Wstęp teoretyczny

Czym jest algorytm genetyczny?

Algorytm ewolucyjny, opisany w Twojej notatce, jest zaawansowaną metodą obliczeniową, która znajduje zastosowanie w rozwiązywaniu złożonych problemów, gdzie tradycyjne podejścia mogą okazać się niewystarczające. Inspiracją dla algorytmu ewolucyjnego jest naturalny proces ewolucji biologicznej, a jego działanie opiera się na selekcji, mutacji, krzyżowaniu i sukcesji.

Jak działa algorytm genetyczny?

Algorytm ewolucyjny przetwarza populacje osobników, z których każdy jest propozycją rozwiązania postawionego problemu. Działa on w środowisku, które można zdefiniować na podstawie rozwiązywanego problemu. W środowisku każdemu osobnikowi jest przyporządkowana wartość liczbową, określającą jakość reprezentowanego przez niego rozwiązania, zwaną przystosowaniem osobnika. Każdy osobnik jest wyposażony w informacje stanowiące jego genotyp, będące przepisem na utworzenie fenotypu - zestawu cech określanych przez genotyp, podlegających ocenie środowiska. Wartość liczbową tej oceny nazywa się przystosowaniem osobnika. Mówimy o kodowaniu fenotypu przez genotyp. Fenotyp jest punktem w przestrzeni rozwiązań problemu, genotyp zaś - punktem w przestrzeni kodów. Środowisko można opisać funkcją przystosowania, za pomocą której osobnikowi przypisuje się przystosowanie na podstawie jego fenotypu. Funkcja ta może być stacjonarna lub zmienna w czasie; może też zawierać element losowości. Ze względu na fakt, że fenotyp jest wynikiem dekodowania genotypu, w dalszych rozważaniach będziemy przyjmować, że funkcja przystosowania jest określona dla genotypów. Genotyp osobnika składa się z chromosomów. Co najmniej jeden z chromosomów zawiera kod określający fenotyp, pozostałe mogą natomiast zawierać informacje istotne dla algorytmu ewolucyjnego, lecz nie mające bezpośredniego wpływu na przystosowanie osobnika. Każdy z chromosomów z kolei składa się z elementarnych jednostek zwanych genami.

Działanie algorytmu ewolucyjnego sprowadza się do wykonywania pętli, w której następują po sobie reprodukcja, operacje genetyczne, ocena i sukcesja. W literaturze przedmiotu reprodukcję i sukcesję określa się łącznym mianem selekcji. Reprodukcja w połączeniu z operatorami genetycznymi modeluje rozmnażanie, podczas którego materiał genetyczny rodziców jest przekazywany potomkom. Podczas reprodukcji zostają powielone losowo wybrane osobniki z populacji bazowej. Możliwe jest zarówno wielokrotne powielenie tego samego osobnika, jak i to, że niektóre osobniki nie zostaną wybrane ani razu do powielenia. Losowość wyboru do reprodukcji uwzględnia jednak wartości przystosowania osobników - charakteryzujące się większym przystosowaniem mają większe szanse powielenia.

Powstałe w wyniku reprodukcji kopie, zwane osobnikami rodzicielskimi, poddawane są operacjom genetycznym, które polegają na dokonaniu losowych modyfikacji ich genotypów. Mutacja polega na perturbacji genotypu jednego osobnika rodzicielskiego. Przyjmuje się najczęściej, że niewielkie perturbacje są bardziej prawdopodobne niż duże. Z kolei krzyżowanie jest operatorem genetycznym działającym na wielu osobnikach rodzicielskich i prowadzącym do wygenerowania jednego lub wielu osobników potomnych, których chromosomy powstają w wyniku wymieszania odpowiednich chromosomów pochodzących z różnych osobników rodzicielskich. Osobniki utworzone w wyniku działania operatorów genetycznych stanowią populację potomną.

Populacja potomna jest poddawana ocenie środowiska, po czym następuje sukcesja - tworzy się nowa populacja bazowa, mogąca zawierać osobniki zarówno z populacji potomnej, jak i ze starej populacji bazowej. Inicjacja pętli

ewolucji polega na utworzeniu początkowej populacji bazowej poprzez wygenerowanie genotypów osobników i obliczenie ich przystosowania. Proces inicjacji jest najczęściej losowy; może być przy tym uwzględniany wpływ środowiska. Cykl ewolucji może kończyć się wówczas, gdy przystosowanie osobników jest odpowiednio duże, lub gdy stwierdzi się, że stan populacji bazowej świadczy o stagnacji algorytmu. W przestrzeni genotypów funkcje przystosowania możemy sobie wyobrażać jako łańcuch wzgórz. Działanie algorytmu ewolucyjnego sprowadza się do premiowania takich osobników, które są lepiej przystosowane do środowiska, a zatem są położone bliżej wierzchołka jednego ze wzgórz. Premiowanie to ma jednak wbudowany element losowości, umożliwiający reprodukcję nawet bardzo słabo przystosowanych osobników. Ten właśnie mechanizm, dopuszczający pogorszenie populacji bazowej w kolejnej generacji, umożliwia wyjście z pułapek ewolucyjnych, czyli z takiej sytuacji, w której niewielkie zmiany fenotypu prowadzą do pogorszenia uzyskanego rozwiązania.

-- Wykłady z algorytmów ewolucyjnych „Jarosław Arabas”

Jakie są zastosowania algorytmu genetycznego?

- Optymalizacja: Algorytmy ewolucyjne są doskonałe w znajdowaniu optymalnych lub blisko optymalnych rozwiązań w przypadkach, gdzie przestrzeń rozwiązań jest zbyt duża lub zbyt skomplikowana, aby przeszukać ją w całości.
- Inżynieria i Projektowanie: Mogą być używane do projektowania skomplikowanych systemów, takich jak sieci komunikacyjne, struktury budowlane czy układy elektroniczne, gdzie wiele zmiennych i ograniczeń musi być uwzględnionych.
- Sztuczna Inteligencja i Machine Learning: Używane do automatycznego tworzenia i dostosowywania algorytmów lub modeli uczenia maszynowego.
- Gry komputerowe i symulacje: W modelowaniu zachowań postaci lub systemów ekologicznych, gdzie wymagana jest adaptacja i ewolucja w odpowiedzi na zmieniające się warunki.

-- https://www.baeldung.com/cs/genetic-algorithms-applications

Schemat blokowy algorytmu genetycznego

Opis wskazanej reprezentacji (kodowania) rozwiązania: kodowanie binarne

Kodowanie binarne w algorytmach genetycznych to metoda reprezentacji kandydatów na rozwiązania problemu za pomocą ciągów binarnych, znanych również jako chromosomy. W tej metodzie, każdy kandydat na rozwiązanie, czyli każdy osobnik w populacji, jest zakodowany jako ciąg bitów 0 i 1.

Podstawy kodowania binarnego:
- Reprezentacja chromosomu: Każdy chromosom jest ciągiem binarnym, gdzie każdy bit może reprezentować cechę problemu lub wartość zmiennej decyzyjnej.
- Długość chromosomu: Długość ciągu binarnego jest z góry określona i zależy od rozmiaru problemu oraz poziomu dokładności, z jaką chcemy reprezentować rozwiązania.
- Zakodowane wartości: Bity w chromosomie mogą bezpośrednio odpowiadać wartościom binarnym zmiennych lub być interpretowane jako liczby w innych systemach liczbowych (np. dziesiętnym) po przeprowadzeniu odpowiedniej konwersji.
-- Algorytmy_genetyczne_populacja-3 (https://zpe.gov.pl)

Metoda selekcji (reprodukcji): metoda turniejowa

Metoda turniejowa to popularna technika selekcji w algorytmach genetycznych, która służy do wybierania osobników do tworzenia nowej populacji, czyli do reprodukcji. Oparta jest na idei "turnieju" między osobnikami populacji, gdzie wybierani są "zwycięzcy" do krzyżowania i tworzenia potomstwa. Oto teoretyczny opis tej metody:

Zasada działania: W metodzie turniejowej, z całej populacji losowo wybiera się pewną liczbę osobników, tzw. uczestników turnieju. Liczba ta jest parametrem metody i jest zwykle znacznie mniejsza niż całkowita liczba osobników w populacji. Uczestnicy turnieju są następnie porównywani pod względem ich przystosowania — miary, która określa, jak dobrze osobnik jest przystosowany do środowiska, czyli jak dobrze rozwiązuje dany problem.

Proces selekcji:
- Wybór uczestników: Losowo wybiera się grupę osobników z populacji. Grupa ta stanowi uczestników pojedynczego turnieju.
- Ocena przystosowania: Dla każdego osobnika w turnieju oblicza się wartość funkcji przystosowania.
- Selekcja zwycięzcy: Osobnik z najwyższą(**w naszym przypadku najnizszą**) wartością funkcji przystosowania jest uznawany za zwycięzcę turnieju. W niektórych wariantach, aby zapewnić różnorodność genetyczną, zwycięzcą może zostać również osobnik z niższym przystosowaniem z pewnym określonym prawdopodobieństwem.
- Reprodukcja: Zwycięzcy są wybierani do krzyżowania, co prowadzi do stworzenia nowych osobników w następnej generacji.

Zalety metody turniejowej:
- Prostota: Metoda jest prosta do zrozumienia i implementacji.
- Elastyczność: Rozmiar turnieju można dostosować, co wpływa na presję selektywną. Mniejsze turnieje zwiększają różnorodność genetyczną, podczas gdy większe promują szybszą konwergencję.
- Skalowalność: Dobrze skaluje się do dużych populacji.
- Różnorodność: Może promować zachowanie różnorodności genetycznej w populacji.

--Algorytmy_genetyczne_populacja-3 (https://zpe.gov.pl)

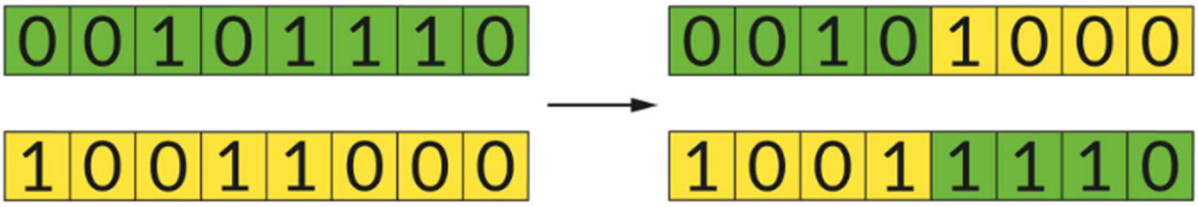
opis wskazanych operatorów genetycznych (krzyżowania oraz mutacji)

Krzyżowanie jednorodne (równomierne):

Krzyżowanie jednorodne, znane również jako krzyżowanie równomierne, jest metodą rekombinacji, w której każdy bit w chromosomie potomka jest wybierany losowo z jednego z dwóch rodziców. To znaczy, że dla każdego miejsca w chromosomie potomka niezależnie rzutuje się monetą, aby zdecydować, od którego rodzica będzie pochodził dany bit.

- Wybór rodziców: Algorytm wybiera pary rodziców, które mają się krzyżować.
- Tworzenie maski: Dla każdej pary rodziców tworzona jest maska binarna o długości równiej długości chromosomów. Maska jest generowana losowo i określa, który bit z której pary rodzicielskiej zostanie przekazany potomkowi.
- Rekombinacja: Dla każdego bitu w chromosomie potomka sprawdzana jest odpowiadająca pozycja w masce. Jeśli bit maski wynosi 0, potomek otrzymuje bit od pierwszego rodzica, a jeśli wynosi 1 - od drugiego rodzica.
- Powtórzenie: Proces jest powtarzany dla każdej pary rodziców w populacji.

Krzyżowanie jednorodne jest szczególnie skuteczne w utrzymaniu różnorodności genetycznej w populacji i może być użyteczne, gdy istotne są interakcje między różnymi bitami chromosomu.



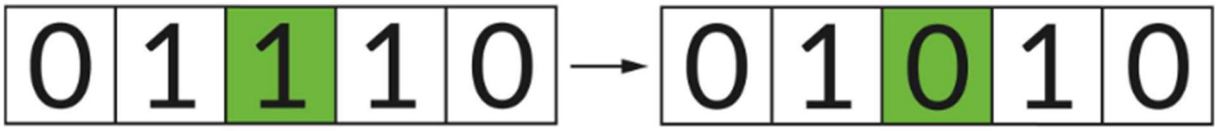
--Algorytmy_genetyczne_populacja-3 (https://zpe.gov.pl)

Mutacja losowa (negacja bitu):

Mutacja losowa, zwana również negacją bitu, jest prostym operatorem mutacji, który zmienia losowo wybrany bit w chromosomie z 0 na 1 lub z 1 na 0. Proces ten jest analogiczny do mutacji w naturze, gdzie drobna zmiana w DNA organizmu może prowadzić do zmiany cechy.

- Wybór osobnika: Algorytm wybiera osobnika, który ma ulec mutacji.
- Wybór bitu: Następnie losowo wybierany jest bit w chromosomie tego osobnika.
- Negacja bitu: Wybrany bit jest negowany, czyli jego wartość jest zmieniana na przeciwną (z 0 na 1 lub z 1 na 0).
- Powtórzenie: Proces może być powtórzony wielokrotnie w zależności od ustalonej stopy mutacji.

Mutacja losowa jest ważnym mechanizmem zapewniającym wprowadzanie nowych cech do populacji, które mogą nie być obecne w żadnym z rodziców. Stosuje się ją zwykle z niskim prawdopodobieństwem, aby nie zakłócić procesu adaptacji, lecz zapewnić dostateczną różnorodność i umożliwić algorytmowi wyjście z lokalnych minimów przestrzeni rozwiązań.



--Algorytmy_genetyczne_populacja-3 (https://zpe.gov.pl)

Implementacja algorytmu genetycznego

```
import random
wielkosc_populacji = 50
dlugosc_chromosomu = 14
ilosc_osobnikow_w_turnieju = 5
prawdopodobienstwo_mutacji = 0.001
prawdopodobienstwo_krzyzowania = 0.7
liczba_generacji = 100
zakres_x = (-50, 50)
srednie_przystosowania = []
najlepsze_przystosowania = []
pi = 3.141592

def random_float():
    return random.random()

def create_array(data):
    if isinstance(data, list):
        return [create_array(element) for element in data]
    else:
        return data

def mean(data):
    if isinstance(data[0], list):
        flattened_data = [item for sublist in data for item in sublist]
        return sum(flattened_data) / len(flattened_data)
    else:
        return sum(data) / len(data)

def argmin(values):
    return values.index(min(values))

def silnia(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * silnia(n - 1)

def cos(x, n=20):
    x = x % (2 * pi)
    suma = 0
    for i in range(n):
        term = (-1) ** i * x ** (2 * i) / silnia(2 * i)
        suma += term
    return suma



population = randint(0, 1, (wielkosc_populacji, dlugosc_chromosomu))

def dekoduj(chromosom, zakres_x, dlugosc_chromosomu):
    mid = dlugosc_chromosomu // 2

    max_bin_value = 2**mid - 1

    x1_value = sum(bit * (2 ** i) for i, bit in
enumerate(reversed(chromosom[:mid])))
    x2_value = sum(bit * (2 ** i) for i, bit in
enumerate(reversed(chromosom[mid:])))
```

```
    x1 = x1_value / max_bin_value * (zakres_x[1] - zakres_x[0]) +
zakres_x[0]
    x2 = x2_value / max_bin_value * (zakres_x[1] - zakres_x[0]) +
zakres_x[0]

    return x1, x2

for i in population:
    print(dekoduj(i,(-50,50), 14))

def funkcja_przystosowania(chromosom):

    x1, x2 = dekoduj(chromosom, zakres_x, dlugosc_chromosomu)
    return x1**2 + 2*x2**2 - 0.3 * cos(3 * pi * x1) - 0.4 * cos(4 * pi *
x2) + 0.7


def selekcja_turniejowa(population, funkcja_przystosowania,
tournament_size):
    wyniki_losowania = []
    for i in range(tournament_size):
        wylosowany = random.randint(0, (len(population)-1))
        wyniki_losowania.append(population[wylosowany])

    najlepszy_osobnik = None
    naj_przystosowanie = []
    for j in range(tournament_size):

naj_przystosowanie.append(funkcja_przystosowania(wyniki_losowania[j]))

    najlepszy_osobnik = min(naj_przystosowanie)
    index = naj_przystosowanie.index(najlepszy_osobnik)
    wybrany_osobnik = wyniki_losowania[index]

    return wybrany_osobnik

def krzyzowanie_jednorodne(parent1, parent2):
    offspring = []
    for i in range(dlugosc_chromosomu):
        if random_float() < 0.5:
            offspring.append(parent1[i])
        else:
            offspring.append(parent2[i])
    return create_array(offspring)

def mutacja_losowa(chromosome, mutation_rate):
    for i in range(len(chromosome)):
        random_number = random_float()
        if random_number < mutation_rate:
            if chromosome[i] == 0:
                chromosome[i] = 1
            else:
                chromosome[i] = 0
    return chromosome


for generation in range(liczba_generacji):
    new_population = []
    while len(new_population) < wielkosc_populacji:
```

```
        parent1 = selekcja_turniejowa(population, funkcja_przystosowania,
ilosc_osobnikow_w_turnieju)
        parent2 = selekcja_turniejowa(population, funkcja_przystosowania,
ilosc_osobnikow_w_turnieju)
        if random_float() < prawdopodobienstwo_krzyzowania:
            nowe_pokolenie_1 = krzyzowanie_jednorodne(parent1, parent2)
            nowe_pokolenie_2 = krzyzowanie_jednorodne(parent1, parent2)
        else:
            nowe_pokolenie_1, nowe_pokolenie_2 = parent1, parent2


        nowe_pokolenie_1 = mutacja_losowa(nowe_pokolenie_1,
prawdopodobienstwo_mutacji)
        nowe_pokolenie_2 = mutacja_losowa(nowe_pokolenie_2,
prawdopodobienstwo_mutacji)

        new_population.append(nowe_pokolenie_1)
        new_population.append(nowe_pokolenie_2)

    population = create_array(new_population[:wielkosc_populacji])

    fitness_scores = create_array([funkcja_przystosowania(individual) for
individual in population])

    srednie_przystosowanie = mean(fitness_scores)
    najlepsze_przystosowanie = min(fitness_scores)
    srednie_przystosowania.append(srednie_przystosowanie)
    najlepsze_przystosowania.append(najlepsze_przystosowanie)

    print(f"Generacja {generation}: Średnie Przystosowanie
{srednie_przystosowanie}, Najlepsze Przystosowanie
{najlepsze_przystosowanie}")

najlepszy_osobnik = population[argmin(fitness_scores)]
najlepsza_wartosc = funkcja_przystosowania(najlepszy_osobnik)

import matplotlib.pyplot as plt

print("\nNajlepsze znalezione rozwiązanie:")
print("Argument (chromosom):", najlepszy_osobnik)
print("Wartość funkcji przystosowania:", najlepsza_wartosc)

fig, ax1 = plt.subplots(figsize=(10, 5))

color = 'tab:blue'
ax1.set_xlabel('Generacja')
ax1.set_ylabel('Średnie przystosowanie', color=color)
ax1.plot(srednie_przystosowania, label='Średnie przystosowanie',
color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()
color = 'tab:red'
ax2.set_ylabel('Najlepsze przystosowanie', color=color)
ax2.plot(najlepsze_przystosowania, label='Najlepsze przystosowanie',
color=color, linestyle='--')
ax2.tick_params(axis='y', labelcolor=color)

plt.title('Średnie i najlepsze przystosowanie w kolejnych generacjach')
fig.tight_layout()
```

```
fig.legend(loc="upper right", bbox_to_anchor=(1,1),
bbox_transform=ax1.transAxes)
plt.show()
```

Funkcje pomocnicze:

```
def random_float():
    return random.random()

def create_array(data):
    if isinstance(data, list):
        return [create_array(element) for element in data]
    else:
        return data

def mean(data):
    if isinstance(data[0], list):
        flattened_data = [item for sublist in data for item in sublist]
        return sum(flattened_data) / len(flattened_data)
    else:
        return sum(data) / len(data)

def argmin(values):
    return values.index(min(values))

def silnia(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * silnia(n - 1)

def cos(x, n=20):
    x = x % (2 * pi)
    suma = 0
    for i in range(n):
        term = (-1) ** i * x ** (2 * i) / silnia(2 * i)
        suma += term
    return suma
```

Funkcja przystosowania

```
def funkcja_przystosowania(chromosom):

    x1, x2 = dekoduj(chromosom, zakres_x, dlugosc_chromosomu)
    return x1**2 + 2*x2**2 - 0.3 * cos(3 * pi * x1) - 0.4 * cos(4 * pi *
x2) + 0.7
```

Selekcja turniejowa

```
def selekcja_turniejowa(population, funkcja_przystosowania,
tournament_size):
    wyniki_losowania = []
    for i in range(tournament_size):
        wylosowany = random.randint(0, (len(population)-1))
        wyniki_losowania.append(population[wylosowany])

    najlepszy_osobnik = None
    naj_przystosowanie = []
    for j in range(tournament_size):

naj_przystosowanie.append(funkcja_przystosowania(wyniki_losowania[j]))

    najlepszy_osobnik = min(naj_przystosowanie)
    index = naj_przystosowanie.index(najlepszy_osobnik)
    wybrany_osobnik = wyniki_losowania[index]

    return wybrany_osobnik
```

Krzyżowanie jendorodne

```
def krzyzowanie_jednorodne(parent1, parent2):
    offspring = []
    for i in range(dlugosc_chromosomu):
        if random_float() < 0.5:
            offspring.append(parent1[i])
        else:
            offspring.append(parent2[i])
    return create_array(offspring)
```

Mutacja losowa

```
def mutacja_losowa(chromosome, mutation_rate):
    for i in range(len(chromosome)):
        random_number = random_float()
        if random_number < mutation_rate:
            if chromosome[i] == 0:
                chromosome[i] = 1
            else:
                chromosome[i] = 0
    return chromosome
```

Główna pętla programu

```
for generation in range(liczba_generacji):
    new_population = []
    while len(new_population) < wielkosc_populacji:

        parent1 = selekcja_turniejowa(population, funkcja_przystosowania,
ilosc_osobnikow_w_turnieju)
        parent2 = selekcja_turniejowa(population, funkcja_przystosowania,
ilosc_osobnikow_w_turnieju)
        if random_float() < prawdopodobienstwo_krzyzowania:
            nowe_pokolenie_1 = krzyzowanie_jednorodne(parent1, parent2)
            nowe_pokolenie_2 = krzyzowanie_jednorodne(parent1, parent2)
        else:
            nowe_pokolenie_1, nowe_pokolenie_2 = parent1, parent2


        nowe_pokolenie_1 = mutacja_losowa(nowe_pokolenie_1,
prawdopodobienstwo_mutacji)
        nowe_pokolenie_2 = mutacja_losowa(nowe_pokolenie_2,
prawdopodobienstwo_mutacji)

        new_population.append(nowe_pokolenie_1)
        new_population.append(nowe_pokolenie_2)

    population = create_array(new_population[:wielkosc_populacji])

    fitness_scores = create_array([funkcja_przystosowania(individual) for
individual in population])

    srednie_przystosowanie = mean(fitness_scores)
    najlepsze_przystosowanie = min(fitness_scores)
    srednie_przystosowania.append(srednie_przystosowanie)
    najlepsze_przystosowania.append(najlepsze_przystosowanie)

    print(f"Generacja {generation}: Średnie Przystosowanie
{srednie_przystosowanie}, Najlepsze Przystosowanie
{najlepsze_przystosowanie}")
```

Opis zadania optymalizacji

Nr 15. Klasyczny algorytm genetyczny z kodowaniem binarnym.

- metoda selekcji (reprodukcji): metoda turniejowa
- operatory genetyczne:
  - krzyżowanie: jednorodne(równomierne),
  - mutacja: losowe (negacja bitu).

$$x* = argmin(f(x_1, x_2))$$

$$f(x_{1,}x_2) = x_1^2 + 2\,x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$$

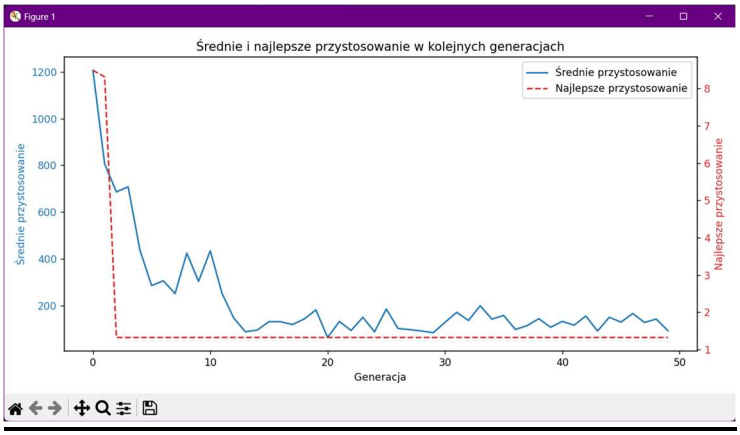
$$x_i \epsilon\, [-50, 50], i = 1{,}2$$

Testy

Zmiana wielkości populacji

NR TESTU	**Wielkość populacji**	Prawdopodobieństwo Mutacji	Prawdopodobieństwo Krzyżowania	Chromosom	Liczba generacji	L. zaw. w turnieju
TEST 1	**500**	0,01	0,4	14 bitów	50	5
TEST 2	**200**	0,01	0,4	14 bitów	50	5
TEST 3	**100**	0,01	0,4	14 bitów	50	5
TEST 4	**50**	0,01	0,4	14 bitów	50	5

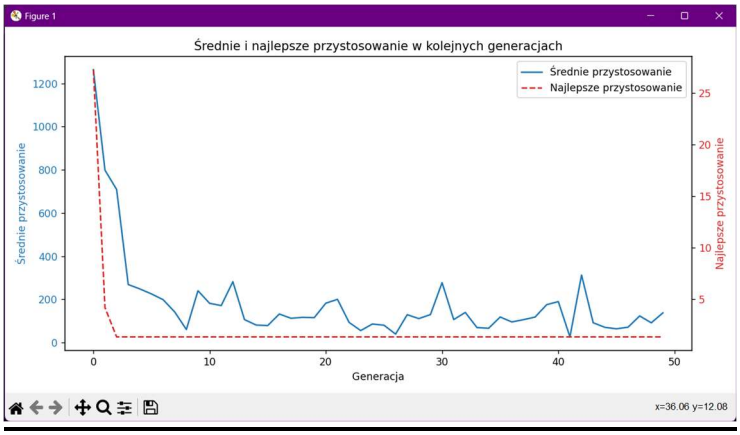
*TEST 1-**500**/0,01/0,4/14(bits)/100/5*



Rysunek 1 Liczba populacji: 500

500/0,01/0,4/14(bits)/100/5

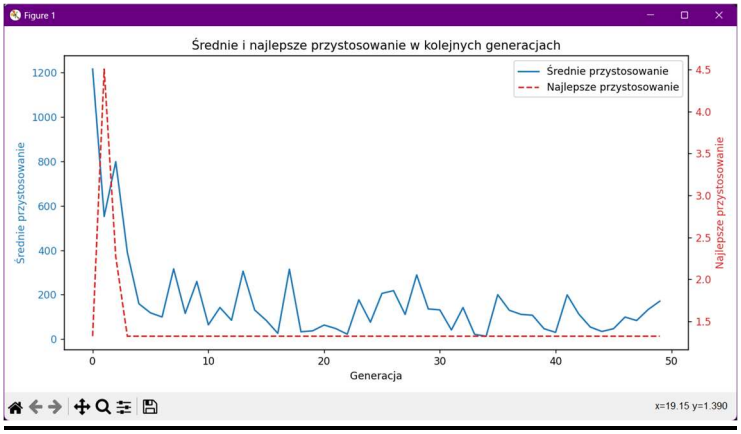
*TEST 2-**200**/0,01/0,4/14(bits)/100/5*



Rysunek 2 Liczba populacji: 200

200/0,01/0,4/14(bits)/100/5

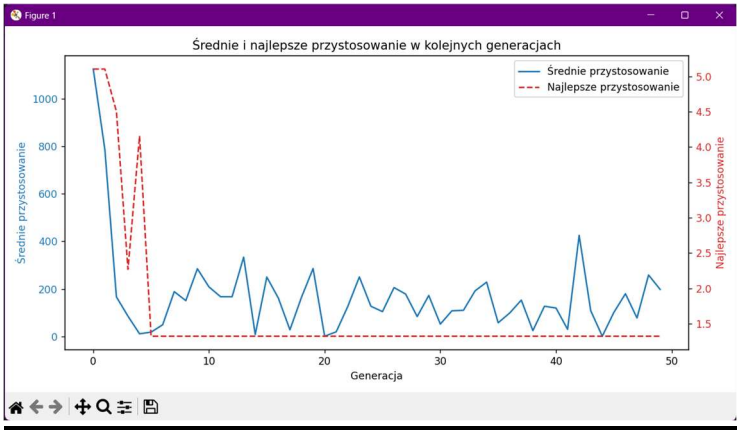
*TEST 3-**100**/0,01/0,4/14(bits)/100/5*



Rysunek 3 Liczba populacji: 100

100/0,01/0,4/14(bits)/100/5

*TEST 4-**50**/0,01/0,4/14(bits)/100/5*



Rysunek 4 Liczba populacji: 50

50/0,01/0,4/14(bits)/100/5

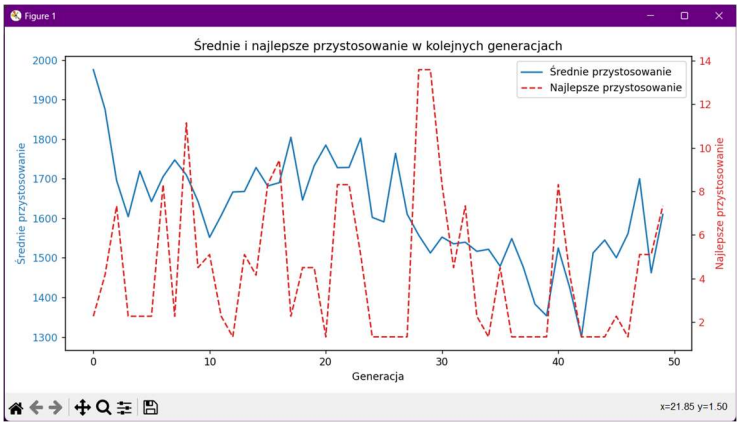
Wyniki:

NR TESTU/popul.	Średnie przystosowanie	Najlepsze przystosowanie	Chromosom	Wartość chromosomu
TEST 1 / 500	93.0538291100619	1.3246035151441193	[1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1]	(0.39, -0.39)
TEST 2 / 200	137.6987195970649	1.3246040236443983	[1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]	(0.39, 0.39)
TEST 3 / 100	170.79074223701056	1.3246038123724664	[0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]	(-0.39, 0.39)
TEST 4 / 50	198.37130084772198	1.3246038123724664	[0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]	(-0.39, 0.39)

Zmiana prawdopodobieństwa mutacji

NR TESTU	Wielkość populacji	**Prawdopodobieństwo Mutacji**	Prawdopodobieństwo Krzyżowania	Chromosom	Liczba generacji	L. zaw. w turnieju
TEST 1	500	**0,1**	0,4	14 bitów	50	5
TEST 2	500	**0,01**	0,4	14 bitów	50	5
TEST 3	500	**0,005**	0,4	14 bitów	50	5
TEST 4	500	**0,001**	0,4	14 bitów	50	5

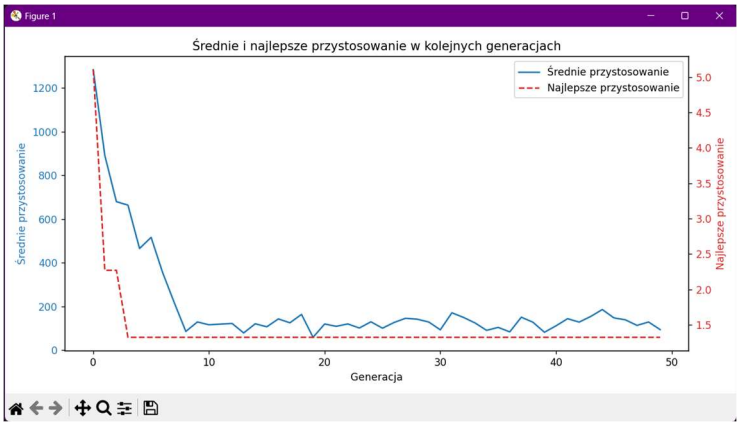
*TEST 1-500/**0,1**/0,4/14(bits)/100/5*



Rysunek 5 Prawdopodobieństwo mutacji: 0,1

500/**0,1**/0,4/14(bits)/100/5

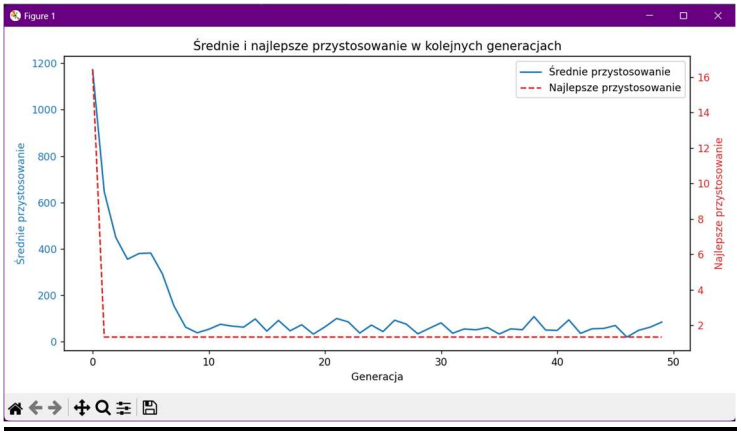
*TEST 2-500/**0,01**/0,4/14(bits)/100/5*



Rysunek 6 Prawdopodobieństwo mutacji: 0,01

500/**0,01**/0,4/14(bits)/100/5

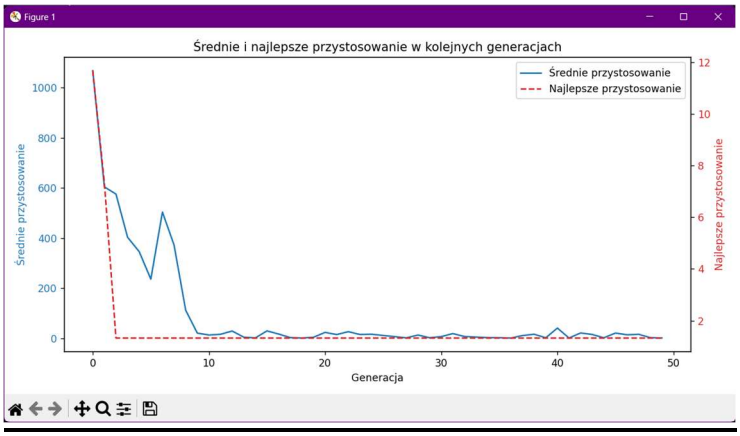
*TEST 3-500/**0,005**/0,4/14(bits)/100/5*



Rysunek 7 Prawdopodobieństwo mutacji: 0,005

500/**0,005**/0,4/14(bits)/100/5

*TEST 4-500/**0,001**/0,4/14(bits)/100/5*



Rysunek 8 Prawdopodobieństwo mutacji: 0,001

500/**0,001**/0,4/14(bits)/100/5

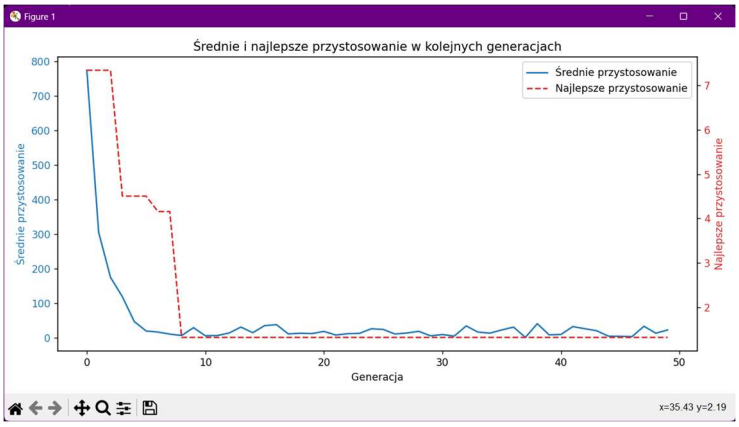
Wyniki:

NR TESTU/Praw. mut.	Średnie przystosowanie	Najlepsze przystosowanie	Chromosom(najlepszy)	Wartość chromosomu
TEST 1 / 0,1	1609.7921543525522	7.337348677327499	[0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0]	(-0.39, -0.39)
TEST 2 / 0,01	94.44886646806383	1.3246033038721872	[0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1]	(-0.39, -0.39)
TEST 3/ 0,005	84.72256049499515	1.3246033038721872	[0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1]	(-0.39, -0.39)
TEST 4/ 0,001	1.8699826122427026	1.3246033038721872	[0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1]	(-0.39, -0.39)

Zmiana prawdopodobieństwo krzyżowania

NR TESTU	Wielkość populacji	Prawdopodobieństwo Mutacji	**Prawdopodobieństwo Krzyżowania**	Chromosom	Liczba generacji	L. zaw. w turnieju
TEST 1	500	0,001	**0,1**	14 bitów	50	5
TEST 2	500	0,001	**0,3**	14 bitów	50	5
TEST 3	500	0,001	**0,6**	14 bitów	50	5
TEST 4	500	0,001	**0,9**	14 bitów	50	5

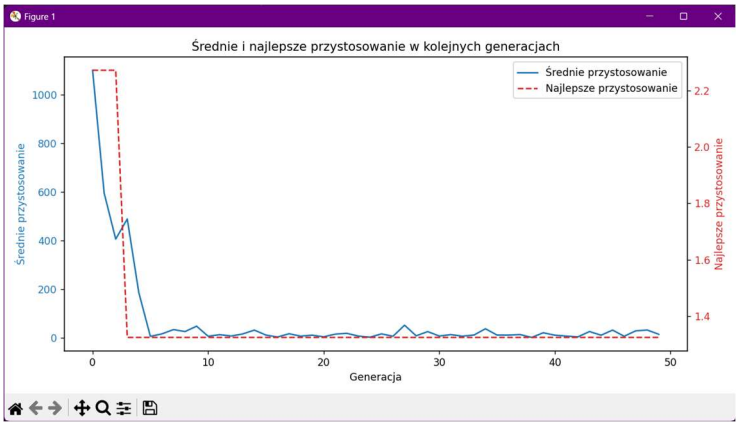
*TEST 1-500/0,001**/0,1**/14(bits)/100/5*



Rysunek 9 Prawdopodobieństwo krzyżowania: 0,1

500/0,001**/0,1**/14(bits)/100/5

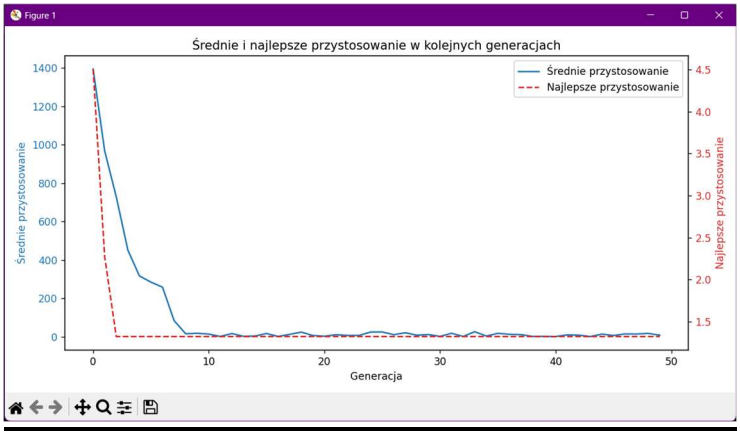
*TEST 2-500/0,001**/0,3**/14(bits)/100/5*



Rysunek 10 Prawdopodobieństwo krzyżowania: 0,3

500/0,001**/0,3**/14(bits)/100/5

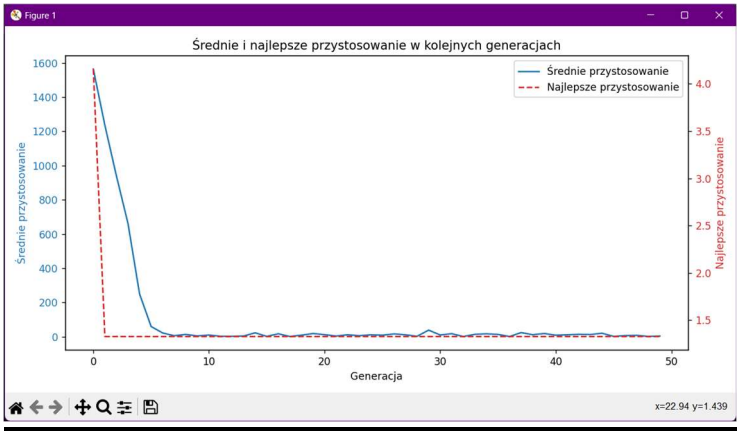
*TEST 3-500/0,001**/0,6**/14(bits)/100/5*



Rysunek 11 Prawdopodobieństwo krzyżowania: 0,6

500/0,001**/0,6**/14(bits)/100/5

*TEST 4-500/0,001**/0,9**/14(bits)/100/5*



Rysunek 12 Prawdopodobieństwo krzyżowania: 0,9

500/0,001**/0,9**/14(bits)/100/5

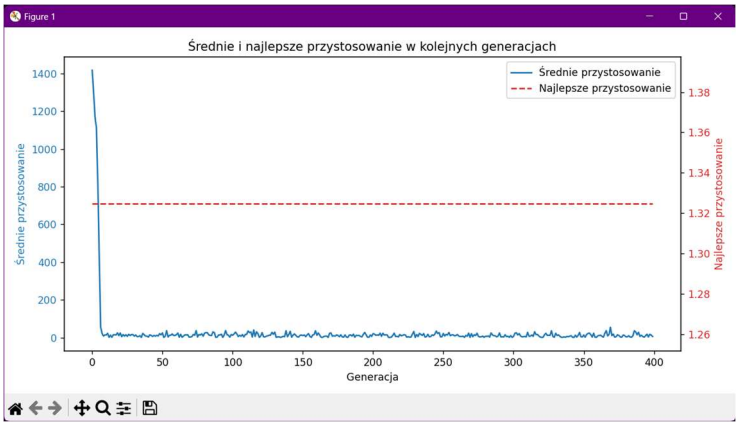
Wyniki:

NR TESTU/Praw.krzyż	Średnie przystosowanie	Najlepsze przystosowanie	Chromosom(najlepszy)	Wartość chromosomu
TEST 1 / 0,1	23.35333655796197	1.3246040236443983	[1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]	(0.39, 0.39)
TEST 2 / 0,3	14.552152081439313	1.3246033038721872	[0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1]	(-0.39, -0.39)
TEST 3 / 0,6	8.552768406691694	1.3246033038721872	[0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]	(-0.39, 0.39)
TEST 4 / 0,9	4.121947359361231	1.3246038123724664	[0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]	(-0.39, 0.39)

Zmiana liczby generacji

NR TESTU	Wielkość populacji	Prawdopodobieństwo Mutacji	Prawdopodobieństwo Krzyżowania	Chromosom	**Liczba generacji**	L. zaw. w turnieju
TEST 1	500	0,001	0,9	14 bitów	**400**	5
TEST 2	500	0,001	0,9	14 bitów	**200**	5
TEST 3	500	0,001	0,9	14 bitów	**50**	5
TEST 4	500	0,001	0,9	14 bitów	**10**	5

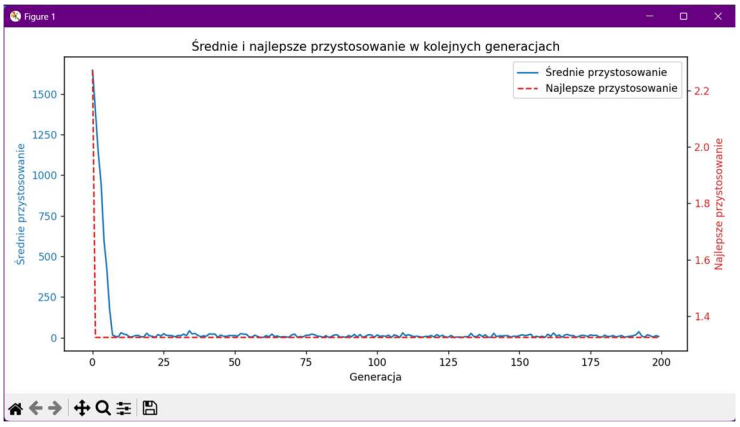
*TEST 1-500/0,001/0,9/14(bits)/**400**/5*



Rysunek 13 Liczba generacji: 400

500/0,001**/**0,9/14(bits)/**400**/5

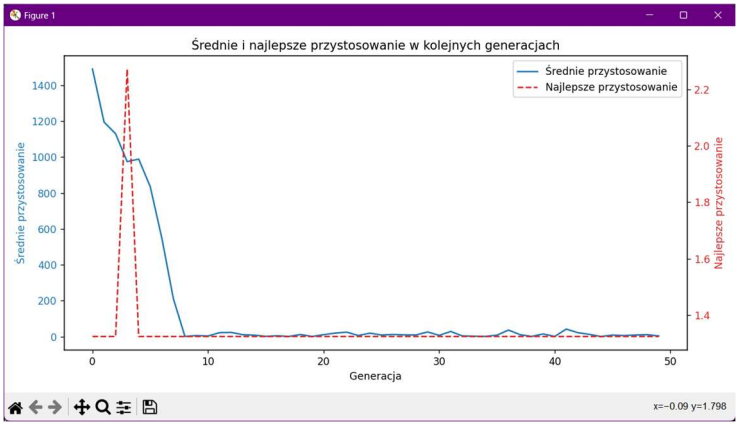
*TEST 2-500/0,001/0,9/14(bits)/**200**/5*



Rysunek 14 Liczba generacji: 200

500/0,001**/**0,9/14(bits)/**200**/5

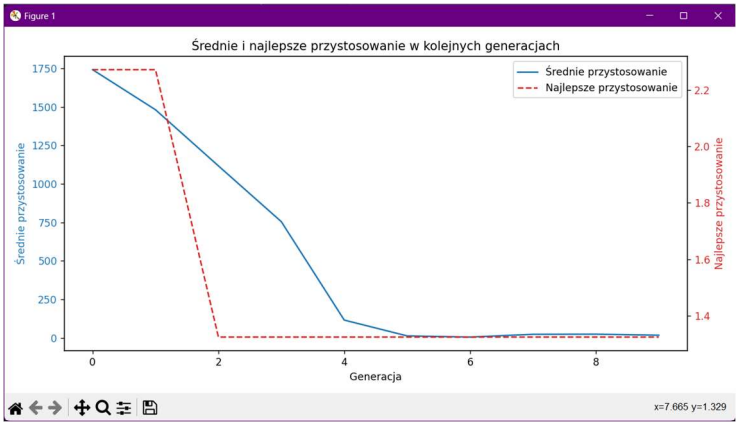
*TEST 3-500/0,001/0,9/14(bits)/**50**/5*



Rysunek 15 Liczba generacji: 50

500/0,001**/**0,9/14(bits)/**50**/5

*TEST 4-500/0,001/0,9/14(bits)/**10**/5*



Rysunek 16 Liczba generacji: 10

500/0,001**/**0,9/14(bits)/**10**/5

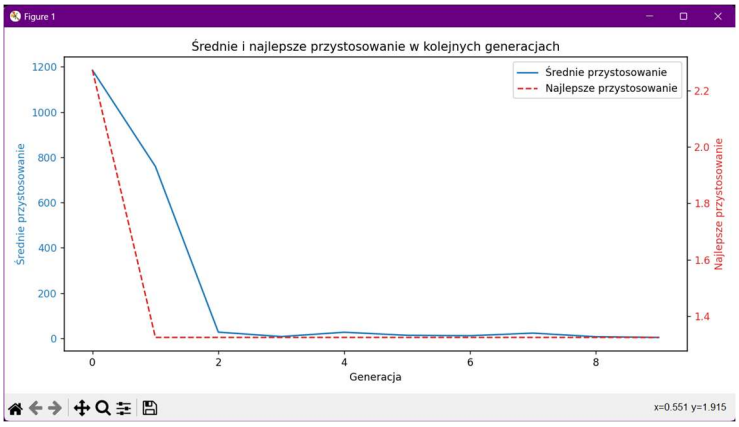
Wyniki:

NR TESTU / Generacje	Średnie przystosowanie	Najlepsze przystosowanie	Chromosom(najlepszy)	Wartość chromosomu
TEST 1/400	6.314421483333738	1.3246035151441193	[1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1]	(0.39, -0.39)
TEST 2/200	7.297264649059781	1.3246033038721872	[0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1]	(-0.39, -0.39)
TEST 3/50	4.311606764228393	1.3246038123724664	[0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]	(-0.39, 0.39)
TEST 4/10	17.141392230508142	1.3246035151441193	[1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1]	(0.39, -0.39)

Zmiana liczby zawodników w turnieju

NR TESTU	Wielkość populacji	Prawdopodobieństwo Mutacji	Prawdopodobieństwo Krzyżowania	Chromosom	Liczba generacji	**L. zaw. w turnieju**
TEST 1	500	0,001	0,9	14 bitów	10	**50**
TEST 2	500	0,001	0,9	14 bitów	10	**20**
TEST 3	500	0,001	0,9	14 bitów	10	**5**
TEST 4	500	0,001	0,9	14 bitów	10	**2**

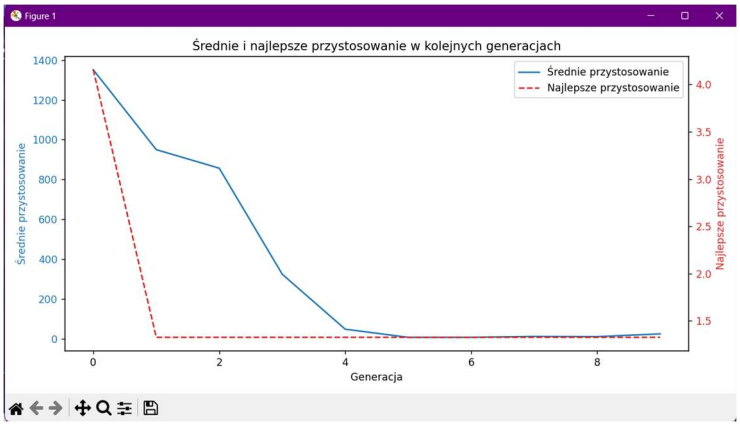
*TEST 1-500/0,001/0,9/14(bits)/10/**50***



Rysunek 17 L. Zaw. W turnieju: 50

500/0,001**/**0,9/14(bits)/10/**50**

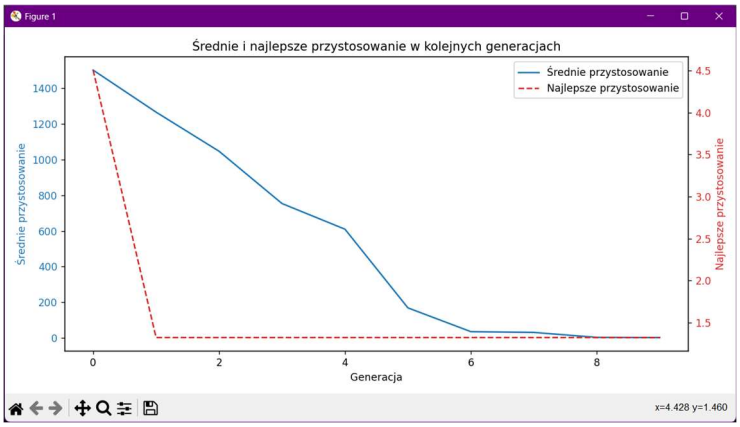
*TEST 2-500/0,001/0,9/14(bits)/10/**20***



Rysunek 18 L. Zaw. W turnieju: 20

500/0,001**/**0,9/14(bits)/10/**20**

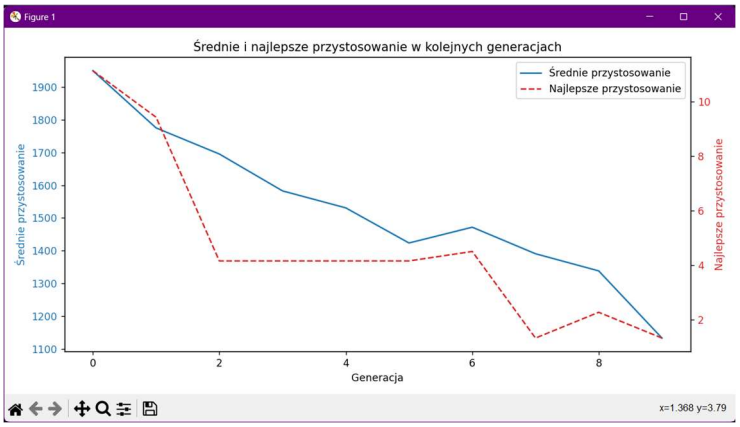
*TEST 3-500/0,001/0,9/14(bits)/10/**5***



Rysunek 19 L. Zaw. W turnieju: 5

500/0,001**/**0,9/14(bits)/10/**5**

*TEST 4-500/0,001/0,9/14(bits)/10/**2***



Rysunek 20 L. Zaw. W turnieju: 2

500/0,001**/**0,9/14(bits)/10/**2**

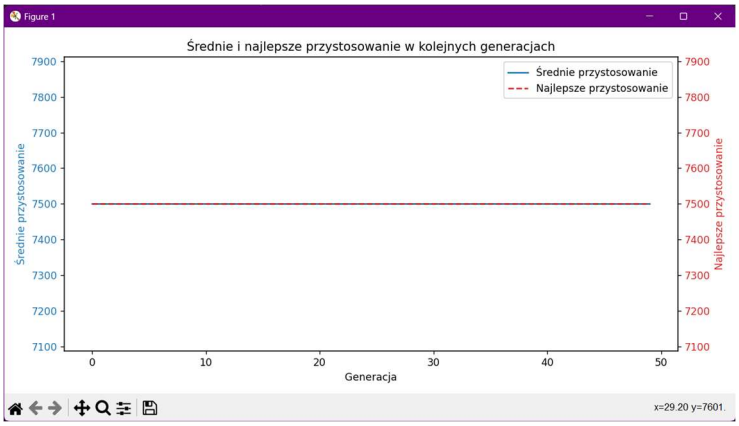
Wyniki:

NR TESTU/ Liczn. Turniej	Średnie przystosowanie	Najlepsze przystosowanie	Chromosom(najlepszy)	Wartość chromosomu
TEST 1/ 50	4.012434029727683	1.3246033038721872	[0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1]	(-0.39, -0.39)
TEST 2/ 20	25.638639564597494	1.3246033038721872	[0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1]	(-0.39, -0.39)
TEST 3 / 5	2.11453365847007	1.3246035151441193	[1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1]	(0.39, -0.39)
TEST 4 / 2	1132.915687226202	1.3246040236443983	[1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]	(0.39, 0.39)

Zmiana liczby genów w chromosomie

NR TESTU	Wielkość populacji	Prawdopodobieństwo Mutacji	Prawdopodobieństwo Krzyżowania	**Chromosom**	Liczba generacji	L. zaw. w turnieju
TEST 1	500	0,001	0,9	**2 bitów**	50	50
TEST 2	500	0,001	0,9	**6 bitów**	50	50
TEST 3	500	0,001	0,9	**10 bitów**	50	50
TEST 4	500	0,001	0,9	**14 bitów**	50	50

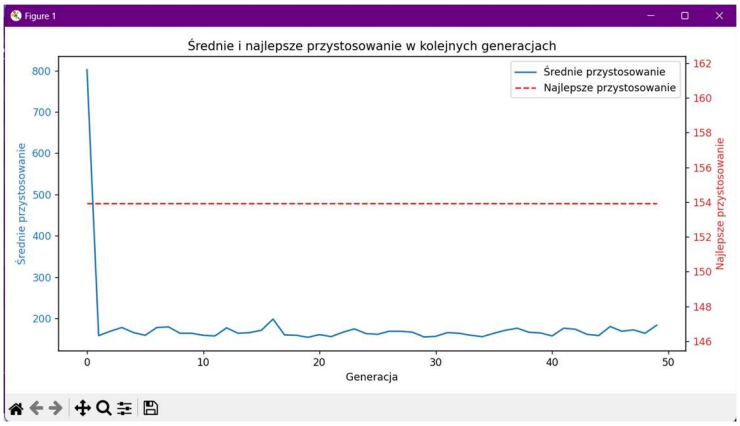
*TEST 1-500/0,001/0,9/**2 bity**/50/50*



Rysunek 21 Liczba genów: 2

500/0,001**/**0,9/**2 bity**/50/50

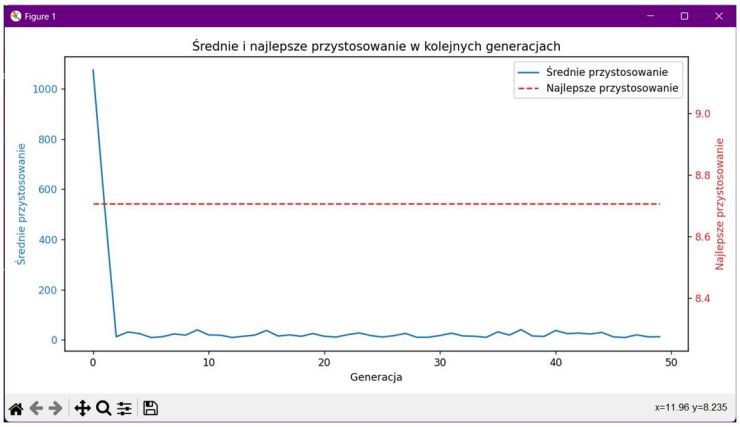
*TEST 2-500/0,001/0,9/**6 bity**/50/50*



Rysunek 22 Liczba genów: 6

500/0,001**/**0,9/**6 bity**/50/50

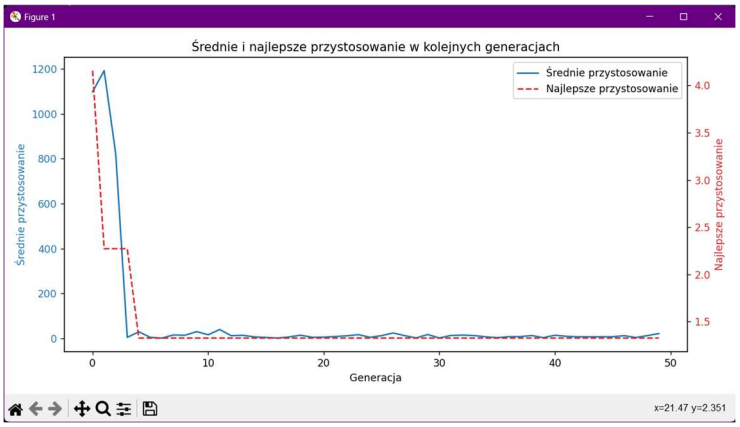
*TEST 3-500/0,001/0,9/**10 bity**/50/50*



Rysunek 23 Liczba genów: 10

500/0,001**/**0,9/**10 bity**/50/50

*TEST 4-500/0,001/0,9/**14 bity**/50/50*



Rysunek 24 Liczba genów: 14

500/0,001**/**0,9/**14 bity**/50/50

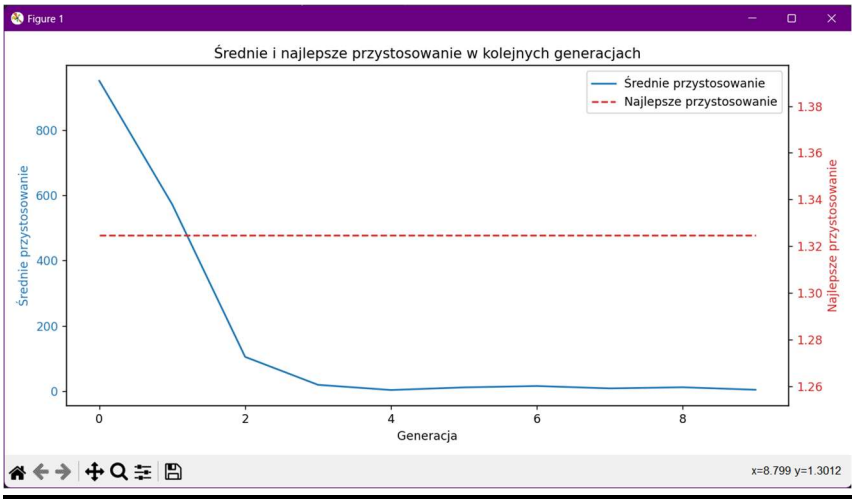
Wyniki:

NR TESTU/ Chromosom	Średnie przystosowanie	Najlepsze przystosowanie	Chromosom(najlepszy)	Wartość chromosomu
TEST 1 / 2	7500.0	7500.0	[0, 1]	(-49.21, -50.0)
TEST 2 / 6	183.30259720391422	153.91698888868405	[0, 1, 1, 1, 0, 0]	(-27.95, -50.0)
TEST 3/ 10	12.3658313080482	8.706103124777018	[1, 0, 0, 0, 0, 1, 0, 0, 0, 0]	(1.97, -50.0)
TEST 4/ 14	21.483906482480688	1.3246038123724664	[0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]	(-0.39, 0.39)

Rozwigzanie

Dla parametrów:

Wielkość populacji	Prawdopodobieństwo Mutacji	Prawdopodobieństwo Krzyżowania	Chromosom	Liczba generacji	L. zaw. W turnieju
500	0,001	0,9	14 bitów	10	50



Chromosom = [0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1]

X1 = -0.39370078740157055

X2 = -0.39370078740157055

F(x1, x2) = 1.3246038123724664

Wnioski

Im większa liczba osobników w populacji, tym bardziej równomierny jest przebieg ich przystosowania. Populacje o wielkości 500 lub 200 osobników wydają się bardziej stabilnie radzić sobie w procesie ewolucji, w porównaniu z mniejszymi grupami, jak 100 czy 50 osobników. Im większa populacja tym więcej jest przeprowadzanych ewolucji dzięki czemu mamy większą szanse na znalezienie najlepszego rozwiązania, a duża grupa będzie niwelować rozwój tych gorszych chromosomów, ponieważ szansa, że zostaną one wyeliminowane w procesie selekcji naturalnej, jest większa.

Mutacje w algorytmie genetycznym odgrywają kluczową rolę, ponieważ pozwalają na wprowadzenie nowych właściwości do populacji, które nie występowałyby inaczej. W naszym badaniu zauważyliśmy, że zbyt częste i intensywne mutacje, nawet jeśli są niewielkie, mogą prowadzić do dużych zmian w przystosowaniu poszczególnych pokoleń. To powoduje, że wyniki są bardzo niestabilne i trudno jest obserwować stopniową poprawę. Jednak gdy zminimalizowaliśmy prawdopodobieństwo mutacji do bardzo niskiego poziomu, jak 0,001, algorytm zaczął działać lepiej. Okazało się, że łagodniejsze podejście do mutacji przyczyniło się do efektywniejszego poszukiwania optymalnych rozwiązań. Podsumowując, nadmierne mutacje mogą zakłócać proces ewolucji, podczas gdy umiarkowane i kontrolowane mutacje wspierają go, prowadząc do bardziej konsekwentnych i korzystnych wyników.

Zbyt niskie prawdopodobieństwo krzyżowania może nie wystarczać do efektywnego mieszania genów. Na wykresach widać, że im większe prawdopodobieństwo tym bliżej średnie przystosowania znajdują się od tych najlepszych. Oznacza to, że im wiekse prawdopodobieństwo tym więcej jest branych najlepszych genów do tworzenia kolejnych populacji, a im mniejsze prawdopodobieństwo tym mniej dobrych genów jest wykorzystywanych do tworzenia następnych pokoleń co potwierdzają wachania średniego przystosowania(niebieskiej lini na wykresie). Wysokie prawdopodobieństwo w naszym algorytmie genetycznym się sprawdza co nie oznacza, że musi się ono sprawdzać w innych problemach.

W naszym przypadku duża liczba generacji jest zbędna, ponieważ algorytm genetyczny dosyć szybko znajduje najlepsze rozwiązanie. Przy ilości generacji 400 widać, że przystosowanie stabilizuje się i wydaję sie juz nie poprawiać co oznacza, że ustawienie tak dużej ilości generacji jest zbędne. Przy 10 i 50 generacjach lepiej widać co się dzieje na początku. W związku z tym zdecydowaliśmy się wybrać 10 generacji dla następnych testów dla lepszego zoobrazowania algorytmu i tego jak na nie wpływają pozostałe parametry.

Liczba osobników w turnieju miała znaczący wpływ na to jak szybko algorytm tworzył populacje z osobnikami o dobrych genach. Im wieksza liczba osobników w turnieju tym częściej wybierano dobre jednostki. Wraz ze zmniejszaniem liczby osobników pogorszały się też populacje co oznacza, że nie było w nich tylu dobrych osobników co w przypadku populacji w algorytmie z większą ilością osobników w turnieju. Częstsze wybieranie dobrych osobników może wpłynąć na prędkość znajdywania najlepszego rozwiązania.

Wartość najlepszego przystosowania zależała od dlugości danego chromosoma. Z wykresów wynika, że nieodpowiednia ilość genów ograniczała algorytm.

Algorytmy genetyczne są unikalne w swoim podejściu do rozwiązywania problemów, ponieważ czerpią z procesów ewolucyjnych, takich jak selekcja, krzyżowanie i mutacja, które występują w naturze. Są niezwykle elastyczne, co pozwala im znaleźć dobre rozwiązania dla złożonych i różnorodnych problemów, często tam, gdzie tradycyjne metody optymalizacji zawodzą. Jednakże, wymagają starannego doboru parametrów, takich jak wielkość populacji czy prawdopodobieństwo mutacji, aby

efektywnie działać. Są też z natury probabilistyczne, co oznacza, że mogą dostarczać różne rozwiązania przy każdym uruchomieniu.