**Dominik Trybuch**

**Sprawozdanie lab.3: Algorytmy genetyczne.**

Na tych laboratoriach mieliśmy za zadanie stworzyć algorytm genetyczny, który obliczał nam maksimum funkcji 𝑓(𝑥) = 𝑥 sin(10𝜋 ∗ 𝑥) + 1 w przedziale 𝑥 ∈ [−1,2]. Jako operatora selekcji użyłem metodę ruletki a jako operatora krzyżowania użyłem krzyżowanie jednopunktowe.

Kod Python :

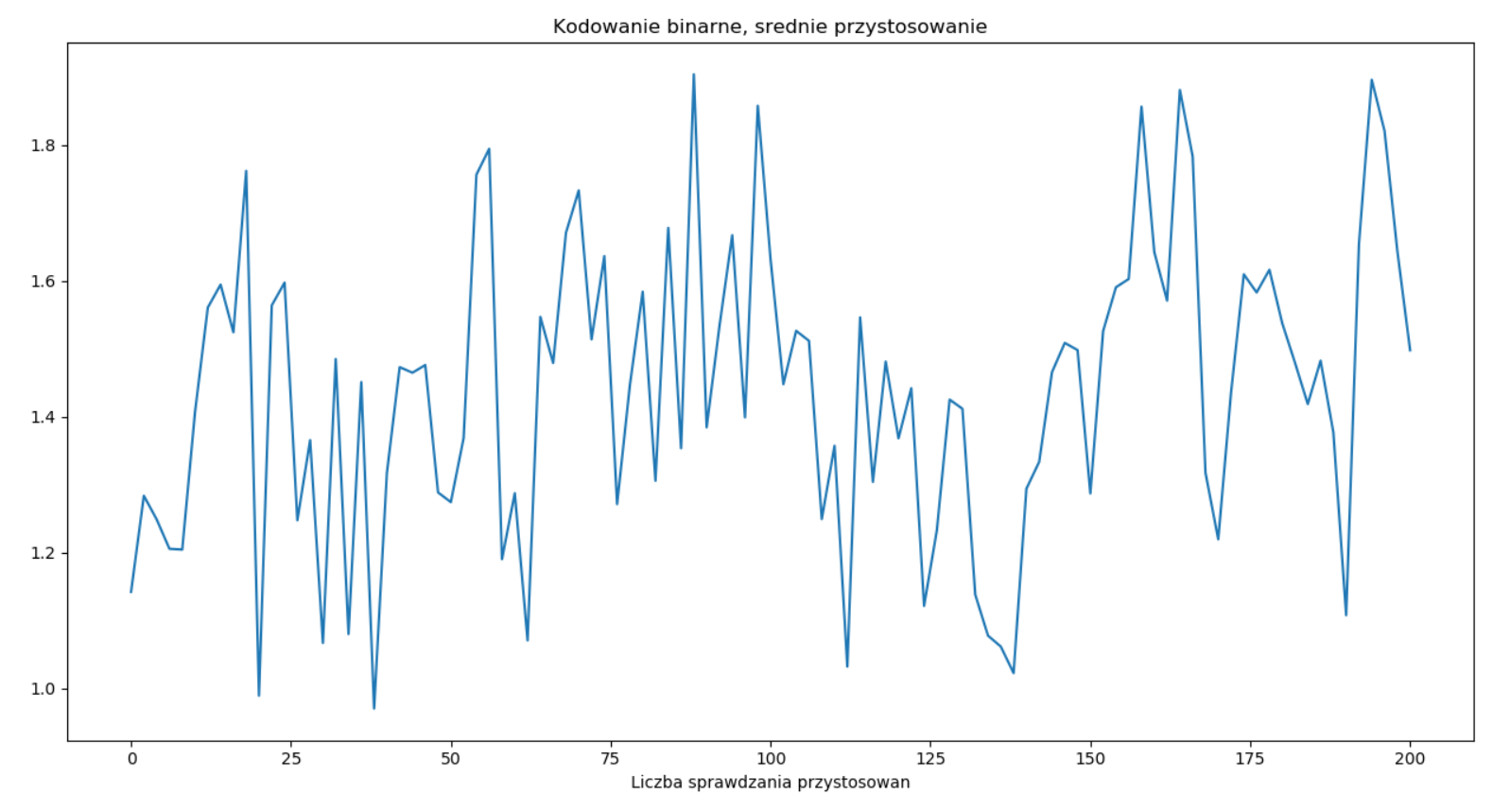
import numpy as np  
import math  
import random  
import matplotlib.pyplot as plt  
import pylab  
  
liczba\_pop = 20 #liczba chromosomow  
dlugosc\_pop = 22 #N genow  
population = np.random.randint(2, size=(liczba\_pop, dlugosc\_pop))  
  
  
class wynik:  
 najlepszy\_wynik = 0  
 index = -3  
 argument = 0  
 y = []  
 x = []  
 srednia = []  
 iteracja = []  
 liczba\_iteracji = 0  
 odch\_standardowe = []  
# print(new\_population)  
  
  
def funkcja(x):  
 return x \* math.sin(10 \* math.pi \* x) + 1  
  
  
def logical\_xor(x,y):  
 if x==y:  
 return 0  
 else:  
 return 1  
  
  
def przeliczanie\_na\_dz(ktora):  
 i = dlugosc\_pop - 1  
 suma1 = 0  
 for z in population[ktora, ]:  
 suma1 = suma1 + math.pow(z \* 2, i)  
 i = i - 1  
 wynik = suma1 / 1398101 - 1  
 return wynik  
  
  
def przeliczanie\_na\_graya(ktora):  
 zamienione = np.zeros((liczba\_pop, dlugosc\_pop), dtype=int) # tablica do zapisania zamienionych z kodu graya na binarne  
 #for q in range(liczba\_pop-1):  
 # zamienione[q, 0] = population[q,0]  
 zamienione[ktora, 0] = population[ktora, 0]  
 for g in range(1, dlugosc\_pop-1):  
 zamienione[ktora, g] = logical\_xor(zamienione[ktora, g-1], population[ktora, g])  
 i = dlugosc\_pop - 1  
 suma1 = 0  
 for z in zamienione[ktora, ]:  
 suma1 = suma1 + math.pow(z \* 2, i)  
 i = i - 1  
 wynik = suma1 / 1398101 - 1  
 return wynik  
  
  
def naj\_przystosowanie():  
 #print(przeliczanie\_na\_dz(2))  
 najlepszy = przeliczanie\_na\_dz(0)  
 #najlepszy = przeliczanie\_na\_graya(0)  
  
 index = 0  
 przystosowanie = []  
 suma = 0  
 argument = 0  
 for x in range(liczba\_pop - 1):  
 przystosowanie.append(funkcja(przeliczanie\_na\_dz(x)))  
 #przystosowanie.append(funkcja(przeliczanie\_na\_graya(x)))  
 suma = suma + funkcja(przeliczanie\_na\_dz(x))  
 #suma = suma + funkcja(przeliczanie\_na\_graya(x))  
 if przystosowanie[x] > najlepszy:  
 najlepszy = przystosowanie[x]  
 index = x  
 argument = przeliczanie\_na\_dz(x)  
 #argument = przeliczanie\_na\_graya(x)  
 if najlepszy > wynik.najlepszy\_wynik:  
 wynik.najlepszy\_wynik = najlepszy  
 wynik.index = index  
 wynik.argument = argument  
  
 wynik.y.append(najlepszy)  
 wynik.x.append(argument)  
 wynik.srednia.append(suma/liczba\_pop)  
 wynik.liczba\_iteracji += 1  
 suma\_odchylenie = 0  
 for i in range(liczba\_pop - 1):  
 suma\_odchylenie += ((suma/liczba\_pop)- przystosowanie[i]) \* ((suma/liczba\_pop)-przystosowanie[i])  
 wynik.odch\_standardowe.append(math.sqrt(suma\_odchylenie/liczba\_pop))  
 #print(przystosowanie)  
 #print("Najlepszy:\n", najlepszy)  
 #print("Index:\n", index)  
 #print("Argument", argument)  
 #print("Suma", suma)  
 #print(population)  
 return przystosowanie  
  
  
def selekcja():  
 przystosowanie = naj\_przystosowanie()  
 szerokosc = []  
 # suma przystosowan  
 suma1 = 0  
 # tablica z ruletka  
 ruletka = np.zeros(liczba\_pop)  
 # tablica przezywajacych  
 survivors = []  
 for i in range(liczba\_pop):  
 przystosowanie.append(funkcja(przeliczanie\_na\_dz(i)))  
 #przystosowanie.append(funkcja(przeliczanie\_na\_graya(i)))  
 suma1 = suma1 + funkcja(przeliczanie\_na\_dz(i))  
 #suma1 = suma1 + funkcja(przeliczanie\_na\_graya(i))  
 for l in range(liczba\_pop):  
 szerokosc.append(funkcja(przeliczanie\_na\_dz(l)) / suma1)  
 #szerokosc.append(funkcja(przeliczanie\_na\_graya(l)) / suma1)  
 if l == 0:  
 ruletka[l] = szerokosc[l]  
 else:  
 ruletka[l] = ruletka[l - 1] + szerokosc[l]  
 #print(ruletka)  
 for q in range(liczba\_pop):  
 losowa = random.uniform(0, 1)  
 for i in range(liczba\_pop):  
 if losowa <= ruletka[i]:  
 survivors.append(i)  
 break  
 # print(survivors)  
 return survivors  
  
  
def zamiana\_pop(populacja):  
 survivors = selekcja()  
 new\_population = np.zeros((liczba\_pop, dlugosc\_pop), dtype=int)  
 for o in range(liczba\_pop-1):  
 new\_population[o] = (populacja[survivors[o]])  
 return new\_population  
  
  
def krzyzowanie():  
 # liczba osobnikow do krzyzowania  
 pcross = 0.2  
 ilosc = math.floor(pcross \* liczba\_pop)  
 rantab = [] # tablica wylosowanych elementow  
 for x in range(ilosc):  
 obecny = random.randint(1, liczba\_pop)  
 for i in range(x):  
 if rantab[i] == obecny:  
 obecny = random.randint(1, liczba\_pop)  
 rantab.append(obecny)  
 if ilosc % 2 != 0:  
 rantab.append((random.randint(1, liczba\_pop)))  
 ilosc += 1  
 # print(rantab)  
 dlugosc = math.floor(ilosc / 2)  
 locus = random.randint(1, dlugosc\_pop)  
 #print(locus)  
 # krzyzowanie wlasciwe  
 licznik = 0  
 while licznik < dlugosc:  
 for i in range(locus, dlugosc\_pop):  
 if population[rantab[licznik] - 1, i] != population[rantab[licznik + 1] - 1, i]:  
 tmp = population[rantab[licznik] - 1, i]  
 population[rantab[licznik] - 1, i] = population[rantab[licznik + 1] - 1, i]  
 population[rantab[licznik + 1] - 1, i] = tmp  
 licznik += 2  
  
  
def mutacja():  
 pmut = 0.2  
 ilosc = math.floor(pmut\*liczba\_pop\*dlugosc\_pop)  
 rantab = np.zeros((ilosc, 2), dtype=int) # tablica wylosowanych elementow  
 for liczba in range(ilosc):  
 rantab[liczba, 0] = random.randint(1, liczba\_pop)  
 rantab[liczba, 1] = random.randint(1, dlugosc\_pop)  
  
 for i in range(ilosc-1):  
 if population[rantab[i, 0]-1, rantab[i, 1]-1] == 0:  
 population[rantab[i, 0]-1, rantab[i, 1]-1] = 1  
 else:  
 population[rantab[i, 0]-1, rantab[i, 1]-1] = 0  
  
  
i = 0  
  
while i < 300:  
 wynik.iteracja.append(wynik.liczba\_iteracji)  
 population = zamiana\_pop(population) #selekcja  
 krzyzowanie()  
 mutacja()  
 naj\_przystosowanie()  
 wynik.iteracja.append(wynik.liczba\_iteracji)  
 i += 1  
  
print("Index: ", wynik.index, "\n Argument: ", wynik.argument, "\n Najlepszy: ", wynik.najlepszy\_wynik)

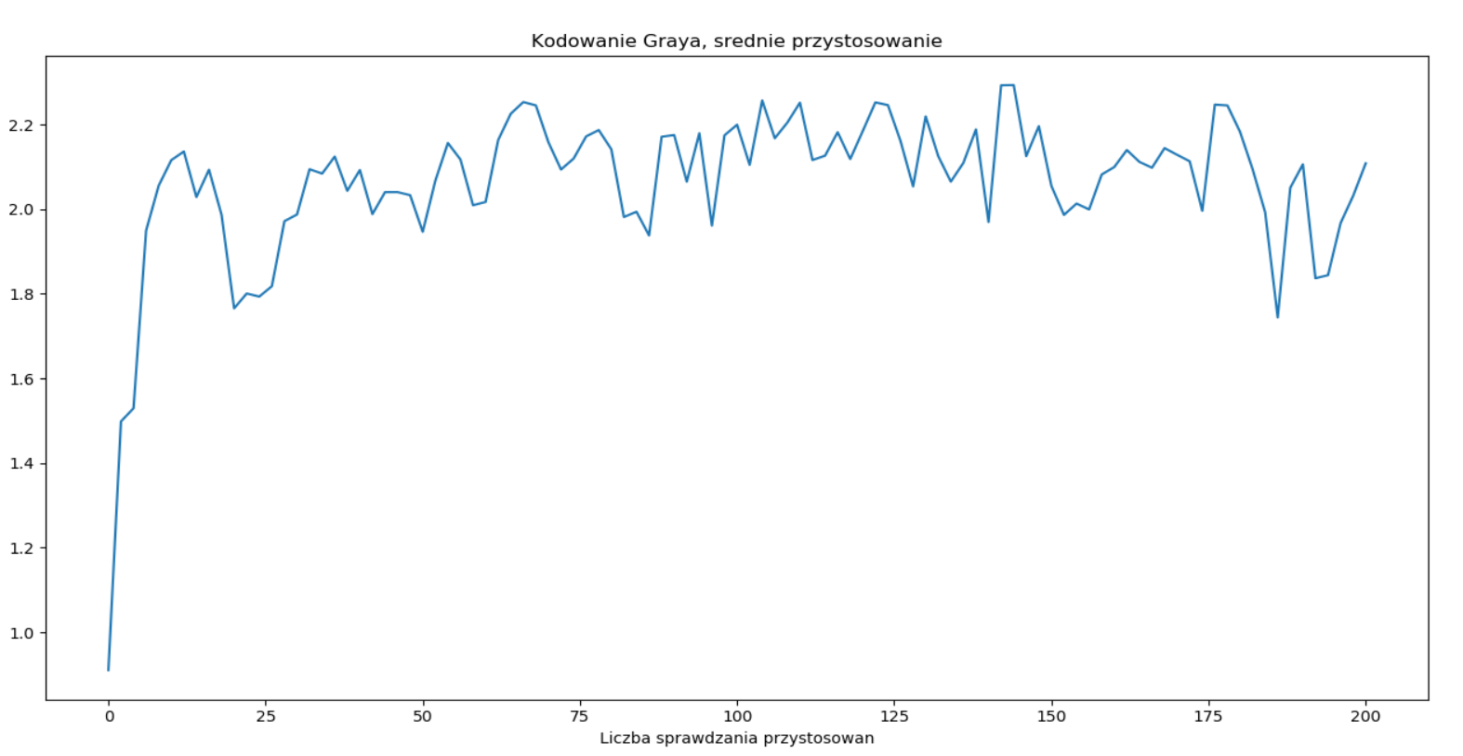
W zależności od prawdopodobieństwa krzyżowania i mutacji otrzymujemy różne wyniki. Jeżeli prawdopodobieństwo jest bardzo małe np. pcross = 0.01 i pmut =0.01, to średni najlepszy wynik wyniósł 2.36. Jeżeli natomiast pcross = 0.2, a pmut pozostało niezmienne to wynik oscylował w granicy 2.85 co jest dobrym wynikiem. Jeżeli natomiast zwiększyłem pmut do 0.1, a pcross zmieniłęm na 0.01 to wynik wyniósł 2.8503 co jest najlepszym wynikiem. Nie powinno się jednak ustawiać pmut na tak wysoką liczbę, gdyż ilość mutacji powinna być znacząco mniejsza od liczby krzyżowań.

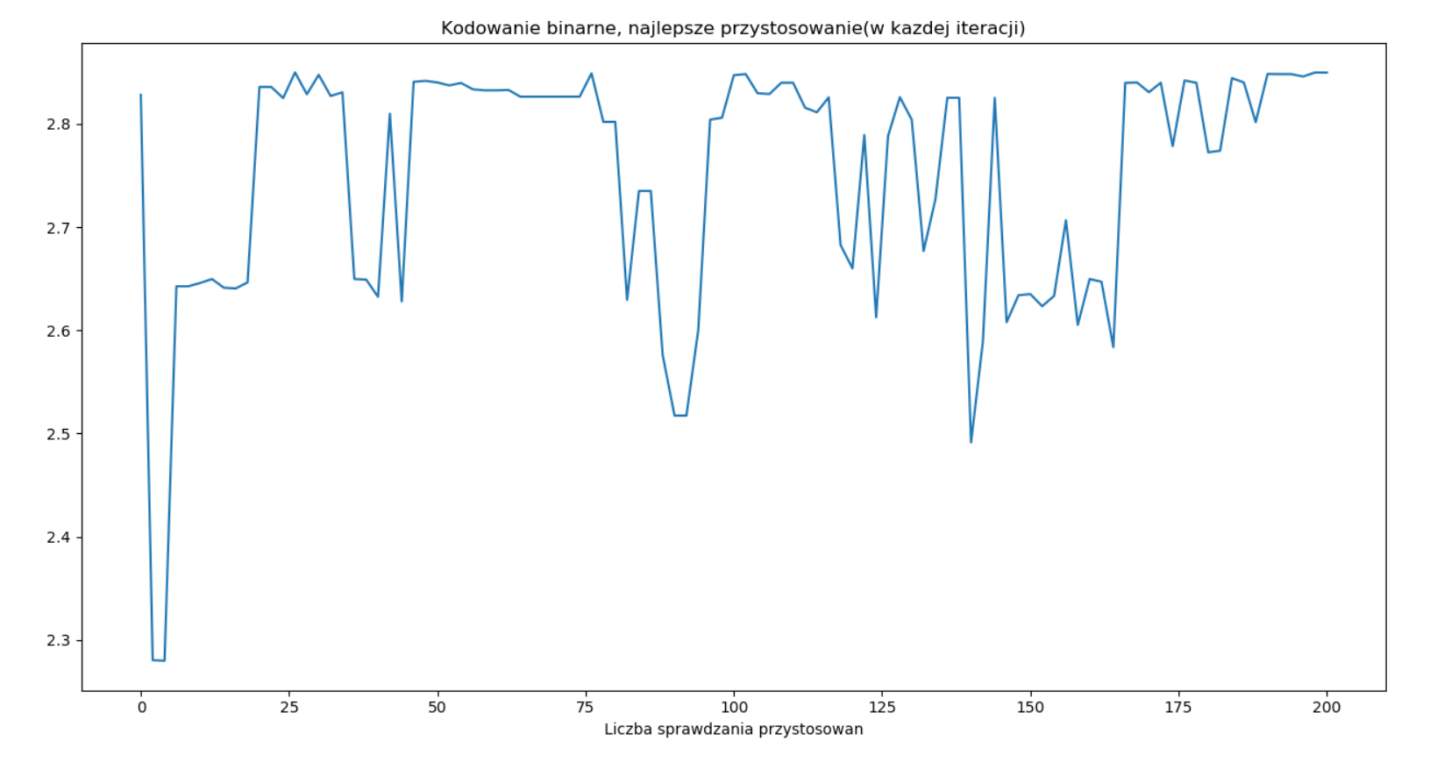
Sprawdziłem także wyniki zakodowania kodem Graya i w porównaniu z kodowaniem binarnym znajdują one lepsze rozwiązanie. Ponadto średnie wyniki są bardziej „stabilne”, czyli wyniki nie są aż tak bardzo rozrzucone i są lepsze jak można zauważyć na wykresach poniżej.

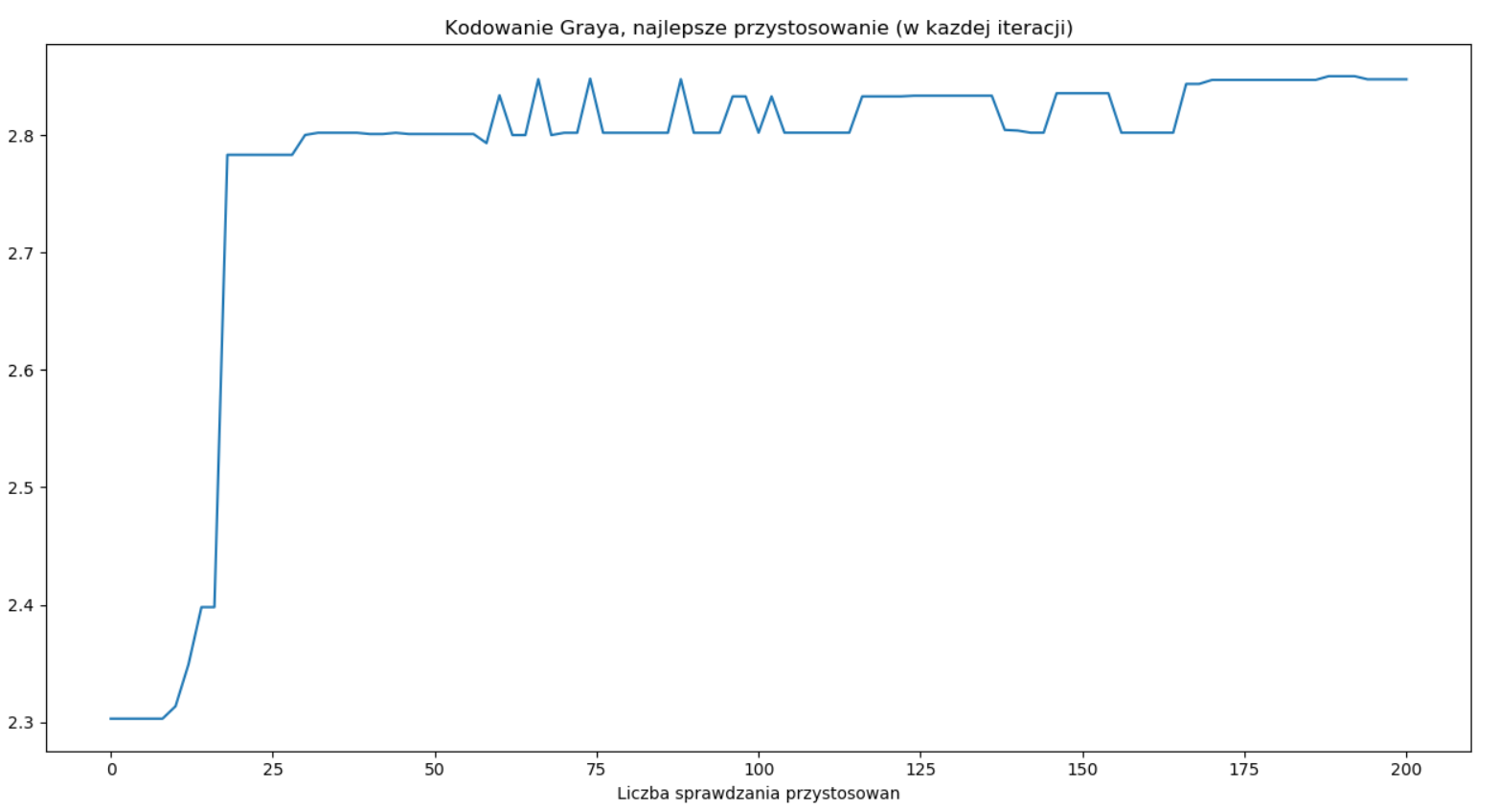
Podobnie najlepsze rozwiązania z użyciem kodu Graya w każdej iteracji są wyższe niż wyniki z użyciem kodowania binarnego. Praktycznie żaden najlepszy wynik od pewnego momentu nie spada poniżej 2.8.

Co więcej przy zastosowaniu kodu Graya, przy niższym prawdopodobieństwie mutacji i krzyżowania, algorytm lepiej znajduje maksimum niż przy użyciu kodowania binarnego z tymi samymi prawdopodobieństwami.









Można także użyć inne metody selekcji np. strategia elitarna, gdzie zachowuje się najlepszego osobnika za bieżącej populacji i wprowadza się go bez zmian do następnej populacji. Inna metodą jest selekcja turniejowa, która dzieli populacje na grupy, a następnie „rozgrywa się turniej” pomiędzy osobnikami poszczególnych grup. Metoda ta bardzo często zastępuje metodę ruletki.   
 Kolejną metodą jest selekcja rankingowa. Polega ona na wyborze osobników zgodnie z przypisanymi im rangami (nadawanymi w zależności od wartości przystosowania). Korzystając z listy rankingowej tworzona jest funkcja określająca liczbę wybieranych kopii chromosomów w zależności od ich rangi.

Alternatywą dla krzyżowania jednopunktowego jest krzyżowanie wielopunktowe. Polega ono na wybraniu więcej niż jednego punktu krzyżowanie chromosomów. Usprawnia on proces krzyżowania, kiedy korzysta się z długich chromosomów. Innym sposobem jest krzyżowanie równomierne, gdzie losuje się wzorzec, który określa które geny potomków są dziedziczone od każdego z rodziców. Jest on łańcuchem binarnym. Wartość 1 wskazuje pozycje w chromosomie rodzica 1 a wartość 0 w chromosomie rodzica 2.