Министерство науки и высшего образования Российской Федерации НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)

Институт прикладной математики и компьютерных наук

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

По дисциплине «Интеллектуальные системы»

ГЕНЕТИЧЕСКИЙ АЛГОРИТМ

Вежнина Надежда Дмитриевна

Направление подготовки 02.03.02 Фундаментальная информатика и информационные системы

Направленность (профиль) «Искусственный интеллект и разработка программных продуктов»

Руководитель работы

А. Д. Брагин

Автор работы

Студент группы №932201

Н. Д. Вежнина

Томск – 2024

**1. Цель работы**

Цель работы - дание программы, реализующей генетический алгоритм для решения задачи оптимизации.

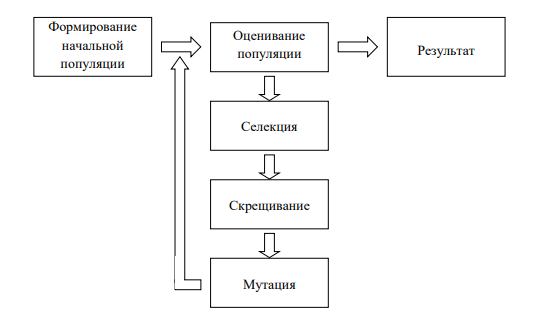
**2. Постановка задачи**

Вариант №8. Найти с помощью генетического алгоритма особь, гены которой соответствуют, в формате RGB, фиолетовому цвету (96, 96, 159).

**3. Метод решения задачи**

В работе используется язык программирования Python. Для решения задачи используется генетический алгоритм. Поиск начинается с произвольных значений целевой функции. Алгоритм использует целочисленное кодирование. Эволюция продолжается, пока не будет найдено решение, но не более 10000 поколений. Размер популяции – 6 особей. В алгоритме используется рулеточная селекция, одноточечный кроссинговер и битовая мутация. Вероятность кроссинговера Pc = 0,9. Вероятность мутации Pm = 0,045. Разрыв поколений T = 1.

**4. Структурная схема алгоритма**



**5. Листинг программы**

import random

population = []

for i in range(6):

a = random.randint(0, 256)

b = random.randint(0, 256)

c = random.randint(0, 256)

population.append([a, b, c])

print(population)

def fit(individual):

return 1 / (abs(individual[0] - 96) + abs(individual[1] - 96) + abs(individual[2] - 159))

population\_fitness = []

for i in range(len(population)):

fitness = fit(population[i])

population\_fitness.append(fitness)

print("Fitness of individual #", i, "equals:", fitness)

fitness\_prob = []

for i in range(len(population\_fitness)):

prob = population\_fitness[i]/sum(population\_fitness)

fitness\_prob.append(prob)

print('Probability of crossing for individual #', i, 'equals:', prob)

def roulette\_wheel\_pop(population, probabilities, number):

chosen = []

for n in range(number):

r = random.random()

prob\_circle = 0

for i in range(len(probabilities)):

prob\_circle += probabilities[i]

if r <= prob\_circle:

chosen.append(population[i])

break

return chosen

def to\_binary(individual):

return "{0:b}".format(individual[0]).zfill(8) + "{0:b}".format(individual[1]).zfill(8) + "{0:b}".format(individual[2]).zfill(8)

def to\_decimal(individual):

x = individual[8:]

return [int(individual[:8], 2), int(x[:8], 2), int(individual[16:], 2)]

def pop\_to\_binary(population):

binary\_pop = []

for individual in population:

binary\_pop.append(to\_binary(individual))

return binary\_pop

def pop\_to\_decimal(population):

decimal\_pop = []

for individual in population:

decimal\_pop.append(to\_decimal(individual))

return decimal\_pop

def get\_uniq\_individual(population):

new\_population = []

for elem in population:

if elem not in new\_population:

new\_population.append(elem)

return new\_population

def crossover(parent\_1, parent\_2):

x = random.randint(1, len(parent\_1) - 1)

return parent\_1[:x] + parent\_2[x:]

def mutation(individual):

Pm = 0.045

result = ''

for digit in individual:

if (random.random() < Pm):

result += '1' if digit == '0' else '0'

else:

result += digit

return result

def population\_crossover(population):

new\_pop = []

for i in range(0, len(population), 2):

if 0.9 > random.random():

new\_pop.append(mutation(crossover(population[i], population[i+1])))

new\_pop.append(mutation(crossover(population[i+1],population[i])))

else:

new\_pop.append(population[i])

new\_pop.append(population[i+1])

return new\_pop

test = 0

for j in range(10000):

population\_fitness = []

for i in range(len(population)):

fitness = fit(population[i])

population\_fitness.append(fitness)

fitness\_prob = []

for i in range(len(population\_fitness)):

prob = population\_fitness[i]/sum(population\_fitness)

fitness\_prob.append(prob)

population\_for\_crossover = roulette\_wheel\_pop(population, fitness\_prob, 6)

binary\_population\_for\_crossover = pop\_to\_binary(population\_for\_crossover)

new\_population = population\_crossover(binary\_population\_for\_crossover)

population\_for\_crossover = get\_uniq\_individual(population\_for\_crossover)

new\_population = pop\_to\_decimal(new\_population)

try:

new\_population.sort(key=fit, reverse = True)

population = new\_population

except ZeroDivisionError:

print('Last population # ', j, ':')

print(new\_population)

for i in range(len(new\_population)):

print('Objective function of individual #', i, 'equals:', abs(population[i][0] - 96) + abs(population[i][1] - 96) + abs(population[i][2] - 159))

break

for i in range(len(population)):

if fit(population[i]) >= 1:

test = 1

print('Last population # ', j, ':')

print(new\_population)

for i in range(len(new\_population)):

print('Objective function of individual #', i, 'equals:', abs(population[i][0] - 96) + abs(population[i][1] - 96) + abs(population[i][2] - 159))

break

if test == 1:

break

**6. Результаты работы генетического алгоритма**

Алгоритм находит оптимальное значение целевой функции в течение числа поколений, варьирующегося в приблизительных пределах от нескольких десятков до тысячи, в зависимости от начальных условий. Полученное решение с высокой точностью соответствует требуемому.

**7. Выводы**

Таким образом, можно сделать вывод о том, что генетический алгоритм является эффективным способом решения оптимизационных задач.

В процессе работы мы получили навыки разработки, настройки и использования генетический алгоритмов.