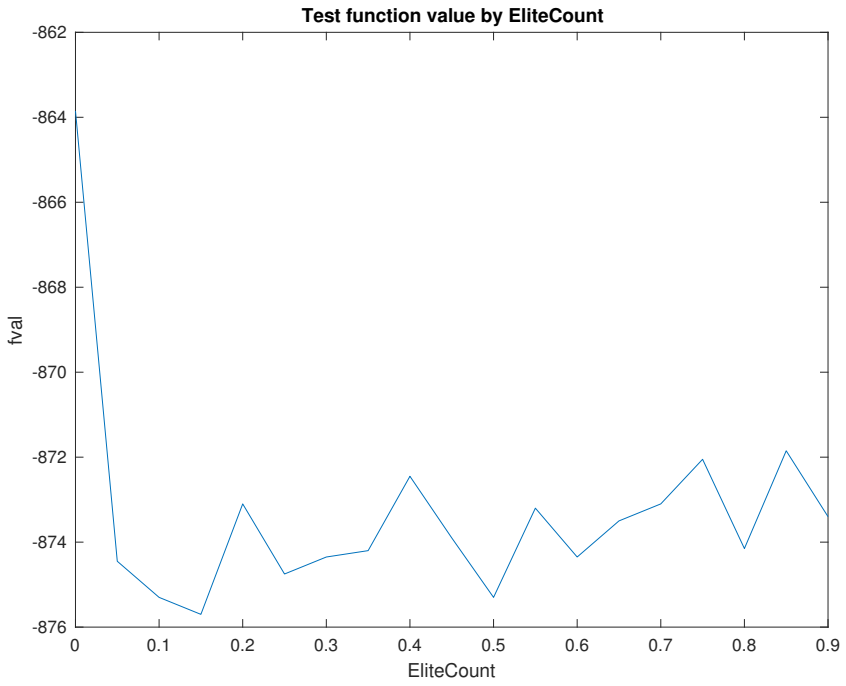
Rysunek 2.7. Uruchomienie algorytmu przy CrossoverFraction=1 dla $n = 32$ przedmiotów.



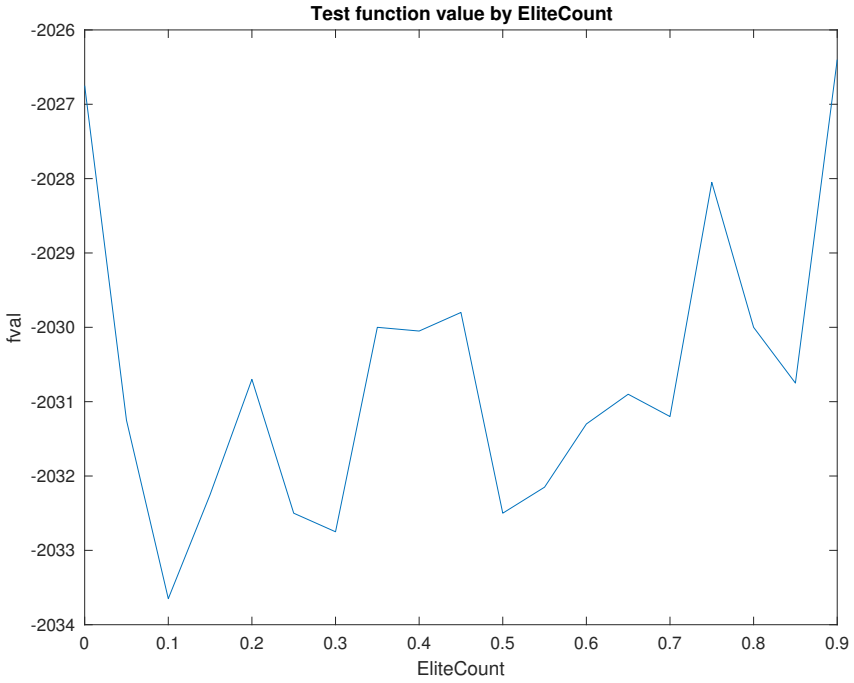
Rysunek 2.8. Uruchomienie algorytmu przy CrossoverFraction=1 dla $n = 64$ przedmiotów.

2.5. EliteCount

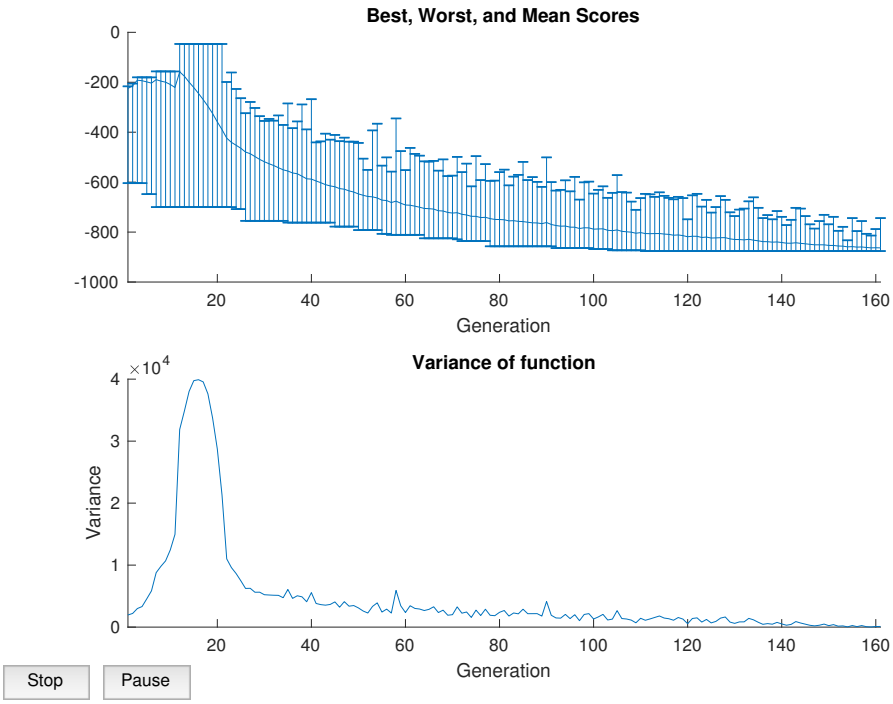
Parametr `EliteCount` odpowiada za ilość elitarnych rozwiązań, które mają zapewnione miejsce w następnym pokoleniu. Nie ma widocznych zależności pomiędzy wartością `EliteCount` a wartością funkcji celu. Zarówno dla 32 jak i 64 przedmiotów, najlepsze wyniki otrzymujemy dla wartości około $\texttt{EliteCount} = 0.1 * PopulationSize$, lecz może być to w pewnym stopniu losowe.



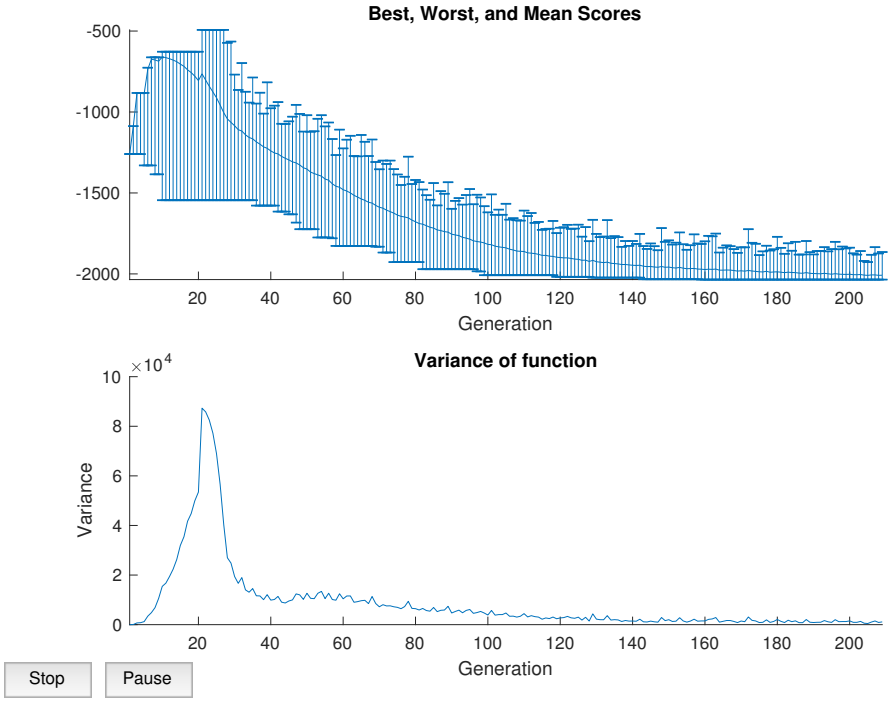
Rysunek 2.9. Zależność funkcji celu od EliteCount dla $n = 32$ przedmiotów.



Rysunek 2.10. Zależność funkcji celu od EliteCount dla $n = 64$ przedmiotów.



Rysunek 2.11. Uruchomienie algorytmu przy $EliteCount = 0.9{*}PopulationSize$ dla $n = 32$ przedmiotów.



Rysunek 2.12. Uruchomienie algorytmu przy $EliteCount = 0.9{*}PopulationSize$ dla $n = 64$ przedmiotów.

3. Algorytm optymalny

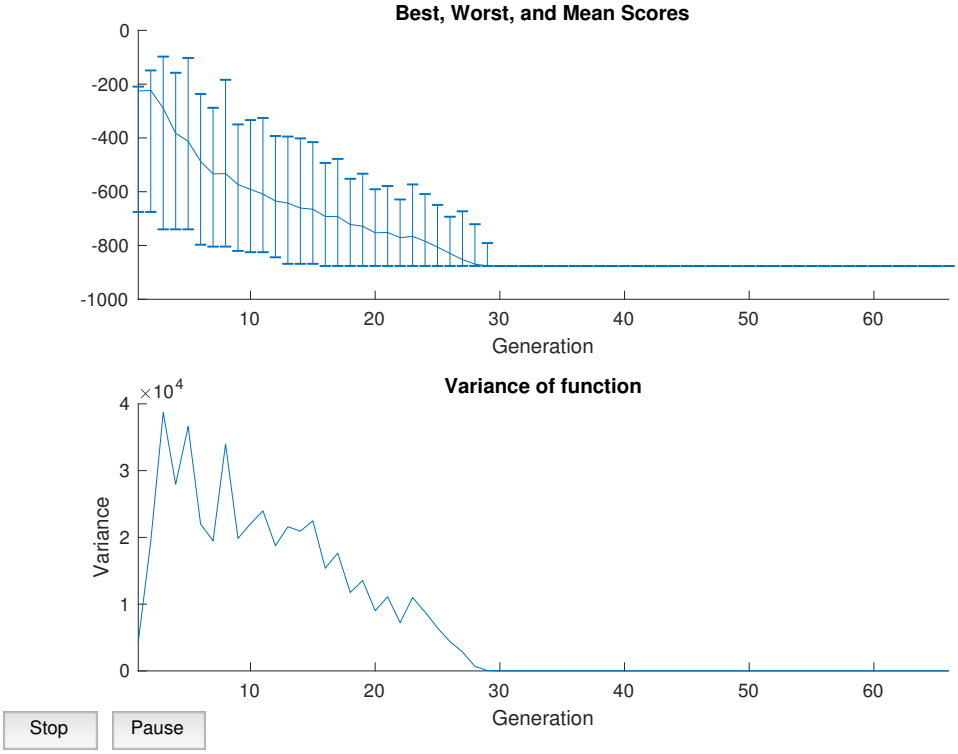
Po przetestowaniu różnych parametrów algorytmu genetycznego wybrałem optymalne wartości:

— PopulationSize = 500
— Selection = Stochastic uniform
— CrossoverFraction = 1
— EliteCount = 10% PopulationSize

Otrzymane wyniki uległy znacznej poprawie, niż w przypadku domyślnego algorytmu. Poniżej zostały zaprezentowane wyniki dla kolejno 32 i 64 przedmiotów.

3.1. N=32 przedmiotów

— Najlepszy wynik: -876
— Najgorszy wynik: -873
— Średni wynik: -875.85



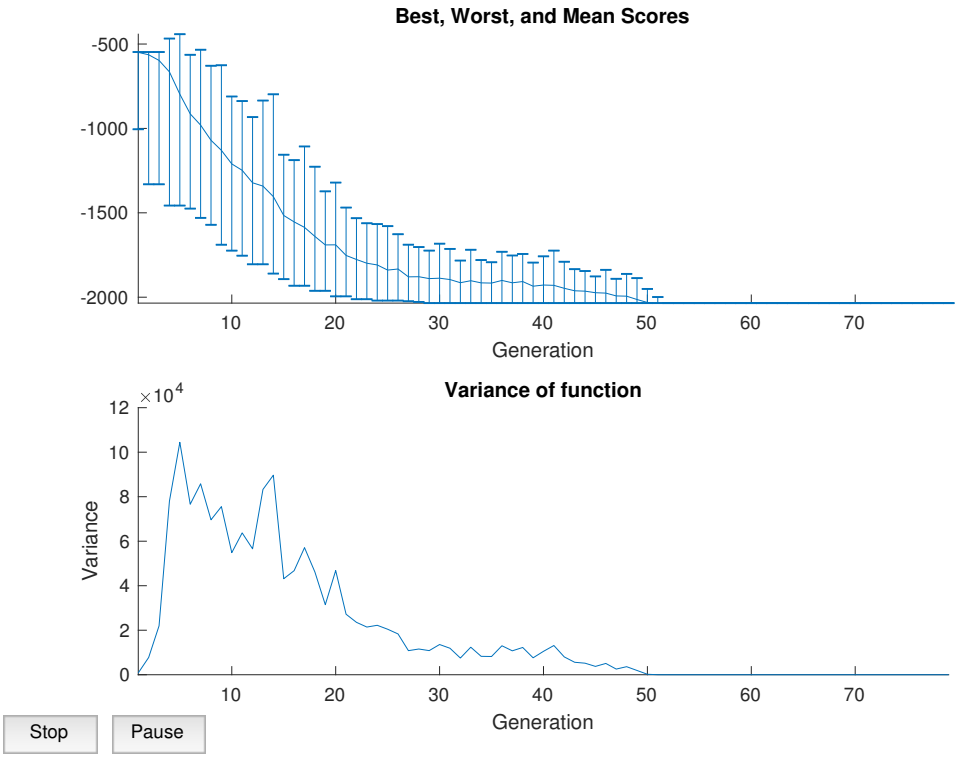
Rysunek 3.1. Problem plecakowy dla $n = 32$ przedmiotów.

Tablica 3.1: Wykaz N=32 przedmiotów

i	p_i	w_i	In Sack
1	10	0,1	0
2	74	0,9	0
3	70	0,7	1
4	56	0,2	1
5	56	0,9	0
6	18	0,3	0
7	52	0,8	0
8	11	0,3	0
9	92	0,8	1
10	13	1	0
11	6	0,7	0
12	27	1	0
13	18	0,4	0
14	6	0,9	0
15	14	0,5	0
16	50	0,1	1
17	96	0,1	1
18	48	0,4	1
19	91	0,7	1
20	21	0,7	0
21	87	0,2	1
22	19	0,3	0
23	85	0,8	1
24	2	0,8	0
25	67	0,8	0
26	51	0,7	0
27	3	0,9	0
28	77	0,8	0
29	72	0,5	1
30	30	0,3	0
31	90	0,9	1
32	39	0,1	1

3.2. N=64 przedmiotów

— Najlepszy wynik: -2035
— Najgorszy wynik: -2034
— Średni wynik: -2034.75



Rysunek 3.2. Problem plecakowy dla $n = 64$ przedmiotów.

Tablica 3.2: Wykaz N=64 przedmiotów

i	p_i	w_i	In Sack
1	7	0,1	0
2	74	0,9	0
3	24	0,7	0
4	94	0,2	1
5	57	0,9	0
6	58	0,3	1
7	19	0,8	0
8	4	0,3	0
9	14	0,8	0
10	32	1	0
11	24	0,7	0
12	86	1	1
13	69	0,4	1
14	73	0,9	0
15	94	0,5	1
16	17	0,1	1
17	7	0,1	0
18	45	0,4	1
19	34	0,7	0
20	35	0,7	0
21	3	0,2	0
22	85	0,3	1
23	42	0,8	0
24	38	0,8	0

Tablica 3.2: Wykaz N=64 przedmiotów

i	p_i	w_i	In Sack
25	6	0,8	0
26	44	0,7	0
27	79	0,9	0
28	8	0,8	0
29	92	0,5	1
30	29	0,3	1
31	85	0,9	1
32	94	0,1	1
33	18	0,2	0
34	5	0,8	0
35	62	0,7	1
36	33	0,6	0
37	99	0,6	1
38	76	0,3	1
39	27	0,6	0
40	57	0,2	1
41	55	0,9	0
42	87	0,2	1
43	91	0,1	1
44	88	0,3	1
45	53	0,3	1
46	72	0,1	1
47	80	0,2	1
48	4	0,5	0
49	71	1	0
50	13	0,5	0
51	1	0,9	0
52	26	0,3	0
53	59	0,9	0
54	7	0,3	0
55	19	0,9	0
56	80	0,1	1
57	88	0,7	1
58	93	0,6	1
59	79	0,1	1
60	3	0,8	0
61	48	0,7	0
62	34	0,4	0
63	82	0,9	1
64	33	0,4	0

4. Wnioski

Dobranie parametrów algorytmu przyniosło oczekiwane efekty, otrzymany algorytm prezentuje znacznie lepsze wyniki niż wersja domyślna. Zarówno dla 32 jak i 64 przedmiotów zdarza się, że algorytm poda wynik, który mógłby być lepszy, lecz są to rzadkie przypadki, co możemy zauważyć patrząc na średnie wyniki kilku iteracji algorytmu.
Jeśli chodzi o działanie algorytmu GA dla 32 i 64 przedmiotów, to mogę stwierdzić, że działa on minimalnie lepiej dla większej liczby przedmiotów. Dla $N = 32$ częściej zdarza się wyliczenie wyniku, który odbiega od optymalnej wartości.