Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Algorytmy Ewolucyjne

Projekt 2

Sofiya Makarenka

Spis treści

1. Treść zadania

Stosując algorytm genetyczny znajdź rozwiązanie problemu plecakowego:

$$\max_{\mathbf{x}} \sum_{i=1}^{n} p_i x_i \qquad \sum_{i=1}^{n} w_i x_i \le W \qquad p_i > 0 \ , \ w_i > 0 \ , \ x_i \in \{0,1\}$$

Założenia:

- liczba przedmiotów: n = 32
- do generacji przedmiotów wykorzystać Skrypt 1; w przypadku wykonania projektu w parze należy wybrać niższy numer albumu; wagi w przedmiotów są losowane z rozkładem równomiernym z przedziału <0.1, 1> z dokładnością do 0.1, a wartości p przedmiotów są losowane z rozkładem równomiernym z przedziału: <1,100> z dokładnością do 1
- maksymalna waga plecaka: W = 30% wagi wszystkich przedmiotów
- dozwolone jest korzystanie ze środowiska MATLAB wraz z dodatkiem Global Optimization Toolbox (optimtool). Wykonanie projektu w Pythonie wymaga uprzedniej konsultacji z prowadzącym projekt.

Dobrać optymalne parametry algorytmu i metodę selekcji

Wyniki przedstaw w postaci sprawozdania (plik programu MS Word lub PDF) z wynikami obliczeń. Sprawozdanie powinno zawierać:

- Wektor binarny stanowiący rozwiązanie problemu wraz z sumaryczną wagą i wartością przedmiotów w plecaku.
- Wartości : o liczności populacji, o liczności potomków elitarnych, skrzyżowanych i zmutowanych, o prawdopodobieństw mutacji
- Kryteria doboru optymalnych parametrów, np. warunku zatrzymania algorytmu
- Dla każdego uruchomienia wykres wartości funkcji celu (min., śr., max., wariancja) w funkcji numeru pokolenia.
- Sprawozdanie nie powinno zawierać niepotrzebnych informacji – takich jak np. teoria i opis metod optymalizacji.

2. Skrypt generujący

```
numerAlbumu=304135;
rng(numerAlbumu);
N=32; items(:,1)=round(0.1+0.9*rand(N,1),1);
items(:,2)=round(1+99*rand(N,1));
```

3. Wektor binarny stanowiący rozwiązanie problemu

Rozwiązanie problemu plecakowego 0/1 można znaleźć wykorzystując algorytm rekurencyjnie dynamiczny. Skrypt, który został użyty podczas wyszukiwania rozwiązania znajduje się w pliku recursive_dynamic.py.

Rozwiązanie problemu za pomocą programowania dynamicznego jest przedstawione w następującym wektorze*:* `[1 1 0 0 0 1 1 0 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 1 0 0 1 0 0 0 1 1]`
Waga plecaka wynosi: `4.9;`
Wartość przedmiotów w plecaku: `981;`

Innym wariantem jest ustawienie “złych” parametrów algorytmu genetycznego (np. %mutacja - 50%) i przyrównania warunków zatrzymania do nieskończoności (nieskończenie wiele pokoleń, nieskończony czas itd., w przypadku matlaba są to wartości bardzo duże ale nie nieskończone).

Do tego jeszcze możemy dołożyć algorytm brute force, ale niestety nie starczyło nam cierpliwości, aby doczekać się końca działania (obliczania) algorytmu, więc nie wykorzystałyśmy tej metody.

4. Warunki zatrzymania algorytmu genetycznego

Skoro wiemy jakie jest rozwiązanie dla naszego przypadku, to możemy wprowadzić kilka zmian do warunków zatrzymania algorytmu. W pierwszej kolejności chodzi nam o parametr `FitnessLimit,` niech ten parametr będzie się równał 981 (przyjmujemy tę wartość, ponieważ takie jest rozwiązanie). Przy takim założeniu, możemy uniknąć zbędnego zwiększenia kosztów obliczeniowych. Zmaleje nam liczba pokoleń przy założeniu, że algorytm się zatrzymuje jak tylko znajdzie rozwiązanie.

Zmieniamy również liczbę maksymalnej liczby generacji i stall generacji. Dążymy do tego, aby nasz algorytm zatrzymywał się wtedy i tylko wtedy, gdy znajdzie się w rozwiązaniu, bo chcemy porównać koszta obliczeniowe. Z tego powodu zwiększamy te dwie ostatnie wartości. Matlab nie pozwala wpisać wartości nieskończonych, więc wpisujemy po prostu bardzo duże liczby względem defaultowych :

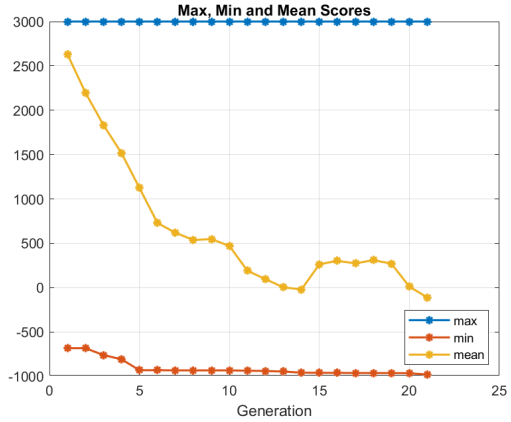
- `MaxGenerations = 1000000.` Maksymalna liczba pokoleń.
- `MaxTime = Inf.` Brak limitu czasu wyszukiwania rozwiązania.
- `FitnessLimit = 981.`
- `MaxStallGenerations = 1000000.` Algorytm zatrzymuje się, jeśli średnia względna zmiana najlepszej wartości funkcji w ciągu MaxStallGenerations jest mniejsza lub równa `FunctionTolerance = 1e-6.`
- `MaxStallTime = Inf.` Algorytm się zatrzymuje, kiedy wartość fitness funkcji się nie zmienia przez czas równy MaxStallTime.
- `StallTest = average change.` Kryterium utknięcia pokolenia jest średnią zmienności funkcji celu.

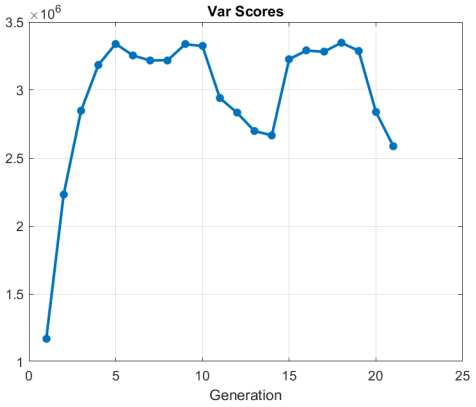
Po pierwszej próbie wyszukiwania rozwiązania z założonymi wcześniej warunkami zatrzymania, na wyjściu uzyskujemy następujące wartości:

Wektor stanowiący rozwiązanie problemu: `[1 1 0 0 0 1 1 0 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 1 0 0 1 0 0 0 1 1]`
Wartość plecaku: `981`
Waga plecaku: `4.9`

Koszt obliczeniowy = `NumberOfGenerations * PopulationSize = 21 * 200 = 4200`

*PopulationSize równa się defaultowo 200, bo liczba zmiennych funkcji fitness jest większa od 5.





Skrypt został uruchomiony 3 razy, bo pamiętamy, że poszukiwania odbywają się na zasadzie losowości.

Widzimy, że udało nam się znaleźć poprawne rozwiązanie, na razie nie wiemy o wielkości wartości kosztu, będziemy próbowały go zmniejszyć w następnych modyfikacjach opcji naszego algorytmu.

Wartość maksymalna pozostaje bez zmian, co wynika ze sposobu zaimplementowania naszej funkcji celu (fitness). Ten fakt, wskazuje nam na to, że w każdej generacji znajdował się plecak, którego masa była wyższa od maksymalnej wagi, która może się tam zmieścić.

Wartość średnia i minimalna maleje, co też pokrywa się z tym czego oczekujemy, skoro algorytmy optymalizacji matlaba szukają minimum.

5. Dobór liczności populacji

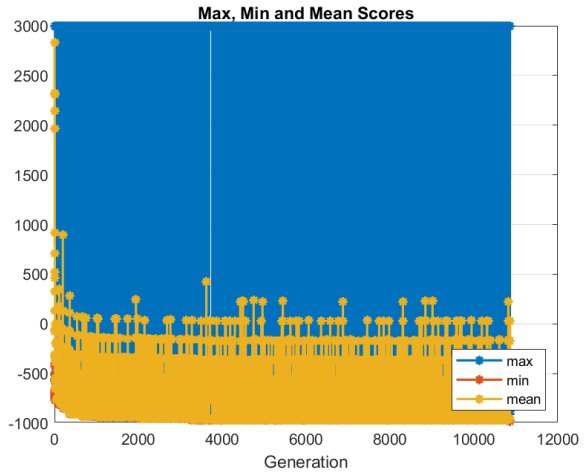
Wartość plecaka dla każdego przypadku pozostaje bez zmian - 981, bo takie mamy warunki zatrzymania algorytmu. Jedynie co się zmienia to koszt obliczeniowy.

Liczność populacji	Koszt obliczeniowy
20	217300
50	415750
200	4200
205	3075
1000	11000
2000	16000

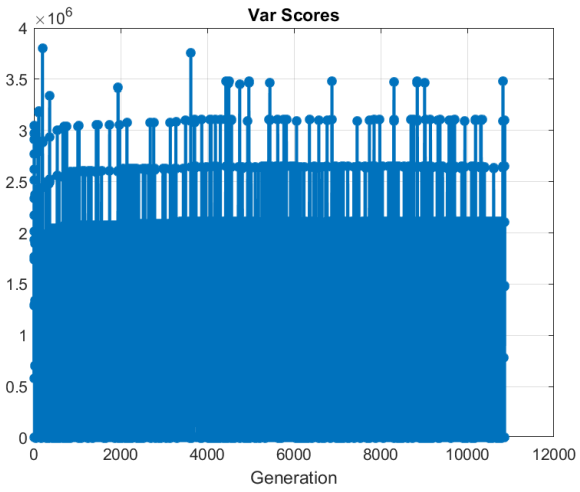
Wektor stanowiący rozwiązanie problemu(dla każdej liczności):

[1 1 0 0 0 1 1 0 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 1 0 0 1 0 0 0 1 1]

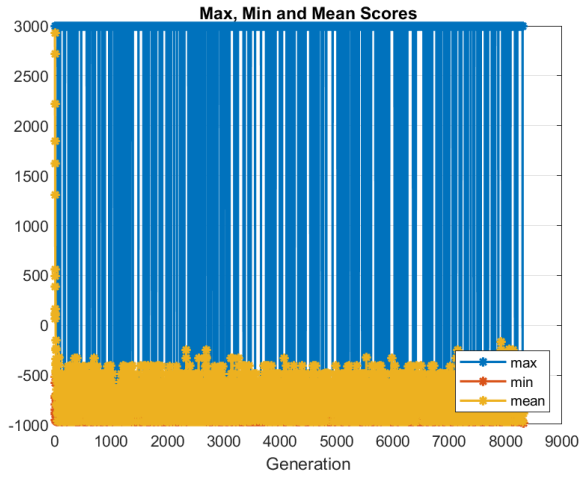
liczność populacji = 20



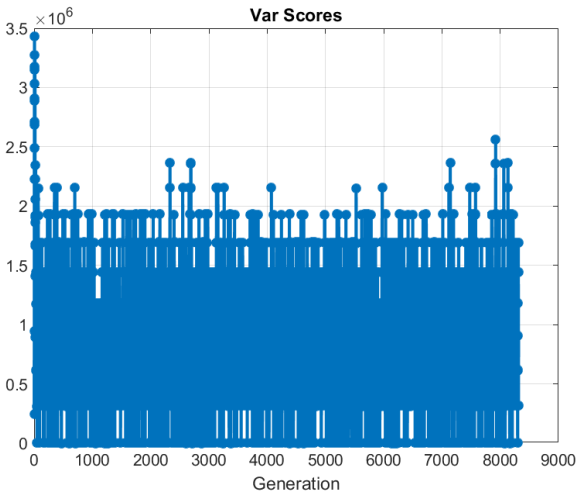
liczność populacji = 20



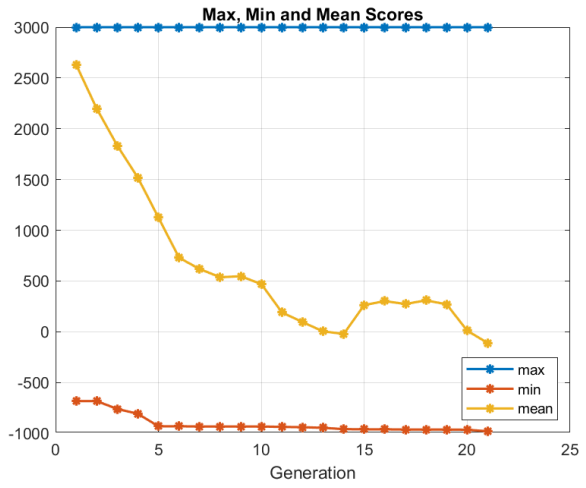
liczność populacji = 50



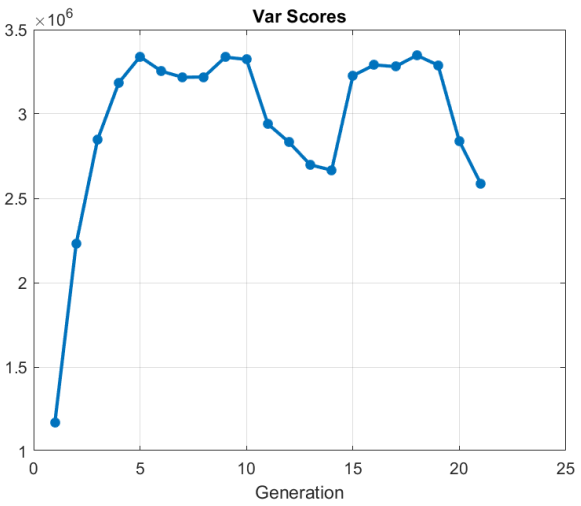
liczność populacji = 50



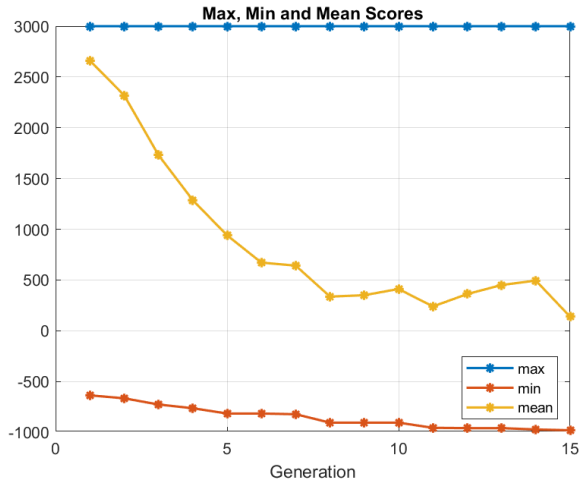
liczność populacji = 200



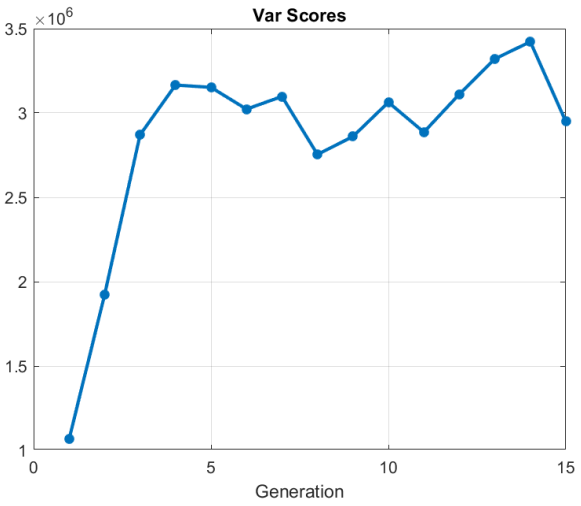
liczność populacji = 200



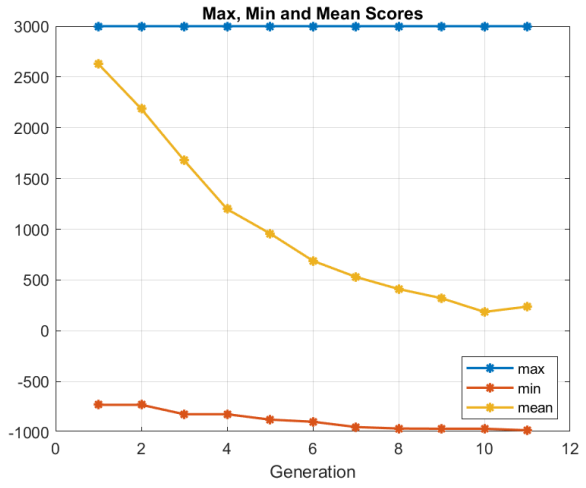
liczność populacji = 205



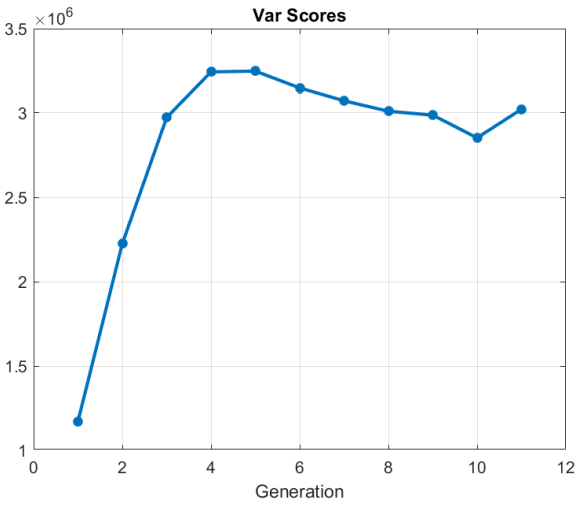
liczność populacji = 205



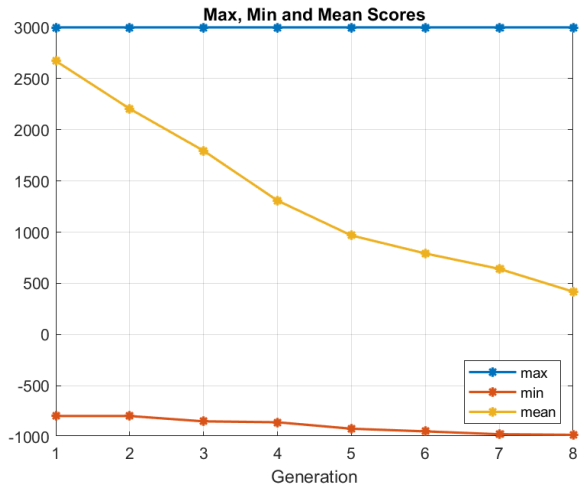
liczność populacji = 1000



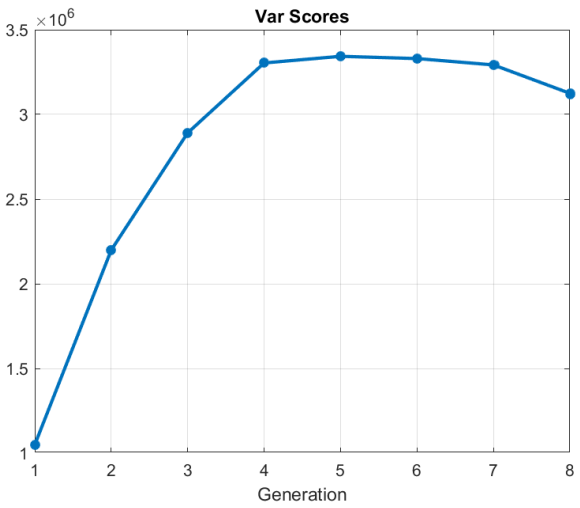
liczność populacji = 1000



liczność populacji = 2000



liczność populacji = 2000



Widzimy, że w przypadku populacji o licznościach 20 i 50 osobników uzyskujemy bardzo duży koszt obliczeniowy. Dzieje się tak dlatego, że liczba pokoleń, która jest nam potrzebna żeby osiągnąć minimum globalne drastycznie wzrasta (dla 20 i 50 osobników to odpowiednio 10865 i 8315 pokoleń).

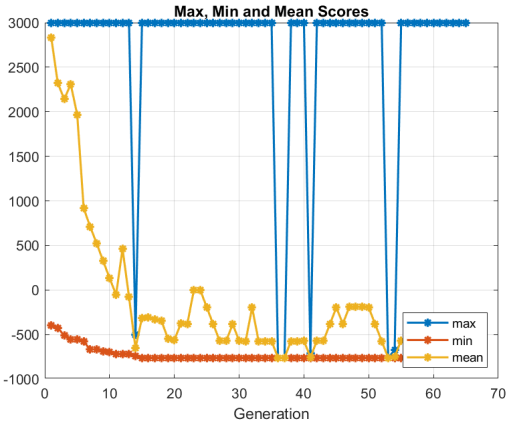
Algorytm dla dużej liczności populacji jak najbardziej znajduje poprawne rozwiązanie(potrzeba jest nawet mniej pokoleń, niż przy warunkach defaultowych), ale cały koszt i tak nam wzrasta, bo wzrasta również liczność.

Balans pomiędzy liczebnością populacji i liczbą pokoleń znalazłyśmy dla populacji *205 osób*.

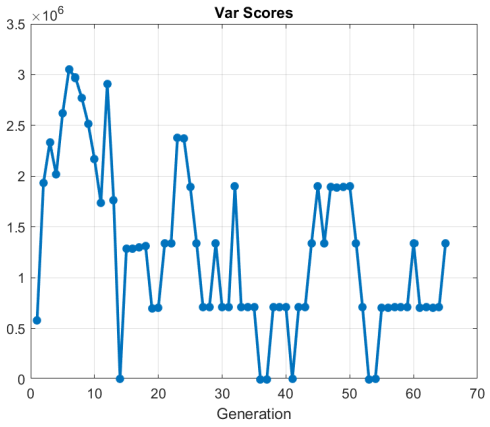
Spróbowałyśmy również wrócić do defaultowych wartości warunków zatrzymania, żeby zbadać zachowanie algorytmu dla małej liczności populacji. Zgodnie z oczekiwaniami, algorytm działa szybciej, ale zatrzymuje się w minimum lokalnym, nie osiągając poprawnego rozwiązania.

Wektor stanowiący rozwiązanie problemu: `[1 0 0 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 0 0 1 0 1 1 0 1 0 1 0 0 0 1]`
Wartość plecaku: `981`
Waga plecaku: `4.9`
Koszt obliczeniowy `= 1300`

liczność populacji = 20



liczność populacji = 20



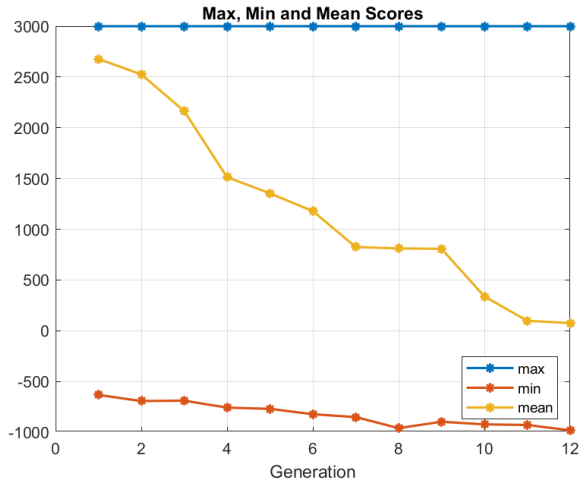
6. Dobór liczności potomków elitarnych

Defaultowo liczba potomków elitarnych równa się `ceil(0.05*PopulationSize)`, dla naszego przypadku wynosi ona 11. W pierwszej kolejności rozpatrzymy przypadek, gdy jedynym warunkiem zatrzymania jest znalezienie poprawnego rozwiązania:

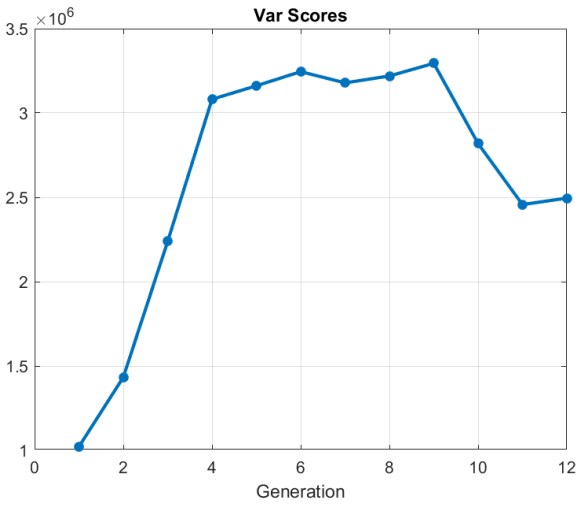
Liczność potomków elitarnych	Koszt obliczeniowy
0(0%)	2460
3(1%)	721395
11(5%)	3075
21(10%)	17400605
82(40%)	1202735
144(70%)	10595220

Wektor stanowiący rozwiązanie problemu(dla każdej liczności):
`[1 1 0 0 0 1 1 0 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 1 0 0 1 0 0 0 1 1]`

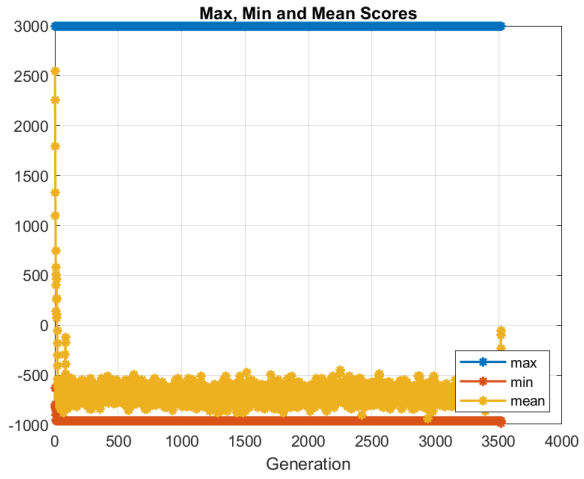
liczność potomków elitarnych = 0



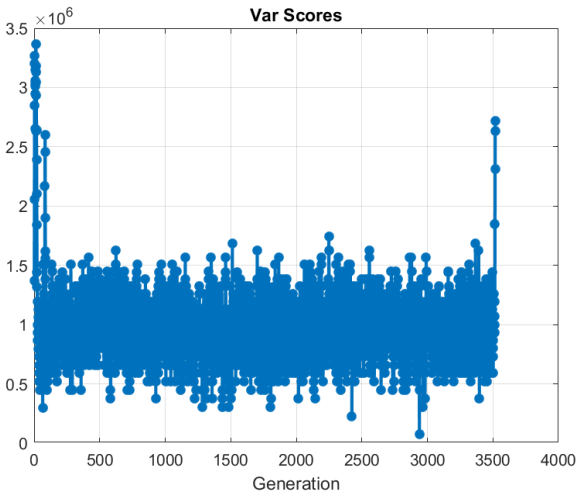
liczność potomków elitarnych = 0



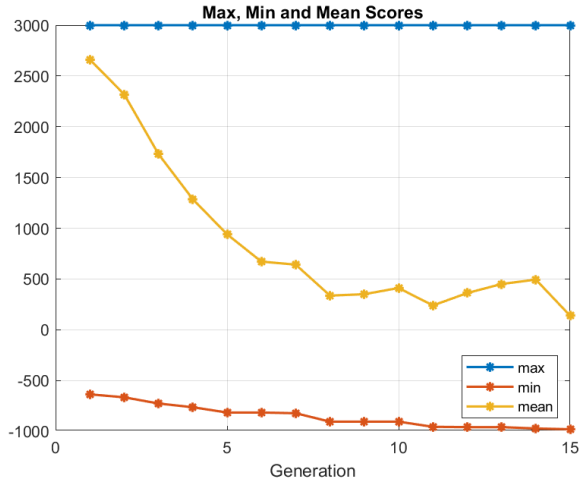
liczność potomków elitarnych = 3



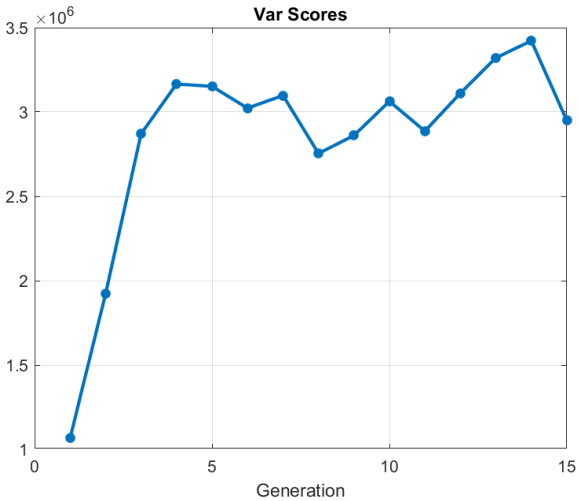
liczność potomków elitarnych = 3



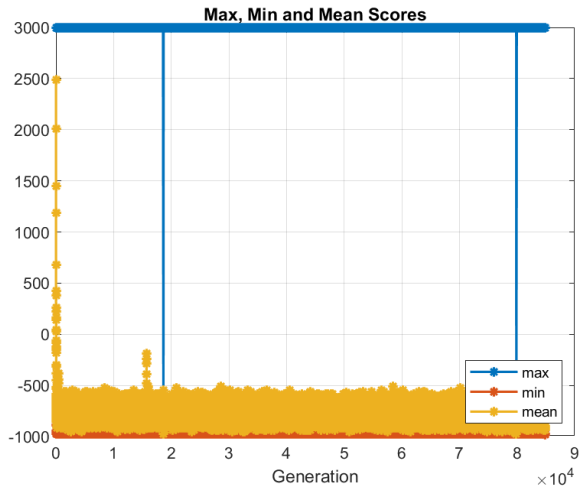
liczność potomków elitarnych = 11



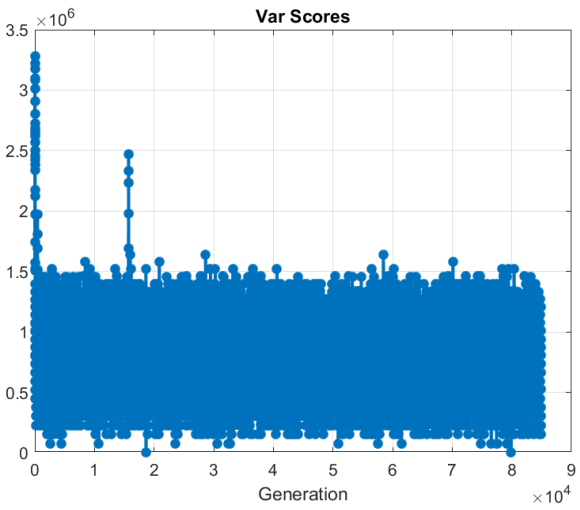
liczność potomków elitarnych = 11



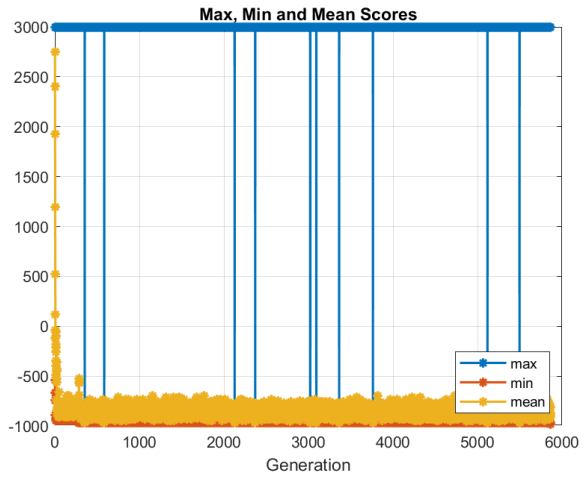
liczność potomków elitarnych = 21



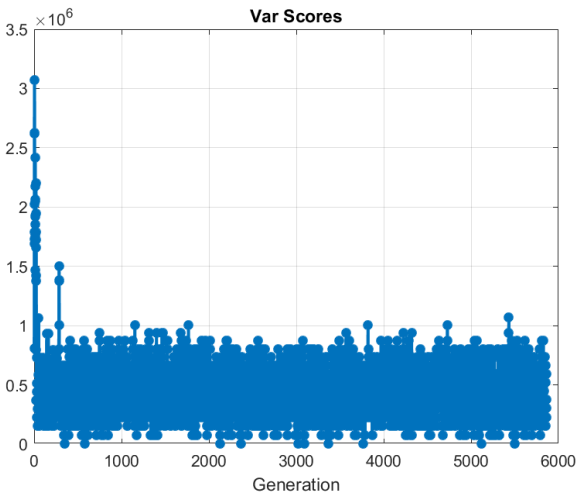
liczność potomków elitarnych = 21



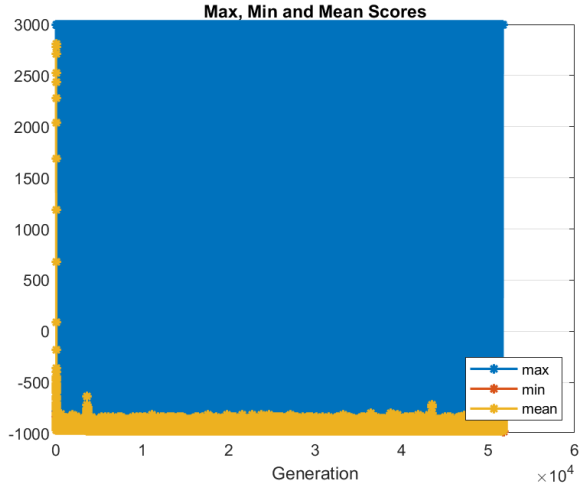
liczność potomków elitarnych = 82



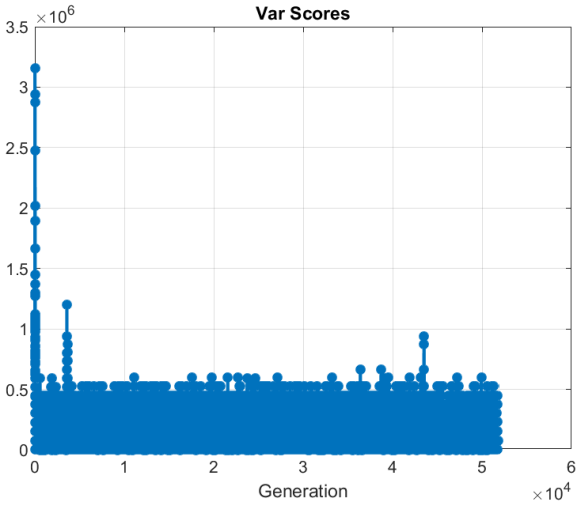
liczność potomków elitarnych = 82



liczność potomków elitarnych = 144



liczność potomków elitarnych = 144



Algorytm działa najszybciej wtedy, gdy nie przenosimy żadnych osobników do następującego pokolenia. Dla defaultowej liczności potomków elitarnych algorytm także działa dobrze. Dla reszty, jak widzimy, algorytm działa bardzo wolno, wartość średnia i minimalna ledwie się zmieniają. W przypadku, gdyby nie było warunku zatrzymania FitnessLimit, algorytm dużo szybciej skończyłby swoje działanie, utykając w minimum lokalnym.

Teraz spróbujmy zbadać algorytm, przywracając wartość FitnessLimit do defaultowej.

Liczność potomków elitarnych	Koszt obliczeniowy	Wartość maksymalna plecaka
0(0%)	12710	981
3(1%)	12710	957
11(5%)	13325	981
21(10%)	14760	973
82(40%)	13120	947
144(70%)	18245	968

Widzimy, że dla defaultowych warunków zatrzymania algorytmu, koszty obliczeniowe rosną wraz procentem osób elitarnych, bo zwiększa się liczba pokoleń potrzebnych do znalezienia rozwiązania. Tylko, że poprawnego rozwiązania i tak nie jesteśmy w stanie znaleźć, bo wartość minimalna przestaje zauważalnie się zmieniać (ciągle wybieramy wielu osobników o lepszej wartości funkcji celu w danym pokoleniu) i algorytm zatrzymuje się w minimum lokalnym.

7. Dobór liczności potomków zmutowanych i skrzyżowanych

W tym poszukiwaniu wróciłyśmy do defaultowych warunków zatrzymania algorytmu, bo czas potrzebny na zakończenie działania algorytmu był zbyt długi. Skoro zmieniłyśmy warunki zatrzymania, to usunęłyśmy kolumnę z kosztem obliczeniowym, bo dla większość przypadków i tak nie będziemy znajdowały dobrego rozwiązania, więc ten koszt już na nic nie wskazuje.

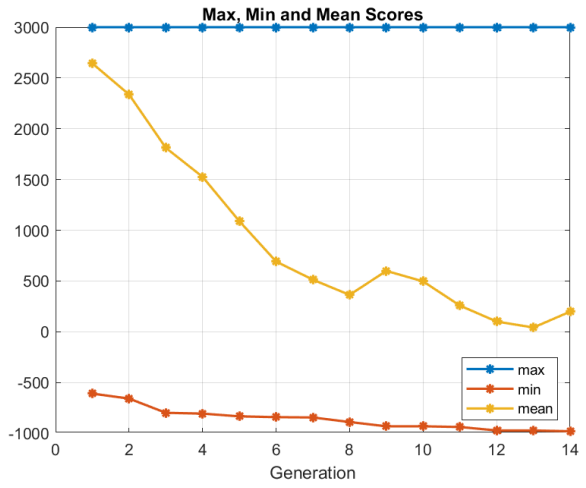
Zakładając, że mamy 205 osobników w populacji i że mamy 11 elitarnych potomków, to pozostała część populacji 205 - 11 = 194 są osobniki zmutowane i skrzyżowane:

Liczność potomków zmutowanych	Liczność potomków skrzyżowanych	Liczność potomków elitarnych
0(0%)	194(100%)	11(5%)
2(1%)	192(99%)	11(5%)
10(5%)	184(95%)	11(5%)
29(10%)	165(85%)	11(5%)
38(20%)	156(80%)	11(5%)
48(25%)	146(75%)	11(5%)
59(30%)	135(70%)	11(5%)
158(70%)	36(30%)	11(5%)
164(85%)	30(15%)	11(5%)
194(100%)	0(0%)	11(5%)

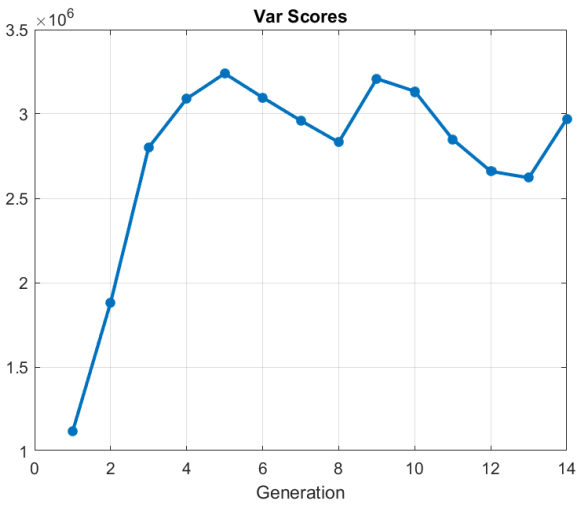
Wektory stanowiące rozwiązanie problemu(pogrubiona czcionka - złe rozwiązanie):

[1 1 0 0 0 1 1 0 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 1 0 0 1 0 0 0 1 1]
[1 1 0 0 0 1 1 0 1 1 0 0 1 0 0 1 1 1 0 0 1 0 1 1 0 0 0 0 0 0 0 1]
[1 1 0 0 0 1 1 0 1 1 0 0 1 0 0 1 1 1 0 0 1 0 1 1 0 0 0 0 0 0 0 1]
[1 1 0 0 0 1 0 0 1 1 0 0 0 0 0 1 1 1 0 0 1 0 1 1 0 1 1 0 0 0 0 1]
[1 1 0 0 0 1 1 0 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 1 0 0 1 0 0 0 1 1]
[1 0 1 0 0 1 1 0 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 1 0 1 1 0 0 0 1 1]
[1 1 0 0 0 1 1 0 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 1 0 0 1 0 0 0 1 1]
[1 1 0 0 0 1 0 0 1 0 0 0 1 0 0 1 1 1 1 0 1 0 1 1 0 0 0 0 0 0 0 1]
[1 1 1 0 0 1 1 0 1 1 0 0 0 0 0 0 1 1 0 0 1 0 1 1 0 0 0 0 0 0 0 1]
[1 1 0 0 0 1 1 0 1 0 0 0 1 0 1 1 1 1 0 0 1 0 1 1 0 1 1 0 0 0 0 1]

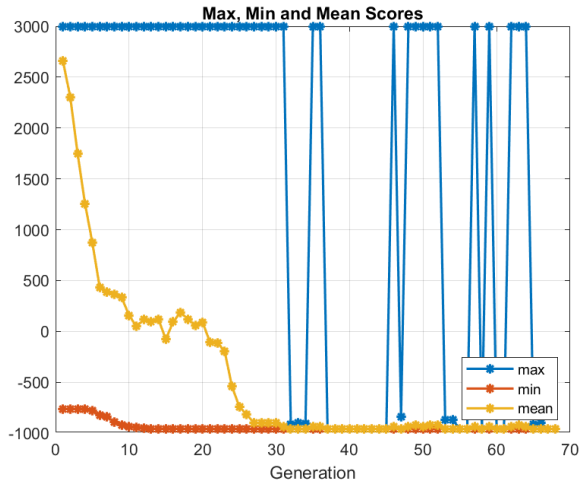
liczność potomków zmutowanych = 0
liczność potomków skrzyżowanych = 194



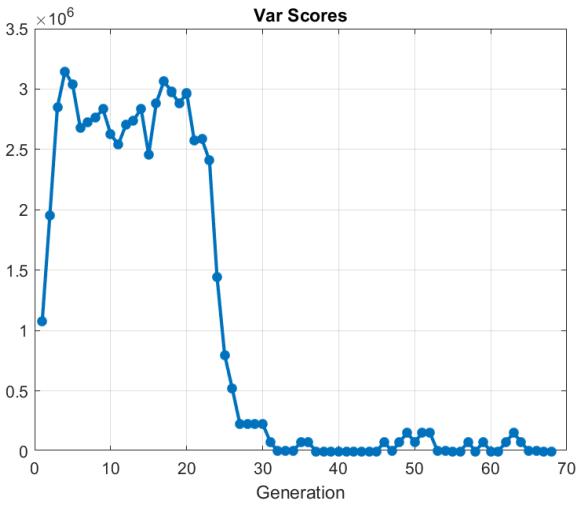
liczność potomków zmutowanych = 0
liczność potomków skrzyżowanych = 194



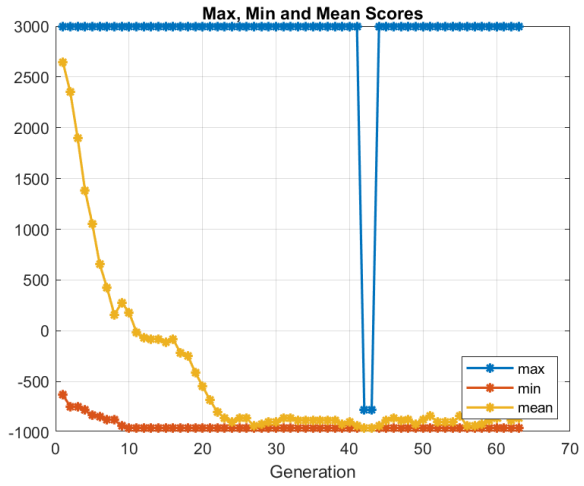
liczność potomków zmutowanych = 2
liczność potomków skrzyżowanych = 192



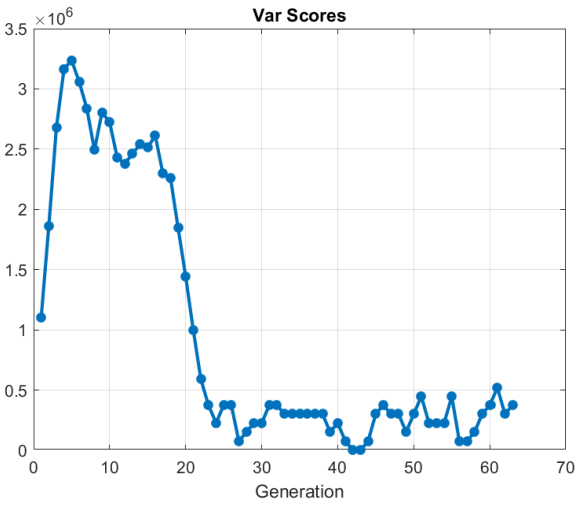
liczność potomków zmutowanych = 2
liczność potomków skrzyżowanych = 192



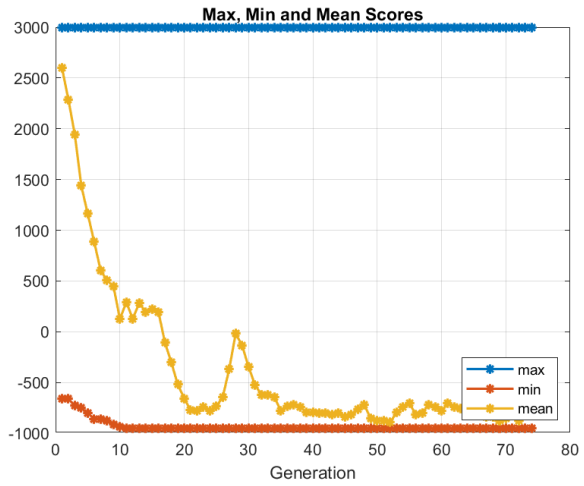
liczność potomków zmutowanych = 10
liczność potomków skrzyżowanych = 184



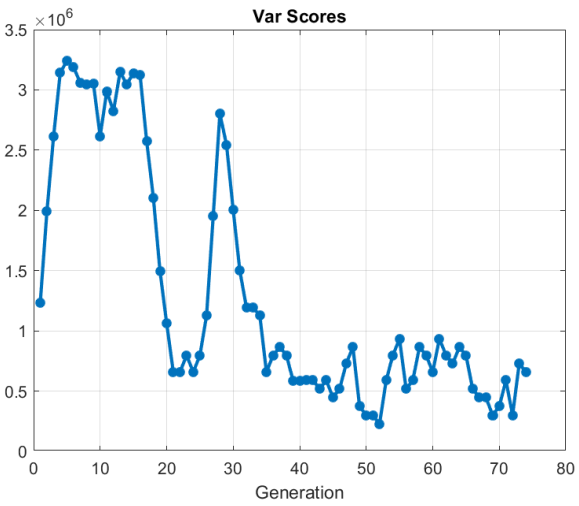
liczność potomków zmutowanych = 10
liczność potomków skrzyżowanych = 184



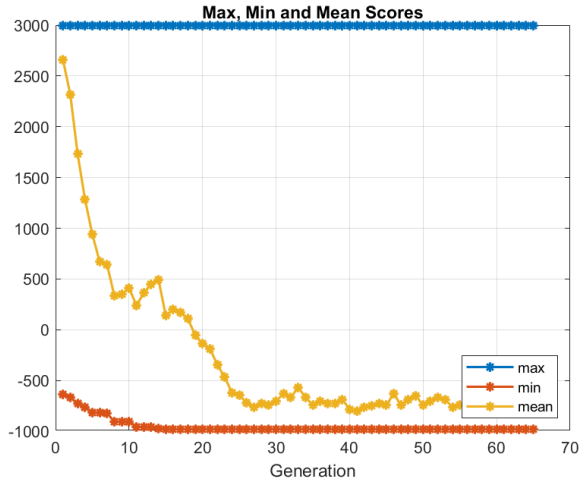
liczność potomków zmutowanych = 29
liczność potomków skrzyżowanych = 165



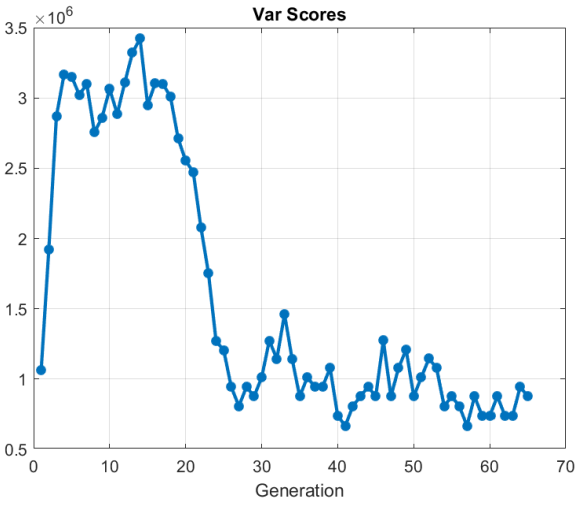
liczność potomków zmutowanych = 29
liczność potomków skrzyżowanych = 1165



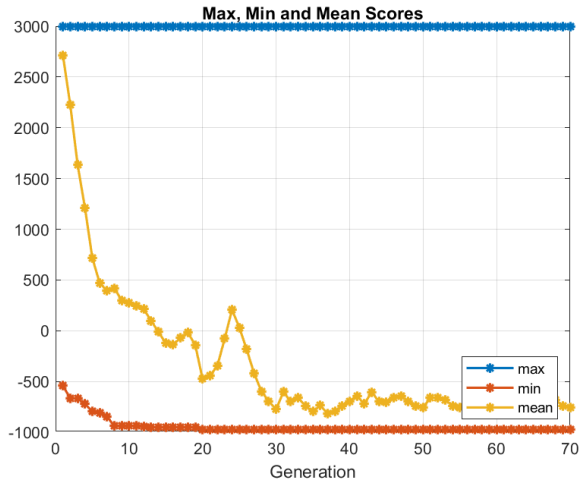
liczność potomków zmutowanych = 38
liczność potomków skrzyżowanych = 156



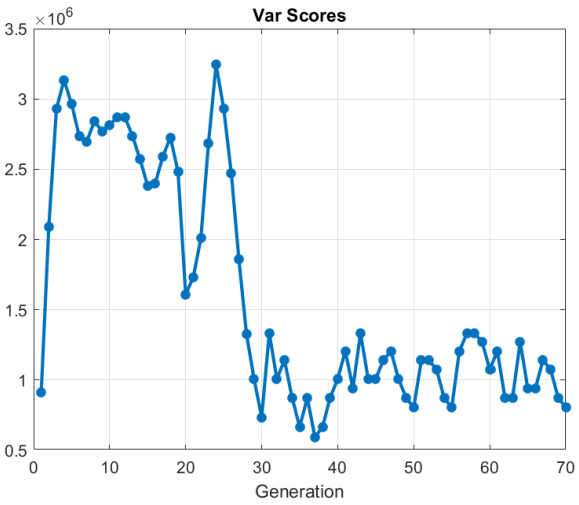
liczność potomków zmutowanych = 38
liczność potomków skrzyżowanych = 156



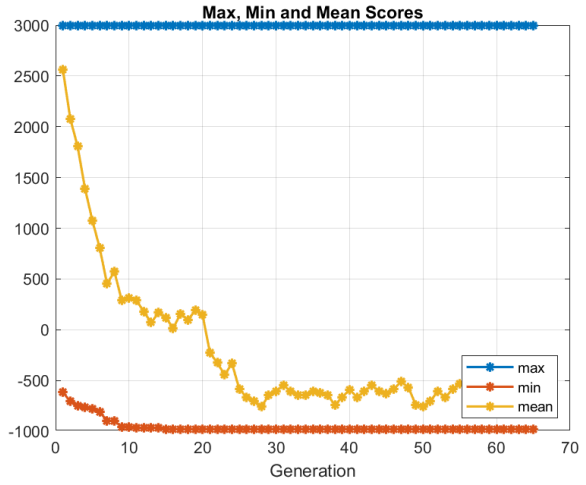
liczność potomków zmutowanych = 48
liczność potomków skrzyżowanych = 146



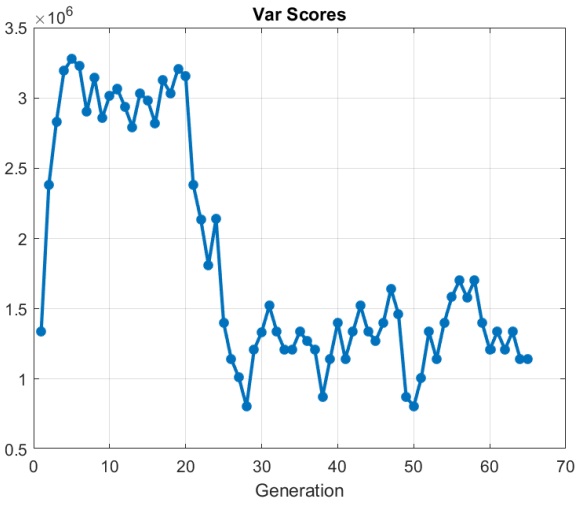
liczność potomków zmutowanych = 48
liczność potomków skrzyżowanych = 146



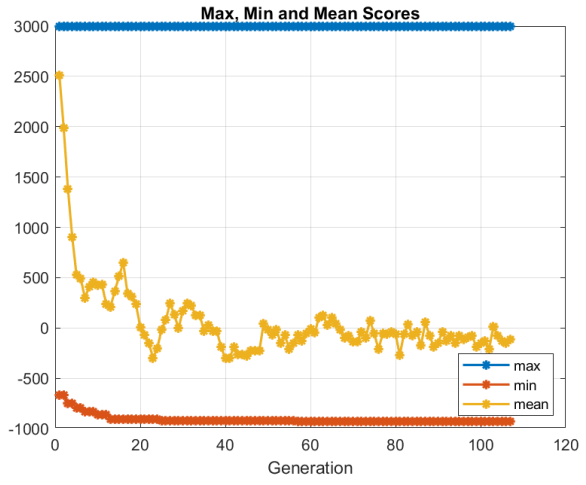
liczność potomków zmutowanych = 59
liczność potomków skrzyżowanych = 135



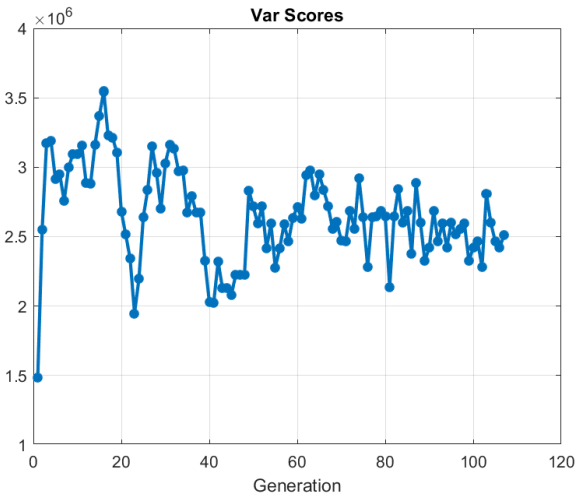
liczność potomków zmutowanych = 59
liczność potomków skrzyżowanych = 135



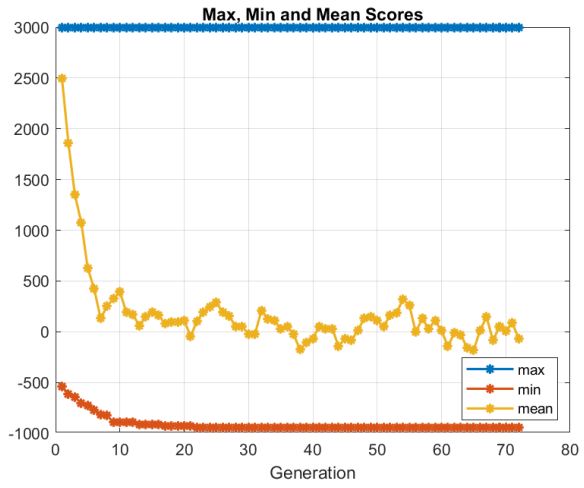
liczność potomków zmutowanych = 158
liczność potomków skrzyżowanych = 36



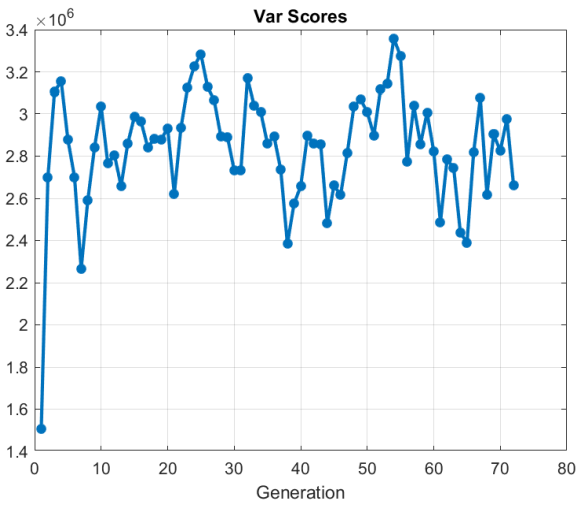
liczność potomków zmutowanych = 158
liczność potomków skrzyżowanych = 36



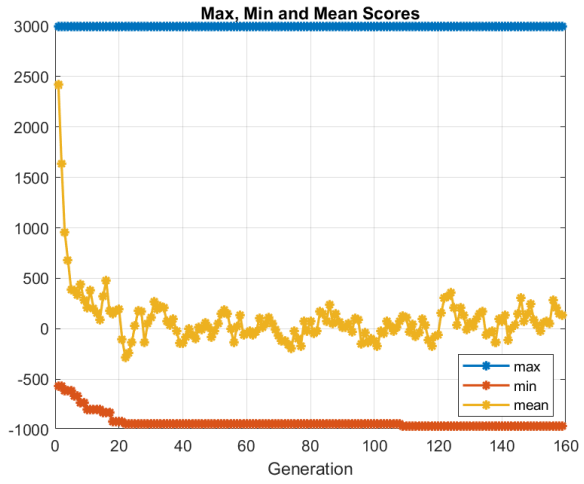
liczność potomków zmutowanych = 164
liczność potomków skrzyżowanych = 30



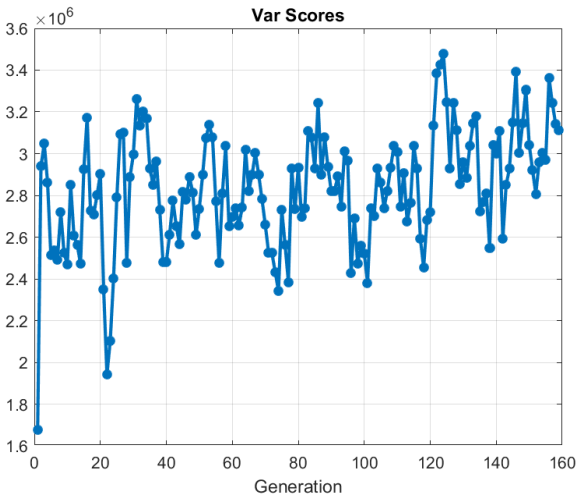
liczność potomków zmutowanych = 164
liczność potomków skrzyżowanych = 130



liczność potomków zmutowanych = 194
liczność potomków skrzyżowanych = 0



liczność potomków zmutowanych = 194
liczność potomków skrzyżowanych = 0



Widzimy, że zbyt mała liczba osobników zmutowanych powoduje drastyczny spadek szybkości algorytmu, bo mamy za mało nowych genów, które wcześniej generowały się dzięki mutacji. Czyli dochodzimy do pewnego pokolenia (w naszym wypadku to ~ 30), gdzie średnia wartość przestaje się zmieniać, bo brakuje nam losowych zmian, które dostarcza mutacja.

Dla liczności mutacji przybliżonej do liczności całej populacji widzimy podobną sytuacje. Ulepszone geny nigdy nie są łączone z genami najlepszego osobnika, ponieważ nie ma krzyżowania.

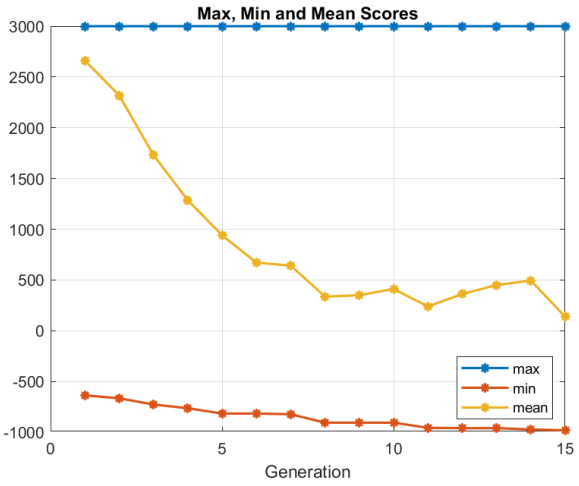
Defaultowe wartości sprawdziły się najlepiej, to jest CrossoverFraction = 0.8 (80%) i MutationFraction = 0.2(20%).

8. Dobór prawdopodobieństwa mutacji

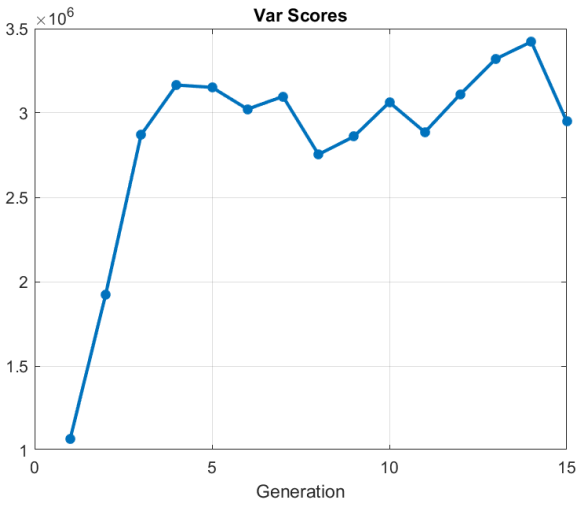
Znowu powracamy do warunku zatrzymania FitnessLimit = 981.

Prawdopodobieństwo mutacji	Koszt obliczeniowy
1%	3075
10%	99835
20%	43255
30%	11480
50%	3280
80%	5740

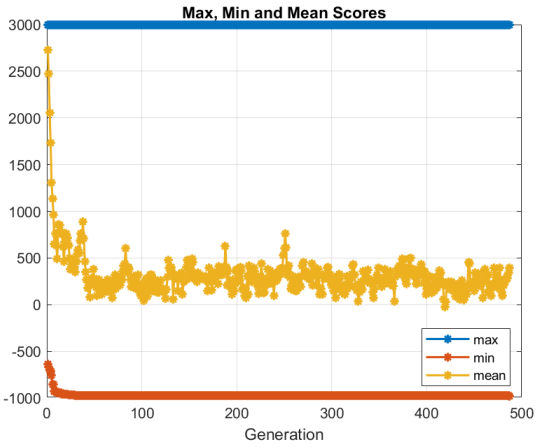
Prawdopodobieństwo mutacji = 1%



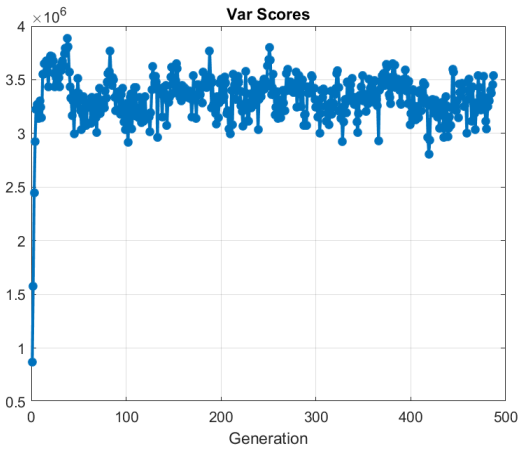
Prawdopodobieństwo mutacji = 1%



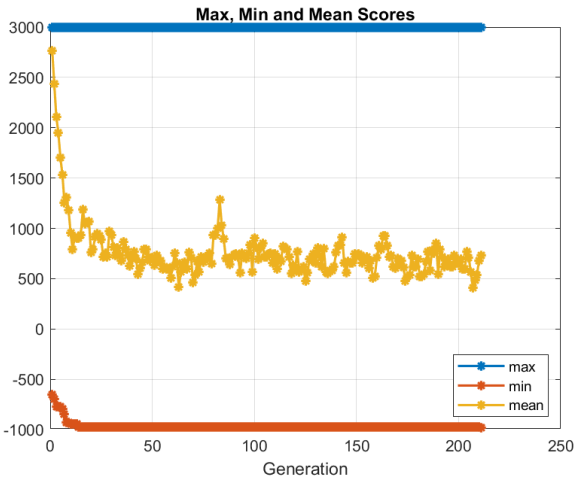
Prawdopodobieństwo mutacji = 10%



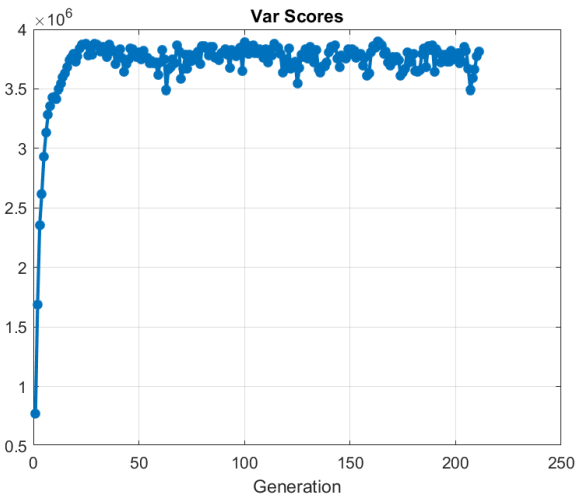
Prawdopodobieństwo mutacji = 10%



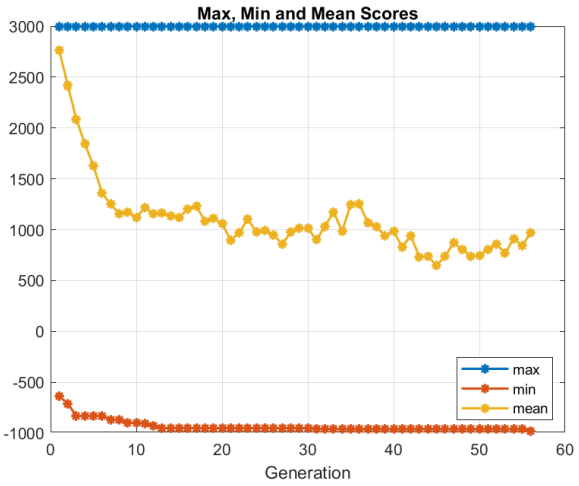
Prawdopodobieństwo mutacji = 20%



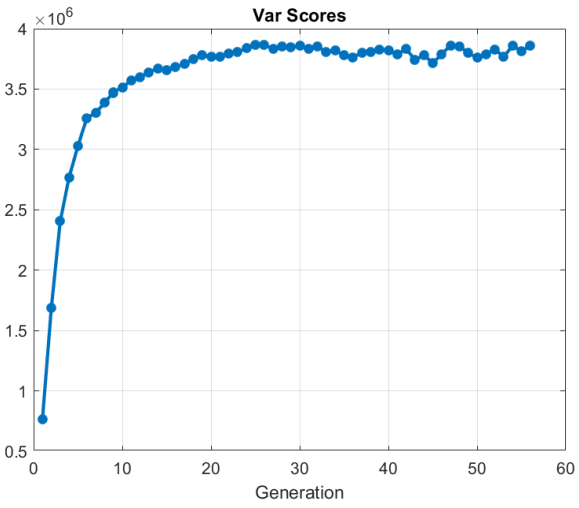
Prawdopodobieństwo mutacji = 20%



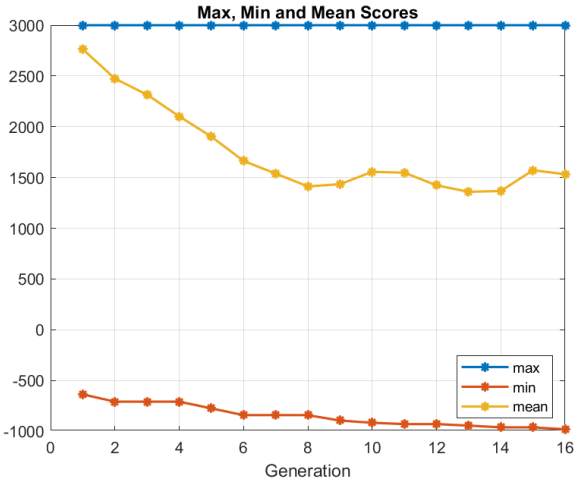
Prawdopodobieństwo mutacji = 30%



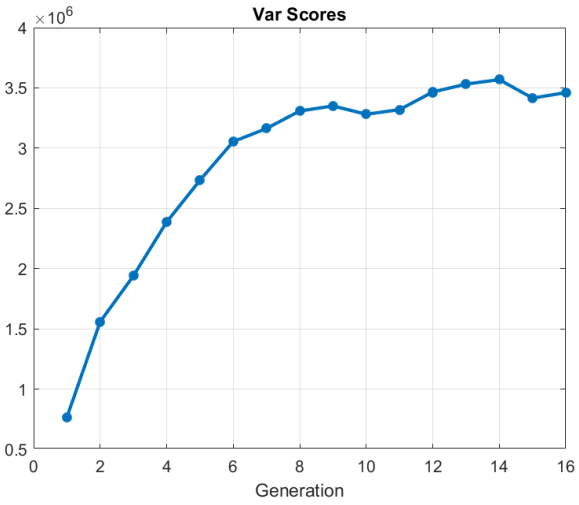
Prawdopodobieństwo mutacji = 30%



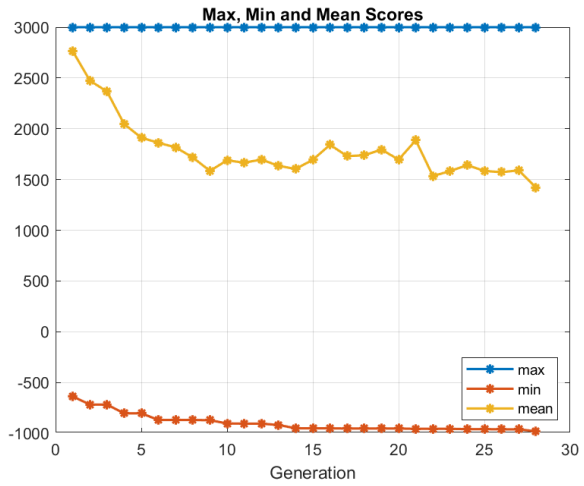
Prawdopodobieństwo mutacji = 50%



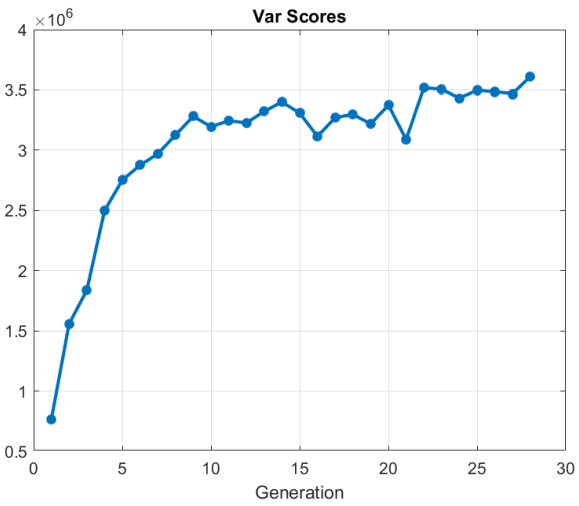
Prawdopodobieństwo mutacji = 50%



Prawdopodobieństwo mutacji = 80%



Prawdopodobieństwo mutacji = 80%



Zdecydowałyśmy się na wybranie niskiego prawdopodobieństwa mutacji (0.05). Dzięki temu uzyskujemy odpowiednio zmniejszającą się wariancję wraz z kolejnymi pokoleniami, a zmienność wariancji nie jest chaotyczna.

9. References

MathWorks documentation for Global Optimization Toolbox(R2022a): ga. Retrieved from: https://www.mathworks.com/help/gads/ga.html

MathWorks documentation for Global Optimization Toolbox(R2022a): Genetic Algorithm Options. Retrieved from: https://www.mathworks.com/help/gads/genetic-algorithm-options.html#f6593

MathWorks documentation for Global Optimization Toolbox(R2022a): Custom Output Function for Genetic Algorythm. Retrieved from: https://www.mathworks.com/help/gads/custom-output-function-for-genetic-algorithm.html#:~:text=The%20custom%20output%20function%20performs,are%20at%20the%20respective%20maxima.

MathWorks documentation for Global Optimization Toolbox(R2022a): Create Custom Plot Function. Retrieved from: https://www.mathworks.com/help/gads/creating-a-custom-plot-function.html

MathWorks documentation for Global Optimization Toolbox(R2022a): Effects of Genetic Algorithms Options. Retrieved from: https://www.mathworks.com/help/gads/options-in-genetic-algorithm.html

MathWorks documentation for Global Optimization Toolbox(R2022a): How the Genetic Algorithm Works. Retrieved from: https://www.mathworks.com/help/gads/how-the-genetic-algorithm-works.html

MathWorks documentation for Global Optimization Toolbox(R2022a): Genetic Algorithm Terminology. Retrieved from: https://www.mathworks.com/help/gads/some-genetic-algorithm-terminology.html

MathWorks documentation(R2022a): Parameterizing Function. Retrieved from: https://www.mathworks.com/help/matlab/math/parameterizing-functions.html

Miazga, P. Algorytmy Ewolucyjne [pdf].

Knapsack problem/0-1. Retrieved from: https://rosettacode.org/wiki/Knapsack_problem/0-1#Recursive_dynamic_programming_algorithm