

|  |
| --- |
| 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](#_Toc138002037)

[1 ГЕНЕТИЧЕСКИЙ АЛГОРИТМ 5](#_Toc138002038)

[1.1 Постановка задачи 5](#_Toc138002039)

[1.2 Описание алгоритма 5](#_Toc138002040)

[1.3 Результат работы программы 6](#_Toc138002041)

[ЗАКЛЮЧЕНИЕ 8](#_Toc138002042)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 9](#_Toc138002043)

[ПРИЛОЖЕНИЯ 10](#_Toc138002044)

ВВЕДЕНИЕ

Генетические алгоритмы (ГА) – адаптивные методы поиска, которые в последнее время часто используются для решения задач функциональной оптимизации. Они основаны на генетических процессах биологических организмов: биологические популяции развиваются в течении нескольких поколений, подчиняясь законам естественного отбора и по принципу «выживает наиболее приспособленный», открытому Чарльзом Дарвином. Подражая этому процессу генетические алгоритмы способны «развивать» решения реальных задач, если те соответствующим образом закодированы. Генетические алгоритмы являются наиболее известным и популярным методом эволюционных вычислений, т.к. используются для широкого круга задач: оптимизации, поиска, управления и т.д. Данные алгоритмы адаптивны, развивают решения, развиваются сами.

**Словарь основных понятий генетического алгоритма.**

Биологические принципы развития являются главной составляющей методов эволюционного моделирования. При этом математический аппарат

«мягких» вычислений играет доминирующую роль в его применении. Биологическая эволюция выступает скорее словарем основных понятий, источником заимствованных идей и законов. Обобщенно можно выделить следующие аспекты заимствований:

понятийный аппарат;

идея коллективного поиска экстремума при помощи популяции особей;

способы представления генетической информации;

генетические операторы;

идея о преимущественном размножении наиболее приспособленных особей.

Учитывая частично заимствованный генетический словарь терминов, рассмотрим правила их интерпретации для оптимизационной модели (табл.1)

Таблица №1. Генетический словарь терминов

|  |  |
| --- | --- |
| Эволюционная модель | Оптимизационная модель |
| Хромосома | Решение, строка, последовательность |
| Ген | Переменная, параметр, характеристика, признак |
| Аллель | Значение фрагмента закодированного параметра |
| Локус | Номер фрагмента закодированного параметра |
| Генотип | Множество закодированных решений задачи, пространство поиска |
| Фенотип | Множество раскодированных решений задачи, пространство решений |
| Особь, индивидуум | Объект, система |
| Пригодность | Качество, оптимальность |
| Fitness-функция | Функция соответствия |
| Популяция | Множество решений |
| Поколение | Итерация работы эволюционного алгоритма |

Таким образом, результат работы генетического алгоритма сильно зависит от того, каким образом настроены его параметры. Отметим, что его параметры часто определяются методом проб и ошибок, на основе анализа получаемых результатов.

1. ГЕНЕТИЧЕСКИЙ АЛГОРИТМ

1.1 Постановка задачи

Реализовать генетический 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 скрещивание и мутацию.

Для скрещивания будем брать двух особей и делить их по гену. Первая честь генов будет унаследована у первого родителя, вторая часть у второго родителя.

Для мутации с вероятность 10% будем менять ближайшие гены местами.

Популяция не ограничивается.

* 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 1-3.

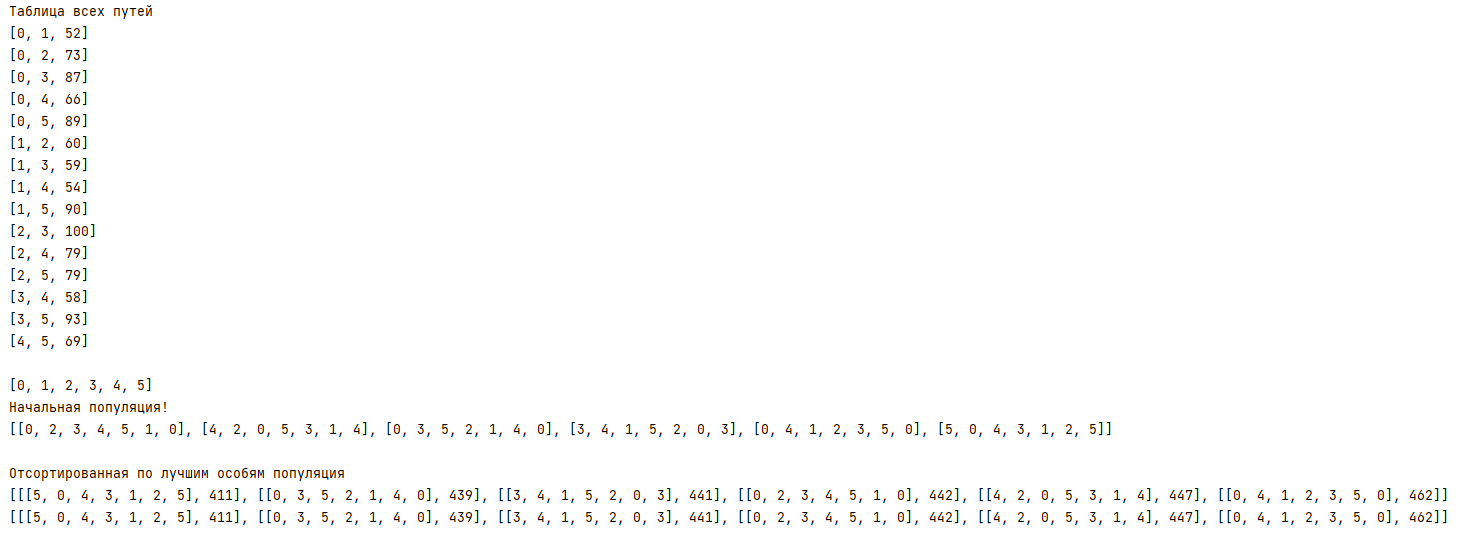


Рисунок 1 – Результат работы программы

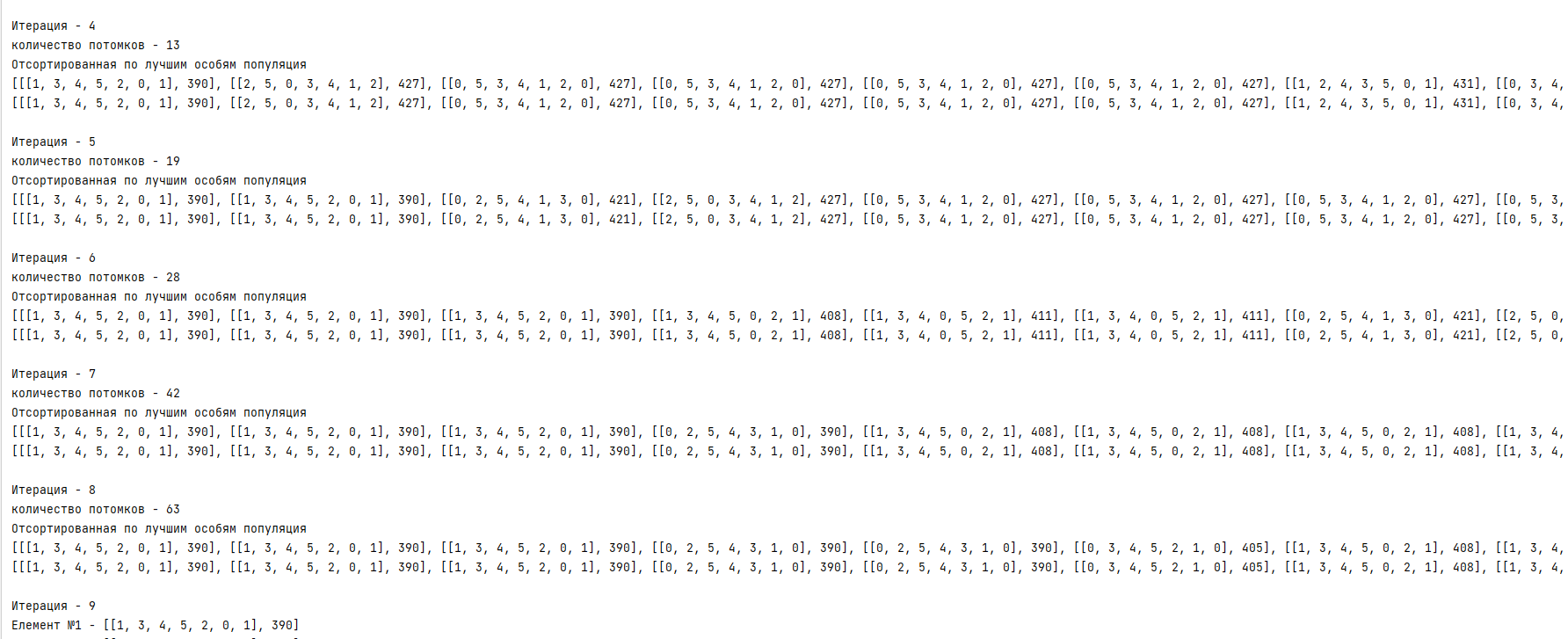


Рисунок 2 – Результат работы программы



Рисунок 3 – Результат работы программы

Генетический алгоритм нашел кратчайший путь на 4 итерации из 10.

ЗАКЛЮЧЕНИЕ

В данной практике был реализован генетический 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 генетического алгоритма

Приложение А

Листинг кода для генетического 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:

new\_generation = crossing(new\_generation, count\_cities, all\_routes)

print('Итерация -', iteration)

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

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

minimum = new\_generation[i]

iteration += 1

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

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

print()

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 скрещивания

def crossing(mass, count\_s, all\_routes):

if len(mass) // 2 == 0:

number\_of\_descendants = len(mass) // 2

else:

number\_of\_descendants = len(mass) // 2 - 1

print('количество потомков -', number\_of\_descendants)

massive = []

element\_to\_split = []

while len(element\_to\_split) != number\_of\_descendants // 2:

pod = []

while len(pod) != 2:

parent\_for\_crossing = random.randint(0, len(mass) - 1)

if parent\_for\_crossing not in pod:

pod.append(parent\_for\_crossing)

element\_to\_split.append(pod)

for i in range(len(element\_to\_split)):

massive.append(random.randint(1, len(mass[0][0]) - 2))

massive\_2 = []

for i in range(len(element\_to\_split)):

massive\_2.append(random.randint(1, len(mass[0][0]) - 2))

for ii, i in enumerate(element\_to\_split):

element = []

# print(mass[i[0]][0])

for j in range(0, massive\_2[ii]):

if mass[i[0]][0][j] not in element:

element.append(mass[i[0]][0][j])

for j in range(massive\_2[ii], len(mass[0][0])):

if mass[i[1]][0][j] not in element:

element.append(mass[i[1]][0][j])

for j in range(0, len(mass[0][0])):

if mass[i[0]][0][j] not in element:

element.append(mass[i[0]][0][j])

Продолжения Листинг А.6

for j in range(0, len(mass[0][0])):

if mass[i[1]][0][j] not in element:

element.append(mass[i[1]][0][j])

if random.randint(0, 100) < 10:

element = mutation(element)

element.append(element[0])

mass.append(element)

# mass\_gen = sort\_by\_long(mass)

return count\_len(mass, all\_routes)

Листинг А.7 – Функция мутации

def mutation(mass):

while True:

index\_1 = random.randint(0, len(mass) - 1)

index\_2 = random.randint(0, len(mass) - 1)

if index\_1 != index\_2:

mass[index\_1], mass[index\_2] = mass[index\_2], mass[index\_1]

return mass

Листинг А.8 – 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.9 – 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)