МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”

Факультет компьютерных наук

Кафедра программирования и информационных технологий

Реализация алгоритма муравьиной колонии, основанного на выборе значения эвристического параметра, контролируемого информационной энтропией для задач коммивояжера

Курсовой проект

09.03.04 Программная инженерия
Информационные системы и сетевые технологии

Зав. кафедрой _____С.Д. Махортов, д.ф.- м.н., доцент __.__.20__
Обучающийся _____А.В. Щербинина, 3 курс, д/о
Руководитель _____Д.И. Соломатин, ст. преподаватель

Воронеж 2023

Содержание

Содержание 2

Введение 4

1 Постановка задачи 6

2 Анализ задачи 7

2.1 Задачи коммивояжера 7

2.2 Подходы к решению задачи коммивояжера 8

2.2.1 Муравьиный алгоритм 8

2.2.2 Генетический алгоритм 9

2.2.3 Имитационный отжиг 11

2.3 Реализация муравьиного алгоритма 13

2.4 Реализация алгоритма муравьиной колонии, основанного на выборе значения эвристического параметра, контролируемого информационной энтропией 15

3 Реализация 17

3.1 Средства реализации 17

3.2 Базовый алгоритм 17

3.3 Реализация алгоритма муравьиной колонии, основанного на выборе значения эвристического параметра, контролируемого информационной энтропией 19

3.4 Интерфейс 20

3.5 Структура проекта 21

4 Проведение экспериментов 25

4.1 Опыт 1 25

4.1.1 Повторение 1 25

4.1.2 Повторение 2 27
4.1.3 Повторение 3 28
4.2 Опыт 2 30
4.3 Опыт 3 31
4.4 Опыт 4 33
Заключение 35
Список источников 36

Введение

Задача коммивояжера - одна из самых известных и важных задач транспортной логистики и комбинаторной оптимизации. Суть задачи сводится к поиску оптимального пути, проходящего через промежуточные пункты по одному разу и возвращающегося в исходную точку.

Условия задачи должны содержать критерий выгодности маршрута (т. е. должен ли он быть максимально коротким, быстрым, дешевым или все вместе), а также исходные данные в виде матрицы затрат (расстояния, стоимости, времени и т. д.) при перемещении между рассматриваемыми пунктами.

Особенности задачи в том, что она довольно просто формулируется и найти хорошие решения для нее также относительно просто, но вместе с тем поиск действительно оптимального маршрута для большого набора данных - непростой и ресурсоемкий процесс.

Для решения задачи коммивояжера ее надо представить как математическую модель. При этом исходные условия можно записать в формате матрицы - таблицы, где строкам соответствуют города отправления, столбцам - города прибытия, а в ячейках указываются расстояния (время, стоимость) между ними. Или в виде графа - схемы, состоящей из вершин (точек, кружков), которые символизируют города, и соединяющих их ребер (линий), длина которых соответствует расстоянию между городами.[1]

Задача коммивояжера является NP-трудной. [2] Это означает, что не существует алгоритма, который находил бы точное решение задачи коммивояжера за полиномиальное время. Единственным же алгоритмом, который может гарантировать нахождение точного решения, является метод полного перебора. Однако работа программы, реализующей этот алгоритм, занимает адекватное время только при очень малой размерности входных данных (при числе пунктов < 20).

Транспортные потоки играют важную роль в жизни любой современной компании. Для любого предприятия важнейшей задачей является как можно быстрее доставить товар клиентам в срок. По этой причине менеджеры компаний, по сути, занимаются решением задач коммивояжера.

В современном мире задача коммивояжера помогает оптимизировать работу транспортных отделов и отделов логистики, упрощает работу почтовых и курьерских служб, повышает эффективность мониторинга объектов, например, станций сотовых операторов и нефтяных вышек.

Способность решить эту задачу позволяет оптимизировать и многие производственные процессы, находить оптимальные стратегии ведения хозяйства, экономить ресурсы.[3]

1 Постановка задачи

Задача коммивояжера имеет свои недостатки. Например, долгий поиск наикратчайшего пути, сильная зависимость от настроечных параметров, которые подбираются только исходя из экспериментов. Следовательно, ее можно улучшить.

Целью данной работы является реализация улучшенного алгоритма муравьиной колонии для задач коммивояжера.

Для достижения данной цели разобьем ее на подзадачи, которые необходимо выполнить поэтапно для достижения желаемого результата.

Во-первых, для демонстрации работы необходимо реализовать базовый алгоритм муравьиной колонии для задач коммивояжера на языке программирования python.

Во-вторых, найти, и исследовать различные подходы к улучшению базового алгоритма, и реализовать один из них, который позволит сократить найденный путь и обеспечит более быструю сходимость относительно базового алгоритма.

В-третьих, провести ряд экспериментов.

2 Анализ задачи

2.1 Задачи коммивояжера

В связи с отсутствием эффективных точных методов решения задачи коммивояжера, становится необходимым использование методов эвристических. Эвристическими называют методы, которые будучи основанными на некой эвристике (правиле), дают решение, близкое к точному[4].

Эвристические алгоритмы не всегда находят оптимальный маршрут, чаще всего выдавая приближенный. Зато, они способны работать с большим числом городов. Учитывая, что задача находит большое применение в различных сферах деятельности человека, лучше иметь приближенный ответ, чем не иметь вовсе.

В условиях задачи, указывается критерий выбора самого выгодного маршрута и соответствующая матрица расстояния, стоимости и так далее. Как правило, это показывает, что маршрут должен проходить через каждый город только один раз.

Задача коммивояжера представляется в виде проблемы поиска в полном взвешенном графе Гамильтонова цикла минимальной стоимости [5].

Сеть дорог для задачи представима графом $G = (V, E)$, где V – вершины графа, E– ребра графа.

Вес ребра Cij эквивалентен расстоянию между смежными вершинами графа. Для удобства программной реализации граф представляется в виде матрицы смежности, размерность которой соответствует количеству вершин в графе.

Матрица смежности должна соответствовать следующим критериям:

— Все ее числа должны быть неотрицательны, то есть $Cij > 0$

— Матрица симметрична, то есть $Cij = Cji$

— Удовлетворяет неравенству треугольника: $Cij + Cj\text{k} = Ci\text{k}$

Смысл значения элемента матрицы dij идентичен смыслу веса ребра Cij в графе $G = (V, E)$, оно определяет расстояние между пунктами i и j. При $i = j$, $dij = 0$ - запрещает переход «в себя».

Математически задача может быть сформулирована следующим образом [5]:

Дана матрица расстояний $C(ij)$, где Cij – стоимость перехода из города i в город j, (i, j = 1,.., n). Найти перестановку $(i1, i2, \dots, in, i1)$ целых чисел от 1 до n, что сводит сумму $C_{i1,i2} + C_{i2,i3} + \cdots + C_{in,i1}$ к минимуму.

Повторяющийся элемент $i1$ в начале и в конце означает, что перестановка зациклена, то есть, мы вернулись в вершину отправления.

2.2 Подходы к решению задачи коммивояжера

2.2.1 Муравьиный алгоритм

Идея муравьиного алгоритма - моделирование поведения муравьёв, связанного с их способностью быстро находить кратчайший путь от муравейника к источнику пищи и адаптироваться к изменяющимся условиям, находя новый кратчайший путь.

При своём движении муравей метит путь феромоном, и эта информация используется другими муравьями для выбора пути. Это элементарное правило поведения и определяет способность муравьёв находить новый путь, если старый оказывается недоступным.

Рассмотрим случай, когда на оптимальном раньше пути возникает преграда. В этом случае необходимо определить новый оптимальный путь. Дойдя до преграды, муравьи с равной вероятностью будут обходить её справа и слева. То же самое будет происходить и на обратной стороне преграды. Однако, те муравьи, которые случайно выберут кратчайший путь, будут быстрее его проходить, и за несколько передвижений он будет более обогащён феромоном. Поскольку движение муравьёв определяется концентрацией феромона, то следующие будут предпочитать именно этот

путь, продолжая обогащать его феромоном до тех пор, пока этот путь по какой-либо причине не станет недоступен.

Очевидная положительная обратная связь быстро приведёт к тому, что кратчайший путь станет единственным маршрутом движения большинства муравьёв. Моделирование испарения феромона отрицательной обратной связи гарантирует нам, что найденное локально, оптимальное решение не будет единственным - муравьи будут искать и другие пути. Если смоделируем процесс такого поведения на некотором графе, рёбра которого представляют собой возможные пути перемещения муравьёв, в течение определённого времени, то наиболее обогащённый феромоном путь по рёбрам этого графа и будет являться решением задачи, полученным с помощью муравьиного алгоритма. [8]

2.2.2 Генетический алгоритм

Генетический алгоритм представляет собой метод оптимизации, основанный на концепциях естественного отбора и генетики. В этом подходе переменные, характеризующие решение, представлены в виде ген в хромосоме. Генетический алгоритм оперирует конечным множеством решений (популяцией) - генерирует новые решения как различные комбинации частей решений популяции, используя такие операторы, как отбор, рекомбинация (кроссинговер) и мутация. Новые решения позиционируются в популяции в соответствии с их положением на поверхности исследуемой функции.

Идею генетического алгоритма подсказала сама природа и работы Дарвина. Делается предположение, что если взять 2 вполне хороших решения задачи и каким-либо образом получить из них новое решение, то будет высокая вероятность того, что новое решение получится хорошим или даже лучшим.

Для реализации этого используют моделирование эволюции (естественного отбора) или если проще - борьбы за выживание. В природе,

по упрощенной схеме, каждое животное стремится выжить, чтобы оставить после себя как можно больше потомства. Выжить в таких условиях могут лишь сильнейшие.

Тогда остается организовать некоторую среду - популяцию, населить её решениями - особями, и устроить им борьбу. Для этого нужно определить функцию, по которой будет определяться сила особи - качество предложенного ею решения. Основываясь на этом параметре можно определить каждой особи количество оставляемых ею потомков, или вероятность того, что эта особь оставит потомка. Причем, не исключен вариант, когда особь со слишком низким значением этого параметра умрёт.

Допустим нужно оптимизировать некоторую функцию $F(X1, X2, .., Xn)$. Пусть ищем её глобальный максимум. Тогда, для реализации генетического алгоритма нужно придумать, как хранить решения. По сути, нам поместить все $X1 - Xn$ в некоторый вектор, который будет играть роль хромосомы. Один из наиболее распространенных способов - кодировать значения переменных в битовом векторе.

Пусть каждая особь состоит из массива X и значения функции F на переменных, извлеченных из этого массива.

Тогда генетический алгоритм будет состоять из следующих шагов:

1. Генерация начальной популяции - заполнение популяции особями, в которых элементы массива X (биты) заполнены случайным образом.
2. Выбор родительской пары, то есть берем K особей с максимальными значениями функции F и составляю из них все возможные пары $(K * (K - 1)/2)$.
3. Кроссинговер - берем случайную точку t на массиве X $(0..L - 1)$. Теперь, все элементы массива с индексами 0-t новой особи (потомка) заполняем элементами с теми же индексами, но из массива X первой родительской особи. Остальные элементы заполняются из массива второй

родительской особи. Для второго потомка делается наоборот - элементы 0-t берут от второго потомка, а остальные - от первого.

4. Новые особи с некоторой вероятностью мутируют

инвертируется случайный бит массива X этой особи. Вероятность мутации обычно полагают порядка 1%.

5. Полученные особи-потомки добавляются в популяцию после

переоценки. Обычно новую особь добавляют взамен самой плохой старой особи, при условии что значение функции на новой особи выше значения функции на старой-плохой особи.

6. Если самое лучшее решение в популяции нас не удовлетворяет,

то переход на шаг 2. Хотя, чаще всего этого условия нет, а итерации генетического алгоритма выполняются бесконечно.

Так моделируется «эволюционный процесс», продолжающийся несколько жизненных циклов (поколений), пока не будет выполнен критерий остановки алгоритма. Таким критерием может быть:

— нахождение глобального решения;

— исчерпание числа поколений, отпущенных на эволюцию;

— исчерпание времени, отпущенного на эволюцию.

Генетические алгоритмы служат, главным образом, для поиска решений в очень больших, сложных пространствах поиска. [9]

2.2.3 Имитационный отжиг

Алгоритм имитации отжига основан на случайном поиске. То есть, случайный поиск выполняется в пространстве решений. Когда проблемное пространство велико и существует больше возможных решений, а точность решения невысока, случайный поиск является очень эффективным решением, поскольку другие методы в настоящее время не обеспечивают удовлетворительной пространственно-временной эффективности.

Так называемый отжиг - это процесс, при котором специалисты-металлурги многократно нагревают или охлаждают металл для получения

определенных кристаллических структур. Контрольным параметром этого процесса является температура T. Основная идея метода имитации отжига заключается в следующем: в процессе изменения системы в сторону тенденции к снижению энергии. Время от времени позволяйте системе переходить в более высокое энергетическое состояние, чтобы избежать локальных минимумов и, наконец, стабильно достигать глобальных минимумов. [10]

Пусть имеется некоторая функция f(x) от состояния x, которую мы хотим минимизировать. x - это перестановка вершин (городов) в том порядке, в котором будем их посещать, а f(x) это длина соответствующего пути.

Возьмём в качестве базового решения какое-то состояние x0 (например, случайную перестановку) и будем пытаться его улучшать.

Введём температуру t - какое-то действительное число (изначально равное единице), которое будет изменяться в течение оптимизации и влиять на вероятность перейти в соседнее состояние.

Пока не придём к оптимальному решению или пока не закончится время, будем повторять следующие шаги:

1. Уменьшим температуру $t_k = T(t_{k-1})$.
2. Выберем случайного соседа x - то есть какое-то состояние y, которое может быть получено из x каким-то минимальным изменением.
3. С вероятностью $p(f(x), f(y), t_k)$, сделаем присвоение x←y.

В каждом шаге есть много свободы при реализации. Основные эвристические соображения следующие:

1. В начале оптимизации наше решение и так плохое, и можем позволить себе высокую температуру и риск перейти в состояние хуже. В конце наоборот - наше решение почти оптимальное, и не хотим терять прогресс. Температура должна быть высокой в начале и медленно уменьшаться к концу.

2. Алгоритм будет работать лучше, если функция f(x) «гладкая» относительно этого изменения, то есть изменяется не сильно.

3. Вероятность должна быть меньше, если новое состояние хуже, чем старое. Также вероятность должна быть больше при высокой температуре.

2.3 Реализация муравьиного алгоритма

Любой муравьиный алгоритм независимо от реализации можно представить в виде цикла: [11]

Пока (условия выхода не выполнены):

Создание муравьёв

Пока все муравьи не вернутся в исходную точку:

Поиск решения

Обновление феромона

Дополнительные действия (опционально)

Рассмотрим каждый шаг подробнее.

1. Создание муравьев

Каждый муравей изначально начинает двигаться со случайно выбранного начального города. На этом же этапе задаётся начальный уровень феромона и эвристического параметра. Они инициализируются небольшим положительным числом для того, чтобы вероятности перехода в следующую вершину не были нулевыми.

2. Поиск решения

Для каждого муравья переход из вершины i в вершину j зависит от трех составляющих: памяти муравья, видимости и виртуального следа феромона.

Память муравья - это список посещенных муравьем городов, заходить в которые еще раз нельзя. Используя этот список, муравей гарантированно не попадет в один и тот же город дважды. Обозначим через $J_k(i)$ список городов, которые еще необходимо посетить муравью k, находящемуся в городе i.

Видимость - величина, обратная расстоянию:$\eta_{ij} = \frac{1}{D_{ij}}$ где D_{ij} (1) - расстояние между городами i и j.

Видимость - это локальная статическая информация, выражающая эвристическое желание посетить город j из города i - чем ближе город, тем больше желание посетить его.

Виртуальный след феромона на ребре ij представляет подтвержденное муравьиным опытом желание посетить город j из города i. В отличие от видимости след феромона является более глобальной и динамичной информацией - она изменяется после каждой итерации алгоритма, отражая приобретенный муравьями опыт. Количество феромона на ребре ij на итерации t обозначим через $\tau_{ij}(t)$.

Муравьи используют феромон и эвристическую информацию для выбора городов-кандидатов с определенной вероятностью P. Вероятность перехода из вершины i в вершину j определяется по формуле (2)

$$P_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in J_k(i)}[\tau_{il}(t)]^{\alpha}[\eta_{il}]^{\beta}} & if\ j \in J_k(i) \\ 0 \end{cases} \quad (2)$$

Где α,β – константные параметры.

При α = 0 выбор ближайшего города наиболее вероятен.

При β = 0 выбор происходит только на основании феромона.

3. Обновление феромона

Уровень феромона обновляется в соответствии с формулой (3)

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t) \quad (3)$$

Где

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta J_{ij}^{k}(t)$$

$$\Delta J_{ij}^{k}(t) = \begin{cases} \frac{Q}{L_k(t)} & \text{, если } (i,j) \in T_k(t) \\ 0 \end{cases}$$

Где m - количество муравьев в колонии.

Q - регулируемый параметр, значение которого выбирают одного порядка с длиной оптимального маршрута.

$T_k(t)$ - маршрут, пройденный муравьем k на итерации t.

L_k(t) - длина этого маршрута.

ρ - коэффициент испарения феромона, $p[0,1]$.

2.4 Реализация алгоритма муравьиной колонии, основанного на выборе значения эвристического параметра, контролируемого информационной энтропией

Рассмотрим и проанализируем улучшения, представленные в статье [11].

Шаги 1-3 совпадают с шагами в классическом муравьином алгоритме. В улучшенном алгоритме добавляется четвертый шаг, на котором происходит перерасчет эвристического параметра.

Когда алгоритм муравьиной колонии начинает работать, количество эвристической информации на каждом пути равняется друг другу, информационная энтропия в это время максимальна, но по мере усиления феромона на пути энтропия будет постепенно уменьшаться. Если энтропия не контролируется в данный момент, то она со временем уменьшится до 0, то есть феромон будет на одном пути максимален, и окончательное решение будет ошибочным. Произойдет преждевременное завершение выполнения программы.

Чтобы преодолеть этот дефект предлагается алгоритм муравьиной колонии, основанный на выборе значения эвристического параметра, контролируемого информационной энтропией. Каждый след представляет собой дискретную случайную величину в матрице феромонов. Энтропия случайной величины определяется по формуле (4)

$$E = -\sum_{i=1}^{r} P_i \log P_i \quad (4)$$

Предлагается использовать значение энтропии в качестве индекса, чтобы указать степень того, сколько информации было извлечено из следов

феромонов, а затем соответствующим образом обновить эвристический параметр. Следовательно, можно, чтобы β обновлялся в соответствии с правилом, заданным формулой (5)

$$\beta = \begin{cases} 5, при\ E' \leq 0{,}85 \\ 4, при\ 0{,}65 \leq E' < 0{,}85 \\ 3, при\ 0{,}15 \leq E' < 0{,}65 \\ 2, при\ E' < 0{,}15 \end{cases} \quad (5)$$

Где $E' = 1 - \frac{E_{max} - E_{curr}}{E_{max}}$ (6)

E_{max} - максимальная энтропия перед началом движения муравьев.

E_{curr} - энтропия на текущем шаге.

3 Реализация

3.1 Средства реализации

Для реализации были выбраны следующие технологии:

— Язык программирования python

— Библиотека matplotlib

— Библиотека numpy

— Библиотека pyqt5

3.2 Базовый алгоритм

1.Создадим класс CommonAlg для реализации муравьиного алгоритма. И сделаем базовую функцию __init__ , которая на вход принимает количество городов numCity, количество муравьёв size_pop, количество итераций max_iter, коэффициент важности феромонов в выборе пути alpha, эвристический коэффициент beta, скорость испарения феромонов rho, матрицу расстояний distance_matrix и инициализируем параметры[12]

```
    def __init__(self, func, numCity, size_pop, max_iter,
distance_matrix=None, alpha=1, beta=3, rho=0.1):
    self.func = func
    self.numCity = numCity
    self.size_pop = size_pop
    self.max_iter = max_iter
    self.alpha = alpha
    self.beta = beta
    self.rho = rho
```

Видимость - величина, обратная расстоянию. Вычислим по формуле (1)

```
    self.prob_matrix_distance = 1 / (distance_matrix + 1e-10 *
np.eye(numCity, numCity))
```

Матрица феромонов, обновляющаяся каждую итерацию. Изначально будет состоять из единиц, чтобы путь получился корректным.

```
    self.antPath = np.ones((numCity, numCity))
```

Путь каждого муравья в определённом поколении. Сначала все пути равны нулю.

```
    self.Table = np.zeros((size_pop, numCity)).astype(int)
```

Общее расстояние пути муравья в определённом поколении. На старте оно отсутствует.

```
self.y = None
```

Переменные для фиксирования лучших поколений.

```
self.generation_best_X, self.generation_best_Y = [], []
self.y_best_history = self.generation_best_Y
self.best_x, self.best_y = None, None
self.y_history_all=[]
```

2.Проходим при помощи цикла for по всем итерациям.

Для начала рассчитываем вероятность перехода из вершины i в вершину j по формуле (2)

```
prob_matrix = (self.antPath ** self.alpha) *
(self.prob_matrix_distance) ** self.beta
```

3.Проходим при помощи цикла for по всем муравьям.

Задаем точку начала пути (стартовый город). Буду задавать ее по порядку.

```
self.Table[j, 0] = 0
```

4.Проходим при помощи цикла for по всем городам.

В цикле задаем память муравья. Города, которые муравей уже посетил, помещается в taboo_set, то есть, те города, в которые нельзя идти муравью. Города, в которые можно посетить поместим в allow_list. Именно из этого перечня будет происходить выбор следующей вершины.

Нормализуем вероятность.

```
prob = prob / prob.sum()
```

Случайно выбираем следующую вершину из списка разрешенных и добавляем ее в список пути.

```
next_point = np.random.choice(allow_list, size=1, p=prob)[0]
            self.Table[j, k + 1] = next_point
```

5.Муравьи заканчивают прокладывать свои пути. Выходим из вложенных циклов прохода по городам и муравьям. Рассчитываем общее расстояние, пройденное каждым муравьем на данной итерации

```
y = np.array([self.func(i) for i in self.Table])
```

6.Фиксируем общее решение

```
self.y_history_all.append(np.average(y))
index_best = y.argmin()
x_best, y_best = self.Table[index_best,
:].copy(),y[index_best].copy()
self.generation_best_X.append(x_best)
self.generation_best_Y.append(y_best)
```

7.Рассчитываем феромон, который будет добавлен к ребру по формуле (3)

```
delta_tau = np.zeros((self.numCity, self.numCity))
```

```
for j in range(self.size_pop):
```
для каждого муравья

```
for k in range(self.numCity -1):
```
для каждой вершины

```
n1, n2 = self.Table[j, k], self.Table[j, k + 1]
```
муравьи перебираются из вершины n1 в вершину n2

```
delta_tau[n1, n2] += 1 / y[j]
```
нанесение феромона

```
n1, n2 = self.Table[j, self.numCity - 1], self.Table[j, 0]
```
муравьи ползут от последней вершины обратно к первой

```
delta_tau[n1, n2] += 1 / y[j]
```
нанесение феромона

```
self.antPath = (1 - self.rho) * self.antPath + delta_tau
```

8.Выбираем и запоминаем минимальное решение среди лучших на каждой итерации.

```
best_generation = np.array(self.generation_best_Y).argmin()
self.best_x = self.generation_best_X[best_generation]
self.best_y = self.generation_best_Y[best_generation]
```

Эта величина и будет ответом.

3.3 Реализация алгоритма муравьиной колонии, основанного на выборе значения эвристического параметра, контролируемого информационной энтропией

Для алгоритма муравьиной колонии, основанный на выборе значения эвристического параметра, контролируемого информационной энтропией, создадим отдельный класс UpgradeAlg.

Сделаем базовую функцию __init__ , которая на вход принимает те же параметры, что и базовый алгоритм.

Перед началом основных расчетов, необходимо подсчитать E_{max} по формуле (4) для каждой пары вершин в матрице смежности.

```
for i in range(len(self.prob_matrix_distance[0])):
                for k in range(len(self.prob_matrix_distance)):
                    if self.prob_matrix_distance[i][k] != 0:
                        Emax += entropy(self.prob_matrix_distance[i], base=10)
```

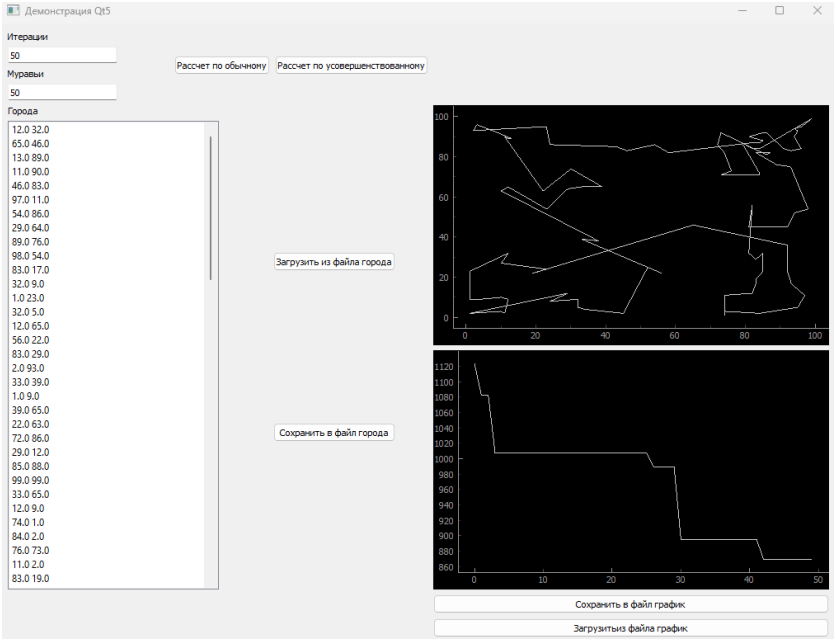
На шаге 4 в циклах рассчитаем еще и энтропию для перехода между двумя городами по формуле (4)

```
Ecurr += entropy(prob, base=10)
```

Перед выполнением шага 5 вычислим коэффициент эвристической информации по формуле (5)

```
E = 1 - (Emax - Ecurr) / Emax
            if E > 0.86:
                self.beta = 5
            elif 0.65 <= E < 0.85:
                self.beta = 4
            elif 0.15 <= E < 0.65:
                self.beta = 3
            elif E < 0.15:
                self.beta = 2
```

3.4 Интерфейс



В левой верхней части экрана представлены два поля ввода. В окошко «Итерации» вводим количество итераций. Аналогично, в окошко «Муравьи» вводим количество муравьев.

Ниже представлен перечень координат городов, по которым будет проводиться построение. Этот список можно загрузить из файла и сохранить в файл, для этого необходимо воспользоваться соответствующими кнопками, которые находятся правее городов.

После ввода всех необходимых данных, можно приступать к поиску пути. Можно найти оптимальный путь по двум алгоритмам: по классическому алгоритму муравьиной колонии и по алгоритму муравьиной колонии, основанный на выборе значения эвристического параметра, контролируемого информационной энтропией. Для того чтобы рассчитать по классическому алгоритму муравьиной колонии, необходимо нажать на кнопку «Расчет по обычному». Для того чтобы рассчитать по алгоритму муравьиной колонии, основанный на выборе значения эвристического параметра, контролируемого информационной энтропией, необходимо нажать на кнопку «Расчет по усовершенствованному».

Результат расчетов отобразится в окнах в правой части экрана. Сверху – построенный маршрут. Ниже - скорость обучения муравьев.

Полученный результат можно сохранить в текстовый файл. Для этого следует воспользоваться кнопкой «Сохранить в файл график».

Также для наглядности можно загрузить из текстового файла предыдущие результаты. Для этого нужно воспользоваться кнопкой «Загрузить из файла график».

3.5 Структура проекта

В проекте есть два основных класса CommonAlg и UpgradeAlg, в которых реализованы классический алгоритм муравьиной колонии и алгоритм муравьиной колонии, основанный на выборе значения эвристического параметра, контролируемого информационной энтропией, соответственно.

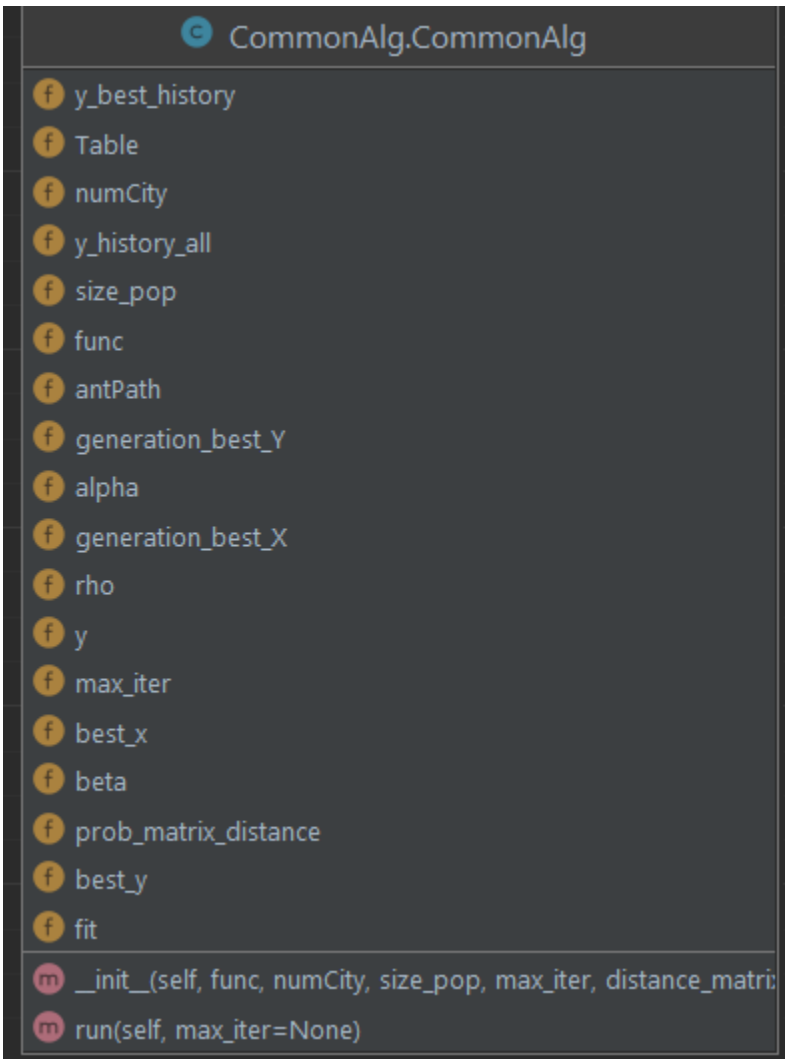


Рисунок 1 - Диаграмма класса CommonAlg

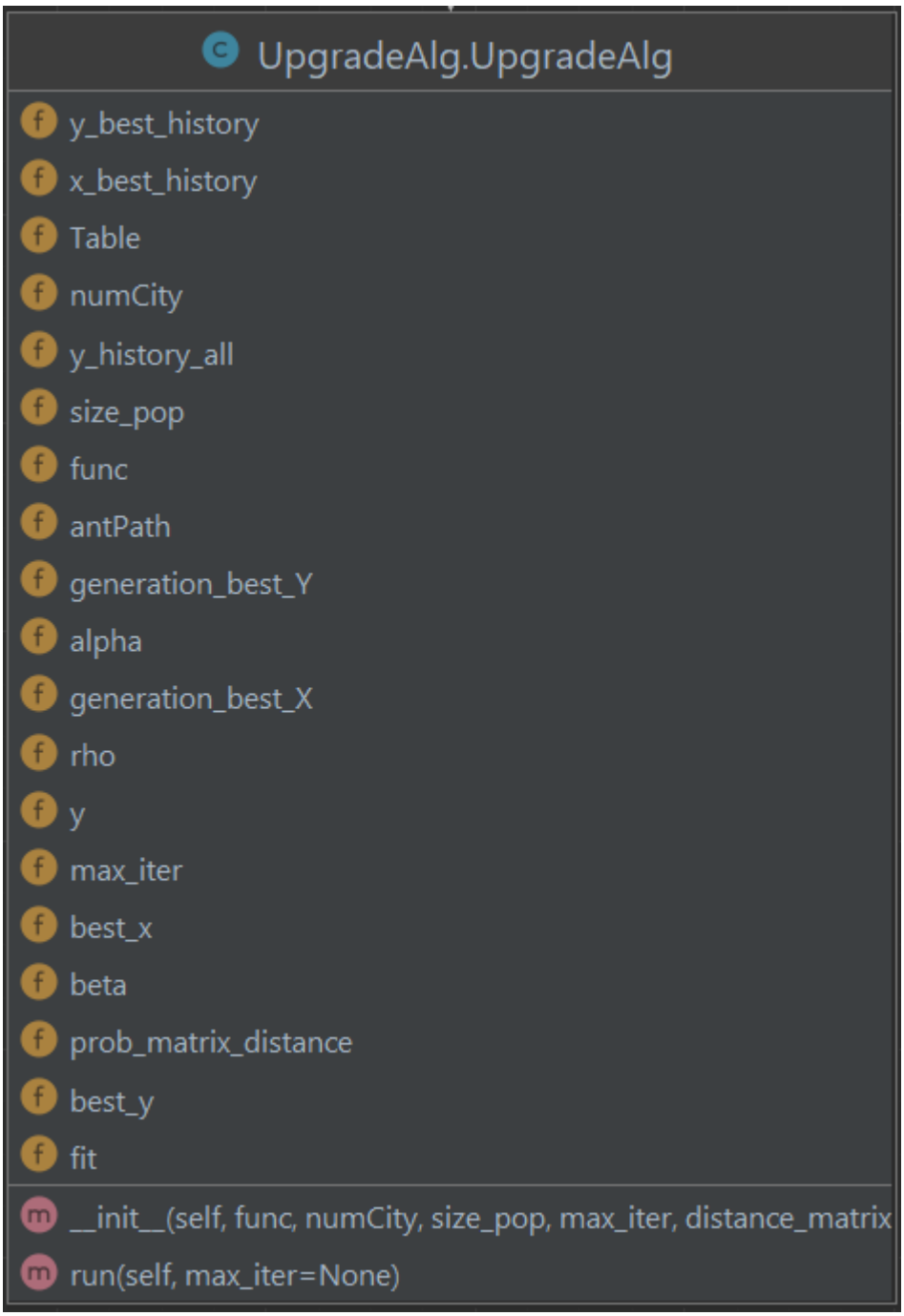


Рисунок 2 - Диаграмма класса UpgradeAlg

В файле main находится точка входа в программу и различные функции, которые нужны для построения различных графиков.

Графический интерфейс реализован при помощи pyqt5 designer в файле MainWindow.ui. Основная логика обработки нажатий реализована в классе MainWindow в файле MainWindow.py. Также есть еще одна точка входа в программу в файле main2, которая привязана к графическому интерфейсу.

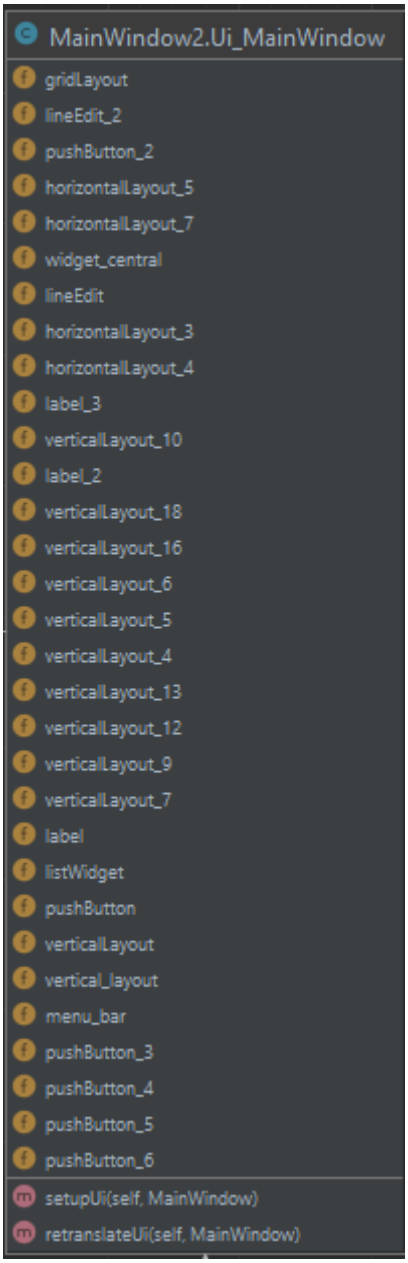


Рисунок 3 - Диаграмма класса MainWindow.ui

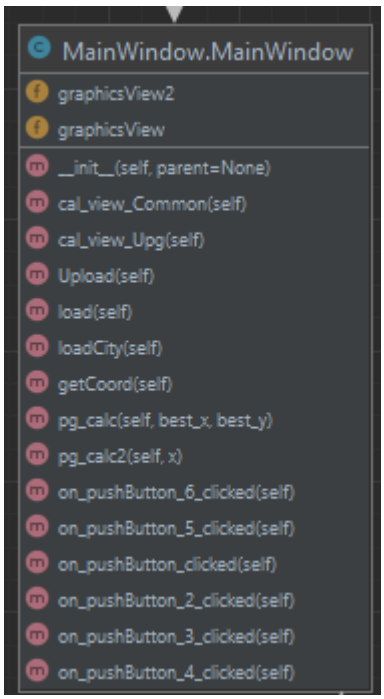


Рисунок 4 - Диаграмма классов MainWindow.py

4 Проведение экспериментов

4.1 Опыт 1

4.1.1 Повторение 1

Входные данные: 100 городов, 50 муравьев, 100 итераций, коэффициент значимости феромона 1, коэффициент значимости эвристического параметра 2, скорость испарения феромона 0.1.

Для классического алгоритма муравьиной колонии получили следующий результат. Длина пути получилась равной 1051.

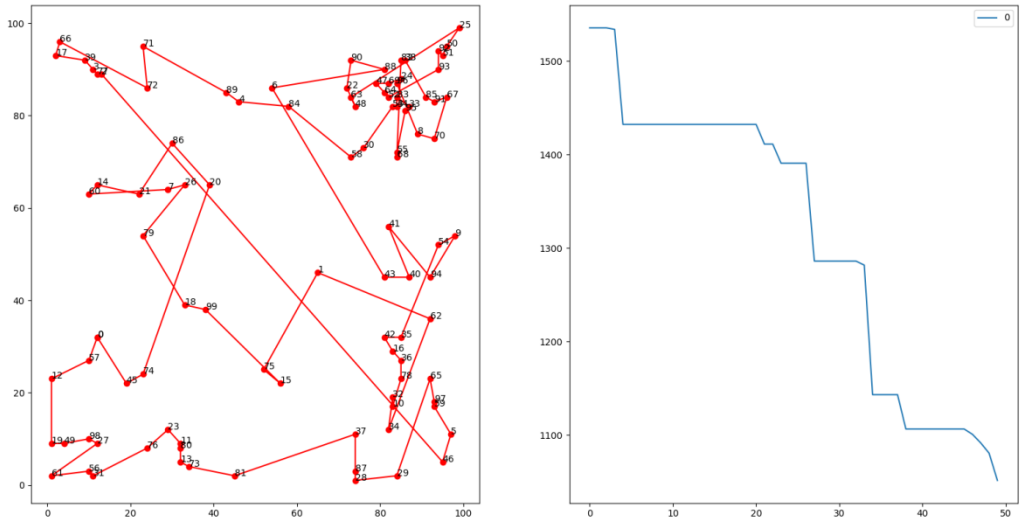


Рисунок 5 - Результаты первого опыта и первого повторения для классического алгоритма

Длина пути для алгоритма муравьиной колонии, основанного на выборе значения эвристического параметра, контролируемого информационной энтропией получилась равной 822.

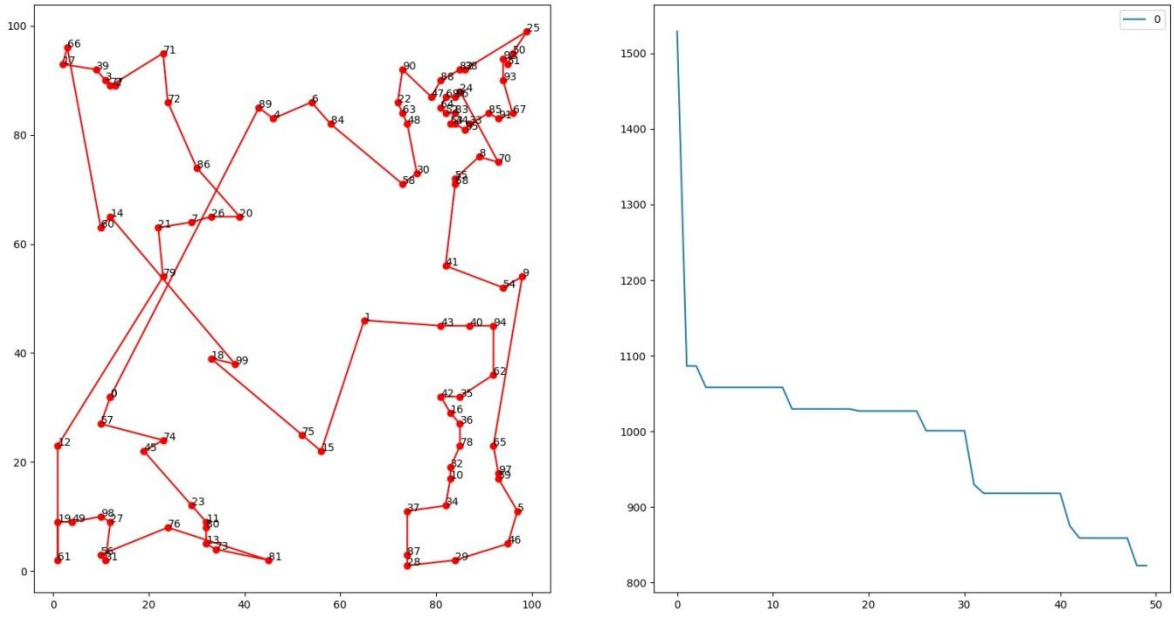


Рисунок 6 - Результаты первого опыта и первого повторения для алгоритма муравьиной колонии, основанного на выборе значения эвристического параметра, контролируемого информационной энтропией

4.1.2 Повторение 2

Входные данные: 100 городов, 100 муравьев, 100 итераций, коэффициент значимости феромона 1, коэффициент значимости эвристического параметра 2, скорость испарения феромона 0.1.

Для классического алгоритма. Длина пути 811.

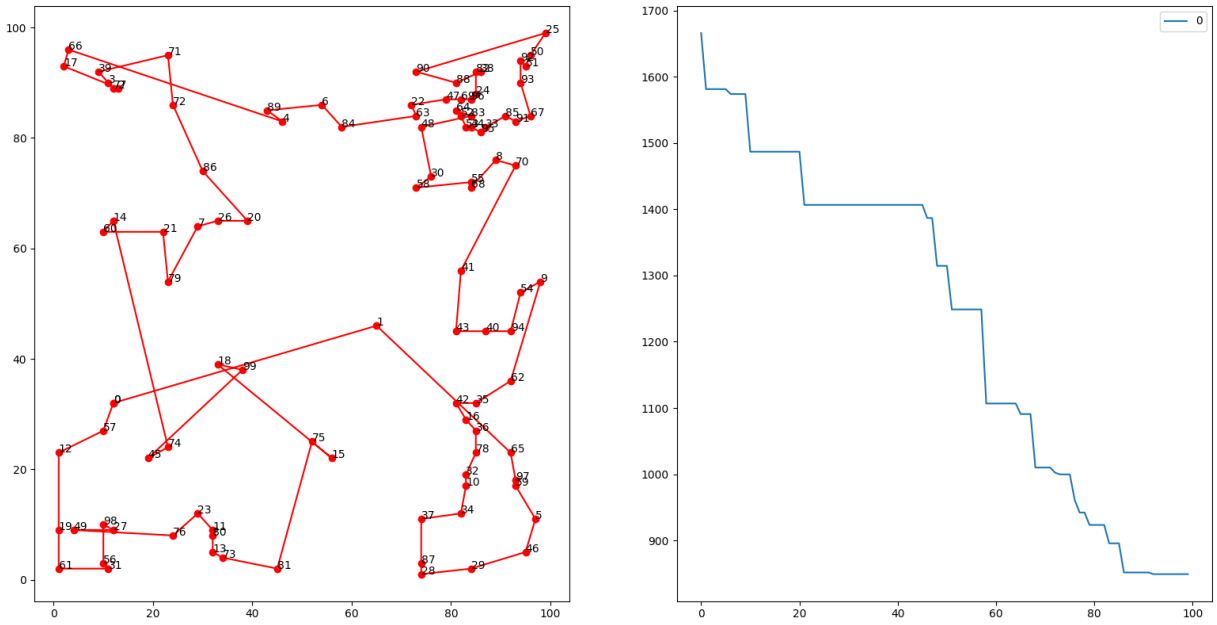


Рисунок 7 - Результаты первого опыта второго повторения для классического алгоритма

Для алгоритма муравьиной колонии, основанного на выборе значения эвристического параметра, контролируемого информационной энтропией алгоритма длина пути 783.

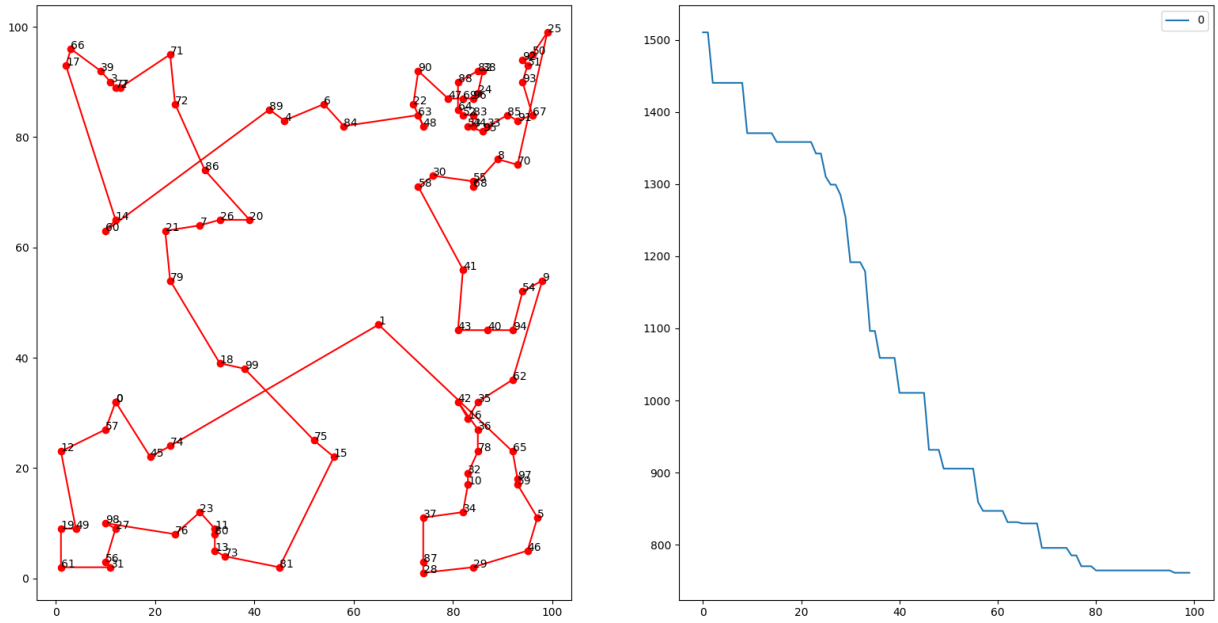


Рисунок 8 - Результаты первого опыта второго повторения для алгоритма муравьиной колонии, основанного на выборе значения эвристического параметра, контролируемого информационной энтропией

4.1.3 Повторение 3

Входные данные: 100 городов, 100 муравьев, 200 итераций, коэффициент значимости феромона 1, коэффициент значимости эвристического параметра 2, скорость испарения феромона 0.1.

Для классического алгоритма. Длина пути 731.

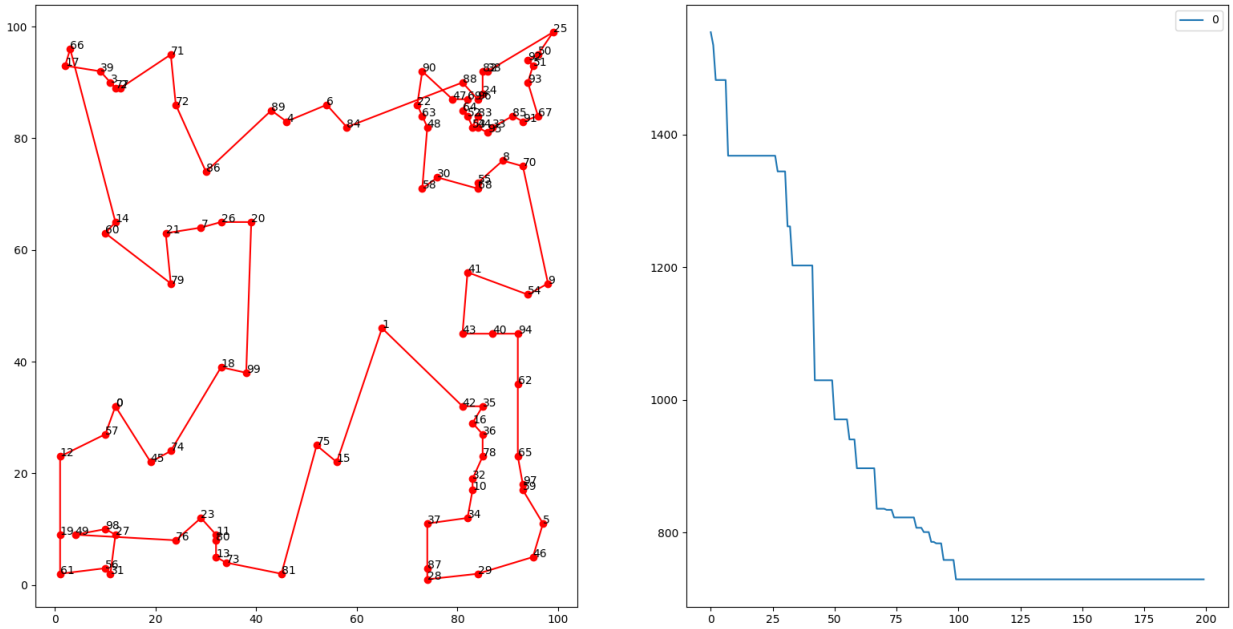


Рисунок 9 - Результаты первого опыта третьего повторения для классического алгоритма муравьиной колонии

Для алгоритма муравьиной колонии, основанного на выборе значения эвристического параметра, контролируемого информационной энтропией длина пути 720.

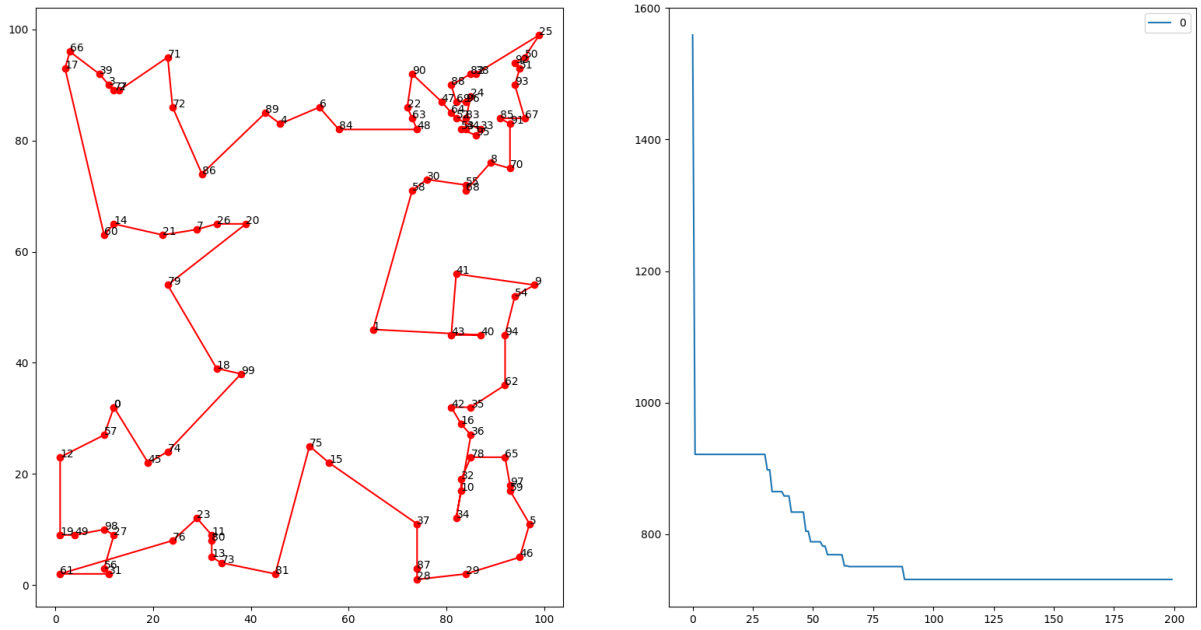


Рисунок 10 - Результаты первого опыта третьего повторения для алгоритма муравьиной колонии, основанного на выборе значения эвристического параметра, контролируемого информационной энтропией

4.2 Опыт 2

На каждом поколении вычислим среднее расстояние. Получим координаты точек. Построим их. Вычислим среднюю линию между этими точками. Входные данные: 100 городов, 50 муравьев, 50 итераций, коэффициент значимости феромона 1, коэффициент значимости эвристического параметра 2, скорость испарения феромона 0.1. Выполним три повторения.

Синий график для алгоритма муравьиной колонии, основанного на выборе значения эвристического параметра, контролируемого информационной энтропией алгоритма, зеленый - для обычного. Выполним три повторения.

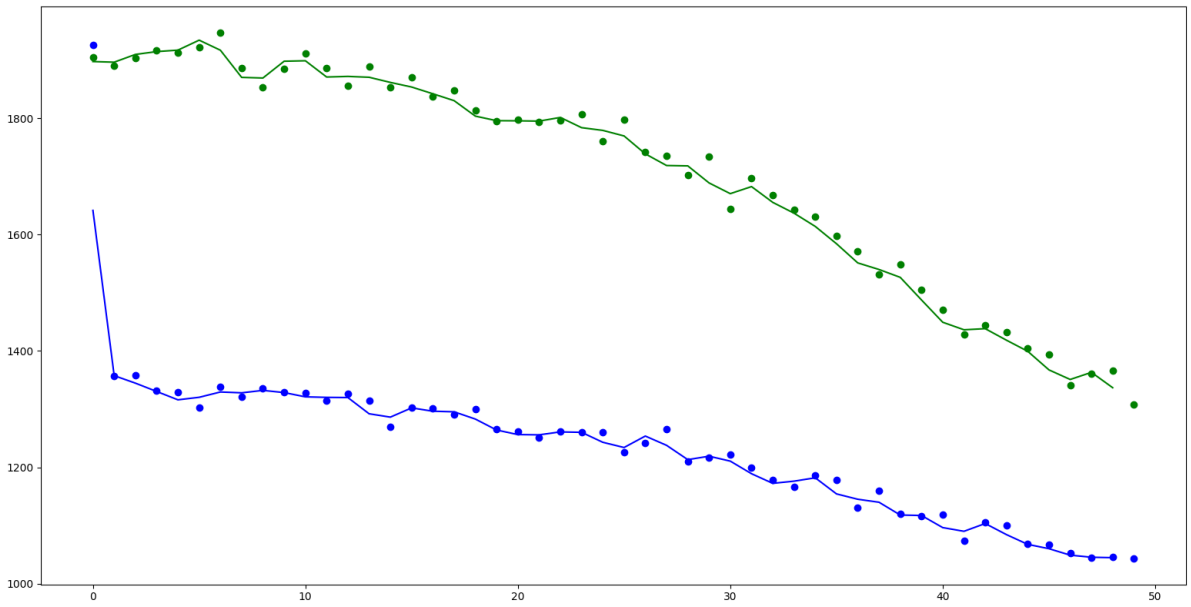


Рисунок 11 - Результаты второго опыта для первого повторения

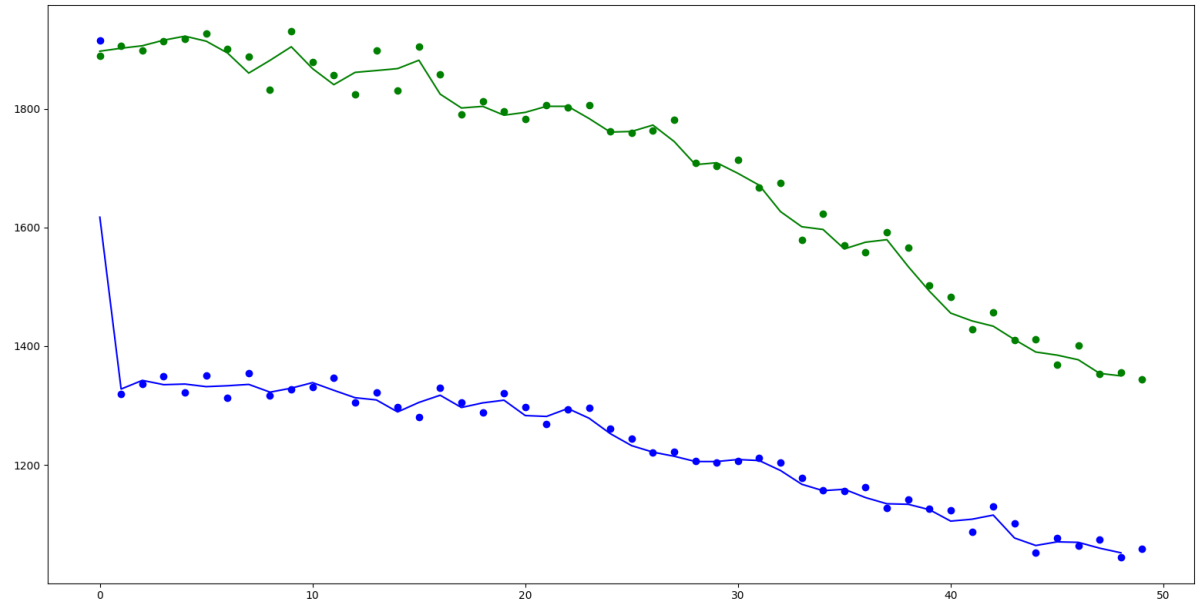


Рисунок 12 - Результаты второго опыта для второго повторения

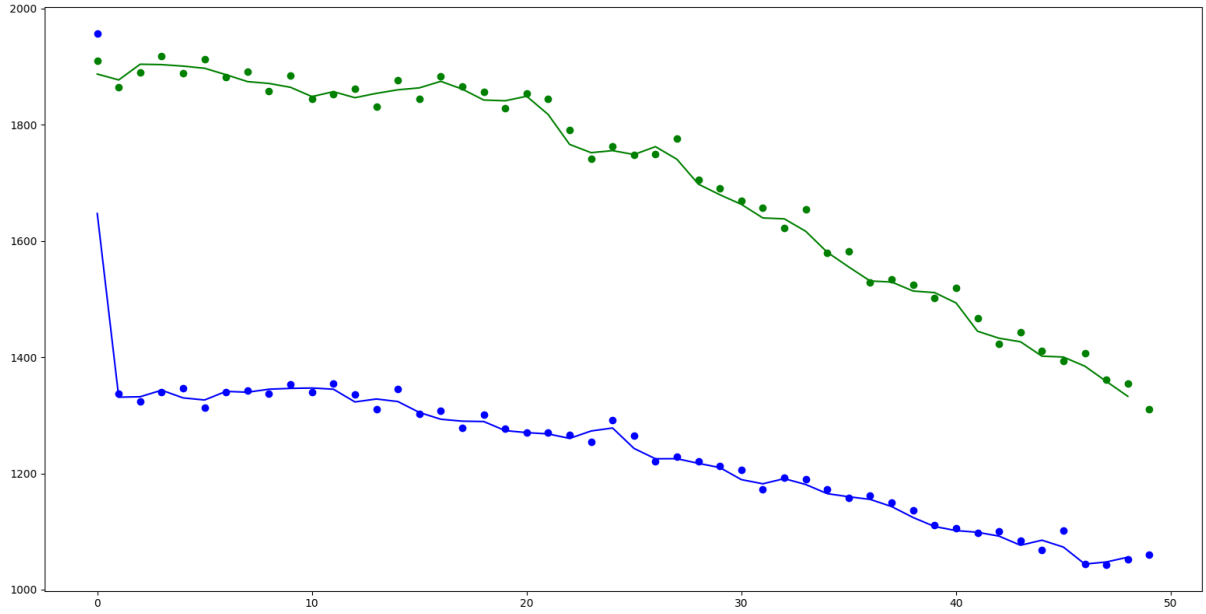


Рисунок 13 - Результаты второго опыта для третьего повторения

4.3 Опыт 3

Проведем серию экспериментов. Построим для каждого эксперимента линию - среднее расстояние для каждого поколения. Получим набор линий. Входные данные: 30 повторений, 100 городов, 50 муравьев, 50 итераций,

коэффициент значимости феромона 1, коэффициент значимости эвристического параметра 2, скорость испарения феромона 0.1.

Синий график для алгоритма муравьиной колонии, основанного на выборе значения эвристического параметра, контролируемого информационной энтропией алгоритма, зеленый - для обычного.

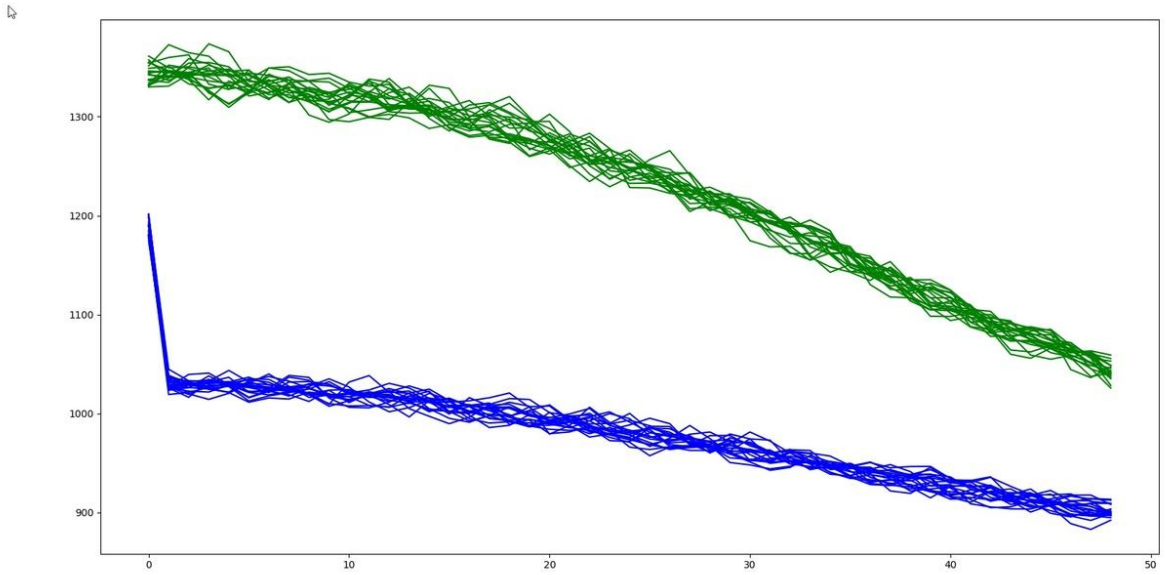


Рисунок 14 - Результаты пятого опыта для первого повторения

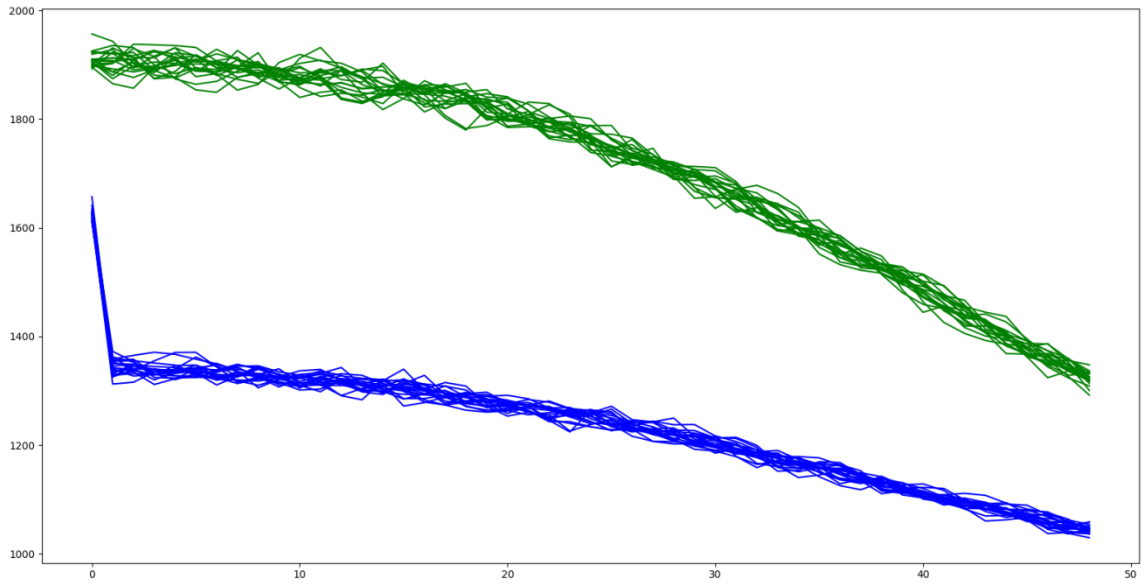


Рисунок 15 - Результаты пятого опыта для второго повторения

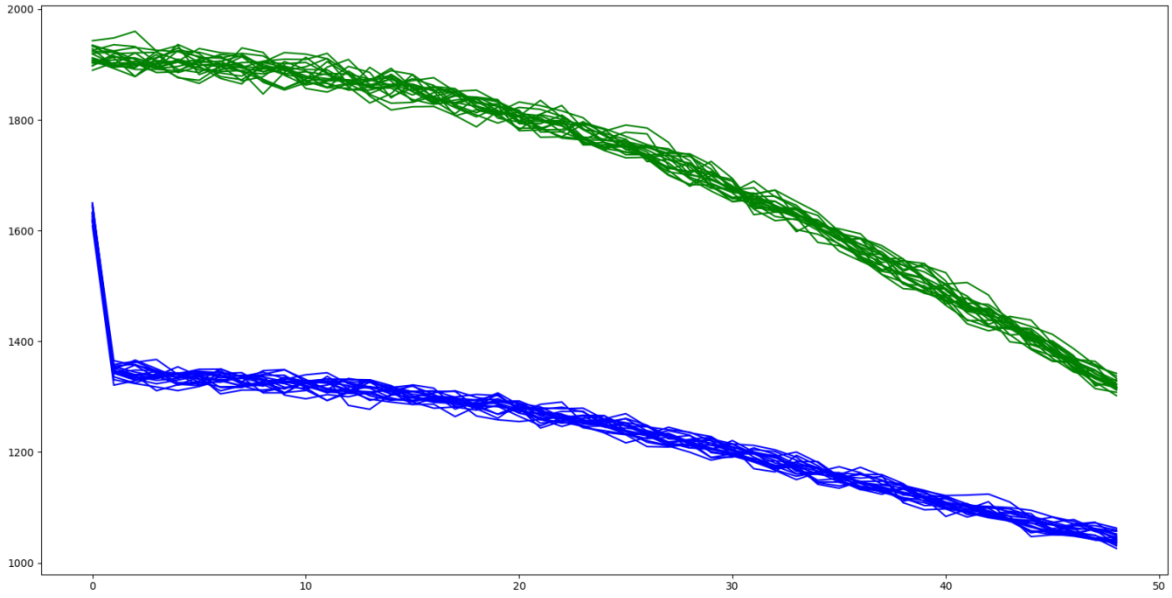


Рисунок 16 - Результаты пятого опыта для третьего повторения

4.4 Опыт 4

Получили набор линий. Теперь усредним этот набор, получая одну линию для каждого алгоритма.

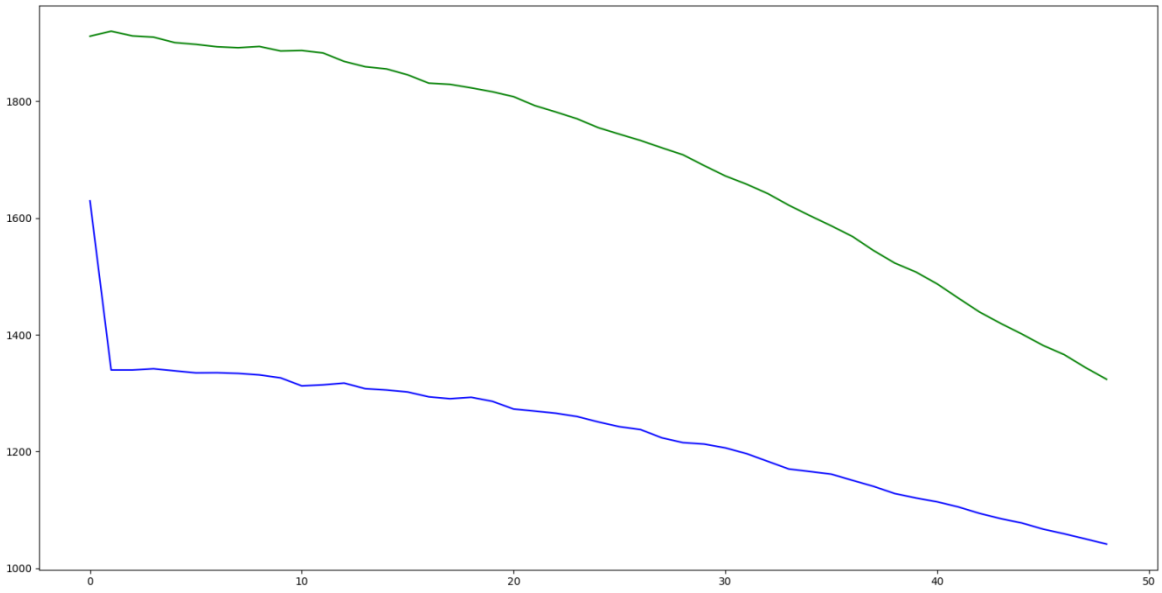


Рисунок 17 - Результаты шестого опыта

Из проведенных опытов видно, что всегда при одинаковых входных данных, алгоритм муравьиной колонии, основанный на выборе значения эвристического параметра, контролируемого информационной энтропией, имеет более быструю сходимость по сравнению с классическим алгоритмом муравьиной колонии.

Заключение

Таким образом, был реализован алгоритм муравьиной колонии, основанный на выборе значения эвристического параметра, контролируемого информационной энтропией. В ходе проведенной серии экспериментов, удалось показать, что при одинаковых входных данных, алгоритм муравьиной колонии, основанный на выборе значения эвристического параметра, контролируемого информационной энтропией находит более короткий маршрут.

Список источников

1. Источник: Галяутдинов Р.Р. Задача коммивояжера - метод ветвей и границ [Электронный ресурс] // Сайт преподавателя экономики. [2023]. URL: https://galyautdinov.ru/post/zadacha-kommivoyazhera (дата обращения: 04.05.2023).
2. Дулькейт В.И. Приближённое решение задачи коммивояжера методов рекурсивного построения вспомогательной кривой // В.И. Дулькейт, Р. Т. Файзуллин // ПДМ. 2009. №1 (3). – 72 с
3. Задача коммивояжера [Электронный ресурс] // Блог 4brain. URL:https://4brain.ru/blog/%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D0%B2%D0%BE%D1%8F%D0%B6%D0%B5%D1%80%D0%B0/ (дата обращения: 04.05.2023).
4. Эвристика - значение, примеры и методы[Электронный ресурс] // Библиотека Невероятных Фактов. [2023]. URL: https://detibib-nevelsk.ru/polezno-znat/evristika-znachenie-primery-i-metody.html?ysclid=lh9etypn5v464247512 (дата обращения: 04.05.2023).
5. Helsgaun K. An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic [Текст] // K. Helsgaun – Roskilde: Roskilde University, 1998.
6. Задачи маршрутизации [Текст] // А. И. Ерзин, Ю. А. Кочетов. // РИЦ НГУ, 2014.
7. Еще раз о решении задачи коммивояжера [Текст] // П. П. Пархоменко // Автоматика и телемеханика. //Москва, 2006. - № 12. - с. 190- 204.
8. Генетические алгоритмы. // Гладков Л.А., Курейчик В.М., Курейчик В.В// ООО «Ростиздат», 2004г.
9. Нейронные сети, генетические алгоритмы и нечеткие системы Д. Рутковская, М. Пилиньский, Л. Рутковский
10. Simulated Annealing Overview [Текст] // Franco Busetti

11. Solving Traveling Salesman Problem by Using Improved Ant Colony Optimization Algorithm// Zar Chi Su Su Hlaing, May Aye Khine // International Journal of Information and Education Technology//[Электронный ресурс]: статья. Режим доступа: URL: http://www.ijiet.org/papers/67-R052.pdf (дата обращения: 09.03.2023).
12. Муравьи и Python: ищем самые короткие пути [Электронный ресурс]: статья. Режим доступа: URL: https://vc.ru/newtechaudit/353372-muravi-i-python-ishchem-samye-korotkie-puti (дата обращения: 20.03.2023).