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

# ОТЧЕТ

**по практической работе №1**
**по дисциплине Учебная практика**
**по теме: Генетический алгоритм поиска МОД.**

Емельянчик А.А.

Студенты гр. 2304 _____ Федоров Е.М. Жилин Д.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2024

**Цель работы.**

Решение задачи нахождения минимального остовного дерева путем использования генетического алгоритма.

**Задание.**

Задача поиска МОД:

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

**План решения задачи.**

Для решения задачи потребуется реализовать генетический алгоритм. Он состоит из следующих шагов:

1. Генерация первой популяции;
2. Пока не достигнут критерий остановки (по пригодности, числу итераций, т. п.):
    a. Отбор будущих родителей из популяции;
    b. Скрещивание родителей (вероятностное);
    c. Мутация потомства (вероятностное);
    d. Получение новой популяции;
3. Наилучшее решение в популяции – итоговое.

Для реализации алгоритма потребуется найти представление решения в виде генотипа, а также определить, каким образом производятся операции над решениями.

Генотип:

Генотип будет представлять собой множество ребер, образующих остовное дерево графа.

Каждый индивид в популяции будет представлен таким множеством ребер.

Первая популяция:

Изначальная популяция создается путем создания деревьев при помощи модифицированного алгоритма Прима. Его суть в том, что на каждом шаге вместо наименьшего ребра выбирается случайное.

Скрещивание:

Для скрещивания создается подграф путем соединения двух деревьев, затем при помощи рандомизированного алгоритма Прима находится случайное остовное дерево в данном подграфе. Тем самым, результат скрещивания всегда будет корректным решением задачи.

Мутации:

В дереве удаляется случайное ребро, образуя две компоненты связности. Затем ищется другое ребро, способное объединить эти компоненты связности. Как и в случае с мутацией, новый граф также является корректным решением.

Приспособленность:

Приспособленность особи (остовного дерева) определяется на основе веса (суммы весов ребер) этого дерева. Очевидным образом приспособленность тем лучше, чем меньше суммарный вес дерева.

Отбор родителей:

При помощи ранжированного отбора находится набор родителей.

**Частичная реализация алгоритма.**

Первая популяция:

Функция *random_prim()* реализует алгоритм Прима для построения случайного остовного дерева:

Если передан аргумент *subgraph_set*, то на его основе создается подграф *subgraph*. Иначе, используется весь граф *self.graph*.

Инициализируется пустое множество *tree_set*, которое будет хранить ребра построенного остовного дерева.

Выбирается случайная начальная вершина *curr* из графа.

Множество *connected* хранит вершины, уже включенные в остовное дерево.

Список *curr_edges* хранит все ребра, инцидентные текущей вершине curr.

Пока количество вершин в *connected* не равно общему количеству вершин в графе:

1. Выбирается случайное ребро *edge* из *curr_edges*.
    a. Если второй конец ребра *edge[1]* уже включен в connected, то ребро пропускается.
    b. Иначе, вершина *edge[1]* добавляется в connected, ребро *edge* добавляется в *tree_set*.
2. Для новой вершины curr добавляются все ее инцидентные ребра в *curr_edges*.

Функция возвращает множество *tree_set*, содержащее ребра построенного остовного дерева.

Скрещивание:

Функция *crossover()* реализует процесс скрещивания двух родительских особей.

Она объединяет множества ребер двух родителей в одно множество *subgraph*, а затем вызывает функцию *random_prim()* для построения нового остовного дерева на основе этого подграфа.

Таким образом, новое потомство наследует признаки от обоих родителей.

Мутация:

Функция *mutate()* реализует процесс мутации особи.

Она принимает множество ребер, образующих остовное дерево, и выполняет следующие действия:

1. Создает копию подграфа, соответствующего входному множеству ребер.
2. Удаляет одно случайное ребро из этого подграфа.
3. Находит две компоненты связности, образовавшиеся после удаления ребра.
4. Перебирает все ребра графа в случайном порядке и добавляет первое ребро, которое соединяет две компоненты.
5. Возвращает новое множество ребер, представляющее модифицированное остовное дерево.

Таким образом, мутация заключается в замене одного ребра в остовном дереве на другое, которое восстанавливает связность дерева.

Алгоритм:

Функция *algorithm()* реализует основной цикл генетического алгоритма.

Она создает начальную популяцию из 100 случайных остовных деревьев, используя функцию *random_prim()*.

Затем в цикле выполняет следующие шаги:

1. Сортирует популяцию по весу остовных деревьев.
2. Вычисляет модифицированную функцию приспособленности, основанную на весе деревьев.
3. Выбирает 50 родителей с вероятностью, пропорциональной их модифицированной приспособленности.
4. Создает 100 новых особей путем скрещивания родителей и применения мутации.
5. Заменяет текущую популяцию на новую.

Алгоритм продолжается до тех пор, пока не будет найдено остовное дерево с минимальным весом (или достигнуто максимальное количество итераций).

**Прототип GUI.**

Графический интерфейс реализован с помощью фреймворка Tkinter. Ниже представлена его частичная реализация:

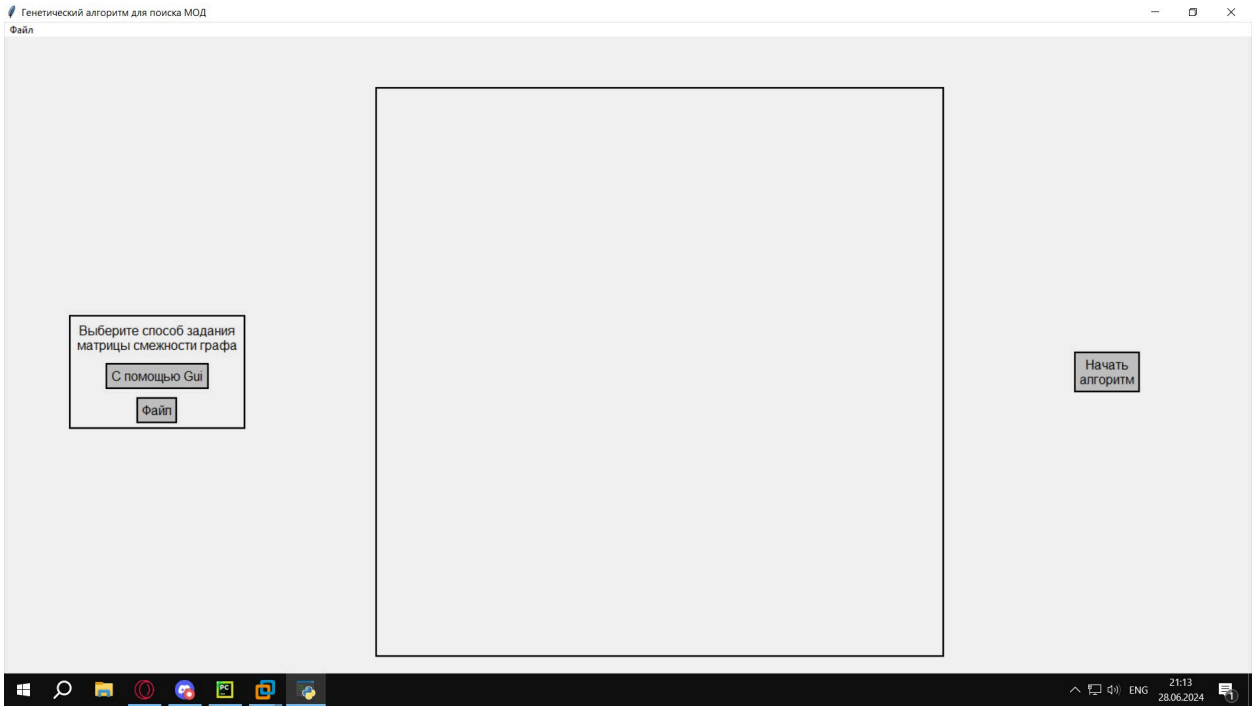


Рис. 1 - Стартовое окно

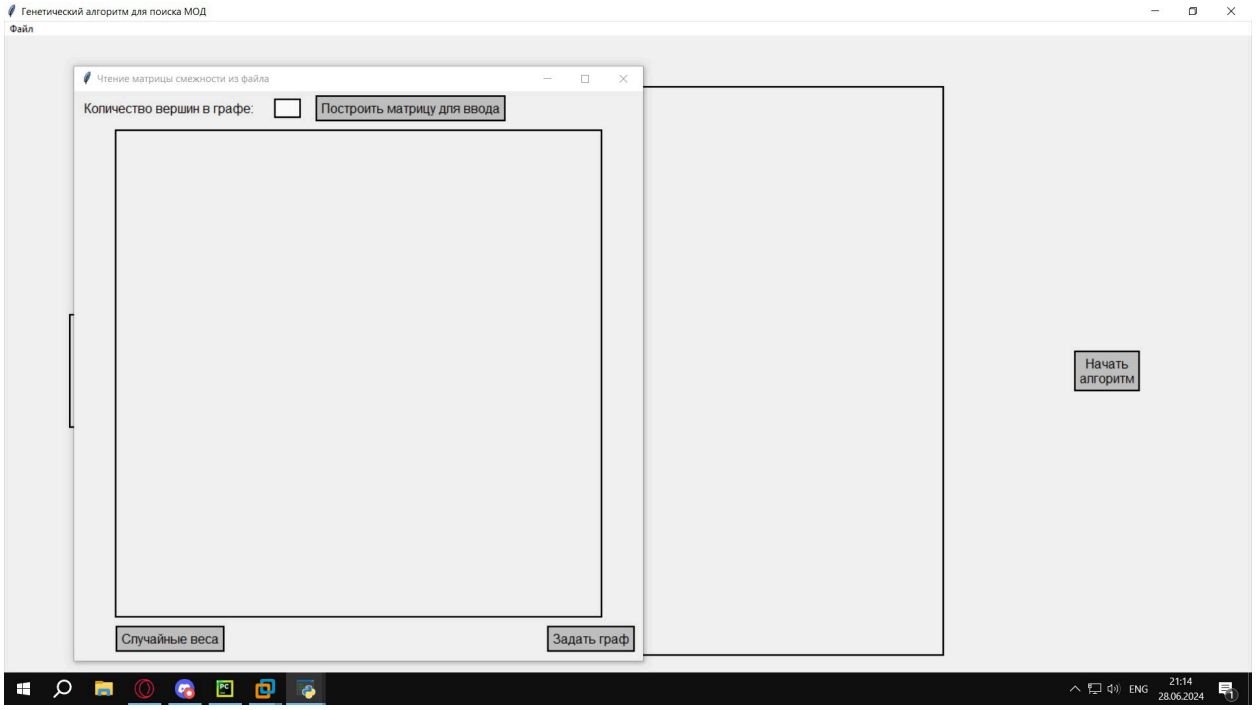


Рис. 2 - Окно ввода размера графа

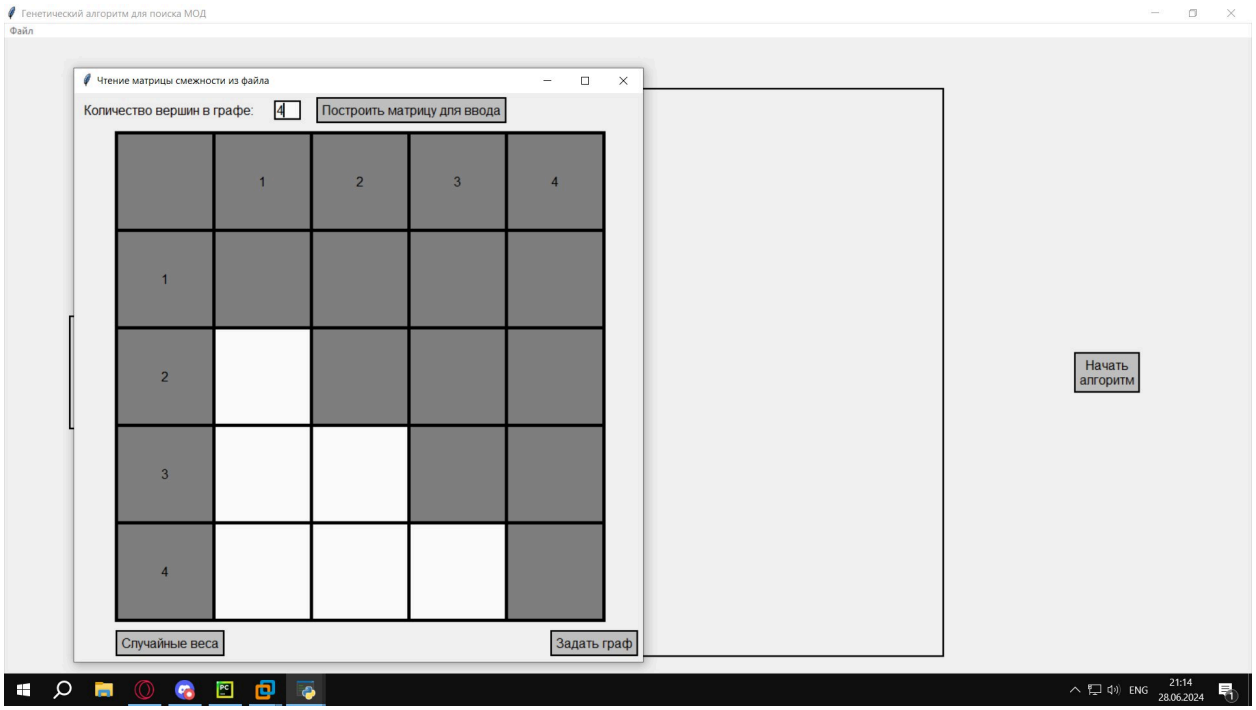


Рис. 3 - Ввод матрицы смежности

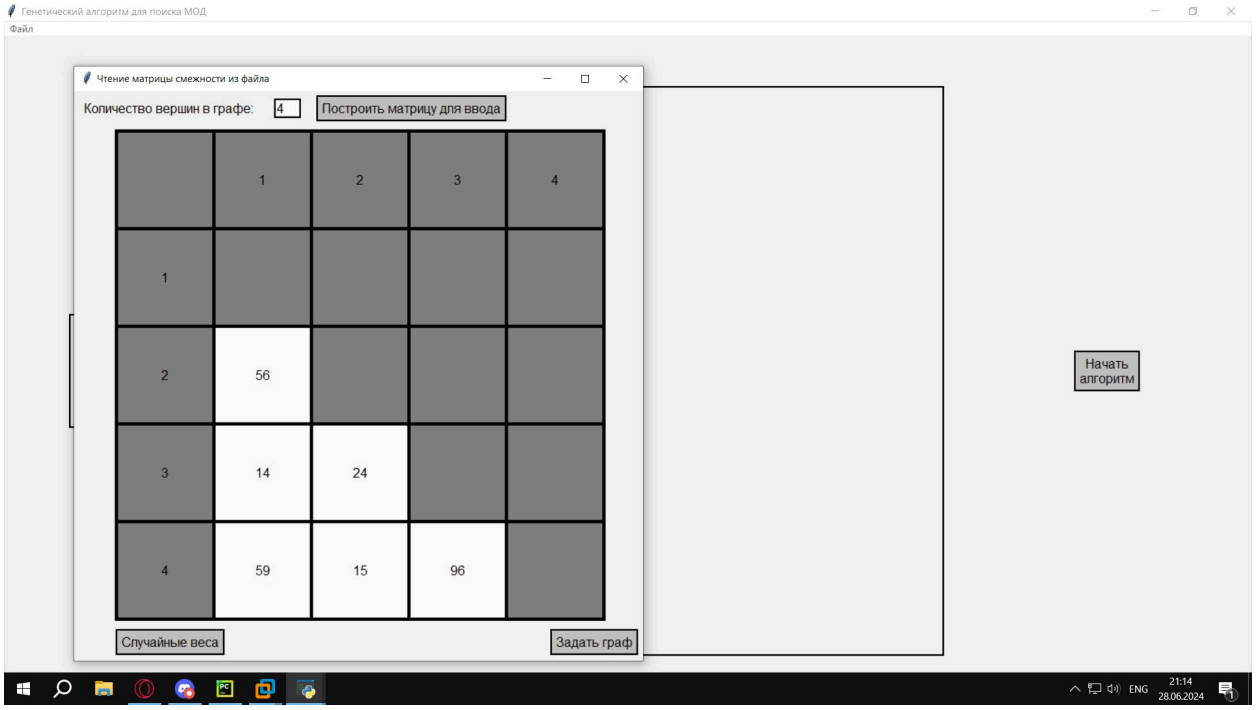


Рис. 4 - Введенная матрица смежности

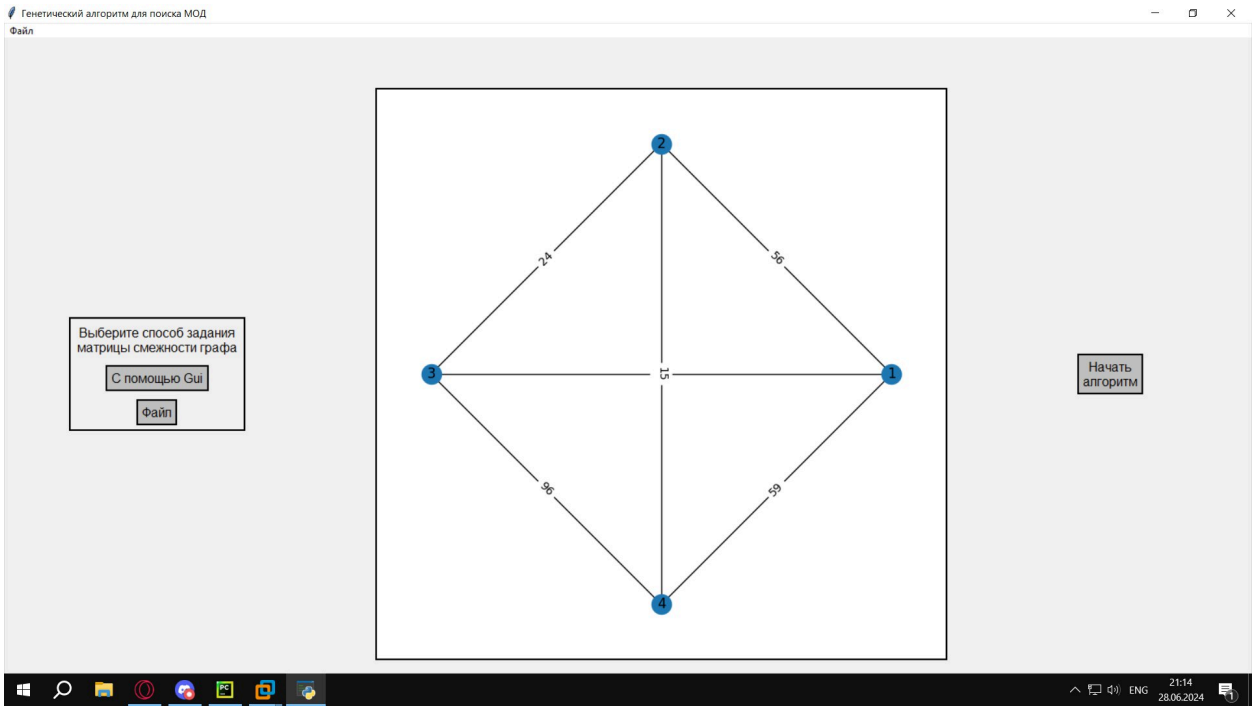


Рис. 5 - Изображение исходного графа

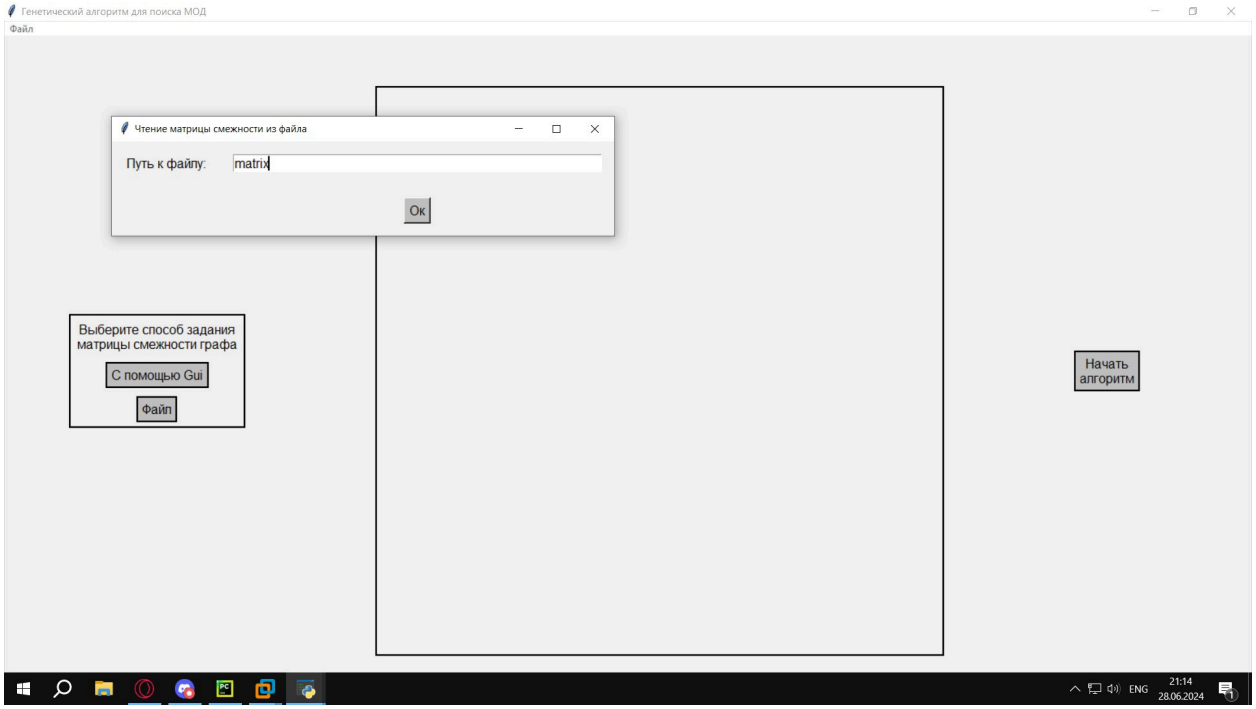


Рис. 6 - Окно ввода пути к файлу

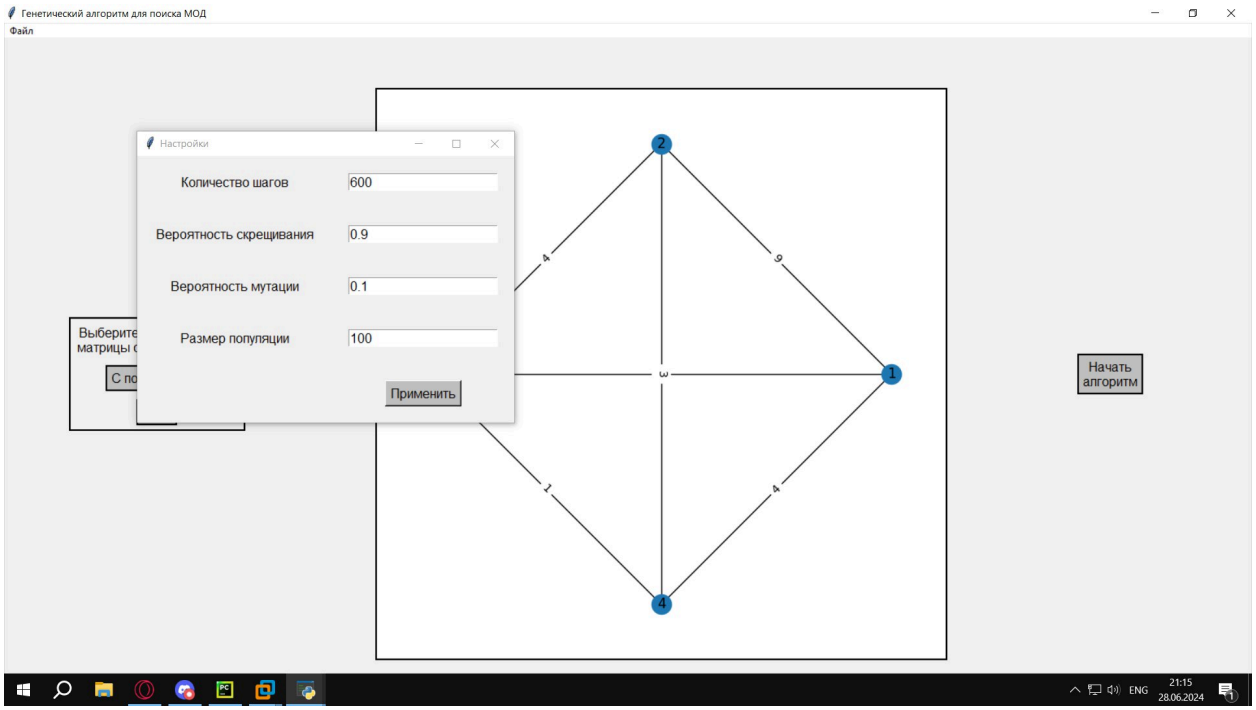


Рис. 7 - Окно настроек

Также разработан примерный макет окна работы алгоритма:


Рис. 8. - Макет окна

На нём отображены следующие элементы:

- Левый нижний угол – график изменения функции качества решения;
- Правый нижний угол – кнопки навигации: переход к предыдущему и следующему шагу, переход к итоговому решению;
- Справа – кнопки переключения между тремя лучшими решениями;
- В центре – Отображения графа выбранного решения;
- Левый верхний угол – кнопка выхода.

**Вывод.**

Было спланировано решение задачи, частично были реализованы генетический алгоритм поиска МОД и GUI.