МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

Тема: Генетические алгоритмы

Студентка гр. 1304 _____ Виноградова М.О.
Студент гр. 1304 _____ Стародубов М.В.
Студент гр. 1304 _____ Макки К.Ю.
Руководитель _____ Жангиров Т.Р.

Санкт-Петербург
2023

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студентка Виноградова М.О. группы 1304

Студент Стародубов М.В. группы 1304

Студент Макки К.Ю. группы 1304

Тема практики: Генетические алгоритмы

Задание на практику:

Решение задачи коммивояжера на C++ с графическим интерфейсом.

Алгоритм: коммивояжера.

Сроки прохождения практики: 30.06.2023 – 13.07.2023

Дата сдачи отчета: 13.07.2020

Дата защиты отчета: 13.07.2020

Студентка	_____	Виноградова М.О.
Студент	_____	Стародубов М.В.
Студент	_____	Макки К.Ю.
Руководитель	_____	Жангиров Т.Р.

АННОТАЦИЯ

С. 38. Ил. 11. Литература 5. Прил. 1

Объектом исследования данной практической работы является алгоритм коммивояжера. Этот алгоритм предназначен для нахождения наиболее оптимального маршрута, проходящего через указанные города хотя бы один раз и возвращающегося в исходный город.

Целью данного проекта является разработка программы с графическим интерфейсом на основе генетических алгоритмов. Этот интерфейс позволит пользователям взаимодействовать с программой и наблюдать пошаговую визуализацию процесса поиска оптимального решения.

В процессе подготовки к практической работе были использованы следующие ресурсы:

Панченко, Т. В. Генетические алгоритмы [Текст] : учебно-методическое пособие / под ред. Ю. Ю. Тарасевича. — Астрахань : Издательский дом «Астраханский университет», 2007. — 87 [3] с.

В работе был представлен процесс решения задачи, документация по использованию программы, а также приведены примеры кода на языке C++ и результаты тестирования.

SUMMARY

The object of research of this practical work is the traveling salesman algorithm(TSP). This algorithm is designed to find the most optimal route passing through the specified cities at least once and returning to the original city.

The purpose of this project is to develop a program with a graphical interface based on genetic algorithms. This interface will allow users to interact with the program and observe a step-by-step visualization of the process of finding the optimal solution.

In the process of preparing for the practical work, the following resources were used:

Панченко, Т. В. Генетические алгоритмы [Текст] : учебно-методическое пособие / под ред. Ю. Ю. Тарасевича. — Астрахань : Издательский дом «Астраханский университет», 2007. — 87 [3] с.

The paper presented the process of solving the problem, documentation on the use of the program, as well as C++ code examples and test results.

СОДЕРЖАНИЕ

	Введение	7
1.	Требования к программе	8
1.1.	Исходные требования к программе*	8
1.2.	Уточнение требований после сдачи прототипа	8
1.3.	Уточнение требований после сдачи 1-ой версии	8
1.4	Уточнение требований после сдачи 2-ой версии	8
2.	Распределение ролей в бригаде	9
3.	Итерация № 1	10
3.1.	План разработки	10
3.1.1.	GUI	10
3.1.2.	Основные модификации	14
3.1.3.	Сторонние библиотеки	16
3.1.4.	Структуры данных	16
3.2.	Результаты по итерации № 1	17
4.	Итерация № 2	18
4.1.	Методы	18
4.1.1.	Описание реализованных модификаций генетического алгоритма	18
4.1.2.	Реализация работы алгоритма	19
4.1.3.	Реализация GUI	25
4.2.	Используемые технологии	28
4.3.	Примеры выполнения алгоритма	29
4.4.	Результаты по итерации № 2	31
5.	Тестирование	32
5.1.	Тестирование графического интерфейса	32
5.2.	Тестирование кода алгоритма	33
	Заключение	34

Список использованных источников 36
Приложение А. Исходный код – только в электронном виде 37

ВВЕДЕНИЕ

Цель практической работы заключается в разработке программы с графическим интерфейсом, основанной на генетических алгоритмах, для решения задачи коммивояжера.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

- Программа должна иметь GUI
- Должна быть возможность задать данные через GUI (CLI), чтение из файла, случайную генерацию.
- Алгоритмы реализуются самостоятельно.
- Настройкой параметров алгоритмов должна производиться пользователем.
- Пошаговая визуализация поиска решения (как меняется аппроксимирующая функция, текущий экстремум, текущее решение оптимизационной задачи). Также должны отображаться 2 следующих лучших решения.
- Должна быть возможность перейти к конечному решению пропустив все шаги.
- Должен присутствовать график изменения функции качества решения с каждым шагом, дополняющийся с каждым шагом.

Доп. баллы даются за:

- Реализацию возможности вернуться на несколько шагов назад.
- Реализацию и возможность выбора модификаций алгоритмов для поиска решения.

Итерация № 3 (06.07)

a. Реализованный GUI с частичной функционалом. Например, присутствует загрузка данных, ввод параметров алгоритма, частичная визуализация.

b. Реализовано хранение данных.

c. Генетический алгоритм частично реализован, реализованы метрики качества.

В отчете описаны используемые технологии. Для GUI описана реализация GUI и как происходит взаимодействие с моделью. Описана реализация структуры данных. Описание реализации ГА.

Итерация № 4 (10.07)

a. GUI полностью функционирует. Реализованы все требования.

b. Генетический алгоритм решает поставленную задачу.

c. Присутствует визуализация работы алгоритма.

В отчете описать реализацию. Привести примеры выполнения алгоритма. Тестирование программы и предоставление корректности работы алгоритма.

Итерация № 5 (13.07)

Итерация финальной сдачи. Если на итерации 4 не было замечаний, то итерация полностью засчитывается. Для данной в отчете итерации описывается пути решения замечаний.

1.2. Уточнение требований после

добавить настройку параметров ГА

2. РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

Разработка алгоритма - Виноградова, Стародубов, Макки.

Реализация графического интерфейса, реализация алгоритма отбора в новую популяцию – Виноградова.

Реализация основных структур данных, реализация алгоритма отбора родителей – Стародубов.

Реализация алгоритма рекомбинации и мутации – Макки.

Распределение ролей представлено на сх. 1.

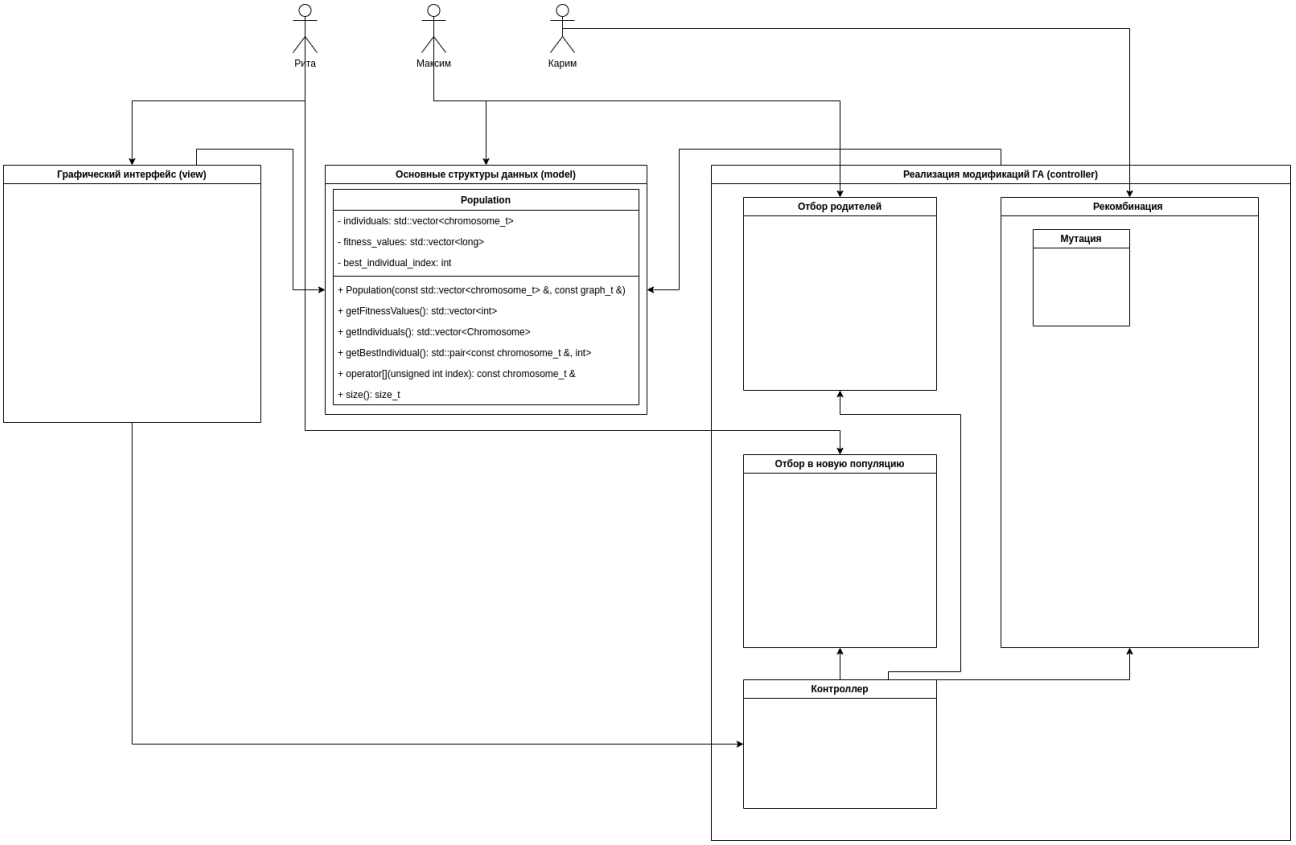


Схема 1 - Распределение ролей

3. Итерация № 1

3.1. План разработки

3.1.1 GUI

Для удобства взаимодействия пользователя с программой был разработан следующий графический интерфейс (рис.1-3)

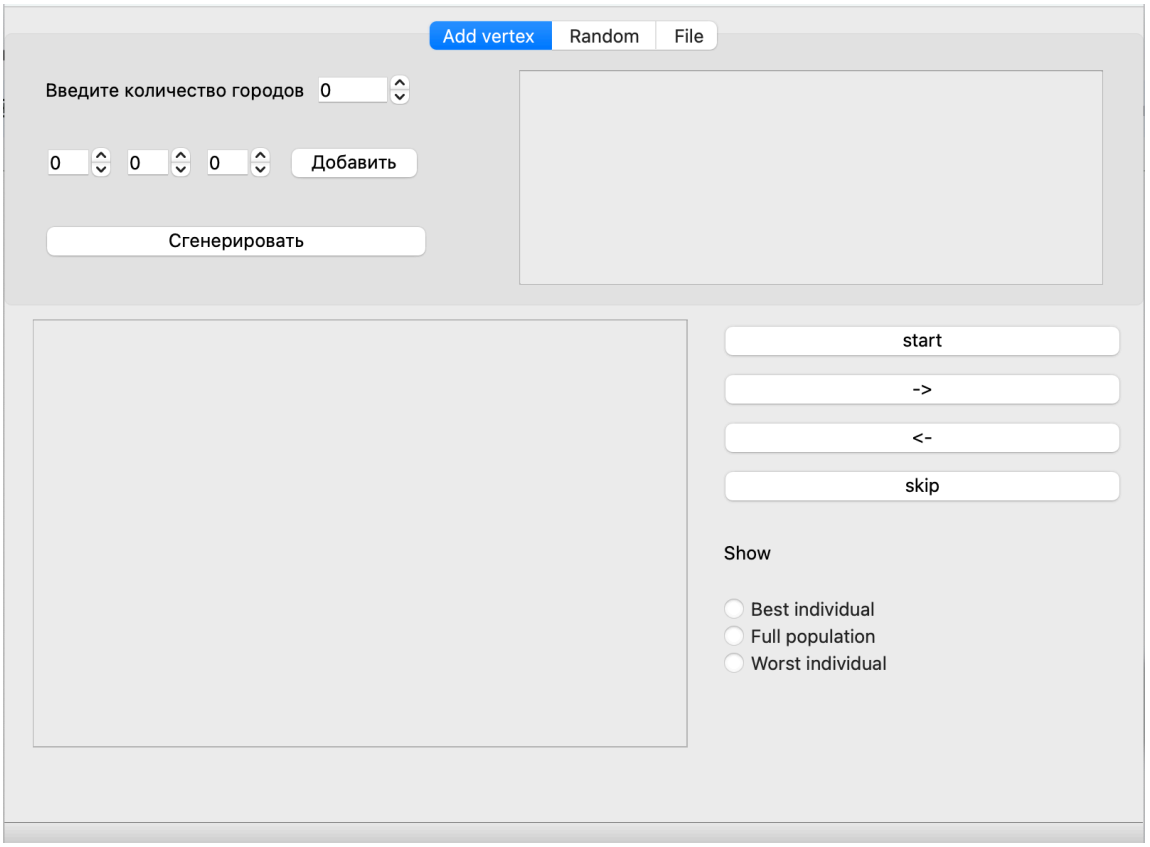


Рисунок 1 - Прототип GUI (вкладка Add vertex)

На вкладке “Add vertex” пользователь может ввести количество городов, а также может указать какие дороги проходят между городами, все добавленные связи будут отображаться в соседней области. После нажатия кнопки “Сгенерировать”, будет создан граф.

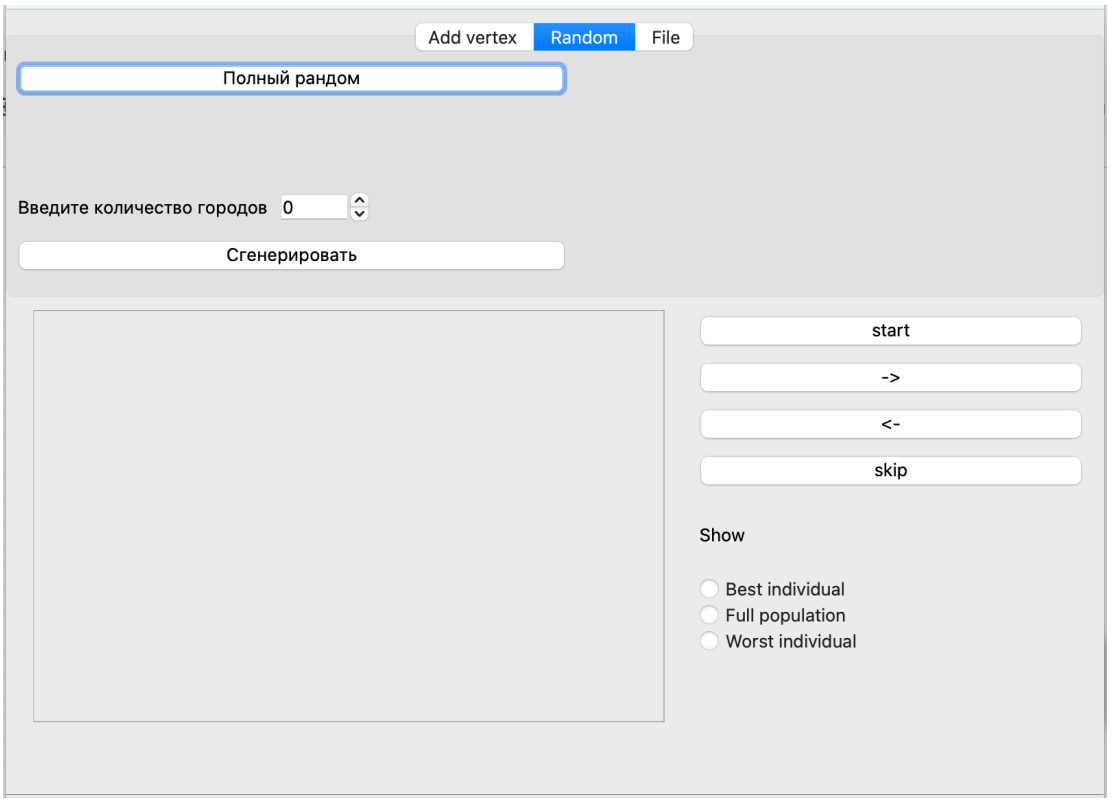


Рисунок 2 - Прототип GUI (вкладка Random)

На вкладке “Random” пользователь может ввести количество городов. После нажатия кнопки “Сгенерировать” будут автоматически сгенерированы “дороги” между городами.

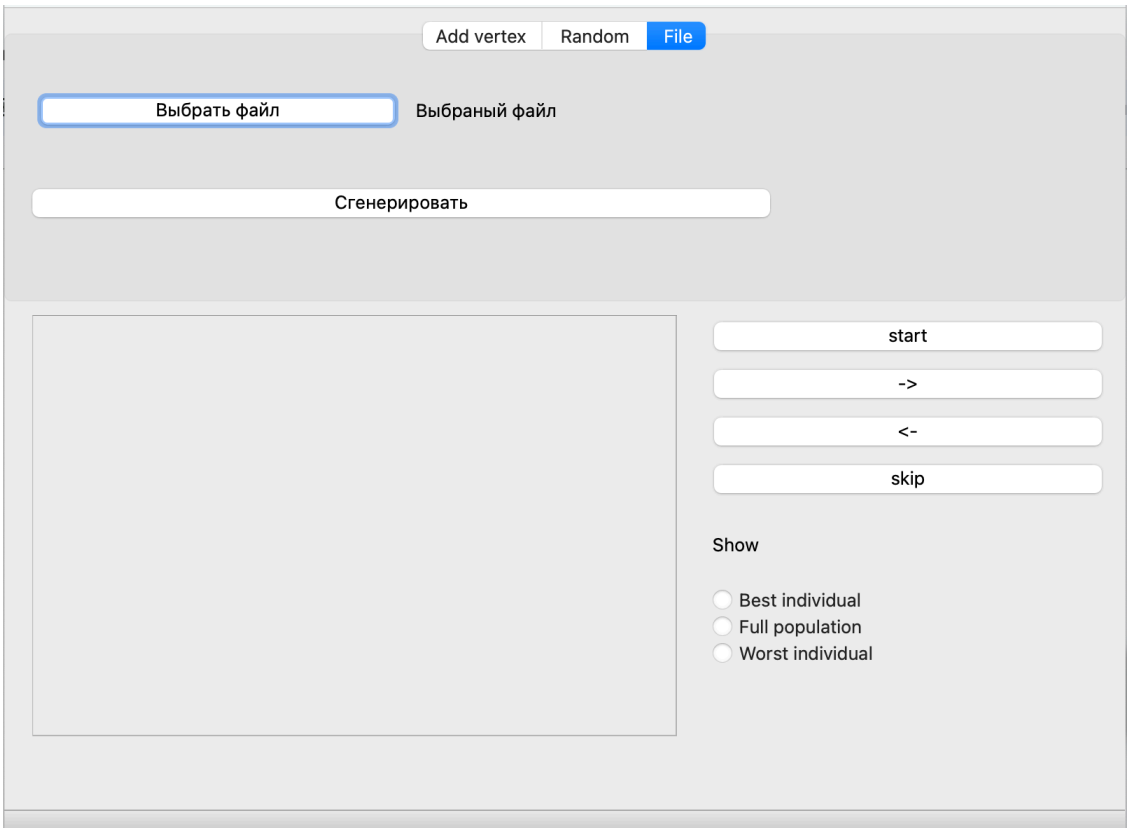


Рисунок 3 - Прототип GUI (вкладка File)

На данной вкладке пользователь может выбрать файл, по содержимому которого будет построен граф. (Требования к файлу: n строк, n столбцов, разделитель пробел).

Также есть область, где будет отображаться, граф/популяция. Справа находятся кнопки управления: start – по входным данными определяет начальную популяцию, -> - шаг вперед, <- – шаг назад, skip – финальное решение. Также можно выбрать данные для отображения: лучшая особь популяции, худшая особь популяции, все особи популяции.

3.1.2 Модификации генетического алгоритма, метрики и настраиваемые параметры

Следующие модификации были выбраны при реализации алгоритма:

1. Инициализация начальной популяции:

Для начала решения задачи коммивояжера случайно выбираются $m \cdot n$ особей, являющихся гамильтоновыми циклами исходного графа, где m – некоторая константа, n – число вершин графа. Каждый цикл представляет собой случайную перестановку вершин графа.

2. Выбор родителей может осуществляться несколькими методами:

Один из которых *панмиксия* – простой оператор отбора, при котором каждой особи в популяции присваивается случайное целое число от 1 до n. Он универсален для различных задач, но менее эффективен с увеличением численности популяции. Вторым методом является *инбридинг*, при котором первый родитель выбирается случайно, а вторым становится член популяции, наиболее близкий к первому. Близость может определяться расстоянием Хемминга (для бинарных строк) или евклидовым расстоянием (для вещественных векторов). *Аутбридинг* же формирует брачные пары из максимально далеких особей.

Селекция в генетическом алгоритме выбирает родителей на основе их приспособленности. Это обеспечивает быструю сходимость, но может привести к преждевременной сходимости к одному решению. Используются разные механизмы отбора, такие как турнирный отбор и метод рулетки. Пороговая величина для селекции может быть вычислена разными способами.

Турнирный отбор выбирает t случайных особей из популяции размером N. Лучшая из них записывается в промежуточный массив, повторяя эту операцию N раз. Особи из промежуточного массива используются для скрещивания. Преимущество турнирного отбора - отсутствие дополнительных вычислений.

В методе рулетки особи отбираются с помощью N «запусков» рулетки, где N – размер популяции. Колесо рулетки содержит по одному сектору для

каждого члена популяции. Размер i-го сектора пропорционален вероятности попадания в новую популяцию $P(i)$, вычисляемой по формуле:

$$P(i) = \frac{f(i)}{\sum_{i=1}^{N} f(i)}$$

В результате анализа данных модификаций мы сделали выбор в пользу метода рулетки, так как в случае панмиксии мы не учитываем наше лучшее решение, а использование инбридинга в нашем случае будет малоэффективным, так как повторяющиеся циклы породят еще один похожий цикл. Реализация аутбридинга требует больших вычислительных затрат.

3. Выбор Рекомбинации:

Из двух методов рекомбинации был выбран метод Кроссинговер, при котором два родителя выбираются для создания потомства. Применяется кроссинговер по одной точке, где случайным образом выбирается точка разреза. Генетический материал от родителей объединяется, и получается потомок. Всего создается $m \cdot n$ потомков.

4. Мутация:

Так как в результате применения кроссинговера полученный цикл может иметь повторяющиеся вершины, необходимо провести корректировку, для этого повторяющиеся вершины в цикле будут заменяться на вершины, отсутствующие в нем. После этого происходит случайная перестановка некоторого количества вершин.

5. Выбор новой популяции:

Новая популяция формируется на основе элитарного признака. Лучшие $m \cdot n$ особей, включая родителей и потомков, добавляются в новую популяцию.

3.1.3 Сторонние библиотеки

Для создания графического интерфейса использовался кроссплатформенный фреймворк Qt. Он предоставляет набор инструментов и библиотек для разработки приложений с красивыми и функциональными интерфейсами, которые могут работать на разных операционных системах. Qt имеет широкий выбор виджетов и элементов управления, а также мощные возможности для обработки событий и работы с сетью и базами данных. Это делает его популярным выбором для разработчиков приложений с графическим интерфейсом.

3.1.4 Структуры данных

Для хранения графа используется матрица смежности, данный способ хранения графа позволяет получить доступ к весу ребра ориентированного графа за константное время.

Для решения задачи с помощью генетического алгоритма необходимо определить основные структуры данных, определяющие особь и популяцию.

Принято решение хранить особь как одну хромосому, представляющую из себя упорядоченный массив целых чисел, каждое число в хромосоме определяется как вершина графа, упорядоченное множество вершин образует цикл.

Популяция определяется как группа особей, для удобства дальнейшей работы с популяцией, в класс добавлены методы, позволяющие узнать значения функции приспособленности для каждой особи, получить информацию об особи с наилучшим значением функции приспособленности.

Для удобного хранения различных поколений популяций определена структура списка популяций, она будет использоваться для отображения предыдущих популяций в графическом интерфейсе.

3.2. Результаты по итерации № 1

В результате первой итерации было произведено распределение участников бригады по разным ролям, в соответствие с которыми будет реализовываться дальнейшая программа. Был разработан прототип графического интерфейса в среде Qt Designer. Также были описаны модификации и метрики генетического алгоритма для поставленной задачи (коммивояжера), структуры данных для хранения. Представлен предполагаемых стек технологий.

4. Итерация №2

4.1. Методы.

4.1.1. Описание реализованных модификаций генетического алгоритма.

Использованы два метода формирования дочерних особей: метод кроссинговера двух случайных особей в популяции с дальнейшей мутацией, выполняемой с помощью обмена местами двух генов в полученной хромосоме, а также метод поворота аллели, выполняющий поворот части генов в хромосоме.

Для отбора особей для кроссинговера и поворота аллелей используется метод рулеточного отбора, позволяющий с большей вероятностью отбирать особей с лучшими значениями функции приспособленности.

Первый метод формирования новых особей позволяет получить более лучшее решение путем объединения частей различных особей, мутация необходима для формирования более разнообразных особей, так как при рулеточном методе отбора одни и те же особи будут отбираться для скрещивания чаще.

Второй метод позволяет более эффективно избегать попаданий в локальные минимумы. В процессе тестирования алгоритма без использования данного метода замечено, что находимые алгоритмом решения отличаются от более оптимальных изменением порядка аллелей в хромосомах, а также разворотом некоторых из этих аллелей. Использование второго метода формирования особей позволяет с большей вероятностью получить лучшее решение при большом числе итераций алгоритма.

С помощью графического интерфейса возможно настроить, какой из данных методов будет использоваться алгоритмом, а также количество мутаций для первого метода.

Отбор особей в новую популяцию происходит с помощью метода элитарного отбора. Данный метод позволяет сохранить лучшее решение, полученное на предыдущей итерации алгоритма, а также игнорирует особи, с

наименьшим значением функции приспособленности. При тестировании алгоритма замечено, что при большом числе итераций алгоритма в популяции формируется множество идентичных особей. Чтобы сохранить разнообразие особей в популяции, новая популяция формируется только из особей с уникальным значением функции качества, если это возможно.

Для используемых модификаций критерием остановки алгоритма является ситуация, когда в течении 75% времени работы алгоритма сохраняется одно и то же решение.

В совокупности, реализованный алгоритм позволяет сохранять разнообразие популяций, а также имеет возможность избегать попаданий в локальные минимумы.

4.1.2. Реализация работы алгоритма.

Класс *Mating*, содержит методы для выполнения скрещивания и мутации хромосом. Описание каждого метода:

1. Конструктор *Mating*(*int chromosome_size*, *int amount_of_crossover_dots*, *double mutation_probability*): Инициализирует объект класса *Mating* с заданными параметрами размера хромосомы (*chromosome_size*), количеством точек скрещивания (*amount_of_crossover_dots*) и вероятностью мутации (*mutation_probability*).

2. Метод *ordered_crossover*(*const chromosome_t*& *first_parent*, *const chromosome_t*& *second_parent*, *int crossover_start*, *int crossover_end*): Выполняет упорядоченное скрещивание двух родительских хромосом (*first_parent* и *second_parent*) в заданном диапазоне точек скрещивания (*crossover_start* и *crossover_end*). Возвращает потомка (новую хромосому) в результате скрещивания.

3. Метод *mutation*(*chromosome_t*& *chromosome, int mutation_rate)*: Выполняет мутацию хромосомы (*chromosome*) путем случайного обмена генами *mutation_rate* раз. Мутация меняет порядок генов внутри хромосомы.

4. Метод *mutationSwitch(chromosome_t& chromosome, int mutation_rate)*: Выполняет мутацию хромосомы (*chromosome*) путем разворота сегмента генов *mutation_rate* раз. Мутация меняет порядок генов внутри хромосомы.

5. Метод *getChildren(const Population& population)*: Получает потомков путем выполнения скрещивания и мутации выбранных родителей из популяции (*population*). Использует метод рулеточного выбора (*RouletteWheelSelection*) для выбора родителей. Возвращает вектор потомков.

6. Метод *getMutated(const Population& population)*: Получает мутированные особи из популяции (*population*) путем выполнения одной мутации на каждой хромосоме. Использует метод рулеточного выбора (*RouletteWheelSelection*) для выбора хромосомы для мутации. Возвращает вектор мутированных особей.

Класс *Algorithm*, реализует генетический алгоритм. Описание каждого метода:

1. *Algorithm(const graph_t &graph, Settings settings):* Конструктор класса, принимает в качестве аргументов граф *graph* и настройки *settings,* производит инициализацию параметров алгоритма.

2. *generateFirstPopulation()*: Метод, создающий первую популяцию для запуска алгоритма. В этом методе происходит инициализация случайных особей.

3. *getCurrentPopulation() const*: Метод, возвращающий ссылку на текущую популяцию.

4. *switchToNextPopulation()*: Метод, переключающийся на следующую популяцию. Если следующая популяция существует, метод переключает текущую популяцию на нее и возвращает 0. Если алгоритм завершил свою работу и не может создать следующую популяцию, метод возвращает 1.

5. *switchToPreviousPopulation()*: Метод, переключающийся на предыдущую популяцию. Если текущая популяция - первая в списке, метод возвращает 1.

6. *switchToLastPopulation()*: Метод, запускающий алгоритм до выполнения условия остановки и переключающий текущую популяцию на последнюю полученную.

7. *generateNextPopulation()*: Метод, создающий следующую популяцию на основе последней популяции. В этом методе происходит вызов методов, производящих скрещивание и мутацию особей, а также отбор лучших особей для следующей популяции. Полученная популяция добавляется в историю популяций, и, если размер истории превышает заданное значение, популяция в начале списка удаляется. Кроме того, в методе проверяется условие остановки алгоритма на основании количества одинаковых лучших решений.

Класс *Population* представляет популяцию особей в генетическом алгоритме. Он содержит методы для работы с этой популяцией. Вот описание каждого метода:

1. *Population(const std::vector<chromosome_t> &individuals, const graph_t &graph):* Принимает вектор особей *individuals* и граф *graph*. Конструктор инициализирует все поля класса и вычисляет длину цикла для каждой особи, чтобы определить значение функции приспособленности.

2. *const std::vector<long> &getFitnessValues() const:* Возвращает вектор значений функции приспособленности для всех особей.

3. *const std::vector<chromosome_t> &getIndividuals() const*: Возвращает вектор хромосом (особей).

4. *std::pair<const chromosome_t &, int> getBestIndividual() const:* Возвращает ссылку на хромосому и значение функции приспособленности наилучшей особи в популяции.

5. *std::pair<const chromosome_t &, int> getWorstIndividual() const*: Возвращает ссылку на хромосому и значение функции приспособленности наихудшей особи в популяции.

6. *const chromosome_t &operator[](unsigned int index) const*: Возвращает хромосому особи с заданным индексом.

7. *std::size_t size() const*: Возвращает количество особей в популяции.

Класс *NewGenerationSelection* реализует метод createNewGeneration, который принимает два аргумента типа *Population*: *ancestors* (предыдущее поколение) и *descendants* (потомки). Метод выбирает лучших особей из предыдущего поколения и потомков для формирования нового поколения.

Основные шаги метода *createNewGeneration*:

1. Создается вектор *all_individuals*, который будет содержать пары из особей (*chromosome_t*) и значений их функции приспособленности (*long*).

2. Резервируется достаточное место в векторе *all_individuals* для хранения особей из предыдущего поколения и потомков.

3. Особи из предыдущего поколения добавляются в *all_individuals* вместе со значениями их функции приспособленности.

4. Особи из потомков также добавляются в *all_individuals* вместе со значениями их функции приспособленности.

5. Определяется компаратор *compare*, который сравнивает пары особей по значениям их функции приспособленности.

6. Вектор *all_individuals* сортируется в порядке возрастания фитнес-значений с использованием компаратора *compare*.

7. Создается вектор *new_population*, который будет содержать особи нового поколения.

8. Резервируется достаточное место в векторе *new_population* для хранения особей нового поколения.

9. Первая особь из *all_individuals* (с наименьшим значением функции приспособленности) добавляется в *new_population*.

10. Инициализируются переменные *previous_added_index* со значением 0 и *next_index* со значением 1.

11. В цикле выполняются следующие действия до тех пор, пока размер *new_population* не достигнет размера предыдущего поколения:

- Проверяется, отличается ли значение функции приспособленности предыдущей добавленной особи от значение функции приспособленности следующей особи из *all_individuals*.
- Если значения различаются, то следующая особь добавляется в *new_population*, и *previous_added_index* устанавливается равным *next_index*.
- Иначе *next_index* увеличивается на 1.

12. Метод возвращает вектор *new_population*, содержащий лучшие особи для нового поколения.

Класс *RouletteWheelSelection*, реализует метод выбора особи из популяции с использованием метода "рулеточного колеса" (*Roulette Wheel Selection*).

Метод *RouletteWheelSelection(const Population &population)* – конструктор класса *RouletteWheelSelection*. Производит инициализацию вектора вероятностей (*probabilities*) на основе значений функции приспособленности (*fitness*) популяции. Для этого он выполняет следующие шаги:

1. Резервирует память для вектора вероятностей (*probabilities*) на основе размера популяции.

2. Получает значения приспособленности популяции с помощью метода *getFitnessValues()* класса *Population*.

3. Инициализирует сумму обратных значений приспособленности (*reverse_fitness_values_sum*) равной 0.

4. Вычисляет обратные значения приспособленности и накапливает их сумму:

- Для каждой особи в популяции:
- Добавляет текущую сумму обратных значений приспособленности в вектор вероятностей (*probabilities*).

- Вычисляет обратное значение приспособленности для текущего индивида и добавляет его к сумме *reverse_fitness_values_sum*.

5. Нормализует вероятности, разделив каждое значение вектора вероятностей на сумму обратных значений приспособленности.

Метод *getIndividual()* возвращает выбранную особь из популяции, используя метод рулеточного колеса. Выполняет следующие шаги:

1. Создает объект *Randomizer* для генерации случайных чисел.

2. Генерирует случайное вещественное число *random_number* в диапазоне от 0 до 1.

3. Инициализирует левую и правую границы (*left_bound* и *right_bound*) для выполнения бинарного поиска в векторе вероятностей (*probabilities*).

4. Выполняет бинарный поиск для нахождения особи в популяции:

- Пока разница между правой и левой границей больше 1:

- Вычисляет середину (*middle*) между правой и левой границей.

- Если случайное число (*random_number*) больше вероятности в середине (*probabilities[middle]*), обновляет левую границу.

- В противном случае обновляет правую границу.

5. Возвращает особь, найденную с помощью бинарного поиска случайного числа в векторе вероятностей. Особь выбирается на основе рулеточного колеса, где вероятности выбора пропорциональны значениям вероятностей вектора *probabilities*.

Класс *Randomizer* содержит два метода для генерации случайных чисел. Каждый метод принимает диапазон значений и возвращает случайное число в этом диапазоне.

Метод *getRandomInt* генерирует случайное целое число в заданном диапазоне. Он использует следующие шаги:

1. Инициализирует *random_device* для получения случайного семени.

2. Создает генератор случайных чисел *mt19937* с помощью семени из *random_device*.

3. Создает равномерное распределение целых чисел *uniform_int_distribution* с заданным диапазоном.

4. Генерирует случайное целое число в заданном диапазоне с помощью распределения и генератора случайных чисел.

5. Возвращает сгенерированное случайное число.

Метод *getRandomDouble* генерирует случайное вещественное число в заданном диапазоне. Данный метод выполняет аналогичные шаги:

1. Инициализирует *random_device* для получения случайного семени.

2. Создает генератор случайных чисел *mt19937* с помощью семени из *random_device*.

3. Создает равномерное распределение вещественных чисел *uniform_real_distribution* с заданным диапазоном.

4. Генерирует случайное вещественное число в заданном диапазоне с помощью распределения и генератора случайных чисел.

5. Возвращает сгенерированное случайное число.

4.1.3. Реализация графического интерфейса.

Класс MainWindow представляет графический интерфейс

Его слоты(функции для взаимодействия с элементами gui):

void *onPushButtonAddEdgeClicked():* обрабатывает нажатие кнопки «Добавить», вызывает проверку через контроллер и в зависимости от полученной информации изменяет вывод информации на экран

void *onFileChoose*(): обрабатывает нажатие кнопки “Выбрать файл”, сохраняет путь до файла и выводит его на экран.

void *onFullRandomClicked*(): обрабатывает нажатие на кнопку “Полный рандом”, вызывает соответствующий метод инициализации графа у контроллера ui.

void *onGraphGenerateAdd*(): обрабатывает нажатие на кнопку “Сгенерировать” на вкладке Add vertex, вызывает соответствующий метод инициализации графа у контроллера ui.

void *onGraphGenerateFile*(): обрабатывает нажатие на кнопку “Сгенерировать” на вкладке File, вызывает соответствующий метод инициализации графа у контроллера ui.

void *onGraphGenerateRandom*(): обрабатывает нажатие на кнопку “Сгенерировать” на вкладке Random, вызывает соответствующий метод инициализации графа у контроллера ui.

void *radioButtonClicked*(): обрабатывает нажатие на radioButton в нижней части экрана (выбор отображение данных), вызывает изменение вывода на экране, в соответствие с выбраным вариантом.

void *onMutationRadioButtonClicked*(): обрабатывает нажатие на radioButton на вкладке настроек, при изменении обновляет настройки в контроллере ui.

void *onSaveClicked*(): обрабатывает нажатие на кнопку Save, обновляет настройки контроллера ui.

void *onStartClicked*(): обрабатывает нажатие на кнопку “start”. Создает контроллер алгоритма, передавая ему граф из контроллера ui.

void *onNextClicked*(): обрабатывает нажатие на кнопку далее, вызывает соответствующий метод у контроллера алгоритма.

void *onPrevClicked*(): обрабатывает нажатие на кнопку предыдущий, вызывает соответствующий метод у контроллера алгоритма.

void *onSkipClicked*(): обрабатывает нажатие на кнопку пропустить, вызывает соответствующий метод у контроллера алгоритма.

void *onPlotButtonClicked*(): обрабатывает нажатие на кнопку “График”, создает новое окно и передает данные о лучших особях.

После генерации графа, он отображается на экране. При работе алгоритма также выводятся актуальные данные.

Методы:

void *initConnectTabFile*(), void *initConnectTabAddVertex*(), void *initConnectTabRandom*(), void *initConnectBlockControl*(), void *initConnectRadioButton*(): методы в который устанавливается связь между сигналами и слотами.

void *drawGraph*(): метод предназначен для вывода на экран графа. Сначала обновляется холст, далее создаются ребра и рисуются вершины.

void *drawIndividual*(): метод предназначен для вывода на экран одной особи (в зависимости от выбора пользователя отображается лучшая или худшая особь).

void *drawPopulation*(): метод предназначен для вывода на экран списка популяции (вывод в виде текста).

void *chooseWhatToDraw*(): метод определяет, что будет выводиться на экран в соответствие с режимом выбранным пользователем.

void *drawPoints*(int n): рисует вершины правильного n-угольника.

Class Dialog - всплывающее окно при вызове графика изменения оптимального решения.

Методы:

void *drawPlot(const std::vector<int> elements*) - метод предназначен для рисования графика по полученному списку решений.

Class ContollerUI - контроллер графического интерфейса.

Методы:

bool *initFile(std::string sourceFile)*: метод для инициализации графа по переданному файлу, считывает данные из файла и заполняет ими сам граф.

bool *initAdd(std::vector<std::pair<int, std::pair<int, int>>> list_edge, int n)*: метод для инициализации графа по переданным ребрам (инцидентные вершины и вес).

bool *initRandom(int n = 0)*: если передано значение 0, то размер графа выбирается случайно. Сам граф также заполняется случайными значениями.

const *graph_t & getGraph()*: метод который возвращает ссылку на граф.

void *initMax(int n)*: метод который заполняет граф размера n на n максимальными значениями.

При инициализации граф проверяется на корректность.

Class CheckData - класс предназначен для проверки корректности данных.

bool checkGraph(graph_t& graph): метод предназначен для проверки графа на корректность (симметричность относительно главной диагонали, на главной диагонали значения INT_MAX).

int addEdge(std::vector<std::pair<int, std::pair<int, int>>>& list_edge, int distance, int departure, int arrival, int n): метод предназначен для проверки добавленного ребра на корректность вершин, если ребра уже есть, то данные обновляются.

struct Settings - структура для хранения пользовательских настроек храниться количество точек кроссинговера, частота мутации, множитель количества особей в популяции.

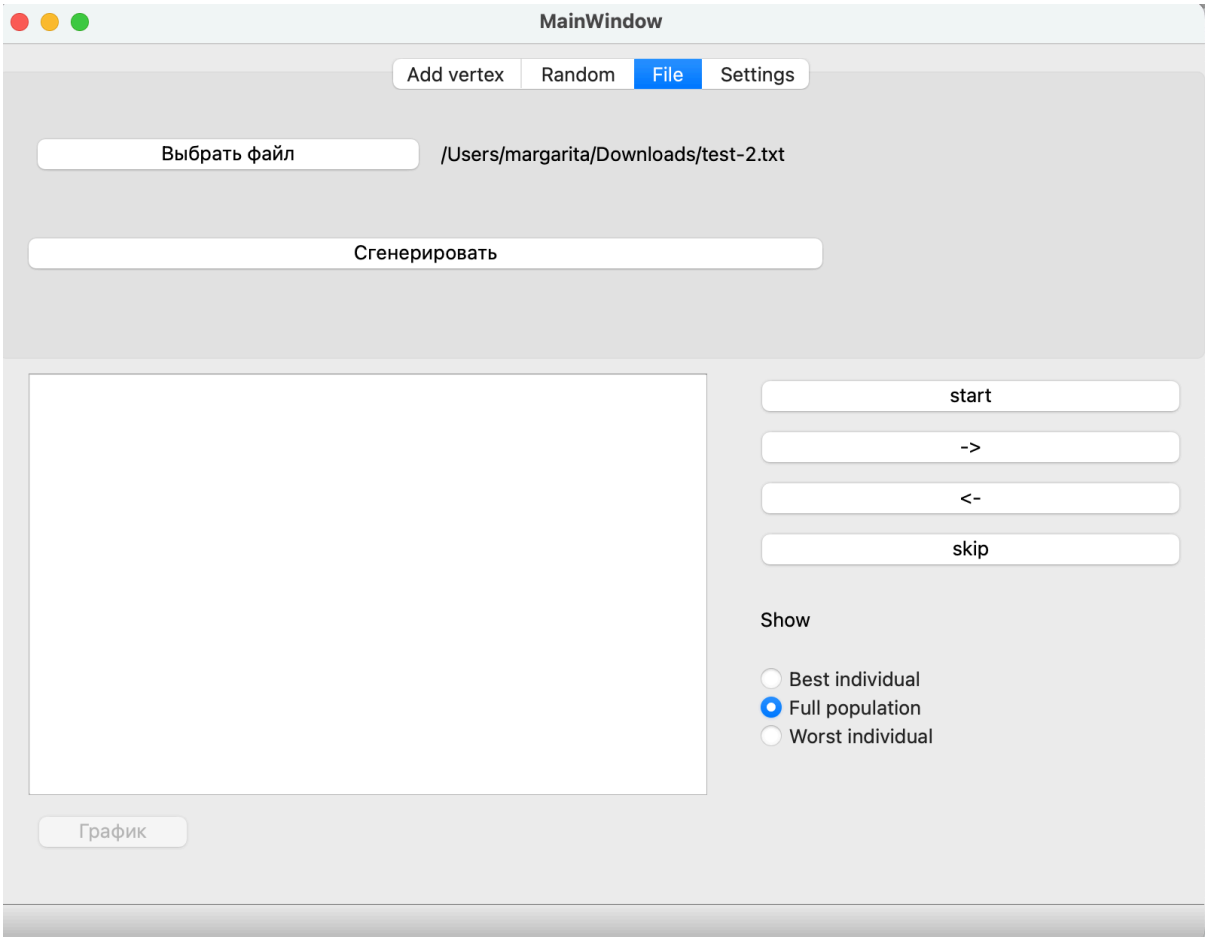
4.2. Используемые технологии.

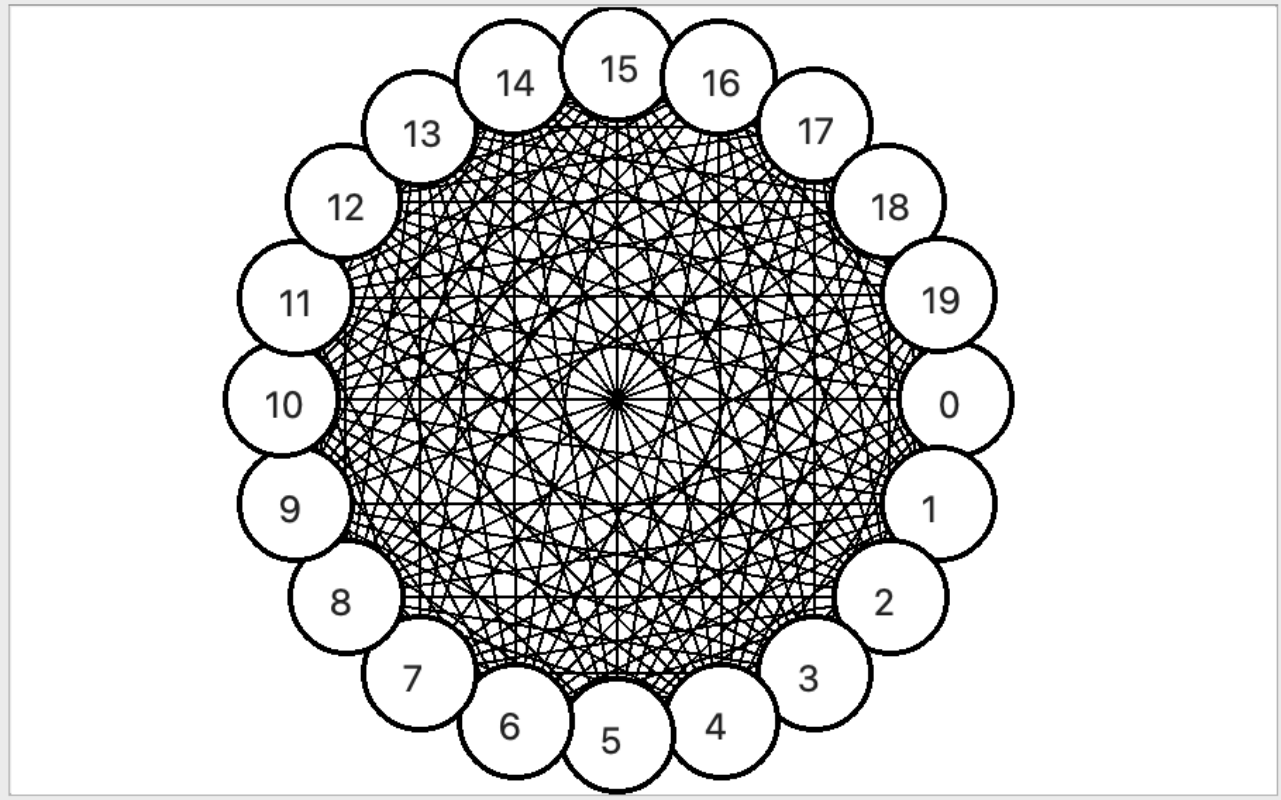
Для реализации графического интерфейса используется кроссплатформенный фреймворк *Qt*, предоставляющий возможность удобного проектирования графического интерфейса.

Дополнительно, для отображения графика изменения лучшего решения с возрастанием числа поколений популяций, использована сторонняя библиотека *QCustomPlot*.

4.3. Примеры выполнения алгоритма

Инициализация через файл:





1) 9 -> 13 -> 8 -> 14 -> 7 -> 3 -> 18 -> 19 -> 4 -> 17 -> 15 -> 10 -> 5 -> 1 -

2) 3 -> 19 -> 17 -> 2 -> 7 -> 18 -> 13 -> 1 -> 5 -> 16 -> 15 -> 6 -> 11 -> 4 -

3) 7 -> 19 -> 11 -> 3 -> 2 -> 18 -> 6 -> 16 -> 0 -> 4 -> 17 -> 15 -> 13 -> 9 -

4) 16 -> 11 -> 17 -> 14 -> 2 -> 15 -> 9 -> 3 -> 10 -> 7 -> 12 -> 1 -> 19 -> 18

5) 9 -> 19 -> 0 -> 5 -> 12 -> 14 -> 18 -> 2 -> 11 -> 16 -> 6 -> 13 -> 3 -> 17

6) 15 -> 3 -> 9 -> 19 -> 11 -> 0 -> 8 -> 12 -> 14 -> 6 -> 13 -> 1 -> 18 -> 5 -

7) 10 -> 14 -> 2 -> 1 -> 9 -> 19 -> 11 -> 16 -> 17 -> 6 -> 8 -> 0 -> 3 -> 4 ->

8) 14 -> 13 -> 18 -> 3 -> 12 -> 0 -> 5 -> 10 -> 17 -> 15 -> 9 -> 7 -> 1 -> 16

9) 14 -> 8 -> 6 -> 11 -> 18 -> 7 -> 15 -> 0 -> 19 -> 2 -> 1 -> 16 -> 12 -> 10

start

->

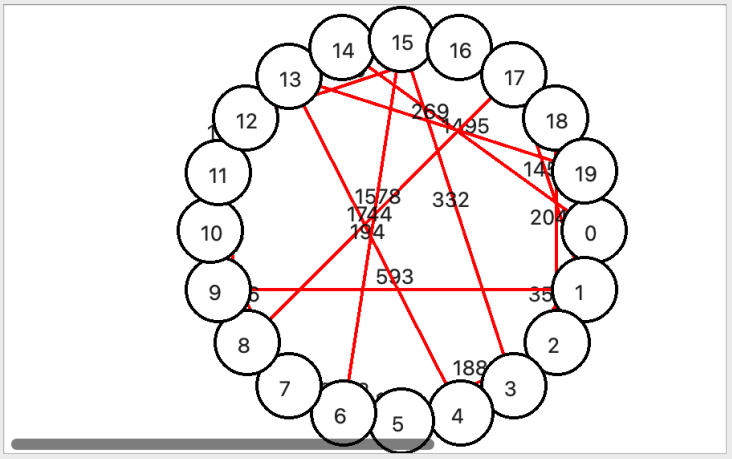
<-

skip

Show

- Best individual
- Full population
- Worst individual

График



start

->

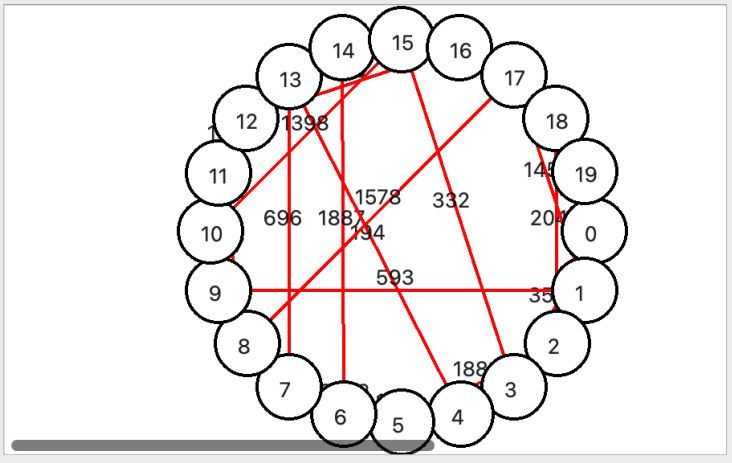
<-

skip

Show

- Best individual
- Full population
- Worst individual

График



start

->

<-

skip

Show

- Best individual
- Full population
- Worst individual

График

4.4. Результаты по итерации № 2

Итерация № 3 (06.07):

В результате третьей итерации были достигнуты следующие результаты. Был реализован графический пользовательский интерфейс (GUI) с частичным функционалом, включающим возможность загрузки данных, ввода параметров алгоритма и частичной визуализации. Также было успешно реализовано хранение данных. Отчет по данной итерации содержит описание используемых технологий. Для GUI представлена реализация и описано взаимодействие с моделью. Также описана реализация структуры данных и генетического алгоритма, включая модификации и метрики, применяемые для поставленной задачи.

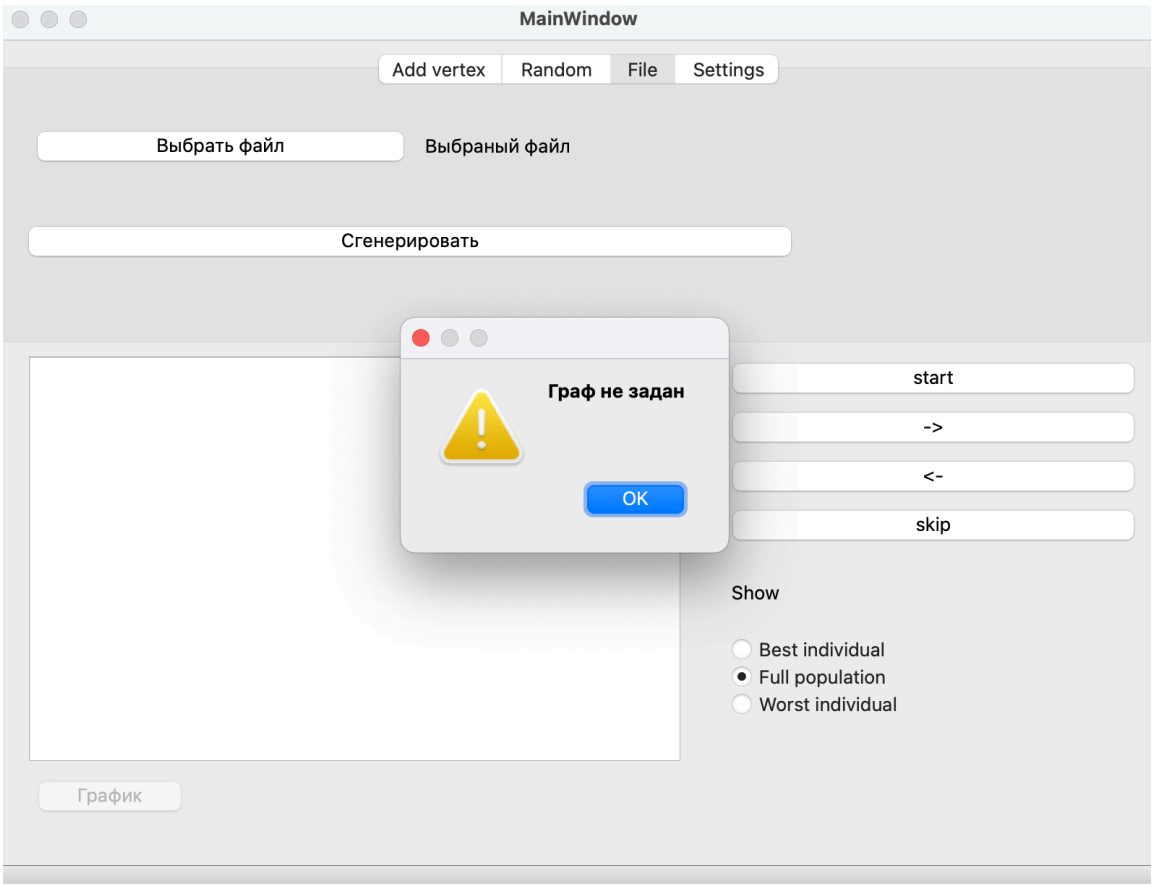
Итерация № 4 - 5 (10.07 - 13.07):

В результате четвертой итерации были достигнуты следующие результаты. Графический пользовательский интерфейс (GUI) полностью функционирует, все требования были успешно реализованы. Генетический алгоритм успешно решает поставленную задачу, и визуализация работы алгоритма присутствует. Отчет содержит детальное описание реализации, включая примеры выполнения алгоритма. Также проведено тестирование программы для проверки корректности работы алгоритма.

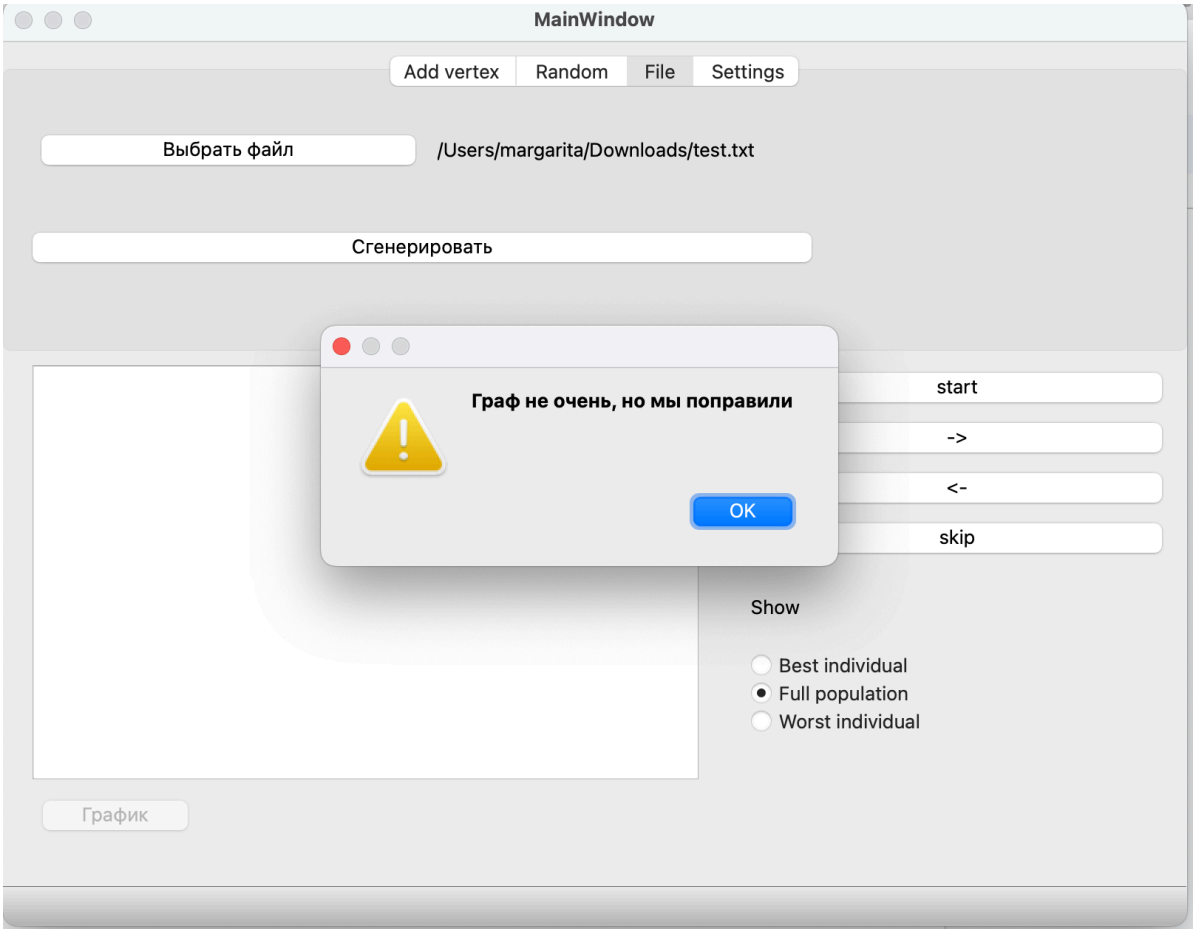
5. ТЕСТИРОВАНИЕ

5.1. Тестирование графического интерфейса

Попытка начать алгоритм без заданного графа:



Пример получения некорректного графа:



5.2. Тестирование кода алгоритма

Таблица тестирования

№	файл	кол-во вершин	время нахождения точного решения, с	точное решение	число итераций ГА	решение ГА
1	test1	5	0.000108272	15733	12	15733
2	test2	10	0.0324929	15414	85	15414
3	test3	15	3.42378	17904	1275	17904
4	test4	20	950.9	27779	6700	27785

Тестовые данные в приложении А.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была успешно выполнена реализация генетического алгоритма для решения задачи коммивояжера. Генетический алгоритм является эффективным методом оптимизации, основанным на принципах эволюции и генетики. Он позволяет находить приближенное или оптимальное решение задачи коммивояжера, которая включает в себя нахождение кратчайшего пути для посещения набора городов и возврата в исходный город.

Результаты работы генетического алгоритма в данном проекте были успешно визуализированы и представлены в графическом пользовательском интерфейсе (GUI), который обеспечивает удобное взаимодействие пользователя с программой. GUI был разработан с использованием среды Qt Designer, что позволило создать функциональный интерфейс с возможностью загрузки данных и ввода параметров алгоритма.

Были применены различные модификации генетического алгоритма, а также определены и реализованы метрики качества для оценки работы алгоритма. Это позволило получить оптимальные или близкие к оптимальным решения для задачи коммивояжера, основываясь на принципах отбора, скрещивания, мутации и элитарности.

В процессе работы были использованы современные технологии и инструменты, такие как Qt Framework для разработки GUI, а также язык программирования C++ для реализации генетического алгоритма и обработки данных.

Тестирование программы позволило убедиться в корректности работы генетического алгоритма и подтвердить его способность находить оптимальные или приближенные решения для задачи коммивояжера.

В целом, выполнение данной работы по решению задачи коммивояжера с использованием генетического алгоритма было успешным. Разработанный программный продукт обладает функциональностью, позволяющей находить

приближенные или оптимальные решения задачи коммивояжера и визуализировать результаты. Представленный в отчете алгоритм и его реализация могут быть использованы в различных сферах, требующих оптимизации маршрутов и планирования путешаний.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Панченко, Т. В. Генетические алгоритмы [Текст] : учебно-методическое пособие / под ред. Ю. Ю. Тарасевича. — Астрахань : Издательский дом «Астраханский университет», 2007. — 87 [3] с.
2. https://www.geeksforgeeks.org/traveling-salesman-problem-using-genetic-algorithm/
3. https://jaketae.github.io/study/genetic-algorithm/
4. https://doc.qt.io/
5. https://www.qcustomplot.com/index.php/tutorials/basicplotting

ПРИЛОЖЕНИЕ А
ДАННЫЕ ДЛЯ ТЕСТИРОВАНИЯ

test1:

2147483647 5461 7416 310 4151

5461 2147483647 3925 6407 2147483647

7416 3925 2147483647 6981 940

310 6407 6981 2147483647 5149

4151 2147483647 940 5149 2147483647

test2:

2147483647 779 8519 6968 6665 8576 1854 1024 7701 9778

779 2147483647 2807 6023 889 742 8158 1616 4140 392

8519 2807 2147483647 2350 4421 1538 6826 2396 7455 6931

6968 6023 2350 2147483647 3202 5821 8 2614 5702 8651

6665 889 4421 3202 2147483647 2147483647 9773 2147483647 2068 7456

8576 742 1538 5821 2147483647 2147483647 2147483647 4652 8293 5090

1854 8158 6826 8 9773 2147483647 2147483647 4595 1404 2138

1024 1616 2396 2614 2147483647 4652 4595 2147483647 201 6217

7701 4140 7455 5702 2068 8293 1404 201 2147483647 8659

9778 392 6931 8651 7456 5090 2138 6217 8659 2147483647

test3:

2147483647 7773 4969 254 6490 1421 7590 9982 1607 2548 9602 1870 308 1002 9047

7773 2147483647 5147 9324 9869 6466 4905 1991 3877 4770 2147483647 1558 7013 2147483647 5631

4969 5147 2147483647 6785 8706 4934 8289 5097 2147483647 5354 9061 4485 4145 7157 1330

254 9324 6785 2147483647 562 7486 3588 6925 5518 8977 7891 2760 2147483647 2542 139

6490 9869 8706 562 2147483647 9721 954 1888 7521 6197 2445 2290 7353 8256 7201

1421 6466 4934 7486 9721 2147483647 880 7179 1173 27 296 7308 2330 2993 2147483647

7590 4905 8289 3588 954 880 2147483647 2147483647 513 6428 1448 936 9928 5491 4215

9982 1991 5097 6925 1888 7179 2147483647 2147483647 1152 2013 7956 2608 1579 5828 5256

1607 3877 2147483647 5518 7521 1173 513 1152 2147483647 2633 4561 1119 22 2799 5422

2548 4770 5354 8977 6197 27 6428 2013 2633 2147483647 2147483647 6905 2734 4894 8562

9602 2147483647 9061 7891 2445 296 1448 7956 4561 2147483647 2147483647 4017 3063 2704 6331

1870 1558 4485 2760 2290 7308 936 2608 1119 6905 4017 2147483647 9686 3995 2147483647

308 7013 4145 2147483647 7353 2330 9928 1579 22 2734 3063 9686 2147483647 1616 7355

1002 2147483647 7157 2542 8256 2993 5491 5828 2799 4894 2704 3995 1616 2147483647 9328

9047 5631 1330 139 7201 2147483647 4215 5256 5422 8562 6331 2147483647 7355 9328 2147483647

test4:

2147483647 5182 716 485 9621 3312 6740 7342 7257 7880 3559 8101 7151 7163 261 2147483647 7786 3430 2147483647 583
5182 2147483647 4790 6630 4873 2147483647 54 5750 443 9264 8026 585 5458 6866 9162 9958 7798 6226 9123 928
716 4790 2147483647 5861 8863 7322 2816 7211 7235 8097 1926 2022 1616 4493 5442 1839 5581 7082 856 1805
485 6630 5861 2147483647 2276 6629 2147483647 9398 1546 4489 2614 1652 8866 5514 6905 2351 5777 7890 7200 2147483647
9621 4873 8863 2276 2147483647 2504 2147483647 6150 7526 4435 1724 4778 217 3610 2069 1493 1142 2937 4177 3572
3312 2147483647 7322 6629 2504 2147483647 7455 9667 7036 5377 5348 5581 5829 6797 4484 2821 6909 9498 5409 5500
6740 54 2816 2147483647 2147483647 7455 2147483647 7969 129 6391 8747 4888 8257 2024 3818 8087 840 1970 5513 151
7342 5750 7211 9398 6150 9667 7969 2147483647 4219 9748 2191 2067 8461 2647 2147483647 4779 6358 2956 6986 2112
7257 443 7235 1546 7526 7036 129 4219 2147483647 9699 1965 9555 3513 3985 3474 8638 6969 7132 6096 3171
7880 9264 8097 4489 4435 5377 6391 9748 9699 2147483647 4056 6562 6190 8134 2147483647 1953 5359 6081 3040 6757
3559 8026 1926 2614 1724 5348 8747 2191 1965 4056 2147483647 2315 2147483647 2334 6382 3652 2147483647 9322 7381 1947
8101 585 2022 1652 4778 5581 4888 2067 9555 6562 2315 2147483647 3758 3376 7089 4512 1438 3636 3993 5357
7151 5458 1616 8866 217 5829 8257 8461 3513 6190 2147483647 3758 2147483647 4154 645 2545 7445 9061 155 4967
7163 6866 4493 5514 3610 6797 2024 2647 3985 8134 2334 3376 4154 2147483647 6158 249 5733 8272 5266 7985
261 9162 5442 6905 2069 4484 3818 2147483647 3474 2147483647 6382 7089 645 6158 2147483647 8287 9965 1439 5311 3448
2147483647 9958 1839 2351 1493 2821 8087 4779 8638 1953 3652 4512 2545 249 8287 2147483647 6357 5915 3152 447
7786 7798 5581 5777 1142 6909 840 6358 6969 5359 2147483647 1438 7445 5733 9965 6357 2147483647 5198 9305 9269
3430 6226 7082 7890 2937 9498 1970 2956 7132 6081 9322 3636 9061 8272 1439 5915 5198 2147483647 8724 9334
2147483647 9123 856 7200 4177 5409 5513 6986 6096 3040 7381 3993 155 5266 5311 3152 9305 8724 2147483647 3363
583 928 1805 2147483647 3572 5500 151 2112 3171 6757 1947 5357 4967 7985 3448 447 9269 9334 3363 2147483647