

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА**

**по дисциплине**

**«Разработка систем поддержки принятий решений»**

Студент группы: ИКБО-14-20 Вежновец Ф.Ю. *(Фамилия студента)*

Руководитель самостоятельной работы Холмогоров В.В.

*(Фамилия преподавателя)*

|  |  |  |
| --- | --- | --- |
|  |  |  |

Москва 2023

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc138002076)

[1 ЭВОЛЮЦИОННЫЙ АЛГОРИТМ 4](#_Toc138002077)

[1.1 Постановка задачи 4](#_Toc138002078)

[1.2 Описание алгоритма 4](#_Toc138002079)

[1.3 Результат работы программы 4](#_Toc138002080)

[ЗАКЛЮЧЕНИЕ 6](#_Toc138002081)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 7](#_Toc138002082)

[ПРИЛОЖЕНИЯ 8](#_Toc138002083)

ВВЕДЕНИЕ

Эволюционные стратегии (ЭС), также как и предыдущие парадигмы, основаны на эволюции популяции потенциальных решений, но, в отличие от них, здесь используется генетические операторы на уровне фенотипа, а не генотипа, как это делается в ГА. Разница в том, что ГА работают в пространстве генотипа – кодов решений, в то время как ЭС производят поиск в пространстве фенотипа – векторном пространстве вещественных чисел. В ЭС читываются свойства хромосомы «в целом», в отличие от ГА, где при поиске решений исследуются отдельные гены. В природе один ген может одновременно влиять на несколько свойств организма. С другой стороны, одно свойство особи может определяться несколькими генами. Естественная эволюция основана на исследовании совокупности генов, а не отдельного (изолированного) гена.

Ранние эволюционные стратегии (ЭС) основывались на популяции, состоящей из одной особи, и в них использовался только один генетический оператор – мутация. Здесь для представления особи (потенциального решения) была использована идея, не представленная в классическом генетическом алгоритме, которая заключается в следующем

1. ЭВОЛЮЦИОННЫЙ АЛГОРИТМ

1.1 Постановка задачи

Реализовать эволюционный алгоритм для задачи коммивояжёра.

* 1. Описание алгоритма

В качестве особи мы принимаем путь, а в качестве генов мы принимаем номера городов, через которые будем идти. Лучшей особью будет считаться та у которой путь через все города будет наименьшим.

В алгоритме мы будем использовать мутацию.

Для мутации будем ген и менять его с другим. Чем дальше заменяемый ген тем меньше вероятность его замены (Рисунок 1).

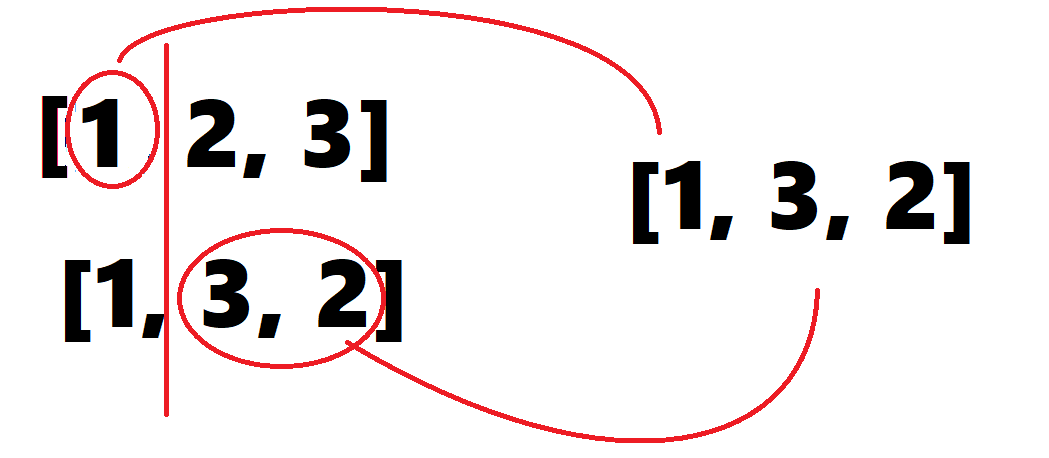


Рисунок 1 - Мутация

Популяция ограничена, будем отбирать только 5 самых лучших особей.

* 1. Результат работы программы

Сгенерируем рандомно длины путей орт города к городу. Всего городов 6, начальная популяция состоит из 6 особей, 10 итераций. Результат работы алгоритма представлен на рисунках 2-3.

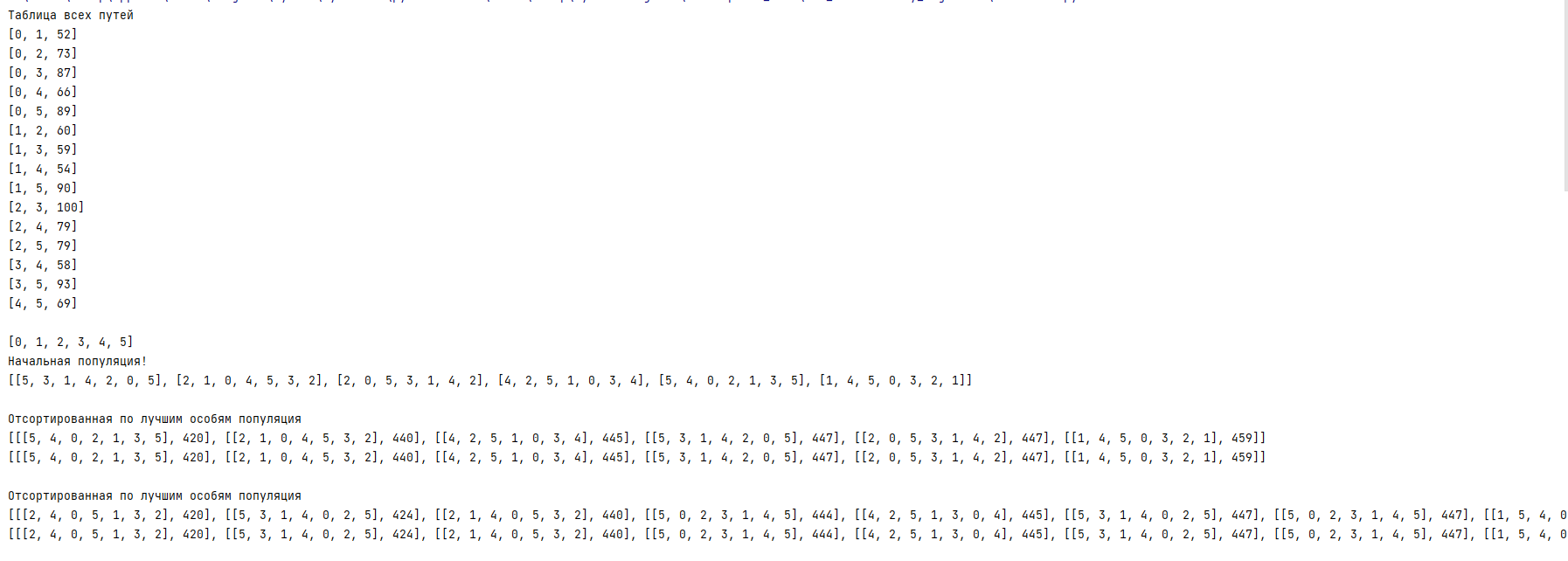


Рисунок 2 – Результат работы программы

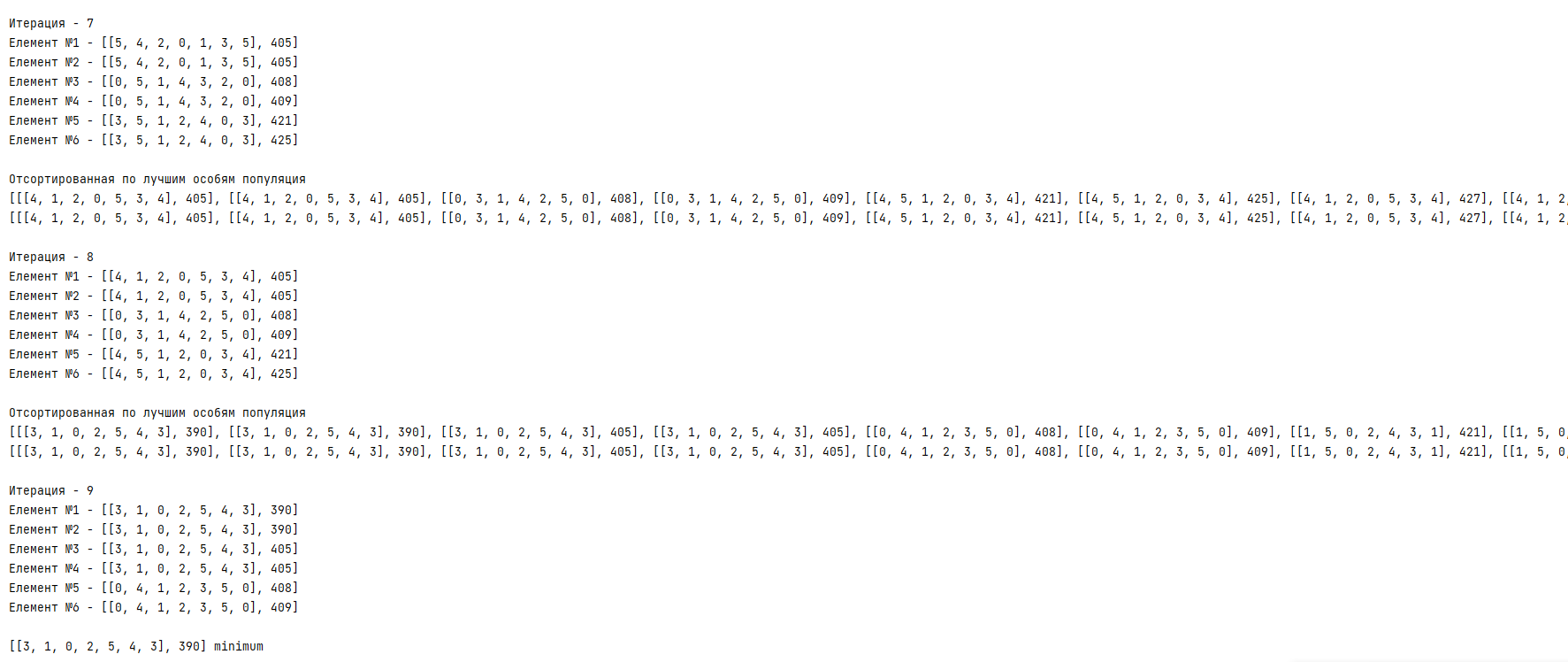


Рисунок 3 – Результат работы программы

Эволюционный алгоритм нашел кратчайший путь на 9 итерации из 10.

ЗАКЛЮЧЕНИЕ

В данной практике был реализован эволюционный алгоритм, на языке высокого уровня python.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Линейное программирование: практикум [Электронный ресурс]: учебно-метод. пособие / А. Б. Сорокин, Е. В. Бражникова, О. В. Платонова. — М.: МИРЭА, 2017. — Электрон. опт. диск (ISO).
2. Методы оптимизации: гибридные генетические алгоритмы [Электронный ресурс]: учебно-метод. пособие / А. Б. Сорокин. — М.: МИРЭА, 2016. — Электрон. опт. диск (ISO).

ПРИЛОЖЕНИЯ

Приложение А – Листинг кода для эволюционного алгоритма

Приложение А

Листинг кода для эволюционного алгоритма

Листинг А.1 – Используемые библиотеки

import random

Листинг А.2 – Функция main

if \_\_name\_\_ == '\_\_main\_\_':

count\_cities = 6

all\_routes = generating\_all\_paths(count\_cities)

new\_generation = generate\_start\_routes(all\_routes, count\_cities)

iteration = 0

minimum = [[], 1000000]

while iteration < 10:

mutation\_generation = []

for element\_id, element in enumerate(new\_generation):

mutation\_generation.append(mutation\_element(element, count\_cities))

for i in mutation\_generation:

new\_generation.append(i)

new\_generation = count\_len(new\_generation, all\_routes)

new\_generation = leave\_the\_giant\_enough(new\_generation)

print('Итерация -', iteration)

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

print(f"Елемент №{i + 1} - {new\_generation[i]}")

print()

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

if new\_generation[i][1] < minimum[1]:

minimum = new\_generation[i]

iteration += 1

print(minimum, 'minimum')

Листинг А.4 – Функция создание всех путей и их длин

def generating\_all\_paths(count):

mass\_all\_routes = []

for i in range(count - 1): # генераця всех путей

for j in range(i, count):

mass\_all\_routes.append([i, j, random.randint(20, 50)])

# mass\_all\_routes = [[0, 1, 52], [0, 2, 73], [0, 3, 87], [0, 4, 66], [0, 5, 89], [1, 2, 60], [1, 3, 59], [1, 4, 54],

# [1, 5, 90], [2, 3, 100], [2, 4, 79], [2, 5, 79], [3, 4, 58], [3, 5, 93], [4, 5, 69]]

print('Таблица всех путей')

for i in mass\_all\_routes:

print(i)

print()

return mass\_all\_routes

Листинг А.5 – Функция генерации начальной популяции

def generate\_start\_routes(routes, cities):

mass\_routes = []

start\_gen\_routers = []

for i in range(cities):

start\_gen\_routers.append(i)

print(start\_gen\_routers)

new\_data = itertools.permutations(start\_gen\_routers, cities)

for mass in new\_data:

mass\_routes.append(list(mass))

start\_routers = []

for i in range(6):

index = random.randint(0, len(mass\_routes) - 1)

if mass\_routes[index] not in start\_routers:

mass\_routes[index].append(mass\_routes[index][0])

start\_routers.append(mass\_routes[index])

print('Начальная популяция!')

print(start\_routers)

print()

# start\_routers =

return count\_len(start\_routers, routes)

Листинг А.6 – Функция мутации

def mutation\_element(element, count\_cities):

switch = True

# print(element)

elements = element[0]

elements.pop(-1)

direction = random.randint(0, 1) # если 1 то; + если 0 то -

first\_element\_to\_mutation = random.randint(0, count\_cities - 1)

second\_element\_to\_mutation = first\_element\_to\_mutation

count = 0

if direction:

second\_element\_to\_mutation += 1

else:

second\_element\_to\_mutation -= 1

while switch:

on\_off = random.randint(0, 1)

if second\_element\_to\_mutation == count\_cities:

second\_element\_to\_mutation = 0

if second\_element\_to\_mutation == -7:

second\_element\_to\_mutation = -1

if on\_off == 0:

if direction:

second\_element\_to\_mutation += 1

else:

second\_element\_to\_mutation -= 1

else:

switch = False

elements[first\_element\_to\_mutation], elements[second\_element\_to\_mutation] = \

elements[second\_element\_to\_mutation], elements[first\_element\_to\_mutation]

elements.append(elements[0])

return elements

Листинг А.7 – Функция сортировки по кратчайшему путти

def sort\_by\_long(mass):

new\_mass = []

while len(mass) != 0:

minimum = [0, 10000]

for i in range(len(mass)):

if mass[i][1] < minimum[1]:

minimum = [i, mass[i][1]]

new\_mass.append(mass[minimum[0]])

mass.pop(minimum[0])

print('Отсортированная по лучшим особям популяция')

for i in range(0, 2):

print(new\_mass)

print()

return new\_mass

Листинг А.8 – Функция расчёта длины путти

def count\_len(mass, routes):

for i in range(len(mass)):

if len(mass[i]) != 2 and mass[i][0] is not list:

way = mass[i]

long\_way = 0

for point in range(len(way) - 1):

for long in routes:

if way[point] == long[0] and way[point + 1] == long[1] or \

way[point] == long[1] and way[point + 1] == long[0]:

long\_way += long[-1]

break

mass[i] = [way, long\_way]

return sort\_by\_long(mass)