Labirynt

Teza problemu:

> Czy istnieje ścieżka z punktu startu do punktu wyjścia w maksymalnie 40 krokach?

Algorytm Genetyczny

Do wykonania algorytmu genetycznego skorzystano z jednej z pythonowych paczek oferujących wsparcie dla algorytmów genetycznych, dokładnie z Pyeasyga.

Do testowania algorytmu wykorzystano ręcznie wykonane labirynty o wymiarach

12x12 oraz 23x21.

Labirynt reprezentuje tablica wypełniona:

1 – ściana

0 – możliwość ruchu

2 – start

3 – exit

Strukturę naszego chromosomu przedstawiamy w postaci ciągu liczb binarnych wynoszących długość 80. Chromosom przedstawia sekwencję 40 ruchów, tzn. każde dwa bity kodują jeden ruch.

Odpowiednio przypisując:

00 – ruch w lewo

01 – ruch w prawo

10 – ruch w górę

11 – ruch w dół

Chromosom jest tym lepszy im bliższa jest jego pozycja do wyjścia, ponieważ skupiamy się tutaj na znalezieniu najbliższego położenia od końca, w 40 krokach. Najsłabszym chromosomem oznaczamy te które są najdalej od wyjścia.

Odległość reprezentowana jest jako suma szerokości i wysokości od pozycji zatrzymania do punktu wyjścia oraz jako wartość w linii prostej.

Funkcja fitness v.1

Funkcja przyjmuje chromosom, który reprezentuje drogę. Funkcja symuluje, aż do napotkania pierwszej kolizji ze ścianą. Resztę ruchów pomija. Zostaje zwracana odległość miedzy punktem kolizji, a polem exit. Wadą tej funkcji jest, że mała jest szansa na zalezienie idealnego genomu, który wskaże drogę od startu do mety.

```
def fitness1(individual, data):
    pozycja = s
    for x in range(len(data)):
        if x % 2 == 0 or x == 0:
            if x == 0 or x % 2 == 0:
                krok = ruch(individual[x], individual[x + 1])

                if labirynt[pozycja + krok] == 1:
                    print ('%s%s SCIANA! %s' % (fg(1), bg(15), attr(0)))
                    print ("Pozycja: " + str(pozycja))
                    rezultat = odleglosc(pozycja)
                    print ("Odleglosc do exit: " + str(rezultat))
                    wLinii = liniaProsta(pozycja)
                    print ("W linii prostej do exit: " + str(wLinii))
                    print (" ")
                    return rezultat
                pozycja = kP(pozycja, krok)

                if pozycja == e:
                    print ("Sukces!")
                    print(individual)
                    print (" ")
                    return meta

    if pozycja == e:
        print ("Sukces")
        print (individual)
        print (" ")
        return meta

    return meta
```

Funkcja fitness v.2

Funkcja przyjmuje chromosom, który reprezentuje drogę. Funkcja symuluje drogę do czasu, aż skończą się ruchy lub napotkamy pole exit. Kiedy napotykamy ścianę ruch nie zostaje wykonany, co oznacza, że przepada, a ilość ruchów się zmniejsza. Funkcja fitness, w ten o to sposób odpowiedzialna jest za blokowanie ruchu przez ścianę. Zwracana jest odległość miedzy pozycją zatrzymania, a punktem wyjścia.

```
def fitness2(individual, data):
    pozycja = s

    for x in range(len(data)):
        if x == 0 or x % 2 == 0:
            krok = ruch(individual[x], individual[x + 1])
            pozycja = kolejnaPozycja(pozycja, krok)

            if pozycja == e:
                print ('%s%s Sukces! %s' % (fg(46), bg(15), attr(0)))
                print(individual)
                print (" ",pozycja)
                return meta

    if pozycja == e:
        print ("Sukces")
        print (individual)
        print (" ")
        return meta


    print ("Pozycja: " + str(pozycja))
    rezultat = odleglosc(pozycja)
    print ("Odleglosc do exit: " + str(rezultat))
    wLinii = liniaProsta(pozycja)
    print ("W linii prostej do exit: " + str(wLinii))
    print (" ")

    return rezultat
```

Funkcja fitness v.3

Funkcja przyjmuje chromosom, który reprezentuje drogę. Funkcja symuluje drogę do czasu, aż skończą się ruchy lub napotkamy pole exit. Działanie funkcji zostało zoptymalizowane z poprzedniej funkcji fitness poprzez modyfikację niedozwolonych ruchów dokładając do tego kolejne restrykcje. Niedostępne są ruchy wchodzące na ścianę, ale dodatkowo na już odwiedzone pola, co sprawia że nie wykonujemy ruchów wstecz. Jeżeli już do tego dojdzie algorytm przyznaje ujemne punkty do wyniku. Ocena fitness jest lepsza, kiedy zwracany genom ma większą wartość. Na koniec zwracana jest odległość miedzy punktem zatrzymania, a wyjściem.

```
def fitness3(individual, data):
    pozycja = s
    powtorzenia = 0

    for x in range(len(data)):
        if x == 0 or x % 2 == 0:
            p = pozycja
            krok = ruch(individual[x], individual[x + 1])
            pozycja = kolejnaPozycja(pozycja, krok)

            if p == pozycja:
                powtorzenia = powtorzenia - 2

            if labirynt[pozycja] == "+":
                powtorzenia = powtorzenia - 2

            if labirynt[pozycja] != 2:
                labirynt[pozycja] = "+"

            if pozycja == e:
                print ("Sukces!")
                print(individual)
                print (" ")
                return infinity

    if pozycja == e:
        print ("Sukces")
        print (individual)
        print (" ")
        return infinity
    print ("Pozycja: " + str(pozycja))
    rezultat = odleglosc(pozycja)
    print ("Odleglosc do exit: " + str(rezultat))
    wLinii = liniaProsta(pozycja)
    print ("W linii prostej do exit: " + str(wLinii))
    print (" ")

    return rezultat + powtorzenia
```

Brute Force

Algorytm ten nie wykorzystuje algorytmu genetycznego. Na potrzeby testu, który ma ukazać różnicę między efektywnością algorytmu typu Brute Force, a Algorytmem Genetycznym.

Źródło: https://levelup.gitconnected.com/solve-a-maze-with-python-e9f0580979a1

Output dla labiryntu 12 x 12:

```
0.18322253227233887
0  0  0  0  0  0  0  0  0  0  0  0
0  1  2  3  0  7  8  9  0  17 16 0
0  0  0  4  5  6  0  10 0  0  15 0
0  7  6  5  0  7  0  11 12 13 14 0
0  8  0  6  0  0  13 12 0  0  15 0
0  9  10 0  0  15 14 13 0  17 16 0
0  10 11 12 13 14 0  14 15 16 0  0
0  11 0  13 14 0  0  15 0  17 18 0
0  12 0  0  0  18 17 16 0  0  19 0
0  13 0  0  0  0  18 0  0  0  20 0
0  14 0  0  21 20 19 20 21 0  21 0
0  0  0  0  0  0  0  0  0  0  0  0
```

Zestawienie algorytmów

Labirynt 12x12	Populacja 10 Generacja = 20 Mut = 5% Elita = true	Populacja 40 Generacja = 20 Mut = 1% Elita = true	Populacja 60 Generacja = 20 Mut = 0,5% Elita = true
Fitness v.1 (niepowodzenie)	0.093 sek	0.339 sek	0.474 sek
Fitness v.2 (sukces)	0.079 sek	0.187 sek	0.757 sek
Fitness v.3 (sukces)	0.052 sek	0.224 sek	0.312 sek
Brute Force (sukces)	0.183 sek		

Labirynt 23x21	Populacja 10 Generacja = 20 Mut = 5% Elita = true	Populacja 40 Generacja = 20 Mut = 1% Elita = true	Populacja 60 Generacja = 20 Mut = 0,5% Elita = true
Fitness v.1 (niepowodzenie)	0.082 sek	0.313 sek	0.795 sek
Fitness v.2	0.053 sek	0.207 sek Sukces	0.290 sek Sukces
Fitness v.3	0.057 sek	0.191 sek	0.285 sek Sukces
Brute Force (sukces)	0.196 sek		

Zauważmy, że dla mniejszego labiryntu wyniki były sukcesywne, co oznacza, że znalezienie ścieżki jest prostsze. W labiryncie już o większych rozmiarach było trudnej uzyskać zadawalający rezultat, przy częstych próbach raz na pewny czas została odnaleziona ścieżka.

Wykorzystując paczkę pyeasyga pozwala ona na proste i łatwe generowanie populacji potomnej oraz wykorzystanie funkcji fitness.

Rezultat programu ukazuje przedstawienie najlepszej funkcji fitness i uzyskaną ścieżkę przedstawiona graficznie.

Output dla labiryntu 12 x 12:

```
Labirynt o wymiarach: 12 x 12
Najlepszy (130, [0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0
, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1
, 1, 1, 1, 1, 1, 0, 0])
 111111111111
 12++1+++1001
 111+++1+1101
 1000101+0001
 1010110+1101
 1001100+1001
 1000001+++11
 1010011+1++1
 1011100011+1
 1010110101+1
 101000000031
 111111111111
```

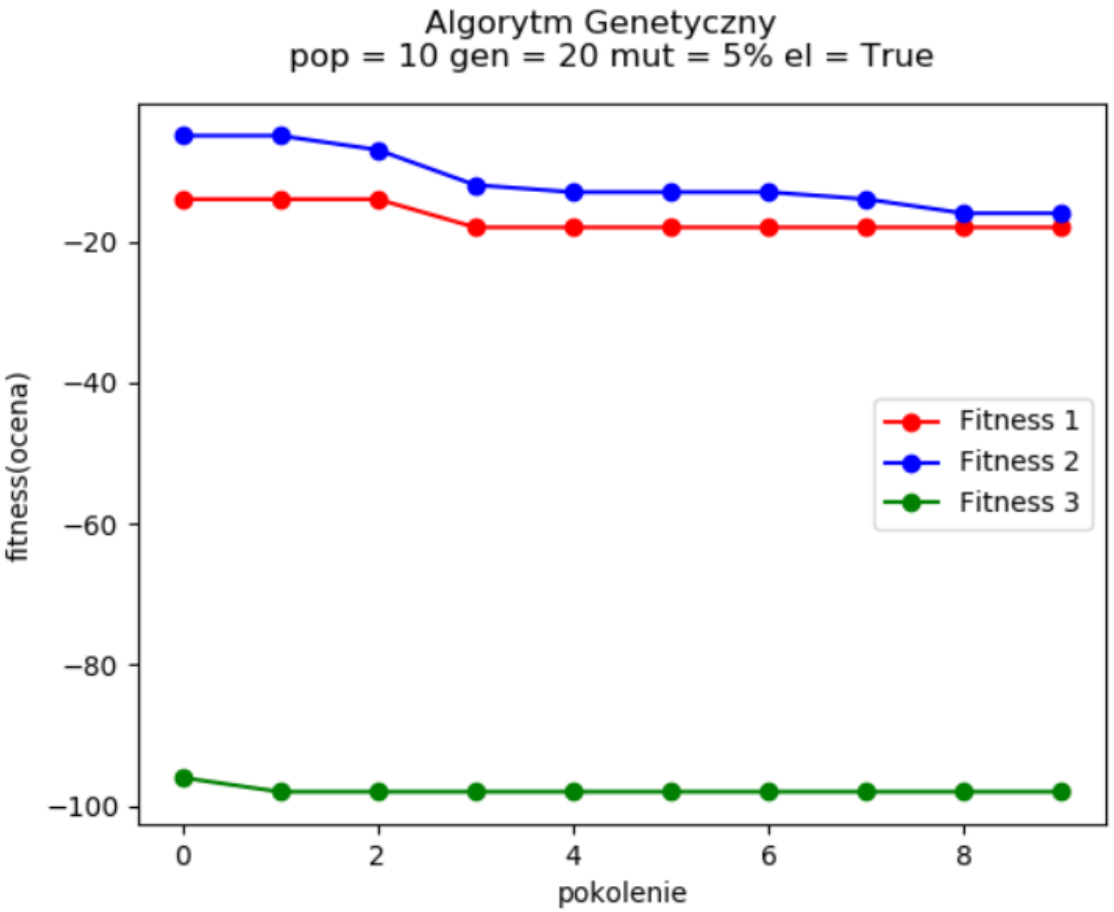
Output dla labiryntu 21 x 23:

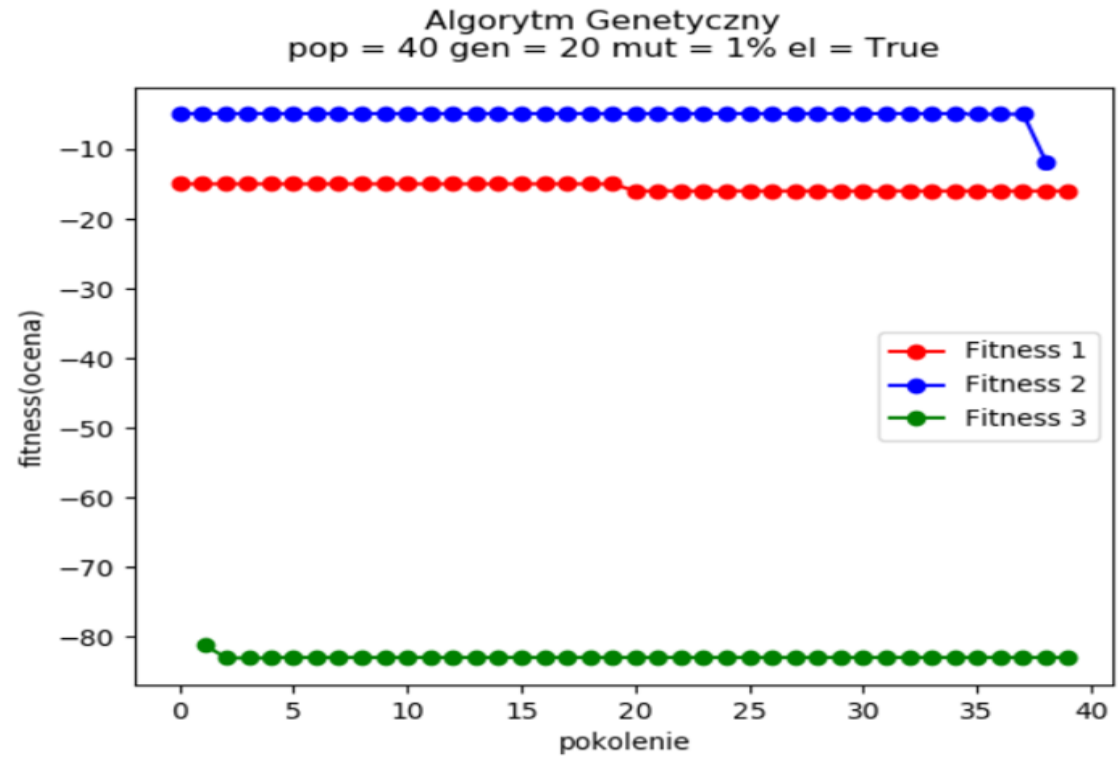
```
Pozycja: 362
Punkt x: -4 , y: -4
Odleglosc do exit: -8
W linii prostej do exit: 5

Labirynt o wymiarach: 23 x 21
Najlepszy (-8, [1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0,
 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0,
 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1])
1111111111111111111111
12+11110011111111000011
10++01100011111110000001
101+1000100000000011101
101+1+00100000000011101
100+++00100000000011101
11111+11001000011100101
10011++++11001111110011
10000110+++++++11000001
100111110111001++++01111
110111100110011111+01111
10000110001001100+10011
101111111010001l0+01111
11111111000001110+11001
10011000011100000+00011
10011000011100000+00011
100110000111000000000011
101111101010000000000031
111111111111111111111111
```

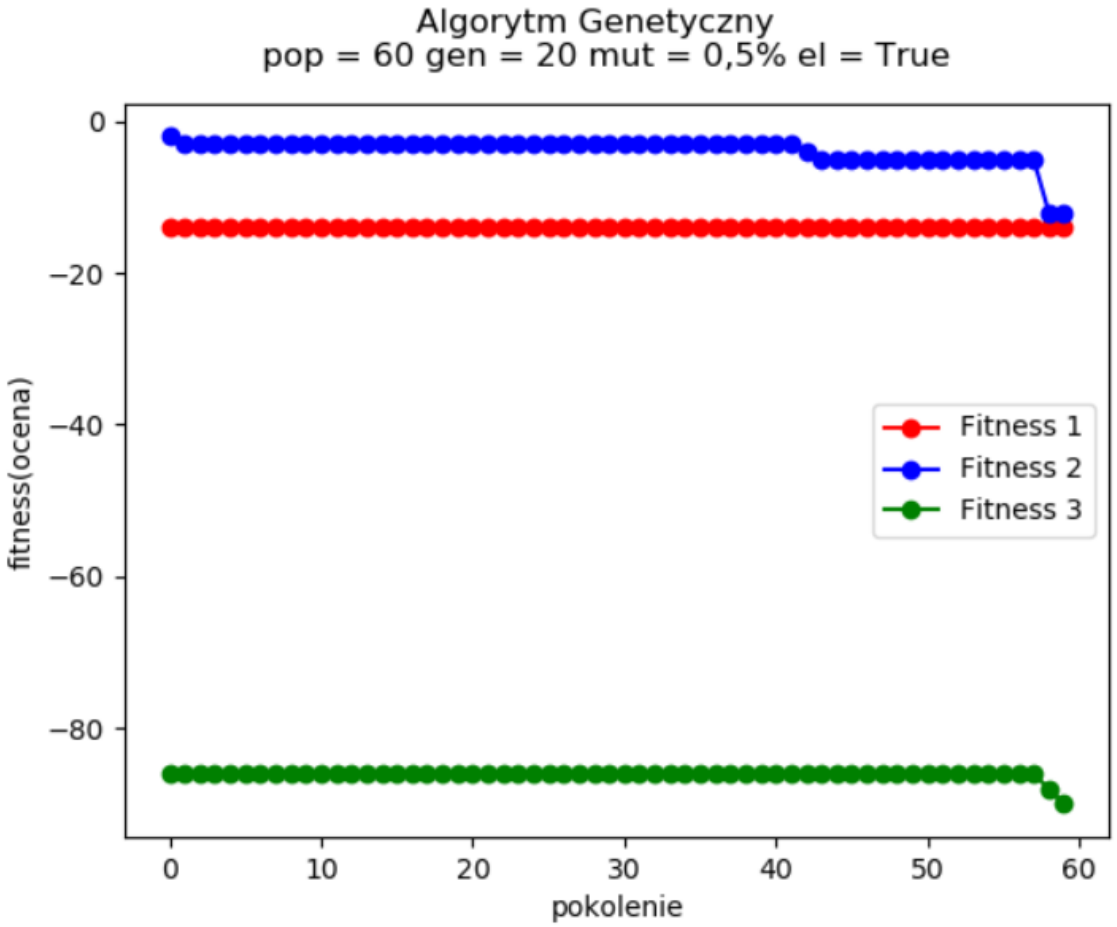
Poniżej przedstawiono reprezentację działania algorytmu genetycznego dla labiryntów.

Użyto do tego paczki matplotlib.pyplot. Ocena fitness przedstawia odległość między punktem zatrzymania, a punktem wyjścia.

Zestawienie wyników funkcji fitness dla labiryntu 12 x 12.

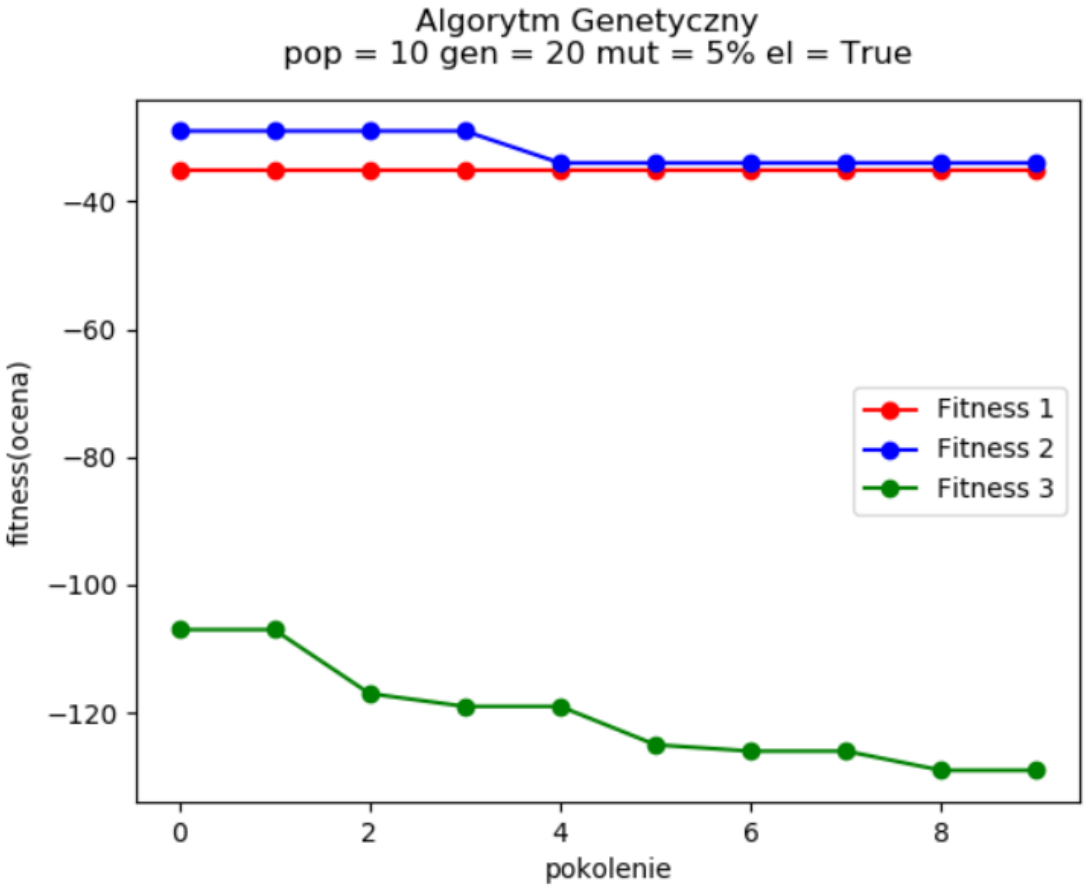


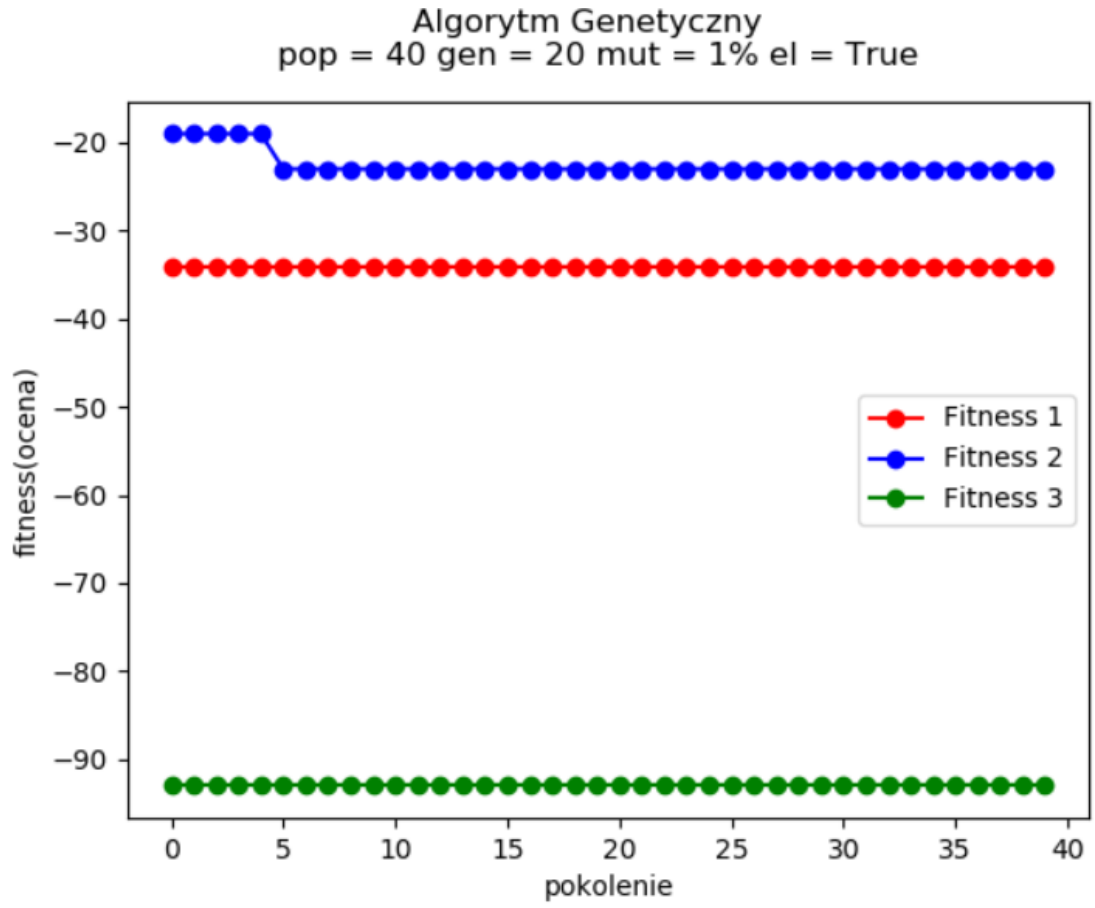


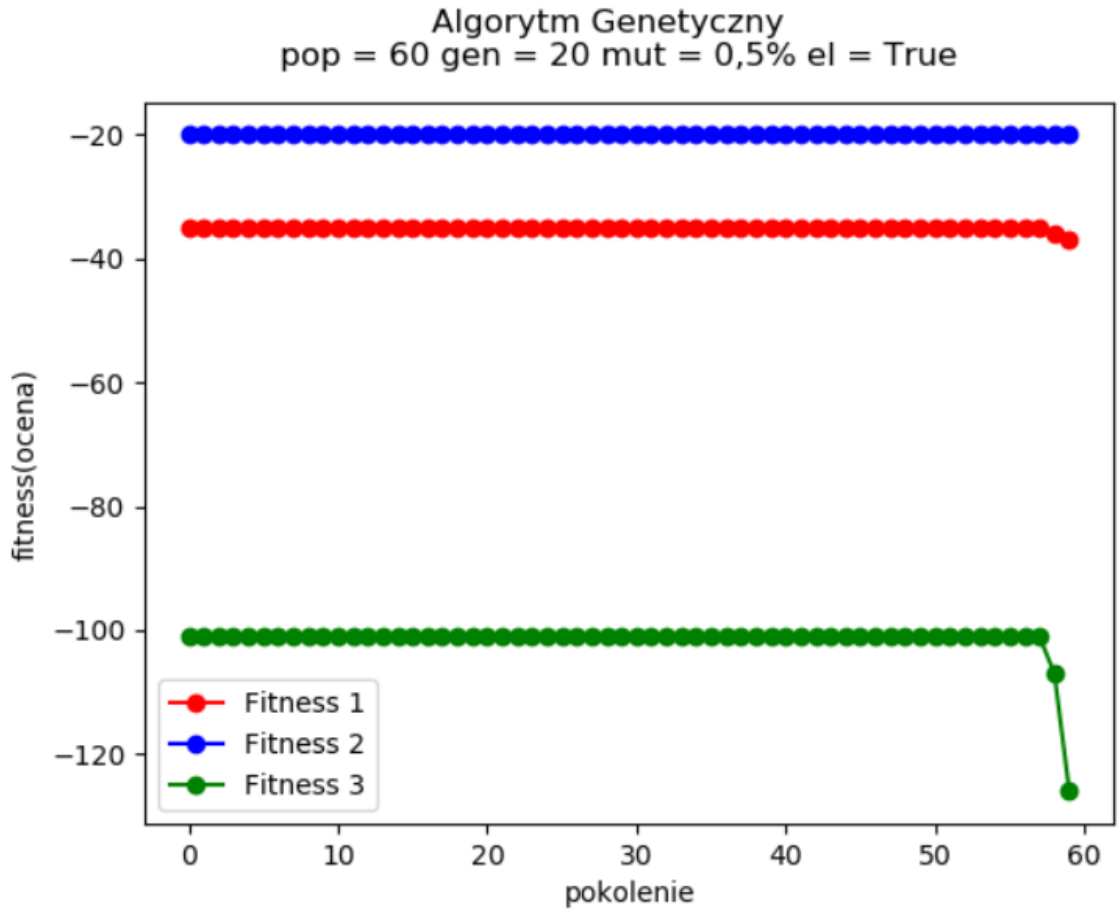


Możemy zaobserwować, że funkcje są zaprezentowane jako stała, chociaż widać niewielką różnice między populacjami. Zdarzają się populacje, gdzie zwraca dużą zmianę, są to pojedyncze przypadki. Funkcja fitness v.2 jest najbliżej rozwiązania naszego problemu w porównaniu z funkcją fitness v.3.

Zestawienie wyników funkcji fitness dla labiryntu 23 x 21.

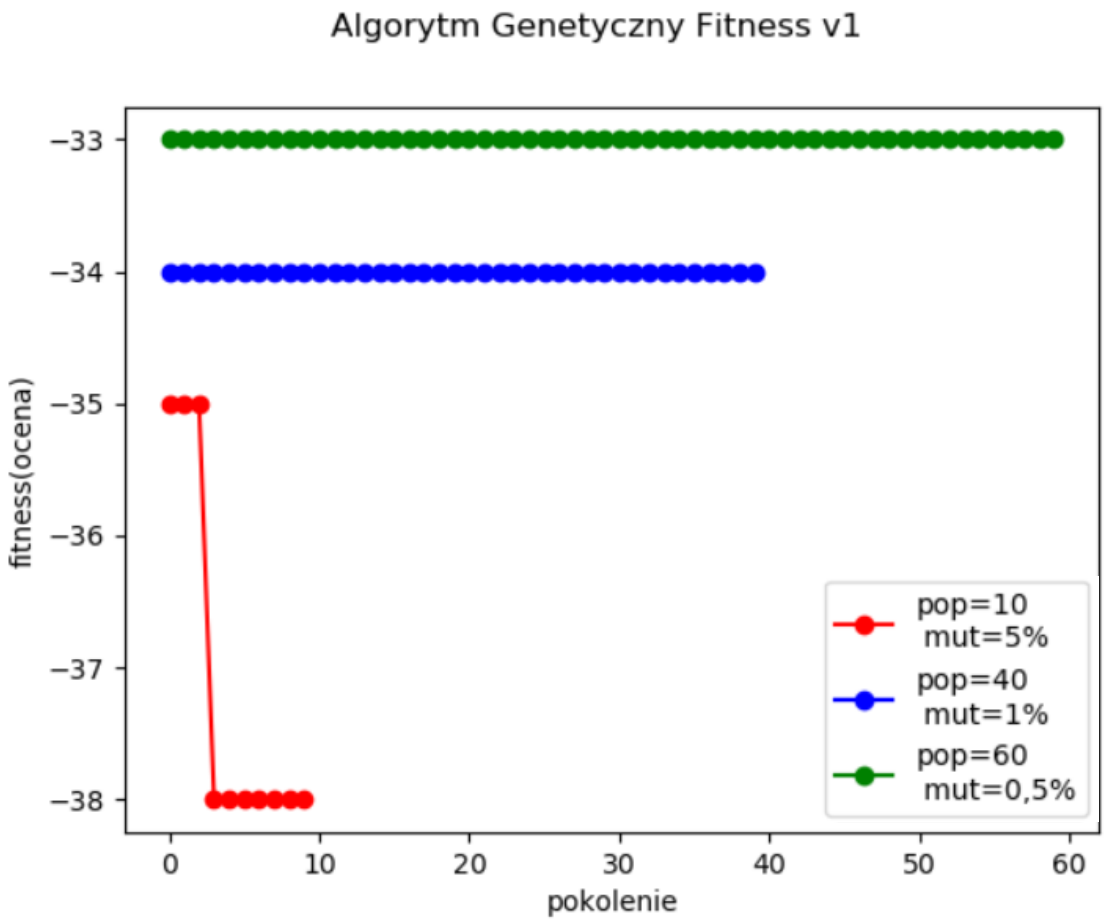


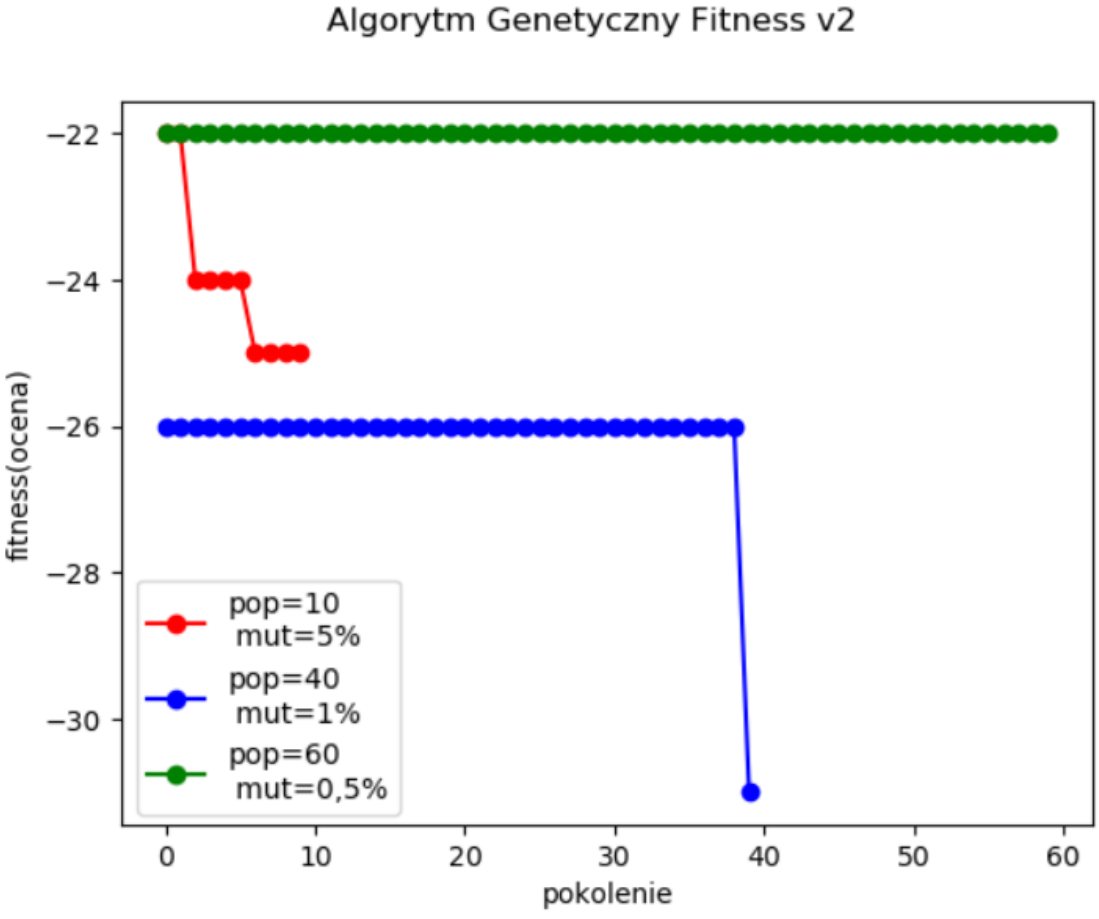


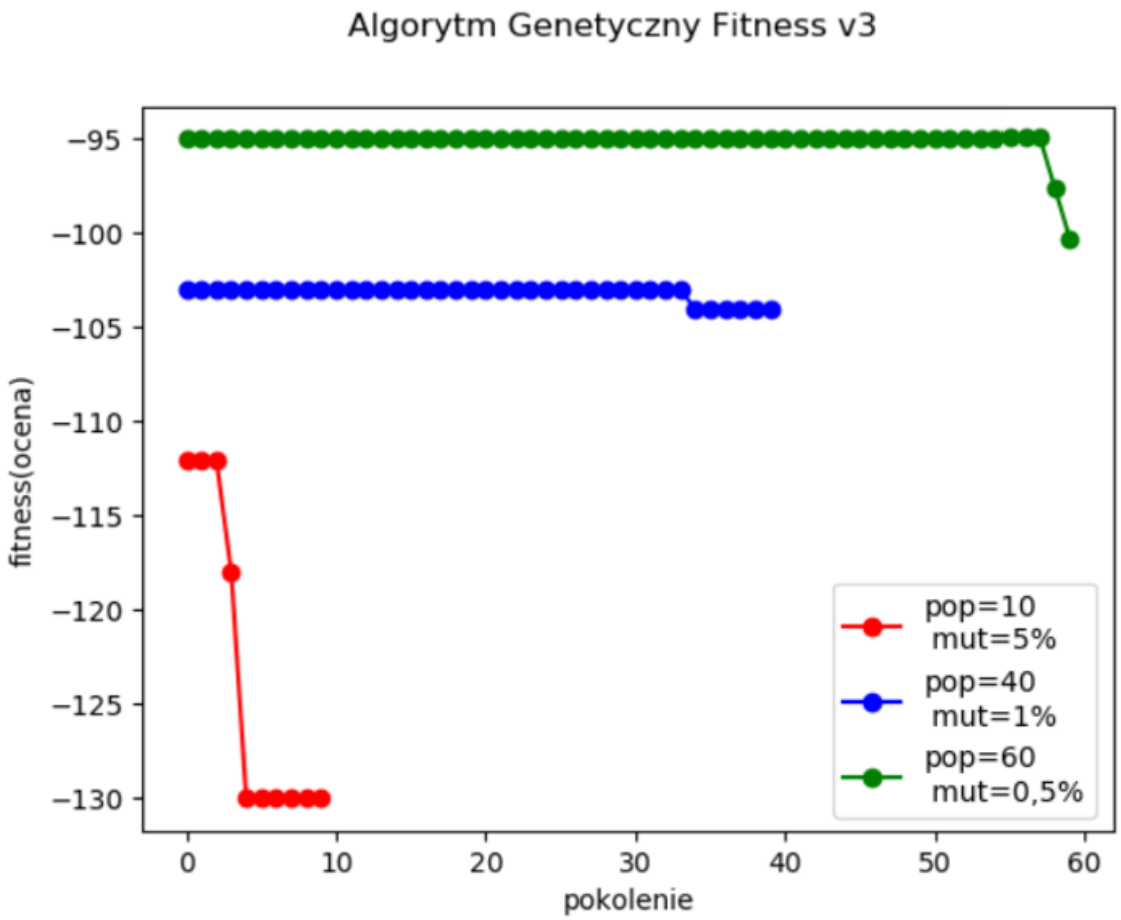


W przypadku większego labiryntu funkcje fitness większości to wykres stały. Przy małej populacji z większą mutacją wykres jest dynamicznie spadkowy, to znaczy że pierwsze pokolenie jest lepsze od kolejnych. Działanie funkcji fitness porównując do labiryntu o wymiarach 12 x 12 jest takie same tzn. funkcja v.3 zwraca najdalsze położenie od punktu wyjścia.

Zestawienie wyników algorytmu genetycznego funkcji fitness dla labiryntu 23 x 21.

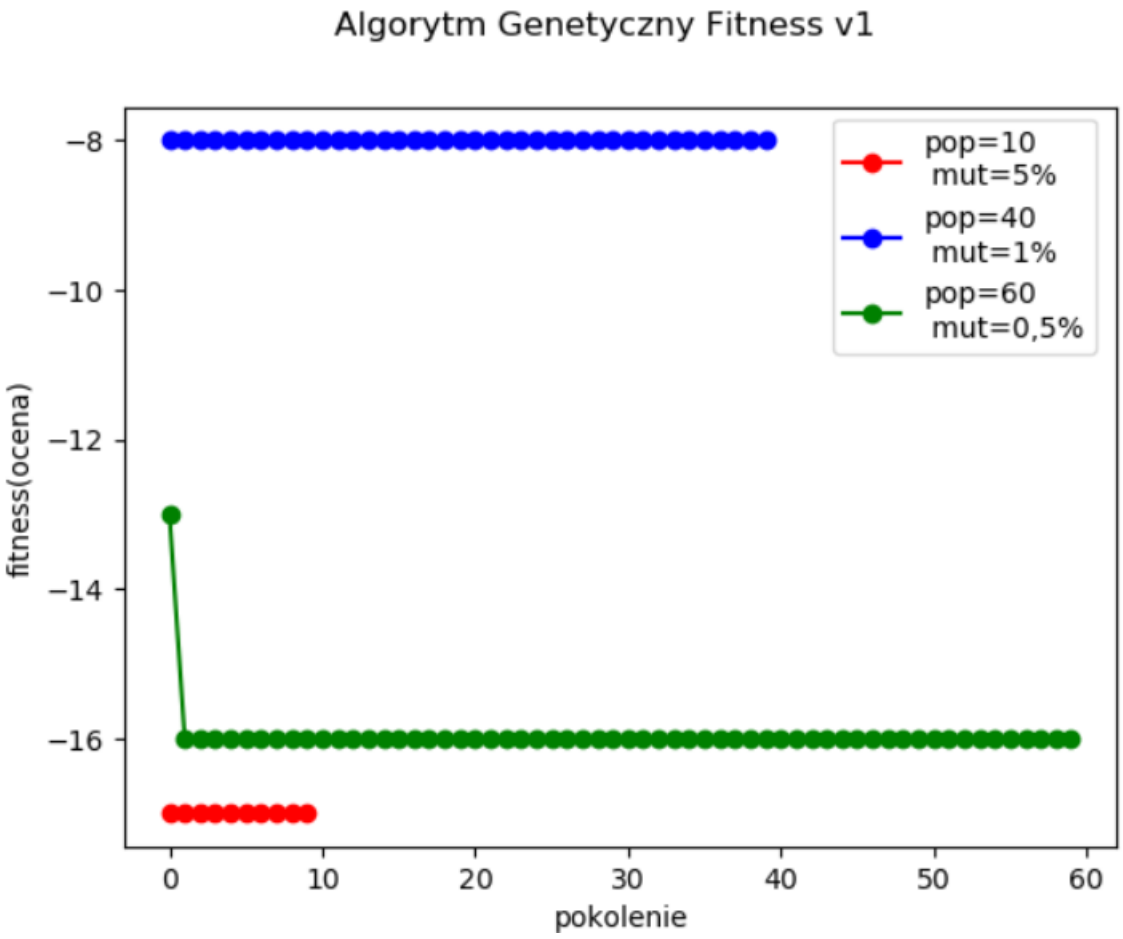


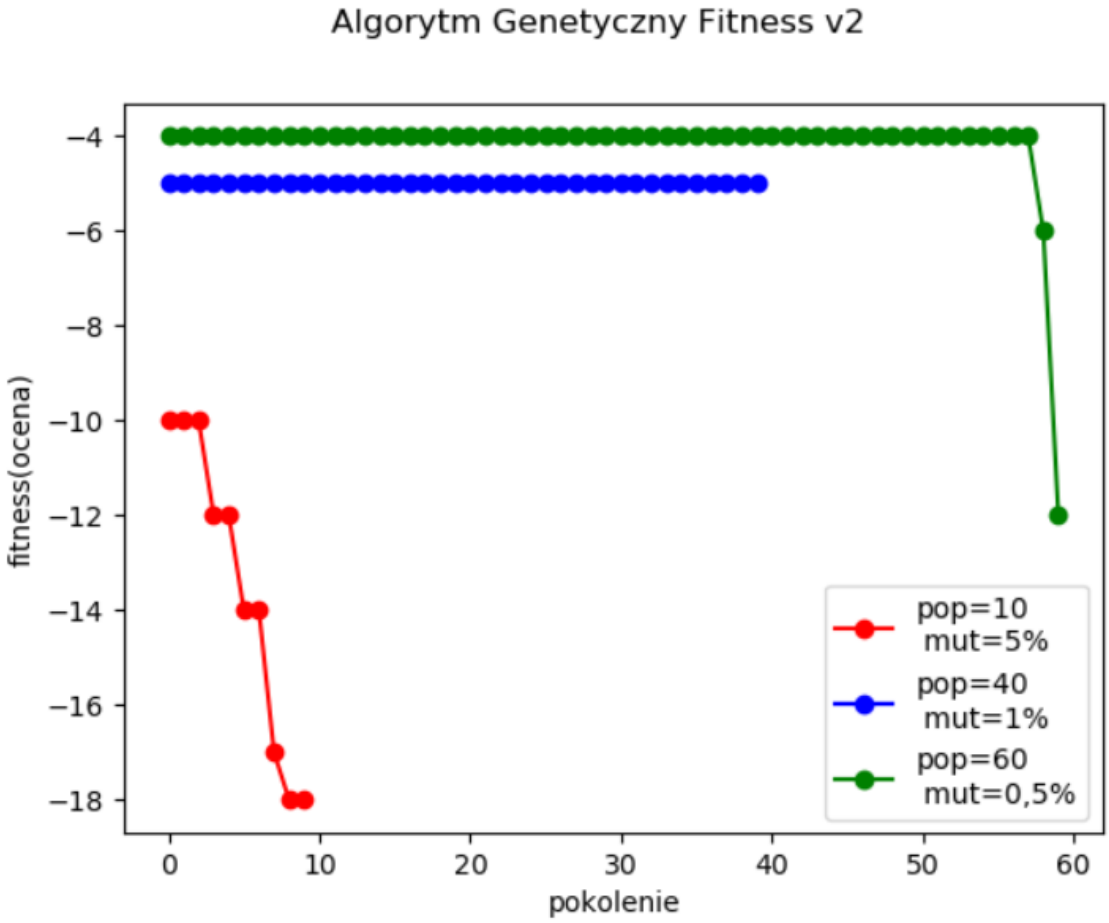


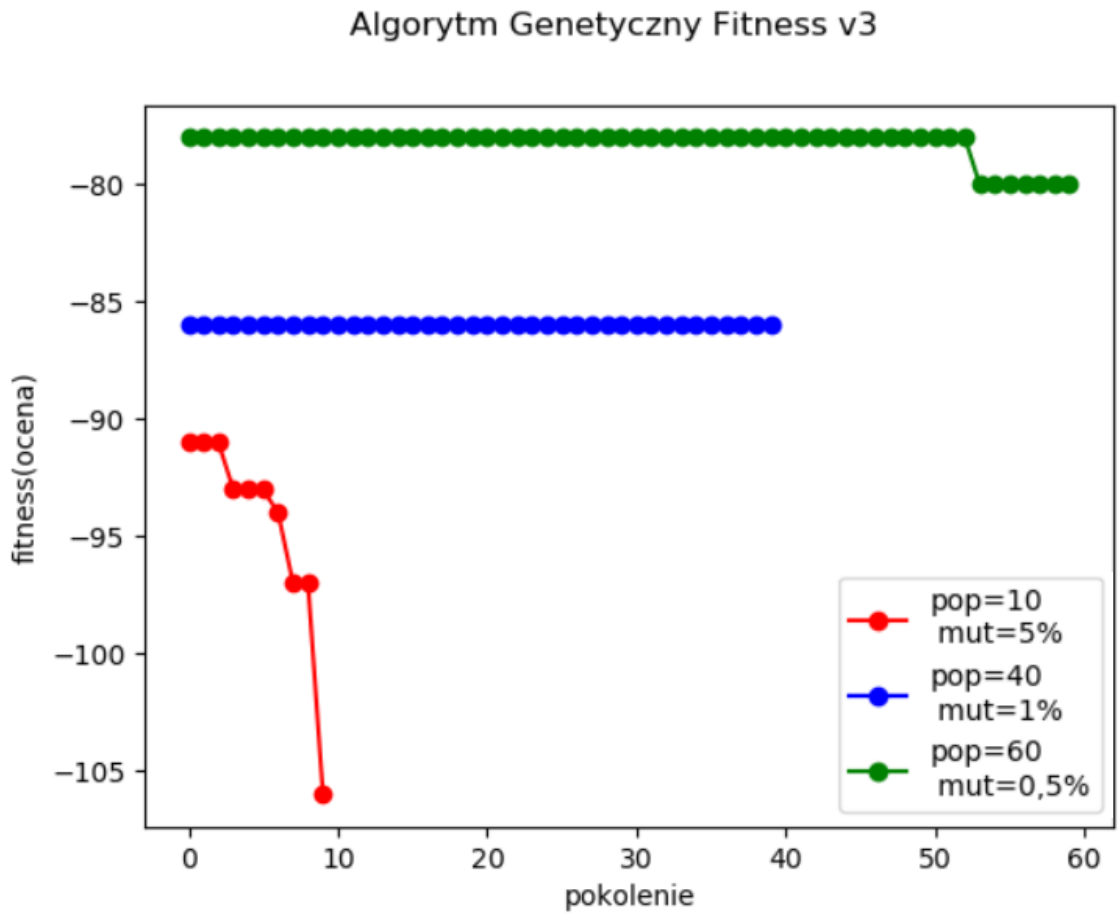


Ukazując jak wyglądają dane funkcje fitness z innymi parametrami algorytmu genetycznego, zauważamy przełom w pokoleniu w populacji 10 z mutacją 5%.. Nie są to satysfakcjonujące zmiany, gdyż punkt zatrzymania w labiryncie jest bliższy startu niż tego co chcemy uzyskać, czyli mety. Funkcja v1 i v3 można stwierdzić, że zwraca przybliżony wynik.

Zestawienie wyników algorytmu genetycznego funkcji fitness dla labiryntu 12 x 12.







W labiryncie 12 x 12 każda funkcja fitness jest odmienna w podanych mutacjach. Nie są to spore różnice jednakże są przedziały do siebie zbliżone. W większości jest reprezentacja stała, najbardziej zauważalne jest w populacji równe 40 z mutacją 1%, ,końcowe pokolenie jak widać się zmienia.

Obserwacje jak zachowuje się funkcja fitness w danym pokoleniu jest bardzo ciekawym zjawiskiem. Zaskakuje tym jak różny, a nawet odległy może być rezultat. Kolejne pokolenia niewiele się różnią, zazwyczaj są to gorsze pokolenia, ponieważ oddalamy się od punktu exit.