

|  |
| --- |
| 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 6](#_Toc136764618)

[1 AССОЦИАТИВНЫЕ ПРАВИЛА 8](#_Toc136764619)

[1.1 Постановка задачи 8](#_Toc136764620)

[1.2 Описание алгоритма 8](#_Toc136764621)

[1.3 Описание предметной области 10](#_Toc136764622)

[1.4 Ассоциотивные правила в программе deductor 11](#_Toc136764623)

[1.5 Расчёт поддержки 12](#_Toc136764624)

[1.6 Расчёт достоверности 13](#_Toc136764625)

[1.7 Расчёт лифта 13](#_Toc136764626)

[1.8 Программная реализация 13](#_Toc136764627)

[2 К-MEANS 15](#_Toc136764628)

[2.1 Постановка задачи 15](#_Toc136764629)

[2.2 Описание алгоритма 15](#_Toc136764630)

[2.3 Ручной расчёт 17](#_Toc136764631)

[2.4 Программная реализация 19](#_Toc136764632)

[3 ЛИНЕЙНАЯ РЕГРЕСИЯ 21](#_Toc136764633)

[3.1 Постановка задачи 21](#_Toc136764634)

[3.2 Описание предметной области 21](#_Toc136764635)

[3.3 Описание алгоритма 21](#_Toc136764636)

[3.4 Ручной расчёт 23](#_Toc136764637)

[3.5 Программная реализация 26](#_Toc136764638)

[4. БАЕСОВСКИЙ КЛАСИФИКАТОР 30](#_Toc136764639)

[4.1 Постановка задачи 30](#_Toc136764640)

[4.2 Описание предметной области 30](#_Toc136764641)

[4.3 Описание алгоритма 30](#_Toc136764642)

[4.4 Ручной расчёт 33](#_Toc136764643)

[4.5 Программная реализация 34](#_Toc136764644)

[5 Логистическая регрессия 35](#_Toc136764645)

[5.1 Постановка задачи 35](#_Toc136764646)

[5.2 Описание предметной области 35](#_Toc136764647)

[5.3 Описание алгоритма 35](#_Toc136764648)

[5.4 Ручной расчёт 36](#_Toc136764649)

[5.5 Программная реализация 37](#_Toc136764650)

[6 PAM 45](#_Toc136764651)

[6.1 Постановка задачи 45](#_Toc136764652)

[6.2 Описание предметной области 45](#_Toc136764653)

[6.3 Описание алгоритма 45](#_Toc136764654)

[6.4 Ручной расчёт 46](#_Toc136764655)

[6.5 Программная реализация 48](#_Toc136764656)

[7 CURE 52](#_Toc136764657)

[7.1 Постановка задачи 52](#_Toc136764658)

[7.2 Описание предметной области 52](#_Toc136764659)

[7.3 Описание алгоритма 52](#_Toc136764660)

[7.4 Ручной расчёт 54](#_Toc136764661)

[7.5 Программная реализация 55](#_Toc136764662)

[8 MST 58](#_Toc136764663)

[8.1 Постановка задачи 58](#_Toc136764664)

[8.2 Описание предметной области 58](#_Toc136764665)

[8.3 Описание алгоритма 58](#_Toc136764666)

[8.4 Ручной расчёт 60](#_Toc136764667)

[8.5 Программная реализация 60](#_Toc136764668)

[9 ID3 65](#_Toc136764669)

[9.1 Постановка задачи 65](#_Toc136764670)

[9.2 Описание предметной области 65](#_Toc136764671)

[9.3 Описание алгоритма 65](#_Toc136764672)

[9.4 Ручной расчёт 67](#_Toc136764673)

[9.5 Программная реализация 68](#_Toc136764674)

[ЗАКЛЮЧЕНИЕ 69](#_Toc136764675)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 70](#_Toc136764676)

[ПРИЛОЖЕНИЯ 71](#_Toc136764677)

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 (Рисунок 1).

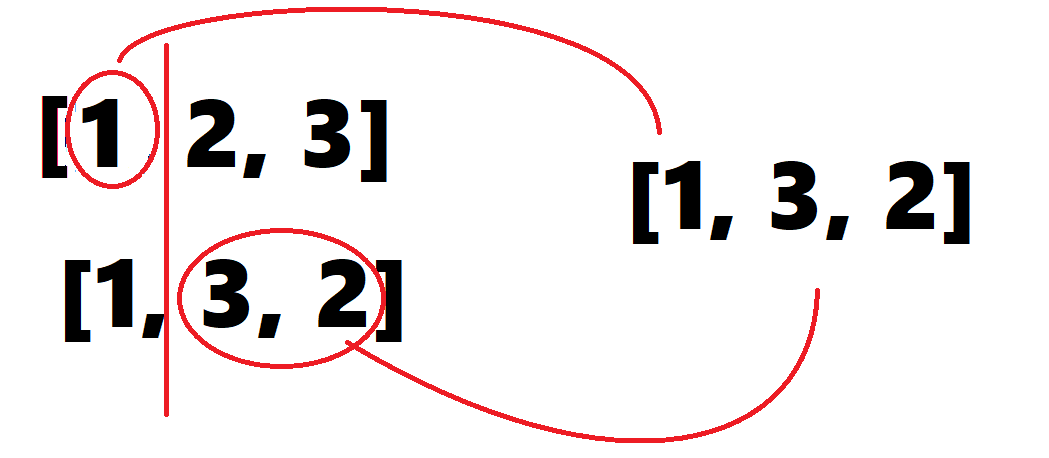


Рисунок 1 - Скрещивание

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 2-3.

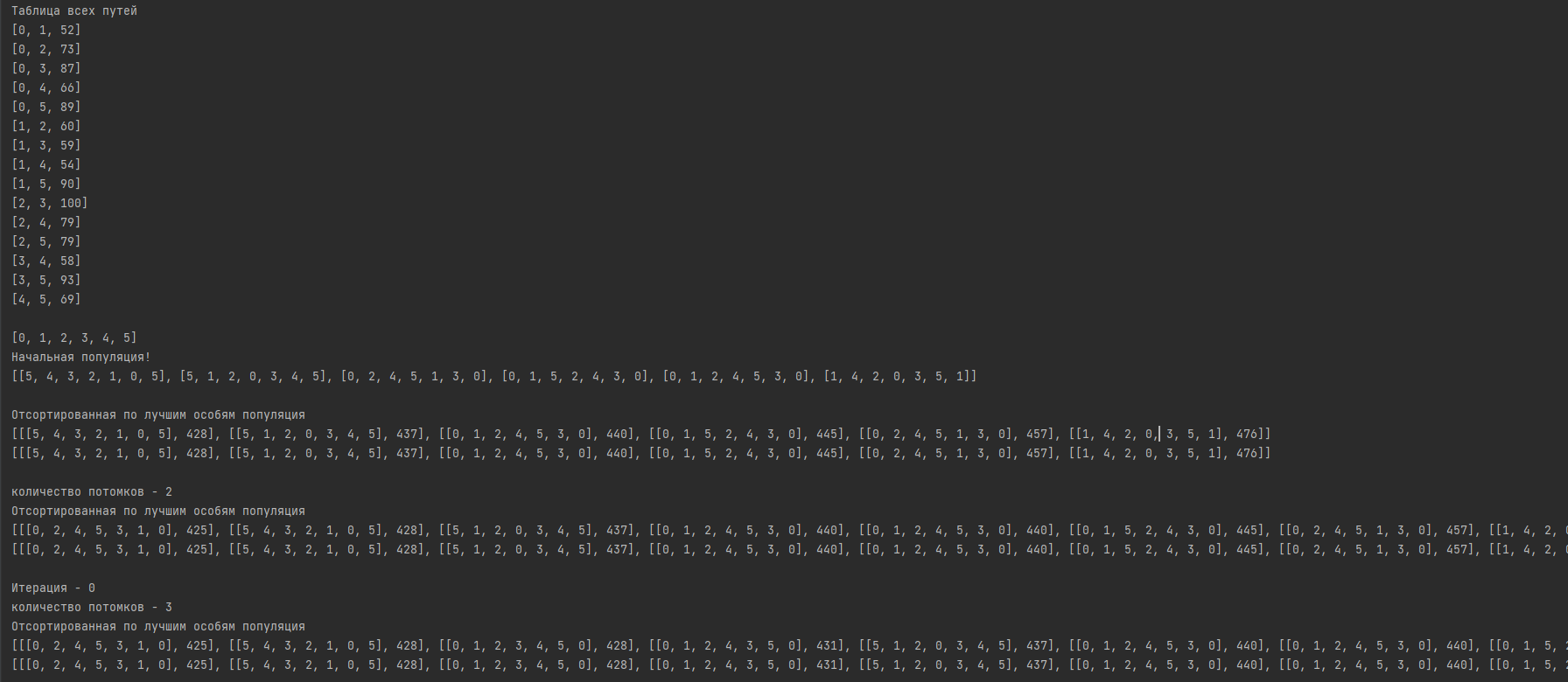


Рисунок 2 – Результат работы программы

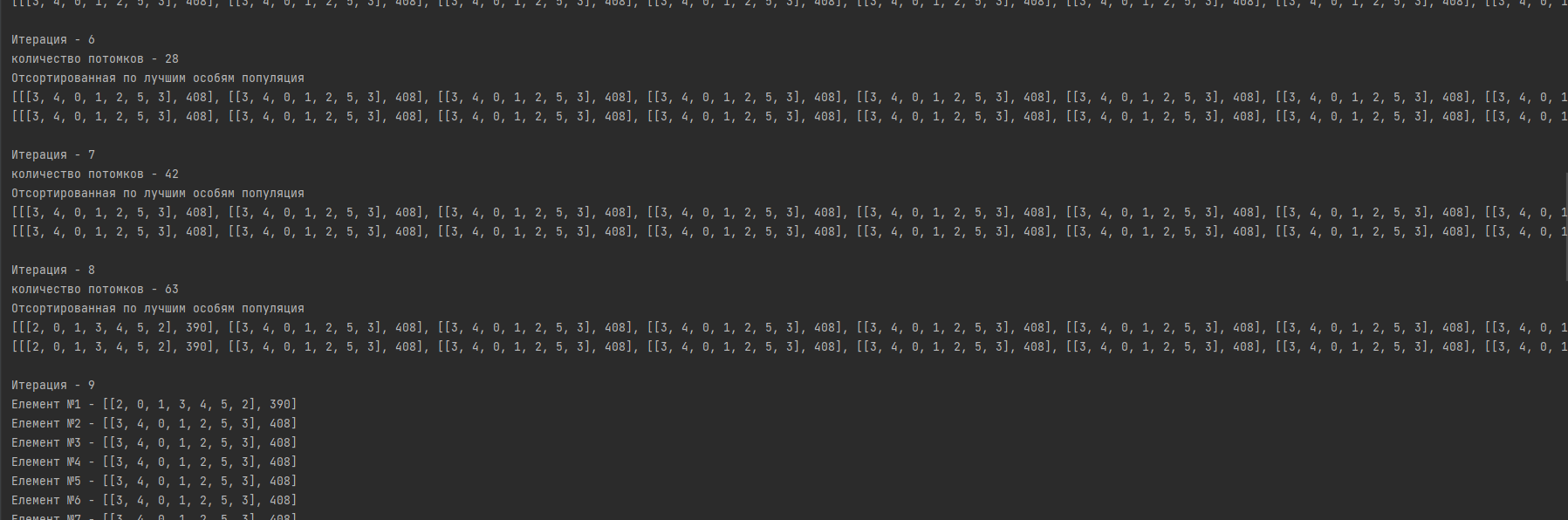


Рисунок 3 – Результат работы программы

Генетический алгоритм нашел кратчайший путь на 8 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