МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и технологий.

**КУРСОВОЙ ПРОЕКТ**

по дисциплине: «Моделирование и оптимальное проектирование технических систем»

на тему: «Решение задачи Коммивояжера генетическим алгоритмом»

Выполнил: ст. гр. 10701220 Дроздов А.С.

Проверил: Прихожий А. А.

Минск 2023

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и сетей.

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту**

по дисциплине: «Моделирование и оптимальное проектирование технических систем»

на тему: «Решение задачи Коммивояжера генетическим алгоритмом»

Выполнил: ст. гр. 10701220 Дроздов А.С.

Проверил: Прихожий А. А.

Минск 2023

**СОДЕРЖИМОЕ**

[ВВЕДЕНИЕ 6](#_Toc153363257)

[1. ПОСТАНОВКА ЗАДАЧИ 7](#_Toc153363258)

[2. МОДЕЛЬ РЕШЕНИЯ ЗАДАЧИ 8](#_Toc153363259)

[3. МЕТОДЫ РЕШЕНИЯ ЗАДАЧИ 9](#_Toc153363260)

[3.1 Метод полного перебора 9](#_Toc153363261)

[3.2 Метод динамического программирования 9](#_Toc153363262)

[3.3 Метод ветвей и границ 10](#_Toc153363263)

[3.4 Жадный алгоритм 11](#_Toc153363264)

[3.5 Генетический алгоритм 11](#_Toc153363265)

[3.6 Муравьиный алгоритм 12](#_Toc153363266)

[3.7 Алгоритм имитации отжига 13](#_Toc153363267)

[4. АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ 14](#_Toc153363268)

[5. ИНФОРМАЦИОННАЯ МОДЕЛЬ 17](#_Toc153363269)

[5.1 Графический интерфейс 17](#_Toc153363270)

[5.2 Входные данные 17](#_Toc153363271)

[5.3 Выходные данные 17](#_Toc153363272)

[6. АРХИТЕКТУРА ПРОГРАММЫ 18](#_Toc153363273)

[7. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ 19](#_Toc153363274)

[8. ТЕСТИРОВАНИЕ ПРОГРАММЫ 20](#_Toc153363275)

[9. ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ 23](#_Toc153363276)

[ЗАКЛЮЧЕНИЕ 31](#_Toc153363277)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 32](#_Toc153363278)

[ПРИЛОЖЕНИЯ А 33](#_Toc153363279)

[ПРИЛОЖЕНИЕ Б 36](#_Toc153363280)

[Файл funcs.py 36](#_Toc153363281)

# ВВЕДЕНИЕ

При изучении задач оптимизации важную роль играет изучение различных алгоритмов решения. При решении задач требуется не только понимать постановку, но и уметь решать их, применяя различные алгоритмы. Однако добиться этого удаётся не во всех случаях: возникает проблема непонимания принципа действия алгоритмов, в связи с чем обучающиеся не всегда способны применить полученные знания на практике.

В этом проекте мы будем исследовать одну из самых известных комбинаторных задач - задачу Коммивояжера. Нашей целью будет нахождение оптимального пути для коммивояжера-туриста, который должен посетить все туристические места из заданного списка ровно один раз и вернуться в исходное место, при этом минимизируется суммарное расстояние.

Для решения этой задачи мы будем использовать генетический алгоритм, который позволяет нам эффективно рассчитывать оптимальные пути с помощью итеративного подхода. Генетический алгоритм (ГА) - это метод оптимизации, вдохновленный процессами естественного отбора и генетической эволюции в биологических системах. Он используется для решения задач оптимизации и поиска наилучших решений в большом пространстве возможных вариантов.

Генетический алгоритм работает в рамках популяции индивидуальных решений, которые представляются в виде геномов или хромосом. Каждый индивидуум представляет собой потенциальное решение задачи. Геномы обычно представляются в виде строк или массивов, где каждый элемент представляет ген или параметр решения.

Генетические алгоритмы обладают свойствами эффективного исследования больших пространств решений и могут находить оптимальные или приближенные решения для многих задач оптимизации.

В ходе проекта мы изучим теоретические основы задачи Коммивояжера и генетический алгоритм, разработаем алгоритм решения, реализуем его на выбранном языке программирования и проведем эксперименты для оценки его эффективности.

# ПОСТАНОВКА ЗАДАЧИ

В ходе выполнения курсового проекта и разработки программного обеспечения необходимо выполнить следующие требования:

1. реализация графического интерфейса программы;
2. возможность задавать вручную параметры генетического алгоритма через графический интерфейс;
3. заполнение матрицы смежности из среды MS Excel;
4. возможность решения задачи Коммивояжера с помощью генетического алгоритма;
5. вывод результатов работы алгоритма в консоли;
6. вывод результатов решения задачи в графическом интерфейсе программы;
7. провести анализ работы алгоритма посредством сравнения затраченного времени на алгоритм и релевантности найденного решения (пути) к кратчайшему маршруту путём изменения вводимых параметров генетического алгоритма;
8. подробное изучение принципов работы генетического алгоритма;
9. понимание его применимости и эффективности для решения задачи коммивояжера.

Для разработки программного обеспечения будет использован язык программирования Python, среда разработки PyCharm Community Edition 2020.2.1. Для создания графического интерфейса программы будет использованы библиотеки PyQt5 и matplotlib (для разметки и вывода данных на рабочую область). Для работы с графом будет использована библиотека networkx, которая позволяет создавать, манипулировать и анализировать графы, которые представляют собой совокупность узлов и ребер между ними. NetworkX широко используется в научных и исследовательских областях, в задачах анализа данных, визуализации и моделирования. Для оценки скорости работы алгоритма будет использована системная библиотека time. Для математических расчётов будет использована библиотека numpy. Для парсинга данных из среды MS Excel используем библиотеку pandas.

# МОДЕЛЬ РЕШЕНИЯ ЗАДАЧИ

Задача коммивояжера (Traveling Salesman Problem, TSP) — это классическая задача оптимизации, которая заключается в поиске самого выгодного маршрута для коммивояжера, который должен посетить набор городов, вернувшись в исходный город, минимизируя при этом общую длину пути.

Задача коммивояжера формулируется следующим образом: дано множество N городов и матрица расстояний, которая указывает расстояние между каждой парой городов. Требуется найти замкнутый маршрут, проходящий через каждый город ровно один раз, такой, чтобы общая сумма расстояний была минимальной. Кроме того, коммивояжер должен вернуться в исходный город.

Маршрут коммивояжёра, который удовлетворяет условию задачи, задаётся следующей перестановкой целых чисел:

Каждому маршруту 𝜋 ставится в соответствие функция стоимости:

где – элементы матрицы смежности С. Тогда искомое решение представимо как 𝜋∗:

В сфере транспорта задача коммивояжера остается актуальной и по сей день. Транспортные, логистические и курьерские службы, а также службы такси стремятся оптимизировать свои маршруты. Современные ГИС-технологии позволяют получить информацию о протяженности маршрутов и затрачиваемом времени на их прохождение. Однако, с увеличением количества критериев оптимальности, таких как пробки и расход топлива, становится сложнее достичь наиболее оптимальных маршрутов. Несмотря на это, получение таких маршрутов может повысить удовлетворенность клиентов и прибыль компаний за счет сокращения расходов.

# МЕТОДЫ РЕШЕНИЯ ЗАДАЧИ

Для решения задачи Коммивояжера было разработана множество различных алгоритмов. Одними из таких методов являются:

1. метод полного перебора;
2. метод динамического программирования;
3. метод ветвей и границ;
4. жадный алгоритм;
5. генетический алгоритм;
6. муравьиный алгоритм;
7. алгоритм имитации отжига.

## Метод полного перебора

Полный перебор (или метод «грубой силы») — метод решения задачи путем перебора всех возможных вариантов. Сложность полного перебора зависит от количества всех возможных решений задачи. Если пространство решений очень велико, то полный перебор может не дать результатов в течение нескольких лет или даже столетий.

Обычно выбор решения можно представить последовательностью выборов. Если делать эти выборы с помощью какого-либо случайного механизма, то решение находится очень быстро, так что можно находить решение многократно и запоминать «рекорд», т. е. наилучшее из встретившихся решений. Этот наивный подход существенно улучшается, когда удается учесть в случайном механизме перспективность тех или иных выборов, т. е. комбинировать случайный поиск с эвристическим методом и методом локального поиска. Такие методы применяются, например, при составлении расписаний для Аэрофлота.

## Метод динамического программирования

Суть метода в том, что выбирается город, а затем для всех остальных городов в связке с ним строятся все возможные маршруты и выбирается самый короткий. В основе лежит простая идея осуществления сравнения различных вариантов допустимых решений задачи не в конце построения множества всех возможных вариантов, а на каждом шаге построения возможных вариантов.

Алгоритм рекурсивно применяется для всех оставшихся городов за вычетом выбранного на предыдущем этапе и так до тех пор, пока не выбранными остаются только два города. Для симметричной задачи часть маршрутов будет повторяться, а если результат сохранять отдельно, то его нужно рассчитать всего один раз. Метод динамического программирования может быть очень затратным по времени и ресурсам при решении задачи коммивояжера на больших размерностях.

## Метод ветвей и границ

Принцип работы метода ветвей и границ заключается в добавлении проверки критерия ограничивающей функции, по которому на некотором уровне можно приостановить построение ветви дерева перестановок. Его идея состоит в том, что ветвятся не все вершины. Сначала вершины просматриваются и каждая из них оценивается. Ветвится та вершина, которая получает лучшую оценку. Трудность состоит в получении этой оценки. Метод является улучшением алгоритма полного перебора и сохраняет его положительные свойства, однако при больших значениях N-го использование всё ещё неэффективно.

Принцип работы метода ветвей и границ можно описать следующим образом:

* создаётся дерево решений, где каждая вершина представляет собой частичное решение задачи. текущую задачу разбивается на подзадачи, добавив ограничения (ветви) к переменным решения;
* оценивается каждая вершина дерева с использованием некоторой верхней или нижней границы. Если текущая вершина не перспективна (не может привести к лучшему решению), она отбрасывается и не исследуется дальше;
* выбирается следующая вершину для исследования, которая обещает лучшее решение;
* процесс повторяется, пока не будет найдено оптимальное решение или не убедитесь, что дальнейший поиск бессмыслен;
* когда все вершины дерева исследованы или достигнут определенный критерий, завершите выполнение алгоритма.

## Жадный алгоритм

Жадный алгоритм для решения задачи коммивояжера основывается на локально оптимальных выборах на каждом шаге. Он начинает с одного случайно выбранного города и затем на каждом шаге выбирает ближайший непосещённый город в качестве следующего пункта назначения. Алгоритм продолжает выбирать ближайший непосёщенный город, пока все города не будут посещены, и затем возвращается обратно в исходный город.

Преимущество жадного алгоритма заключается в его простоте и скорости выполнения. Он не требует сложных вычислений и может быть легко реализован. Однако, жадный алгоритм не гарантирует нахождение оптимального решения и зачастую дает только приближенное решение.

Жадный алгоритм может быть улучшен путем включения различных оптимизаций, таких как использование эвристических правил или предварительное сортировка городов по некоторому критерию.

## Генетический алгоритм

Генетический алгоритм — это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путем последовательного подбора, комбинирования и вариации искомых параметров с использованием механизмов, напоминающих биологическую эволюцию. Является разновидностью эволюционных вычислений. Отличительной особенностью генетического алгоритма является акцент на использование оператора «скрещивания», который производит операцию рекомбинации решений-кандидатов, роль которой аналогична роли скрещивания в живой природе.

Задача кодируется таким образом, чтобы её решение могло быть представлено в виде вектора хромосома. Случайным образом создаётся некоторое количество начальных векторов («начальная популяция»). Они оцениваются с использованием «функции приспособленности», в результате чего каждому вектору присваивается определённое значение («приспособленность»), которое определяет вероятность выживания организма, представленного данным вектором. После этого с использованием полученных значений приспособленности выбираются вектора (селекция), допущенные к «скрещиванию». К этим векторам применяются «генетические операторы, создавая, таким образом следующее «поколение». Особи следующего поколения также оцениваются, затем производится селекция, применяются генетические операторы и т. д. Так моделируется «эволюционный процесс», продолжающийся несколько жизненных циклов (поколений), пока не будет выполнен критерий остановки алгоритма. Генетические алгоритмы служат, главным образом, для поиска решений в очень больших, сложных пространствах поиска.

## Муравьиный алгоритм

Муравьиные алгоритмы представляют собой вероятностную жадную эвристику, где вероятности устанавливаются исходя из информации о качестве решений, полученных на предыдущих шагах. Муравьиный алгоритм - один из эффективных полиномиальных алгоритмов для нахождения приближённых решений задач поиска маршрутов на графах посредством мета эвристической оптимизации.

Имитация самоорганизации муравьиной колонии составляет основу муравьиных алгоритмов. Под самоорганизацией понимается множество динамических механизмов, которые обеспечивают достижение глобальной цели системой в результате низкоуровневого развития этой системы. Проводя эксперименты по выбору муравьями между двумя маршрутами неравной длины, ведущими от колонии муравьев к источнику питания, биологи заметили, что, как правило, муравьи используют кратчайший маршрут.

Одним из способов передачи информации между муравьями является стигмергия –разнесённый во времени тип взаимодействия, при котором один субъект некоторым образом изменяет часть окружающей среды, а остальные субъекты используют эту информацию позже, когда находятся в окрестности. Стигмергия осуществляется через феромон –особое вещество, откладываемое как след при перемещении муравья. Феромон со временем испаряется, что позволяет муравьям подстраиваться под изменение внешней среды. Именно эти механизмы самоорганизации заложены в принцип действия муравьиных алгоритмов.

## Алгоритм имитации отжига

Алгоритм имитации отжига — метод решения задачи глобальной оптимизации. Он основывается на имитации физического процесса, который происходит при кристаллизации вещества, в том числе при отжиге металлов. Предполагается, что в определённый момент атомы вещества практически выстроены в кристаллическую решётку, но ещё допустимы переходы отдельных атомов из одной ячейки в другую. Активность атомов тем больше, чем выше температура, которую постепенно понижают, что приводит к тому, что вероятность переходов в состояния с большей энергией уменьшается. Устойчивая кристаллическая решётка соответствует минимуму энергии атомов, поэтому атом либо переходит в состояние с меньшим уровнем энергии, либо остаётся на месте.

В программной реализации под энергией понимается длина получившегося маршрута. Изначально маршрут задаётся случайно. На каждой итерации с помощью перестановки пары индексов мы генерируем новый маршрут. На основе его длины, длины предыдущего маршрута и текущего значения температуры алгоритм определяет, совершать ли переход в новое состояние или нет.

В конце каждой итерации система “остывает”: параметр, отвечающий за температуру, уменьшается. Когда алгоритм отработает отведённое количество итераций, он вернёт минимальный найденный маршрут, являющийся решением задачи.

# АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ

Пусть G=(V, E) – неориентированный полный взвешенный конечный граф, где V={v1,…,vk} – множество вершин (городов); E={(i,j) | i,jV, ij} – множество ребер (участков пути), соединяю-щих вершины (города). Каждое ребро (i,j) имеет вес w(i,j), описывающий длину участка пути.

Пусть s есть исходная (начальная) вершина, выйдя из которой коммивояжер обходит по разу все другие вершины и возвращается в s. Пусть далее pi=(v1,v2,…,vi) – пройденный к настоящему моменту времени промежуточный маршрут, где v1=s. Маршрут p=(v1,v2,…,vk,v1) является одним из завершенных маршрутов. Полное множество P всех завершенных маршрутов получается в ре-зультате произвольной перестановки вершин v2,…,vk в маршруте p.

Длина промежуточного маршрута оценивается выражением pi

.

Обозначим через W(p) длину завершенного маршрута p. Задача коммивояжера формулиру-ется как

.

Псевдокод алгоритма будет выглядеть следующим образом:

**def genetic\_algorithm**():

# Создаем начальную популяцию случайных маршрутов **population**

# Основной цикл генетического алгоритма **for**:

# Оцениваем приспособленность (длину маршрута) каждого маршрута в популяции

**fitness\_scores** = []

# Выбираем лучшие маршруты для размножения **selected\_routes**:

# Селекция (выбор родителей):

#Вероятность выбора маршрута пропорциональна его приспособленности **probabilities**

# Выбираем родителей с помощью рулеточной селекции **for**:

random.choices(population, probabilities)

# Создаем новую популяцию путем скрещивания и мутации

population = crossover():

#Добавляем лучшие маршруты без изменений в новую популяцию

#Скрещиваем родителей, чтобы создать новых потомков **for**:

# Копируем участок маршрута от одного родителя

# Заполняем оставшиеся гены из другого родителя **for**:

#Получаем ребёнка с двумя генотипами

#Получаем новую популяцию

population = mutation():

# Меняем местами две случайные вершины в маршруте

#Получаем новую популяцию

# Находим лучший маршрут из финальной популяции

best\_route = min(population)

best\_length = fitness(best\_route)

Важно отметить, что задача коммивояжера является NP-трудной, и для больших наборов городов может быть трудно найти точное оптимальное решение в разумное время. Поэтому зачастую применяются приближенные методы, которые дают хорошее приближение оптимального решения или эвристические методы, такие как генетический алгоритм, которые находят удовлетворительные решения за разумное время с процентом погрешности за счёт своей специфики.

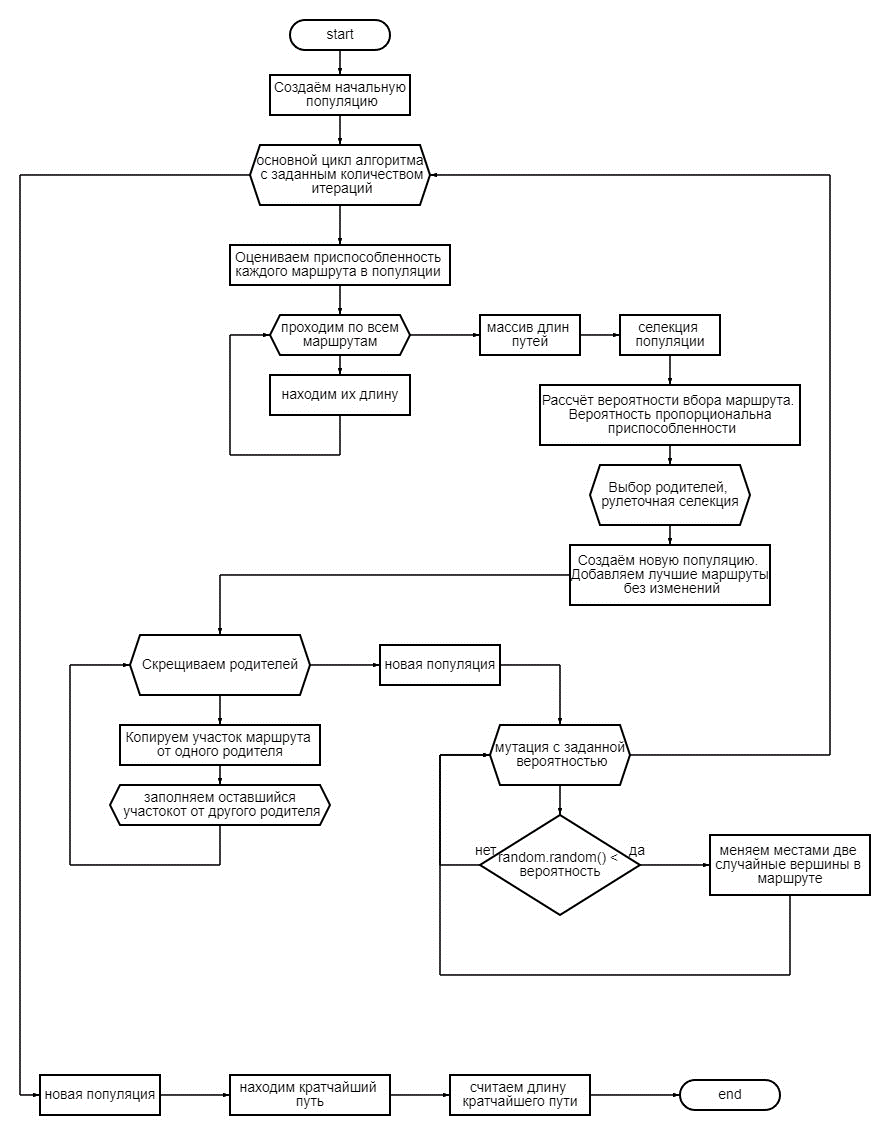


Рисунок 4.1 – блок-схема алгоритма

# ИНФОРМАЦИОННАЯ МОДЕЛЬ

Для создания графа используется матрица размером, в которую занесены длины путей между туристическими объектами Беларуси. Матрица парсится из файла TuristPath.xlsx среды MS Excel.

Кратчайший путь в графе представляет собой оптимальный маршрут, который приближен к минимальной величине длин маршрута или равен ей, что приемлемо для достижения цели.

Кратчайший путь в коде программы представлен в виде списка вершин в той последовательности, в которой был произведен обход, в самой программе – визуально отображен в виде графа.

## Графический интерфейс

Диалоговое окно PyQt представляет собой прямоугольную область, в которой отображаются следующие элементы:

* Кнопка «run» для запуска алгоритма;
* Поля ввода параметров размера популяции, количества итераций и вероятности мутаций.

## Входные данные

* Матрица смежности из файла Excel.
* Размер популяции.
* Количество итераций алгоритма.
* Вероятность мутации популяции.

## Выходные данные

* Количество точек на графе
* Затраченное время на работу алгоритма
* Последовательность обхода графа
* Длина кратчайшего пути обхода по графу

# АРХИТЕКТУРА ПРОГРАММЫ

Программа имеет следующую физическую структуру проекта:

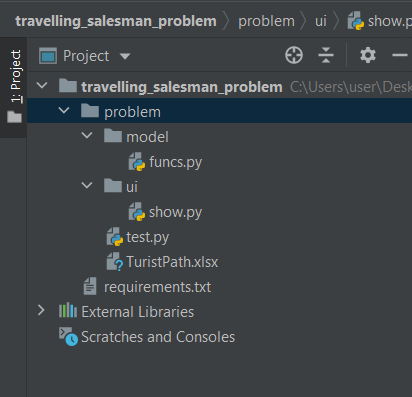


Рисунок 6.1 – Физическая структура проекта

Основной файл является отправной точкой программы. Здесь производится запуск главного цикла приложения.

В папке содержится один файл show.py который отвечает за визуальную составляющую программы. На рисунке 6.2 изображена диаграмма класса MainWindow() их этого файла.

В папке содержится файл , который заключает в себе все вызываемые функции алгоритма.

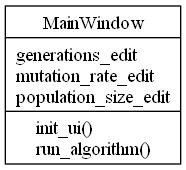


Рисунок 6.2 – Диаграмма класса MainWindow()

Класс производит определение и ввод параметров генетического алгоритма, а также самой формы ввода и кнопки запуска алгоритма. Ещё класс хранит основную функцию запуска алгоритма.

# ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ

Графический интерфейс пользователя представлен в двух формах (рисунок 7.1 и рисунок 7.2). Диалоговое окно PyQt содержит в себе следующие элементы:

* кнопка «run», при нажатию на которую создаётся сам граф, запускается алгоритм решения и вывод решения в виде графического окна networkx;
* текстовое поле для ввода размера популяции алгоритма;
* текстовое поле для ввода количества итераций алгоритма;
* текстовое поле для ввода вероятности мутации популяции.

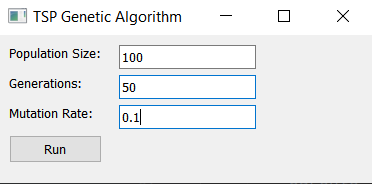


Рисунок 7.1 – графический интерфейс пользователя

Окно networkx (рисунок 7.2) содержит в себе:

* рамку вывода выходных значений алгоритма(количество точек, длина кратчайшего из найденных путей, сам путь и затраченное время);
* непосредственно граф с нанесёнными весами рёбер и отмеченными вершинами, а также маршрут коммивояжера.

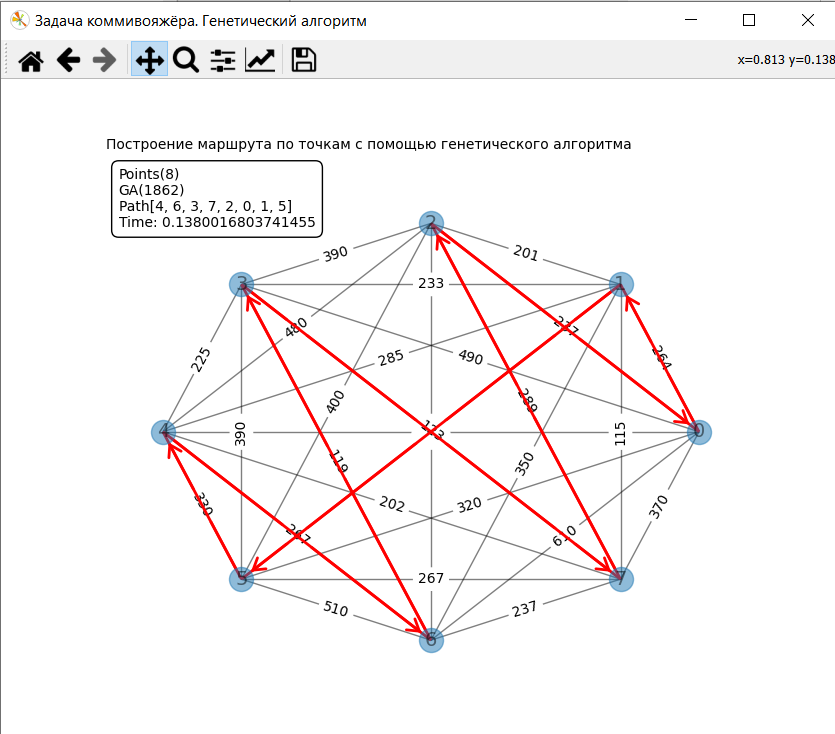


Рисунок 7.1 – графический интерфейс пользователя, окно networkx

# ТЕСТИРОВАНИЕ ПРОГРАММЫ

Тестирование проводилось на ноутбуке под операционной системой Windows 10 Корпоративная, процессор AMD Ryzen 5 4500U with Radeon Graphics 2.38 GHz, ОЗУ 8 ГБ.

Тестируя программу, проверим полную функциональность программного приложения. Необходимо, чтобы работа программы была осмысленной, логичной и обрабатывала все исключительные ситуации. Проверим работоспособность каждой функции.

При попытке запустить алгоритм, при этом не введя данных в поля программа выдает предупреждающее сообщение (рисунок 8.1)

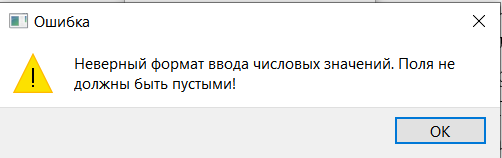


Рисунок 8.1 – предупреждающее сообщение

При попытке ввода чисел вне диапазона значений (рисунок 8.2) программа выдаст следующее сообщение (рисунок 8.3)

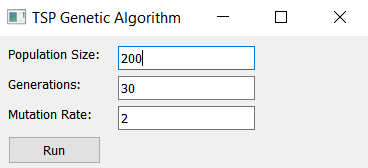


Рисунок 8.2 – введенные данные

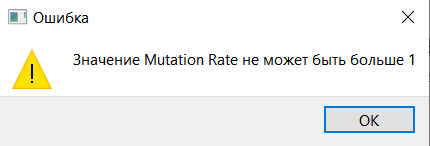


Рисунок 8.3 – сообщение об ошибке

В программе также предусмотрен запрет на ввод любых символов кроме цифр.

При правильно введенных данных: популяция 200, количество итераций 30, вероятность мутации 0.1 (рисунок 8.4), после нажатия кнопки «run» считается матрица смежности из файла Excel, выведется её значение в консоль (рисунок 8.5), начнётся работа генетического алгоритма по поиску оптимального пути и результат выведется на экран в виде графического изображения графа (рисунок 8.6).

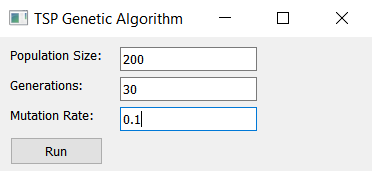


Рисунок 8.4 – правильно заполненная форма вводимых параметров

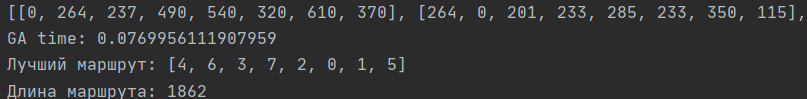


Рисунок 8.5 – вывод матрицы смежности и результатов выполнения в консоли

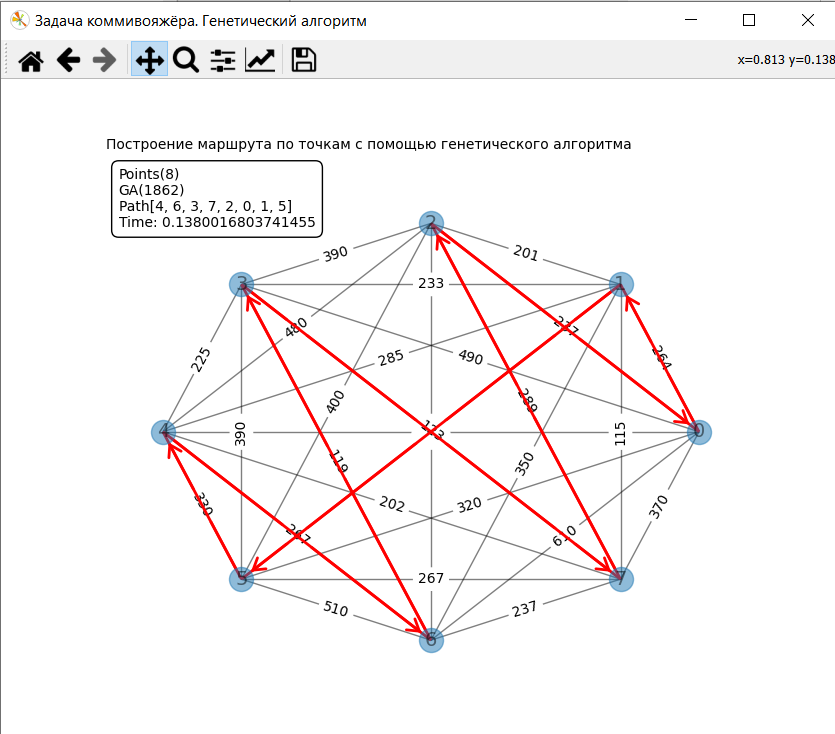


Рисунок 8.5 – результат работы генетического алгоритма и графическое изображение графа с найденным оптимальным путём коммивояжера

# ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

После запуска программы перед пользователям открывается диалоговое окно где нужно ввести параметры алгоритма.

Перед решение задачи пользователь вводит данные в текстовые поля и нажимает на кнопку run (рисунок 9.1)

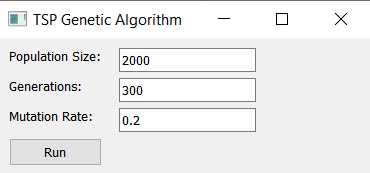


Рисунок 9.1 – окно ввода данных и запуска

После мы можем решить задача коммивояжера для заданной в файле матрицы. После нажатия на соответствующую кнопку запуска открывается новое окно с решением (рисунок 9.2).

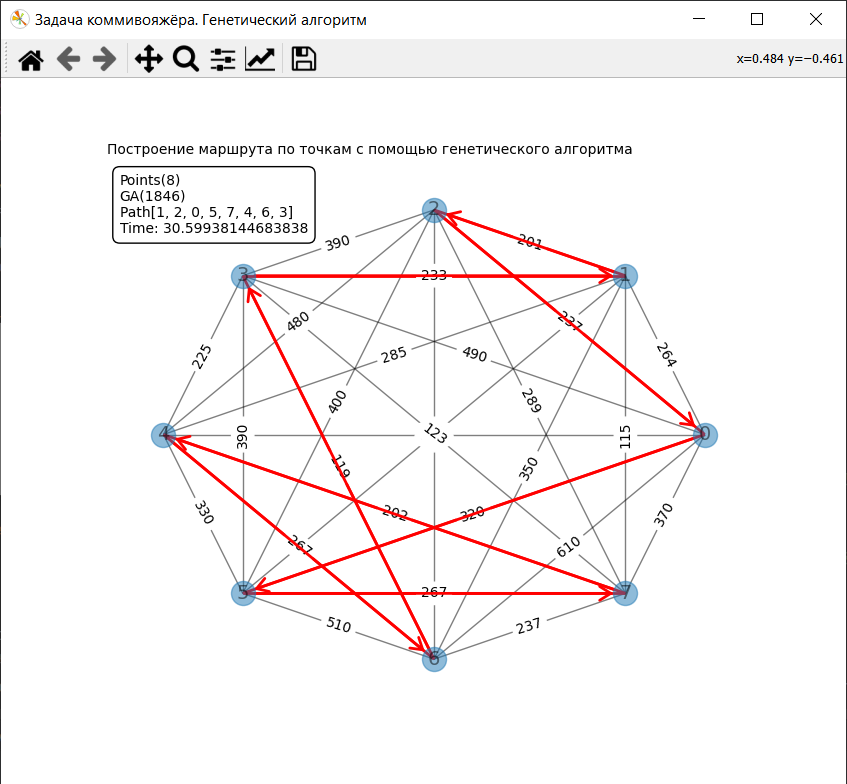


Рисунок 9.2 – решение задачи

Для получения некой выборки решений введём разные параметры запуска алгоритма и прогоним их несколько раз для поиска более оптимального пути.

Сначала выберем небольшой размер популяции и количество итераций (рисунок 9.3)

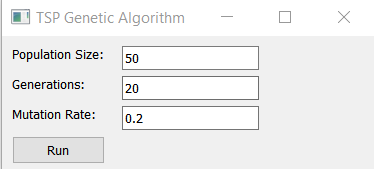


Рисунок 9.3 – вводимые параметры

Прогоним эти значения 3 раза (рисунок 9.4, рисунок 9.5, рисунок 9.6)

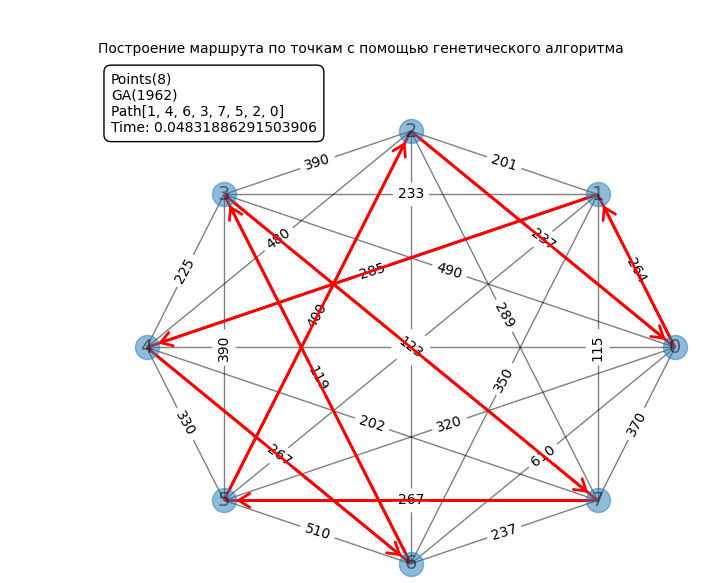


Рисунок 9.4 – результат (1)

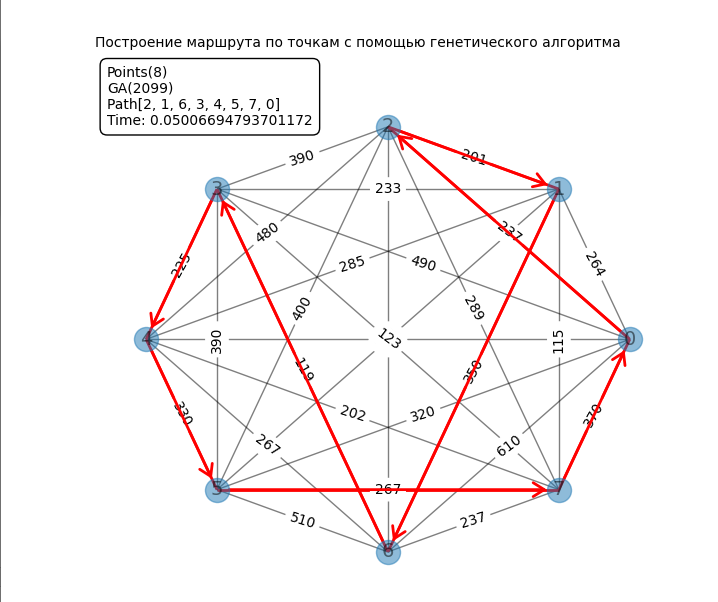


Рисунок 9.5 – результат (2)

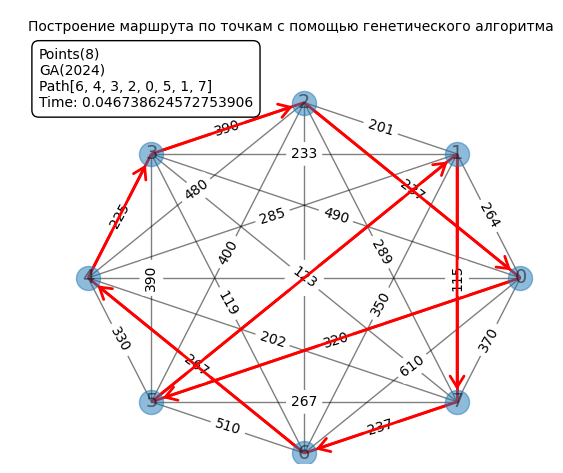


Рисунок 9.6 – результат (3)

Из выборки решений при первом случае вводимых параметров был найден оптимальный кратчайший путь при результате 1, равный 1962 километра, что близко к оптимальному значению. Из результатов видно, что результаты отличаются, что говорит о погрешности алгоритма и проценте случайности. Также, затраченное время довольно мало (рисунок 9.7).



Рисунок 9.7 – примерный результат решения при малых заданных параметрах

Возьмём параметры побольше для более корректной и точной работы алгоритма (рисунок 9.8).

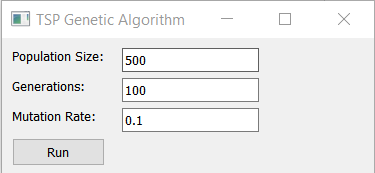


Рисунок 9.8 – вводимые параметры

Прогоним снова значения 3 раза (рисунок 9.9, рисунок 9.10, рисунок 9.11)

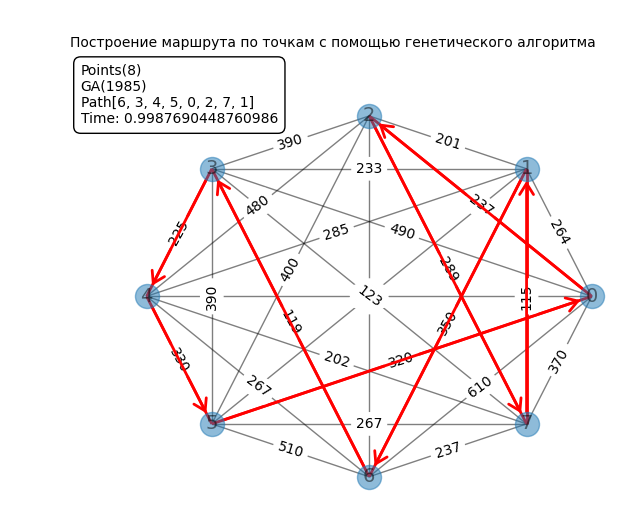


Рисунок 9.9 – результат (1)

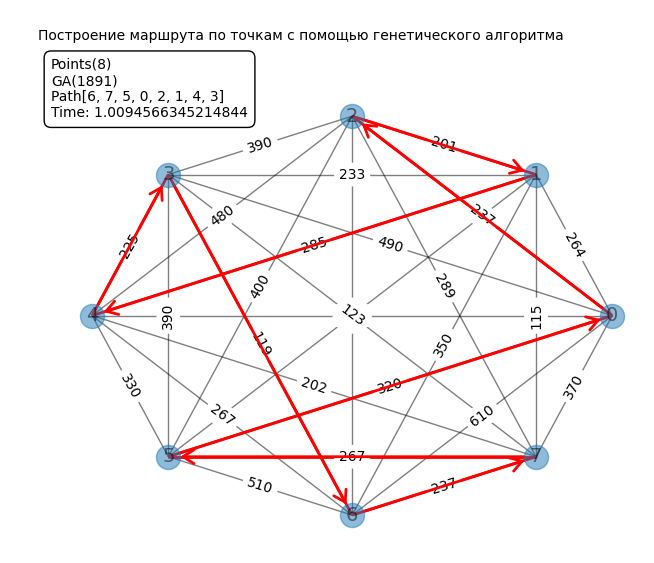


Рисунок 9.10 – результат (2)

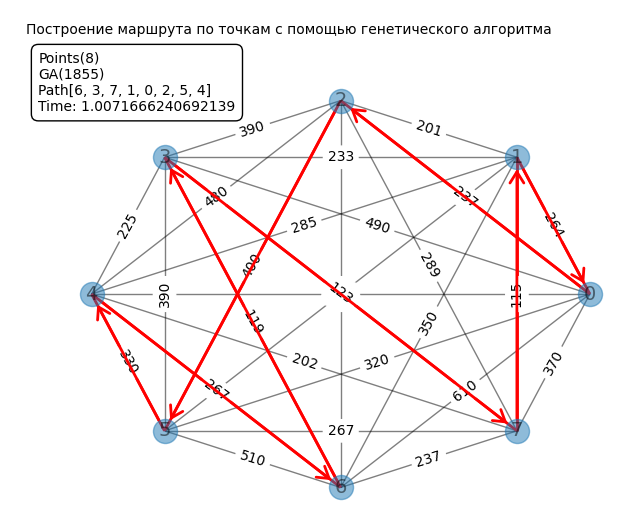


Рисунок 9.11 – результат (3)

Из выборки решений при втором случае вводимых параметров был найден оптимальный кратчайший путь при результате 3, равный 1855 километра, что ещё ближе к оптимальному значению. Время выполнения алгоритма существенно увеличилось, но также мало.

В конце проверим работу алгоритма с относительно большими значениями (рисунок 9.12).

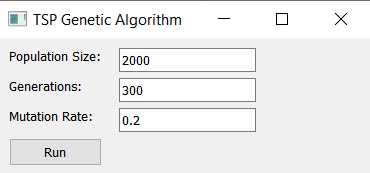


Рисунок 9.12 – вводимые параметры

Прогоним работу алгоритма один раз (рисунок 9.13).

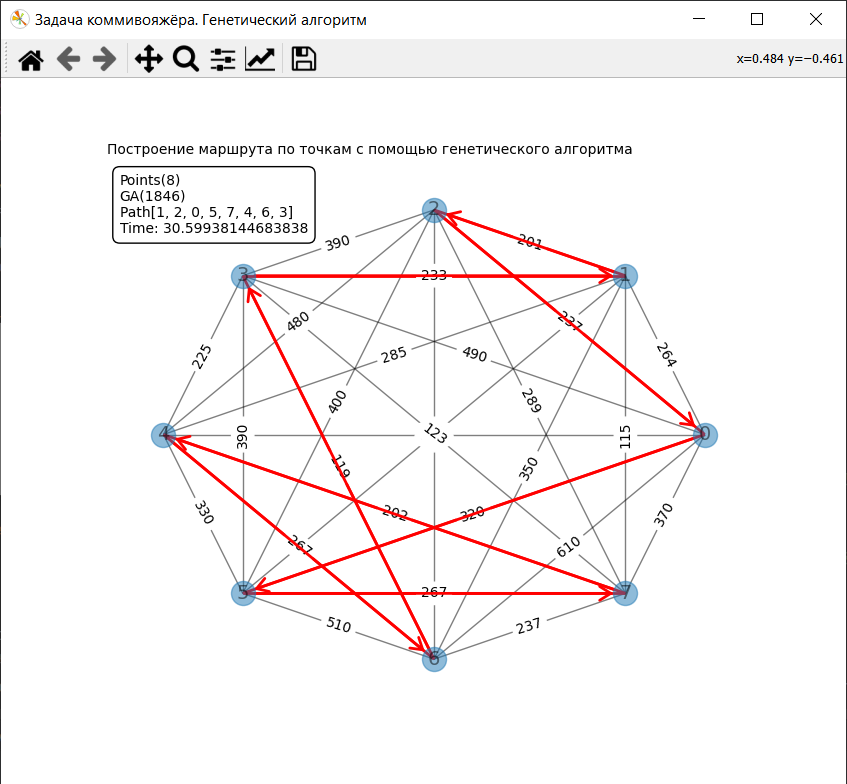


Рисунок 9.12 – результат выполнения алгоритма для больших параметров

В этом случае вводимых параметров был найден оптимальный кратчайший путь при результате равный 1846 километров, что равно значению. Время выполнения алгоритма существенно увеличилось и уже внушительное.

Также стоит отметить как выглядят наши исходные данные в среде MS Excel (рисунок 9.13).

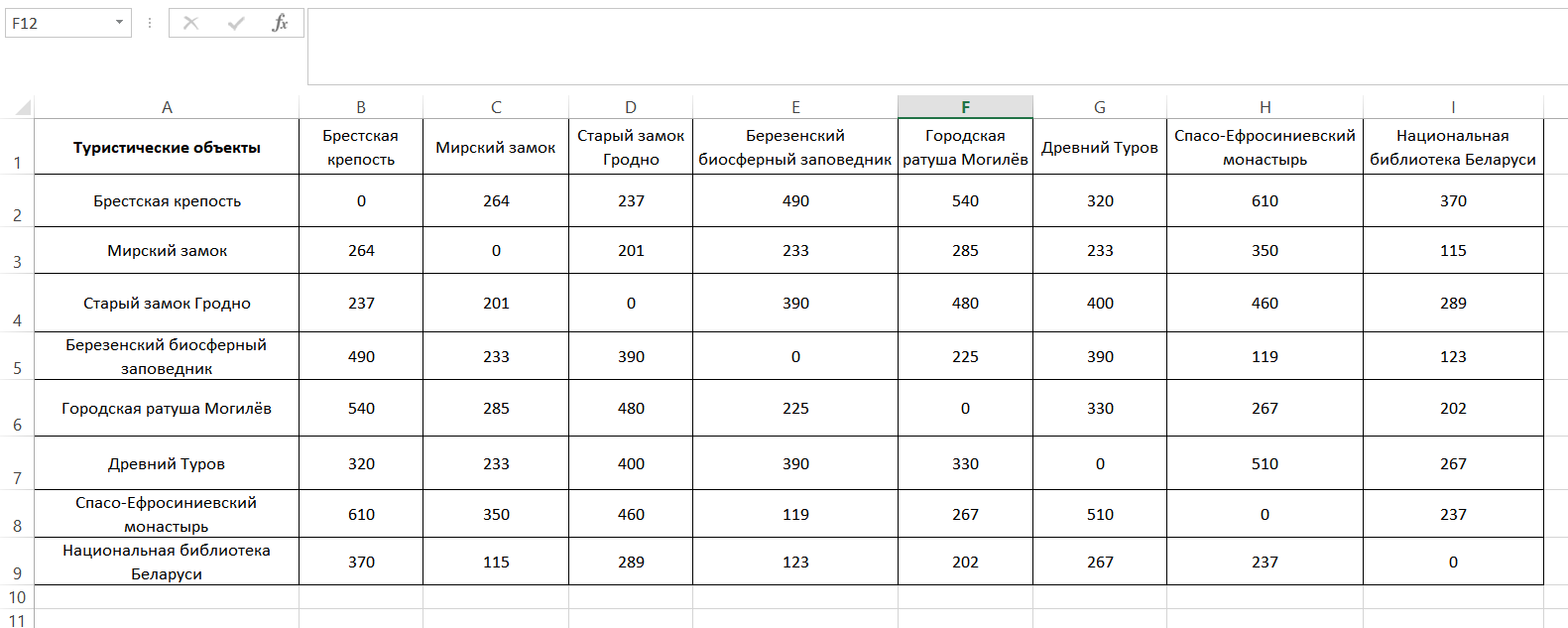


Рисунок 9. 13 – файл TuristPath.xlsx

Данные для заполнения таблицы были взяты из ресурса <https://yandex.by/maps> и выбирался кратчайший путь между отдельными вершинами. На рисунках 9.14-9.16 показаны примеры нахождения значений.

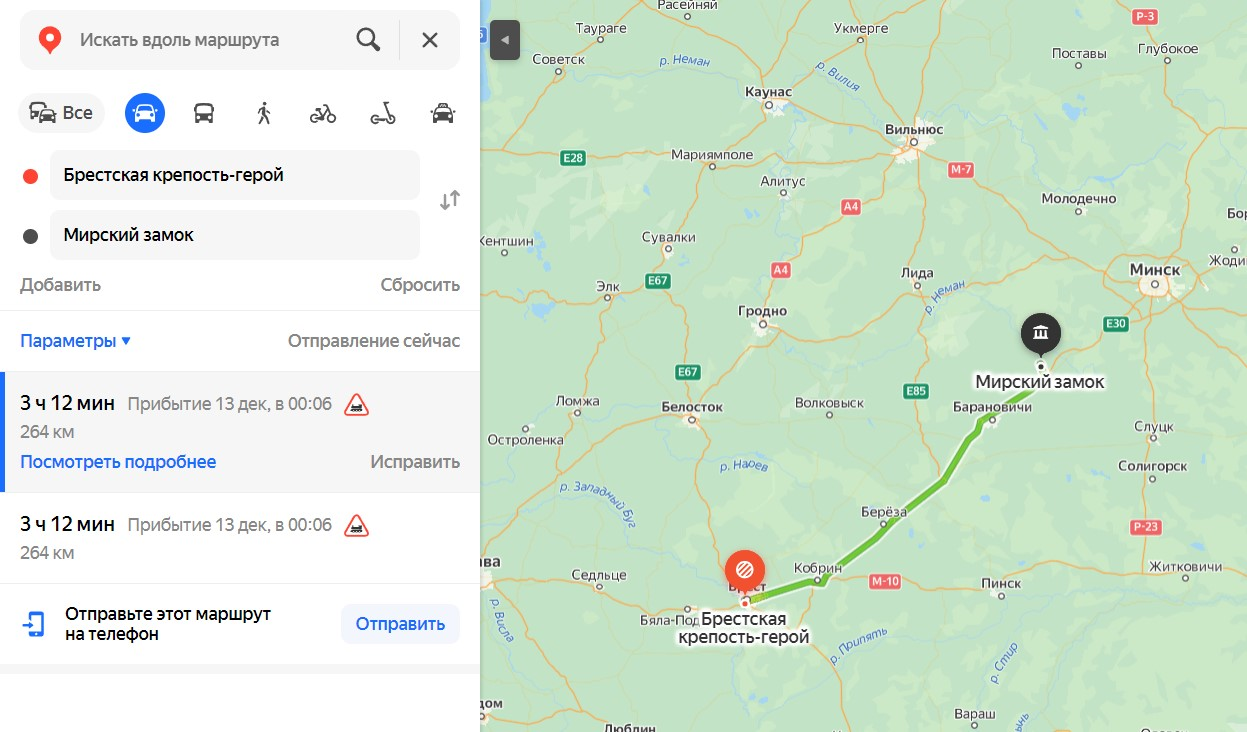


Рисунок 9. 14 – путь Брестская крепость-герой -> Мирский замок

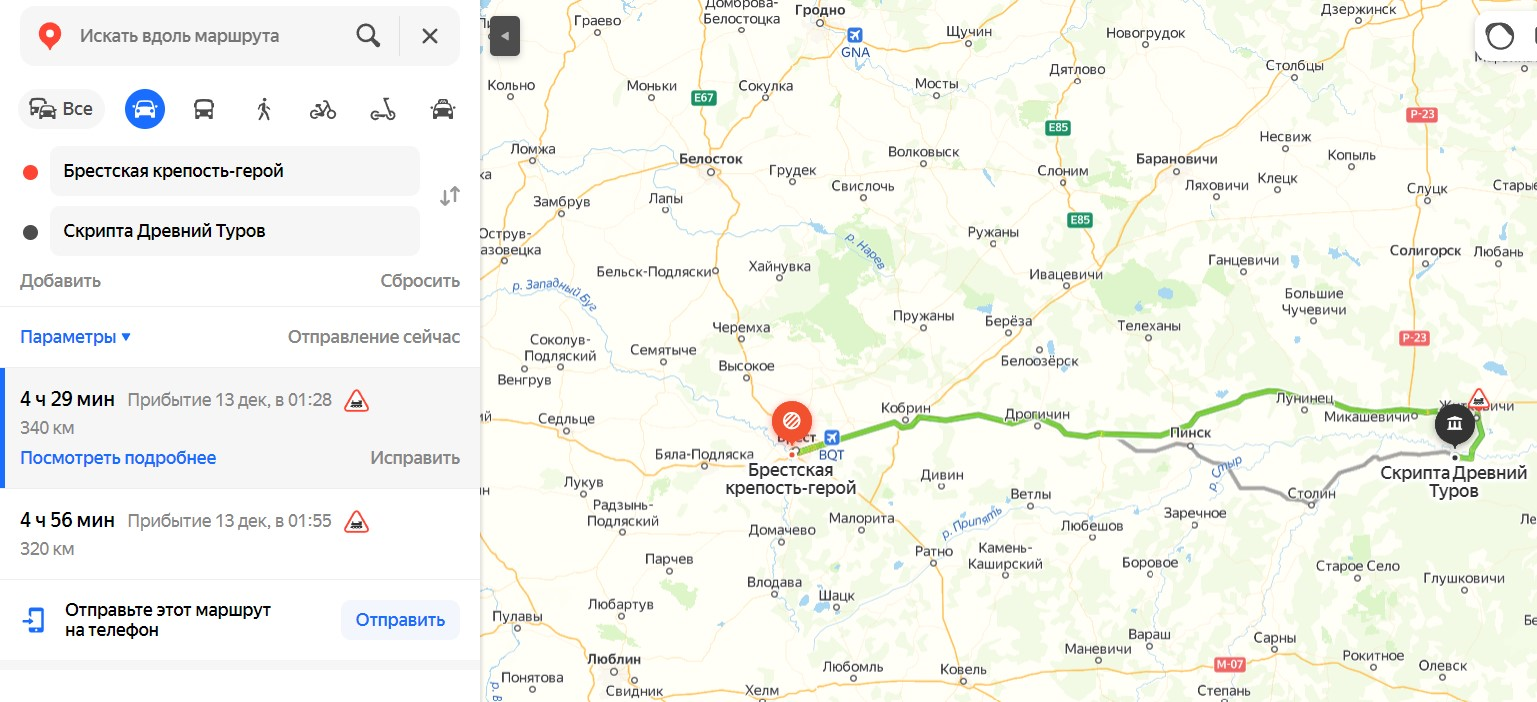


Рисунок 9. 15 – путь Брестская крепость-герой -> Скрипта Древний Туров

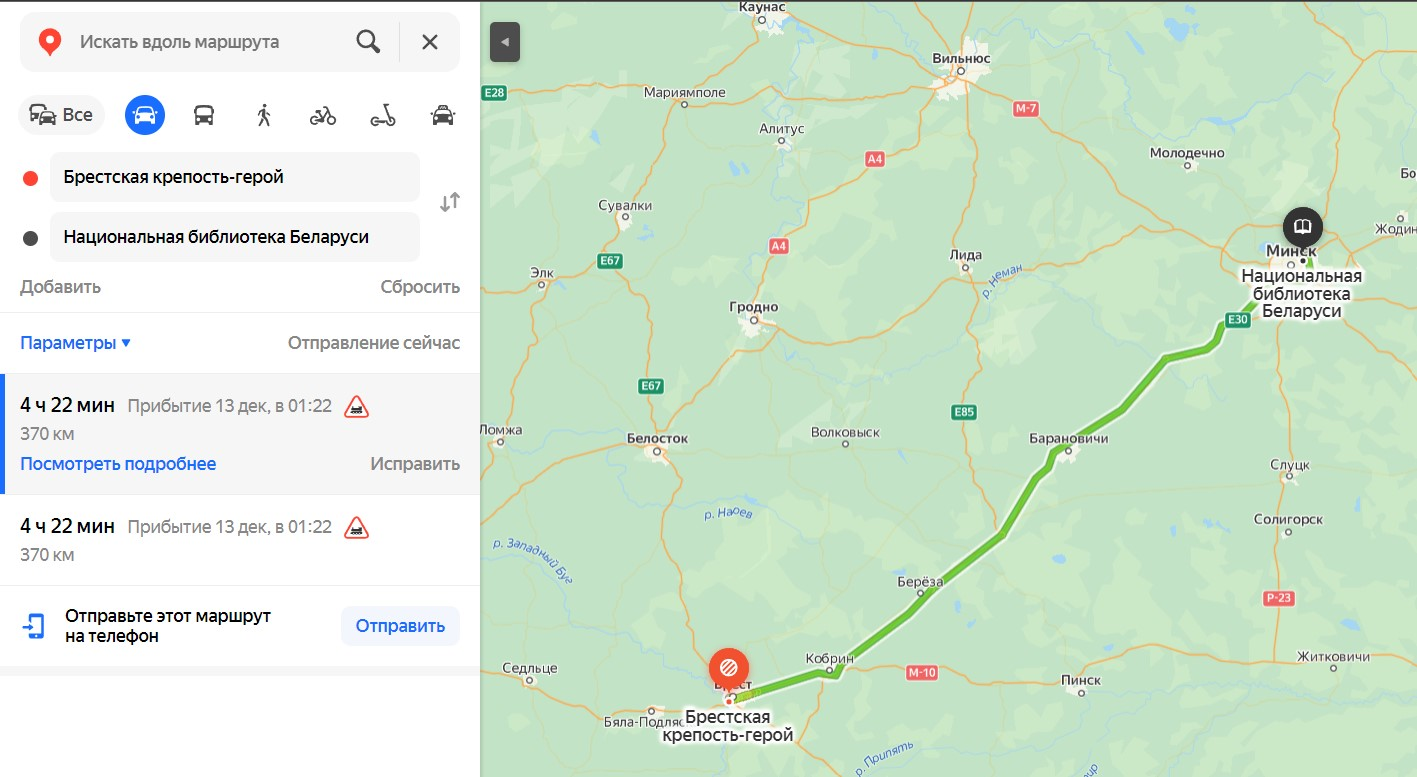


Рисунок 9. 16 – путь Брестская крепость-герой -> Нац. Библиотека Беларуси

Из найденного кратчайшего пути [1.2,0,5,7,4,6,3] можно составить маршрут: Мирский замок - Старый замок Гродно - Брестская крепость -Древний Туров - Национальная библиотека Беларуси - Городская ратуша Могилёв - Спасо-Ефросиниевский монастырь - Березенский биосферный заповедник.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта была исследована задача коммивояжера и разработано программа для её решение генетического алгоритма. Этот метод позволяет эффективно находить оптимальный путь для коммивояжера, перебирая все возможные комбинации маршрутов, скрещивая их, и выбирая наименьшую стоимость.

На основе полученных в результате тестовых запусков программы данных можно сделать следующие вывод: генетический алгоритм может быть очень затратным по времени и ресурсам при решении задачи коммивояжера на больших размерностях. Так как он перебирает все случайные комбинации маршрутов и их число равно числу популяции, а также выполнение алгоритма происходит на большом количестве итераций для генетического отбора, время выполнения алгоритма растет экспоненциально с увеличением параметров. Также при большом числе городов решение генетическим алгоритмом может столкнуться с ограничениями памяти компьютера. Генетический алгоритм дает гарантированное оптимальное решение задачи коммивояжера.

В целом, генетический алгоритм имеет свои плюсы и минусы при решении задачи коммивояжера. К плюсам можно отнести: универсальность

(т.к. может использоваться для решения широкого спектра задач оптимизации), параллелизм (алгоритм легко распараллелить, что позволяет использовать множество вычислительных ресурсов для эффективного поиска оптимального решения, что особенно полезно для больших объёмов данных и итераций), глобальный поиск (алгоритм способен искать решения во всем пространстве возможных решений, не застревая в локальных оптимумах).

К минусам можно отнести: вычислительная сложность, неопределённость сходимости и подбор параметров. Все эти факторы и делают генетический алгоритм эвристическим.

Полученные результаты могут быть использованы в различных сферах, таких как логистика, транспортное планирование, оптимизация маршрутов доставки и других областях.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. **D Lakshmi** What is travelling salesman problem | Locus blog [В Интернете] // blog.locus. - Mara Labs Inc. - 11 2023 г.. - https://blog.locus.sh/travelling-salesman-problem-and-how-can-tech-solve-it/.
2. Matplotlib: Visualization with Python [В Интернете] // Mathplotlib.com. - 11 2023 г.. - https://matplotlib.org/.
3. NetworkX - NetworkX documentation [В Интернете] // networkx.com. - NetworkX developers. - 11 2023 г.. - https://networkx.org/.
4. Python OOPs Concepts [В Интернете] // geeksforgeeks. - Geeksforgeeks Sanchhaya Education Privete Limited. - 11 2023 г.. - https://www.geeksforgeeks.org/python-oops-concepts/.
5. **rebuilder** Задача Коммивояжера генетическим алгоритмом [В Интернете] // Habr.com. - 15 11 2023 г.. - https://habr.com/ru/articles/701458/.
6. PyQt — Python interface to windows [В Интернете] // docs.python. - Python Software Foundation. - 12 2023 г.. - https://docs.python.org/3/library/PyQt5.html.
7. Welcome to python.org [В Интернете] // Python.org. - 12 07 2023 г.. - https://www.python.org/.
8. **А.А. Прихожий** Конспект лекций по дициплине МиОПТС. - Минск : БНТУ Кафедра "Программное обеспечение информационных технологий", 2013.
9. Эвристические алгоритмы [В Интернете] // tproger. - 11 2023 г.. - https://tproger.ru/articles/dynprog-starters.
10. Руководство по программированию на tkinter [В Интернете] // METANIT. - metanit.com. - 12 2023 г.. - https://metanit.com/python/tkinter/.

# ПРИЛОЖЕНИЯ А

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |
|  | Фамилия | Подпись | Дата | *Блок-схема работы алгоритма* | Лист | Листов |
| Студент | Дроздов |  |  |
| Руководитель | Прихожий |  |  | *1* | *4* |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |
|  | Фамилия | Подпись | Дата | *Диаграммы основных классов приложения и его структура* | Лист | Листов |
| Студент | Дроздов |  |  |
| Руководитель | Прихожий |  |  | *2* | *4* |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |
|  | Фамилия | Подпись | Дата | *Графический интерфейс программы* | Лист | Листов |
| Студент | Дроздов |  |  |
| Руководитель | Прихожий |  |  | *3* | *4* |

# ПРИЛОЖЕНИЕ Б

## Файл funcs.py

import random

import pandas as pd

# Генетический алгоритм для решения задачи коммивояжера

def tsp\_genetic\_algorithm(graph, population\_size, generations, mutation\_rate):

num\_vertices = len(graph)

# Создаем начальную популяцию случайных маршрутов

population = []

for \_ in range(population\_size):

route = list(range(num\_vertices))

random.shuffle(route)

population.append(route)

# Основной цикл генетического алгоритма

for \_ in range(generations):

# Оцениваем приспособленность каждого маршрута в популяции

fitness\_scores = []

for route in population:

fitness\_scores.append(fitness(route, graph))

# Выбираем лучшие маршруты для размножения

selected\_routes = selection(population, fitness\_scores)

# Создаем новую популяцию путем скрещивания и мутации

population = crossover(selected\_routes, population\_size)

population = mutation(population, mutation\_rate)

# Находим лучший маршрут из финальной популяции

best\_route = min(population, key=lambda route: fitness(route, graph))

best\_length = fitness(best\_route, graph)

return best\_route, best\_length

# Функция приспособленности (длина маршрута)

def fitness(route, graph):

length = 0

for i in range(len(route)):

length += graph[route[i]][route[(i + 1) % len(route)]]

return length

# Селекция (выбор родителей)

def selection(population, fitness\_scores):

selected\_routes = []

total\_fitness = sum(fitness\_scores)

# Вероятность выбора маршрута пропорциональна его приспособленности

probabilities = [score / total\_fitness for score in fitness\_scores]

# Выбираем родителей с помощью рулеточной селекции

for \_ in range(len(population)):

selected\_routes.append(random.choices(population, probabilities)[0])

return selected\_routes

# Скрещивание (кроссовер)

def crossover(selected\_routes, population\_size):

new\_population = []

# Добавляем лучшие маршруты без изменений в новую популяцию

new\_population.extend(selected\_routes[:population\_size // 2])

# Скрещиваем родителей, чтобы создать новых потомков

for \_ in range(population\_size // 2):

parents = random.sample(selected\_routes, 2)

child = [None] \* len(parents[0])

start, end = sorted(random.sample(range(len(parents[0])), 2))

# Копируем участок маршрута от одного родителя

child[start:end] = parents[0][start:end]

# Заполняем оставшиеся гены из другого родителя

index = end

for vertex in parents[1]:

if vertex not in child:

child[index] = vertex

index = (index + 1) % len(parents[0])

if index == start:

break

new\_population.append(child)

return new\_population

# Мутация

def mutation(population, mutation\_rate):

for i in range(len(population)):

if random.random() < mutation\_rate:

# Меняем местами две случайные вершины в маршруте

random\_indices = random.sample(range(len(population[i])), 2)

population[i][random\_indices[0]], population[i][random\_indices[1]] = population[i][random\_indices[1]], \

population[i][random\_indices[0]]

return population

def get\_graph(file\_path):

# Считывание данных из Excel файла

data\_frame = pd.read\_excel(file\_path, header=None)

# Выбор указанного диапазона столбцов и строк

selected\_data = data\_frame.iloc[1:10, 1:10]

# Преобразование данных в массив по строкам

graph\_array = selected\_data.values.tolist()

return graph\_array