МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по учебной практике

Тема: «Задача поиска МОД»

Студенты гр. 2304 Пашков Г.М.,
Жихарев В.В.,
Ишустин О.В.

Преподаватель _____ Жангиров Т.М.

Санкт-Петербург
2024

Цель работы.

Целью работы является написание программы, удовлетворяющей условию учебной практики, она должна иметь GUI и выполнять поставленную задачу.

Задание.

Задача поиска МОД

Дан взвешенный неориентированный граф (ребра имеют вес больше 0). Необходимо найти минимальное остовное дерево для данного графа.

Требование к решению:

- Программа должна иметь GUI
- Должна быть возможность задать данные через GUI/чтение из файла/случайную генерацию (выбор отдается пользователю)
- Алгоритмы реализуются самостоятельно.
- Настройкой параметров алгоритмов должна производиться пользователем.
- Пошаговая визуализация поиска решения (как меняется аппроксимирующая функция, текущий экстремум, текущее решение оптимизационной задачи в зависимости от популяции). Также должны отображаться 3 наилучших решения.
- Должна быть возможность перейти к конечному решению пропустив отображение всех шагов.
- 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.

Распределение ролей в команде.

- Пашков Г.М. – написание отчета, разработка GUI
- Жихарев В.В. – реализация алгоритма
- Ишустин О.В. – реализация GUI

Описание алгоритма.

Алгоритм поиска минимального остовного дерева (МОД) в взвешенном неориентированном графе может быть выполнен с использованием двух популярных методов: алгоритма Краскала и алгоритма Прима. Оба алгоритма эффективно решают задачу, но применяются в зависимости от структуры графа и предпочтений.

Алгоритм Краскала

1. Инициализация:
    - Сформируйте список всех рёбер графа.
    - Отсортируйте рёбра по весу в порядке возрастания.
    - Создайте отдельные множества для каждой вершины (для отслеживания соединений).
2. Основной процесс:
    - Начните с пустого множества рёбер для МОД.
    - Последовательно перебирайте рёбра в отсортированном порядке.
    - Для каждого ребра проверяйте, соединяют ли его концы разные множества (проверка на цикл):
        - Если рёбра принадлежат разным множествам, добавьте это ребро в МОД и объедините множества.
        - Если рёбра принадлежат одному множеству, пропустите его (оно образует цикл).
3. Завершение:
    - Продолжайте процесс, пока не будет добавлено (V-1) рёбер (где V — количество вершин).

- Полученное множество рёбер — это минимальное остовное дерево.

Плюсы и минусы:

- Плюсы: Эффективен для графов с множеством рёбер.
- Минусы: требуется сортировка рёбер, что может быть затратным для очень больших графов.

Алгоритм Прима

1. Инициализация:
   - Выберите произвольную начальную вершину и добавьте её в множество остовного дерева.
   - Инициализируйте массив минимальных расстояний для всех вершин от начальной вершины.
2. Основной процесс:
   - Повторяйте следующие шаги, пока все вершины не будут включены в остовное дерево:
     - Выберите вершину с минимальным значением расстояния, ещё не включённую в остовное дерево.
     - Добавьте выбранную вершину и соответствующее ребро в МОД.
     - Обновите значения расстояний для всех соседей новой вершины, если ребро имеет меньший вес, чем ранее записанное расстояние.
3. Завершение:
   - Процесс продолжается, пока не будут включены все вершины графа.

- Полученное множество рёбер формирует минимальное остовное дерево.

Плюсы и минусы:

- Плюсы: Эффективен для графов с плотными связями.
- Минусы: может быть менее эффективен для графов с большим количеством рёбер по сравнению с алгоритмом Краскала.

Заключение

Оба алгоритма обеспечивают нахождение минимального остовного дерева с оптимальной сложностью. Алгоритм Краскала предпочтителен для графов с разреженной структурой, тогда как алгоритм Прима подходит для плотных графов. Выбор алгоритма зависит от структуры графа и конкретных требований к производительности.

Выполнение работы.

Генетический алгоритм

Геном представляет собой последовательность, состоящую из 0 и 1, где каждая цифра указывает на присутствие или отсутствие соответствующего ребра в исходном графе.

Целевая функция определяется как сумма весов всех включенных рёбер. Если полученный подграф по геному является несвязным, вводится штраф, вычисляемый по формуле: $P=\ln(k)\cdot S$, где k - количество компонент связности, а S - сумма весов всех рёбер.

Генетический алгоритм включает в себя следующие этапы:

0. Генерация начальной популяции геномов.
1. Вычисление целевой функции для каждого генома в популяции.
2. Выбор следующей популяции того же размера с помощью метода, определенного в программе.
3. Скрещивание случайных пар геномов с установленной вероятностью.
4. Мутация каждого генома в популяции с установленной вероятностью.
5. Повторение шага 1 для новой популяции.

Алгоритм продолжается до тех пор, пока не будет достигнут установленный лимит числа эпох.

Графический интерфейс

Для создания графического интерфейса была использована библиотека Dear ImGui. Графический интерфейс поделён на 4 блока: ввод данных графа (рис. 1), ввод параметров генетического алгоритма (рис. 2), просмотр информации о популяции и управление ГА (рис. 3), состояние ГА (рис. 4). Блоки представляют собой окна. При помощи режима Docking можно гибко настраивать их положение (рис. 5).

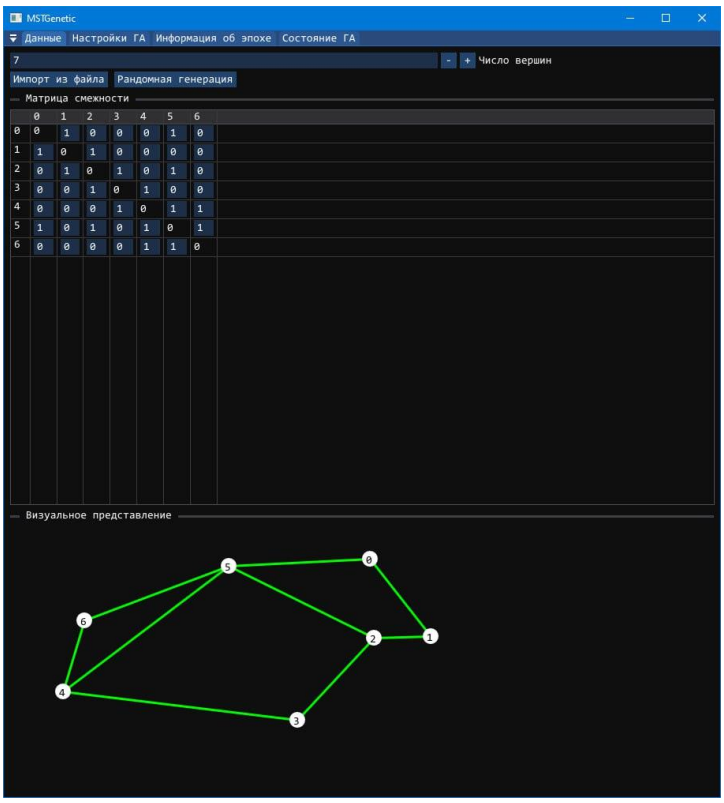

Рисунок 1 - Окно ввода данных графа

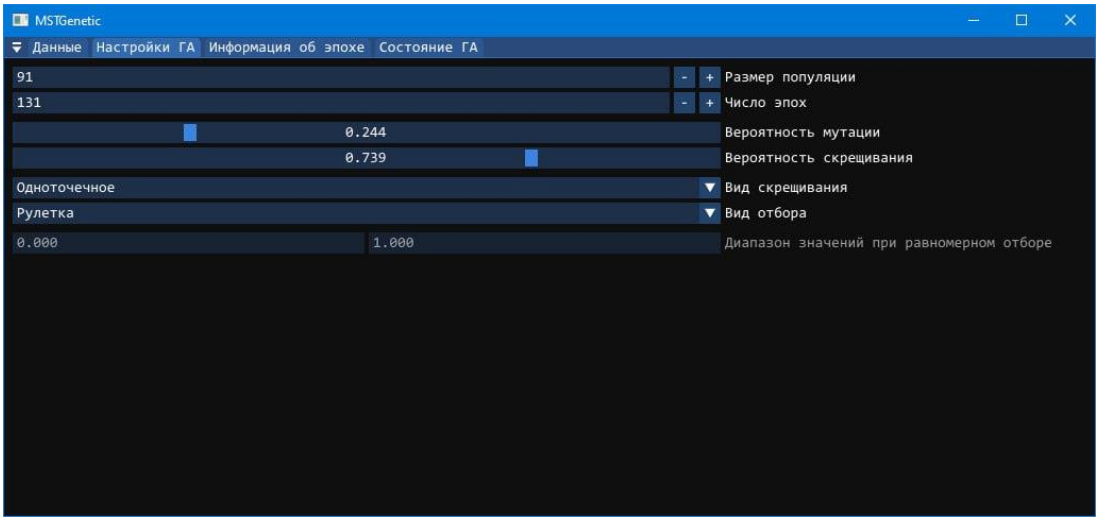

Рисунок 2 - Окно настройки параметров генетического алгоритма

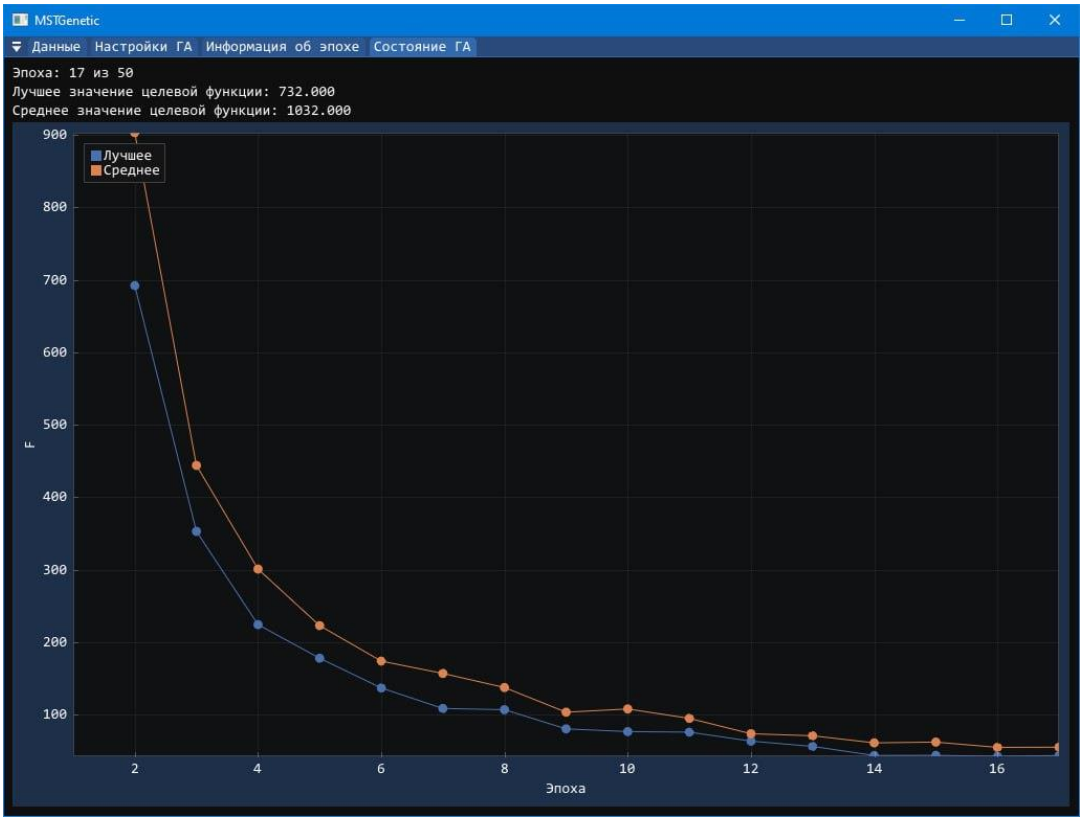

Рисунок 3 - Окно статистики генетического алгоритма

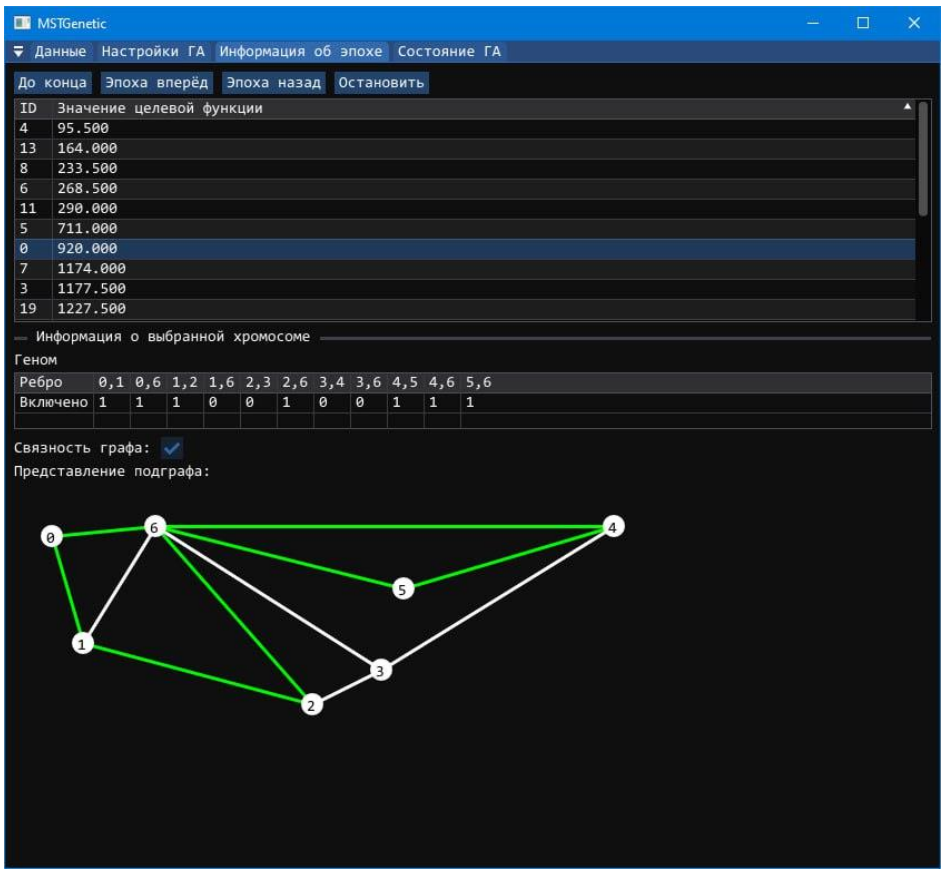

Рисунок 4 - Окно с информацией о популяции текущей эпохи

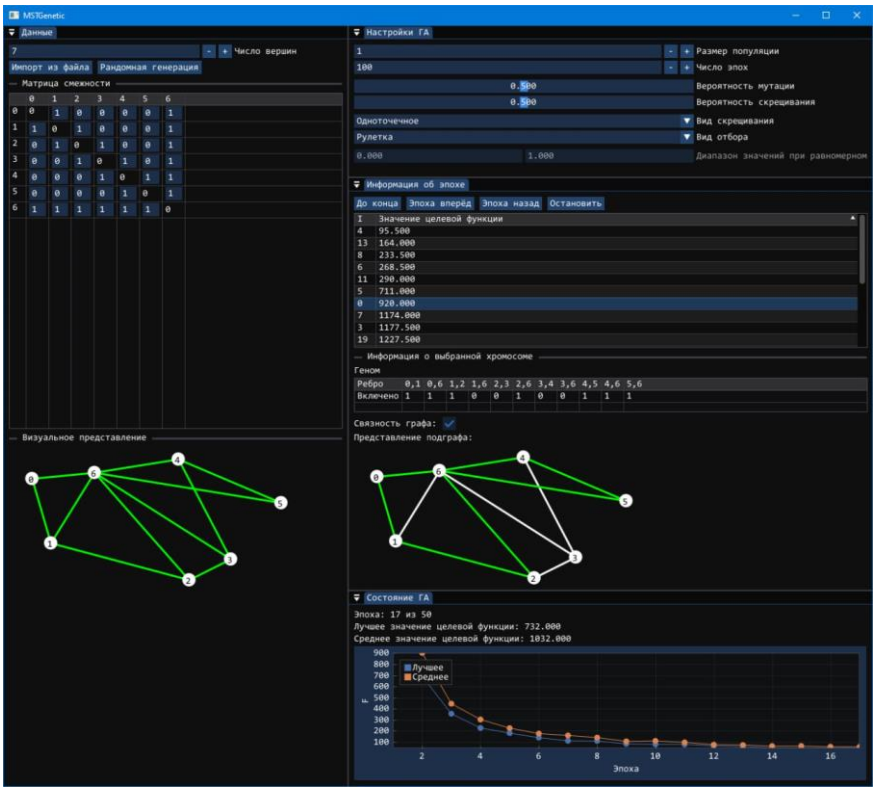

Рисунок 5 - Демонстрация возможностей ImGui в кастомизации отображения окон

Выводы.

В результате работы был написан работоспособный прототип интерфейса, решающий поставленную задачу. Программа было протестирована, результаты тестов удовлетворительны.