Министерство науки и высшего образования Российской Федерации

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)

Институт прикладной математики и компьютерных наук

ОТЧЕТ

по дисциплине «Интеллектуальные системы»

на тему «Нахождение с помощью генетического алгоритма особь, гены которой соответствуют, в формате RGB, фиолетовому цвету»

Выполнили студенты команды №1

Квашнина А., Мирошник С., Киреев Д.,
Ковалева Н., Маслова Е., Урбановский Е.

Томск—2020

1. Цель работы

Написать программу, которая реализует генетический алгоритм для нахождения особи, гены которой соответствуют, в формате RGB, фиолетовому цвету.

2. Постановку задачи.

Для формулирования/формирования/составления/и тд. верной постановки задачи необходимо:

А) Определить количество и тип оптимизируемых переменных задачи, которые необходимо закодировать в хромосоме:

Числовое представление цветовой модели RGB можно задать в виде куба,

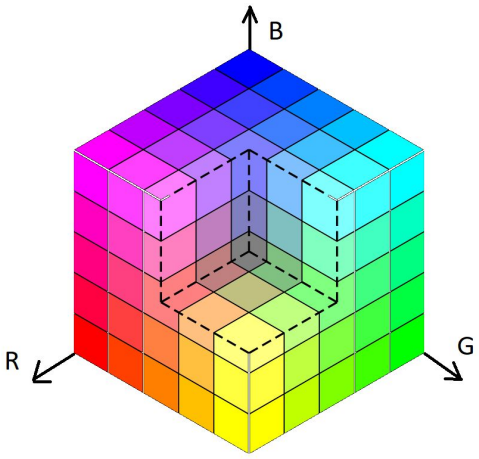

Рис 1.RGB-цветовая модель, представленная в виде куба

где каждая из координатных осей характеризуется значением от 0 до 255, которое можно представить байтом.

Целевыми переменными в данном случае будут являться значения координатных осей R, G, и B.

Б) Определить критерий оценки особей, задав функцию приспособленности (целевую функцию).

Приспособленность i-ой особи f_i можно задать как

$$96*R+96*G+96*B \rightarrow \min, \text{ где } R, G, B \in [0, 255].$$

В) Выбрать способ кодирования его параметры.

Так как значения координатных осей могут быть представлены в виде целого числа, то для кодирования хромосомы было выбрано целочисленное кодирование с тремя генами размерности 8.

Г) Определить параметры (размер популяции, тип селекции, генетические операторы и их вероятности, величина разрыва поколений).

Для тестирования влияния различных характеристик размер популяции пользователь может задать самостоятельно. Для селекции была выбрана селекция усечением с возможностью самостоятельного задания значения коэффициента l (по умолчанию 0,5). В качестве генетических операторов будем использовать 1-точечный кроссинговер и битовую мутацию. Вероятности применения операторов скрещивания и мутации установим равными 0,8 и 0,01, соответственно. Новое поколение будем формировать только из особей-потомков, т.е. величина разрыва поколений T равна 1.

3. Метод решения задачи.

Для решения поставленной задачи необходимо выполнить следующие действия:

1. Создать начальную популяцию
2. Выполнить оценку популяции
3. Выбрать лучшую особь популяции
4. Провести селекцию отсечением
5. Выполнить скрещивание с помощью одноточечного кроссинговера
6. Выполнить алгоритм мутации
7. Вывести результаты

Программная реализация создана с использованием объектно ориентированного подхода на языке Python.

4. Структурную схему алгоритма.

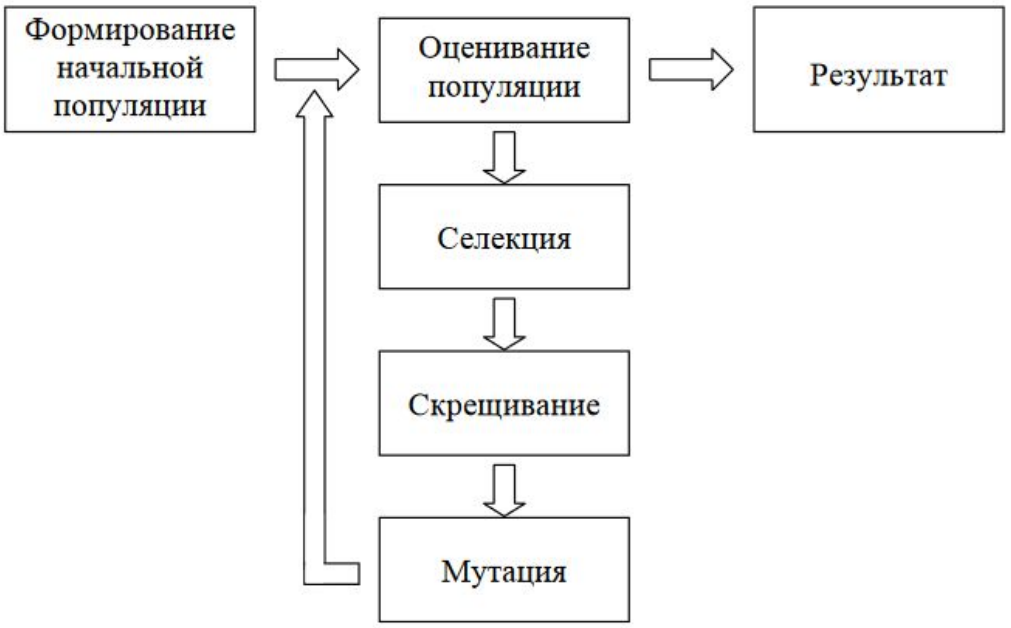

Рис 2. Схема работы генетического алгоритма

5. Листинг программы.

см. Приложение

6. Результаты работы генетического алгоритмы.

Для тестирования выбрано 200 особей, 60 поколений.

Лучшая особь в начальной популяции:

```
best_score = math.inf
best_chromosome = []

for i in x.population:
  if fitness_function(i.convert_chromosome(8)) < best_score:
    best_score = fitness_function(i.convert_chromosome(8))
    best_chromosome = i

print(best_chromosome.convert_chromosome(8), best_chromosome.chromosome, fitness_function(best_chromosome.convert_chromosome(8)))
```

```
[76, 113, 146] 01001100011100011001001O 50
```

Рис 3. RBG представление цвета, хромосома и значение целевой функции

После выполнения алгоритма:

```
best_score = math.inf
best_chromosome = []

for i in x.population:
  if fitness_function(i.convert_chromosome(8)) < best_score:
    best_score = fitness_function(i.convert_chromosome(8))
    best_chromosome = i

print(best_chromosome.convert_chromosome(8), best_chromosome.chromosome, fitness_function(best_chromosome.convert_chromosome(8)))
```

```
[95, 96, 160] 01011111011000001010000 2
```

Рис 4. RBG представление цвета, хромосома и значение целевой функции лучшей особи

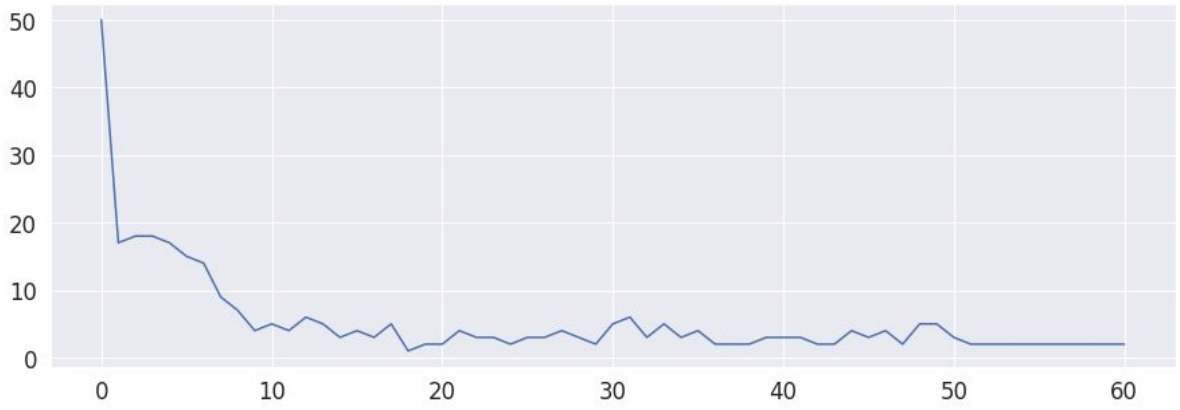

Рис 5. График изменения значения целевой функции лучшей особи в i-ом поколении

Вышло:

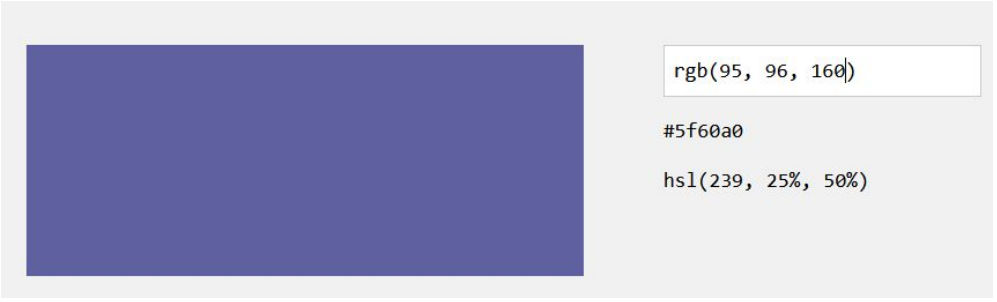

Рис 6. Цвет, полученный использованием сервиса w3schools.com

Цель:

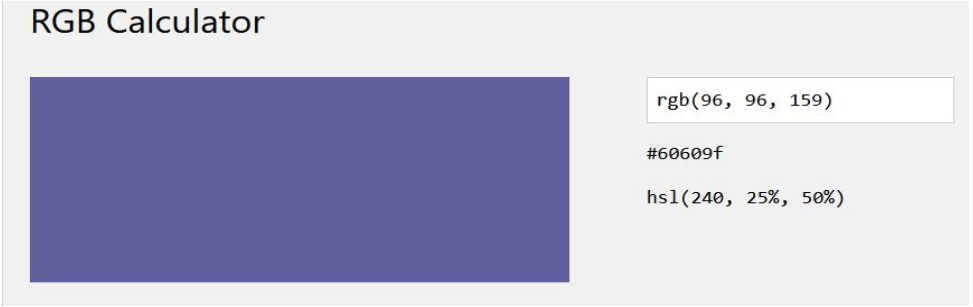

Рис 7. Целевой цвет, полученный использованием сервиса w3schools.com

Результаты теста соответствуют ожиданиям, и программа работает верно.

7. Выводы.

В результате работы была написана программа по нахождению особи, гены которой соответствуют, в формате RGB, фиолетовому цвету. Разработанная программа выводит наиболее подходящую особь.

ПРИЛОЖЕНИЕ

```
def fitness_function(individual):
  return(abs(96 - individual[0]) + abs(96 - individual[1]) + abs(159 - individual[2]))
```

Рис.8 Целевая функция

```
class Individual:
  chromosome = ''
  def __init__(self, min_diap, max_diap, count_gene):
    self.chromosome = []

    for i in range(0, count_gene):
      gene = random.randint(min_diap, max_diap)
      buff = '00000000'

      for j in range(0, len(buff)):
        buff = buff[:len(buff) - 1 - j] + str(gene % 2) + buff[len(buff) - 1 - j + 1:]
        gene = gene // 2
      self.chromosome.append(buff)

    self.chromosome = ''.join(self.chromosome)

  def convert_chromosome(self, size_gene):
    k = 0
    gene_list = []
    while k < len(self.chromosome):
      gene_list.append(int(self.chromosome[0 + k:size_gene + k], 2))
      k = k + size_gene
    return(gene_list)
```

Рис. 9 Класс особи, в котором создается особь популяции и производится операция перевода из двоичной в десятичную запись числа для последующей оценки популяции

```
def create_population(self, size, min_diap, max_diap, count_gene, size_gene):
  self.print_best_score = []
  self.population = []
  self.size_gene = size_gene
  self.count_gene = count_gene
  self.size_population = size
  for i in range(0, self.size_population):
    self.population.append(Individual(min_diap, max_diap, count_gene))
```

Рис. 10 Создание начальной популяции

```
def selection(self, l = 0.5):
  score_fitness_function = []
  for i in self.population:
    score_fitness_function.append(fitness_function(i.convert_chromosome(self.size_gene)))

  mask = []
  for j, i in zip(self.population, score_fitness_function):
    if i < np.quantile(score_fitness_function, 1 - l):
      mask.append(j)

  self.population = mask
```

Рис. 11 Селекция отсечением

```
def crossbreeding(self, Pc = 0.8):
  k = 0
  new_population = []

  while k < self.size_population:
    i = random.randint(0, len(self.population) - 1)
    j = random.randint(0, len(self.population) - 1)
    if Pc > random.uniform(0, 1):
      w = random.randint(1, len(self.population[i].chromosome) - 1)
      child_1 = Individual(1, 1, 1)
      child_2 = Individual(1, 1, 1)
      child_1.chromosome = self.population[i].chromosome[:w] + self.population[j].chromosome[w:]
      child_2.chromosome = self.population[j].chromosome[:w] + self.population[i].chromosome[w:]
      new_population.append(child_1)
      new_population.append(child_2)
    else:
      new_population.append(self.population[i])
      new_population.append(self.population[j])
    k = k + 2

  self.population = new_population
```

Рис. 12 Скрещивание, одноточечный кроссинговер

```
def mutation(self, probability):
  for i in self.population:
    for j in range(0, len(i.chromosome)):
      if probability > random.uniform(0, 1):
        if i.chromosome[j] == '0':
          i.chromosome = i.chromosome[:j] + '1' + i.chromosome[j + 1:]
        else:
          i.chromosome = i.chromosome[:j] + '0' + i.chromosome[j + 1:]
```

Рис. 13 Мутация