

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

ОТЧЕТ
по учебной практике
Тема: Генетические алгоритмы и ПСА

Студент гр. 3344

Клюкин А.В.

Студентка гр. 3344

Якимова Ю.А.

Руководитель

Жангиров Т.Р.

Санкт-Петербург

2025

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Клюкин А.В., Студентка Якимова Ю.А.

Группа 3344

Тема практики: Генетические алгоритмы и ПСА

Задание на практику:

11) Задача о назначениях

Пусть имеется N работ и N кандидатов на выполнение этих работ, причем назначение j -й работы i -му кандидату требует затрат $c_{ij} > 0$. Необходимо назначить каждому кандидату по работе, чтобы минимизировать суммарные затраты. Причем каждый кандидат может быть назначен на одну работу, а каждая работа может выполняться только одним кандидатом.

Сроки прохождения практики: 25.06.2025 – 07.07.2025

Дата сдачи отчета: 04.07.2025

Дата защиты отчета: 04.07.2025

Студентка		Якимова Ю.А
Студент		Клюкин А.В.
Руководитель		Жангиров Т.Р.

АННОТАЦИЯ

Практика направлена на изучение и применение генетических алгоритмов для решения оптимизационных задач. Требуется разработать программу с графическим интерфейсом, которая будет находить оптимальное распределение работ между кандидатами, минимизируя суммарные затраты. Основные этапы включают:

- Реализацию ГА без использования специализированных библиотек.
- Интеграцию GUI для ввода данных различными способами, настройки параметров ГА и визуализации результатов.
- Обеспечение пошагового выполнения алгоритма с отображением лучшего решения, его стоимости и средних показателей поколения.
- Построение графика изменения приспособленности в процессе работы алгоритма.
- Демонстрацию возможности повторного запуска с новыми параметрами и данными.

СОДЕРЖАНИЕ

	Введение	5
1.	Анализ и проектирование	6
1.1.	Постановка задачи	6
1.2.	Представление решения	6
1.3.	Структура проекта	6
1.4.	Прототип GUI	8
1.5.	Порядок реализации, роли и сроки	9
2.	Реализация	11
2.1.	Реализация генетического алгоритма	11
2.2.	Реализация GUI (визуальная часть)	13
3.	Связь между GUI и алгоритмом	16
3.1.	Дополнение функциональности GUI	16
3.2.	Классы и их взаимодействие	17
	Заключение	19

ВВЕДЕНИЕ

Целью данной практики является изучение принципов работы ГА и применение их для решения задачи на оптимизацию. Разработка программы с графическим интерфейсом для телеметрии и демонстрации пошаговой работы алгоритма для анализа результатов.

Задачи:

- Изучить теоретические основы работы ГА
- Разработать алгоритм, решающий задачу с помощью ГА
- Реализовать графический интерфейс
- Связать графический интерфейс с основным алгоритмом
- Протестировать работу программы на различных входных данных

1. АНАЛИЗ И ПРОЕКТИРОВАНИЕ

1.1. Постановка задачи

Задача заключается в оптимальном разделении N работ между N кандидатами, где каждое значение (i, j) имеет стоимость $c[i][j]$. Требуется найти такое распределение, что каждый кандидат выполняет ровно одну работу и суммарная стоимость назначений минимальна.

1.2. Представление решения

Кодирование решения - хромосома. Будет представляться в виде массива, длиной N , где индексом элемента является номер кандидата, а значение элемента - номер работы, которую он выполняет.

Функция приспособленности вычисляется как сумма стоимости назначений из данной матрицы. Из хромосомы берется индекс кандидата и номер работы. Чем меньше сумма, тем лучше решение.

Генерация начальной популяции создается случайным образом с проверкой на валидность.

Скрещивание будет реализовано с помощью метода “Упорядоченное скрещивание”. Метод равномерного или k -точечного скрещивания не подходит, тк возможно нарушение валидности хромосомы.

Мутация будет реализована через метод обмена и обращения.

Отбор подчиняется правилу рулетки.

1.3. Структура проекта

Проект разбит на две части - реализация генетического алгоритма и работа с графической составляющей. С точки зрения структуры, будет разделено на несколько модулей. Основные компоненты:

Главный модуль - `main.py`. Является точкой запуска программы. Инициализирует GUI и управляет запуском ГА.

Модуль генетического алгоритма - `genetic_solver.py`. Реализация ГА, содержащий методы скрещивания, мутации, отбора.

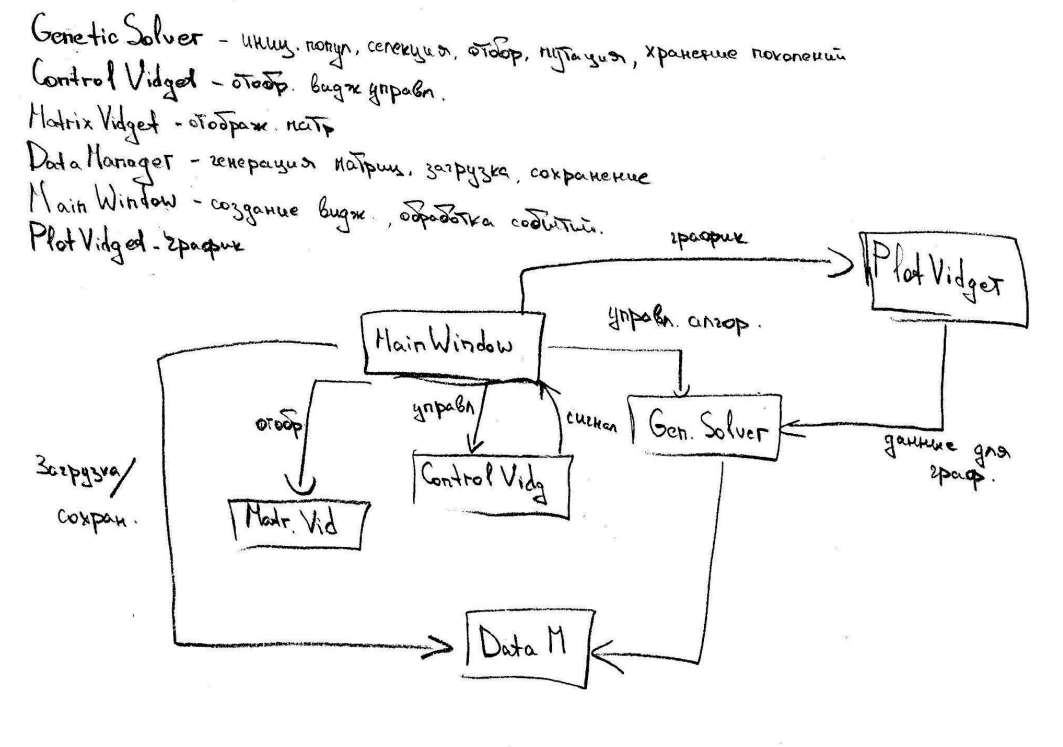
Модуль управления данными - data_manager.py. Работа со входными данными, генерацией, хранением для графического представления.

Модули графического интерфейса - matrix_widget, control_widget, plot_widget

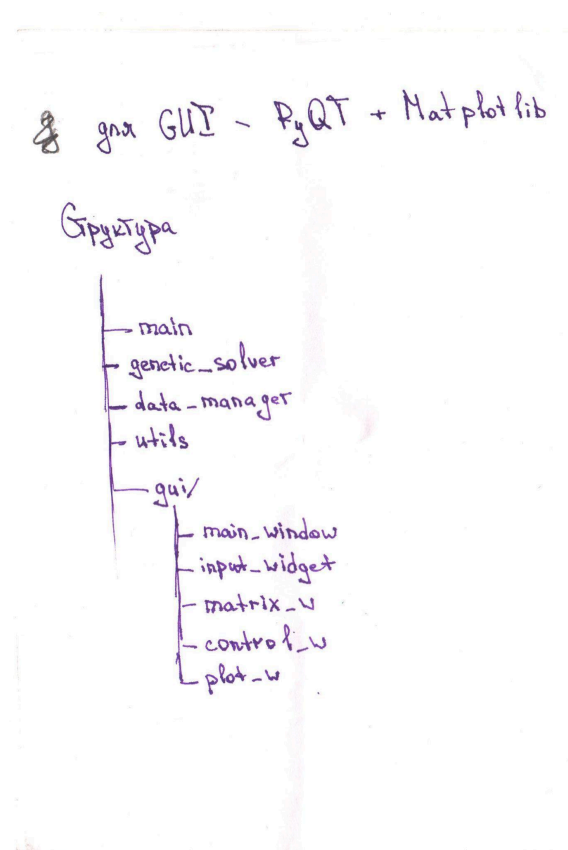
Различные вспомогательные функции - utils.py

Схема взаимодействия модулей

1. Пользователь запускает программу через main.py.
2. main_window.py загружает интерфейс и предоставляет варианты ввода данных (файл, генерация, ручной ввод).
3. data_manager обрабатывает данные и передает матрицу стоимостей в genetic_solver.
4. genetic_solver выполняет алгоритм, обновляя информацию о поколениях.
5. plot_widget и control_widget отображают прогресс и позволяют управлять параметрами.
6. Результаты выводятся в главном окне.



(Рис. 1)



(Рис. 2)

1.4. Прототип GUI

Для создания GUI было решено использовать фреймворк PyQt5. Это решение обусловлено простотой, популярностью фреймворка а также возможностью интеграции Matplotlib в виджеты.

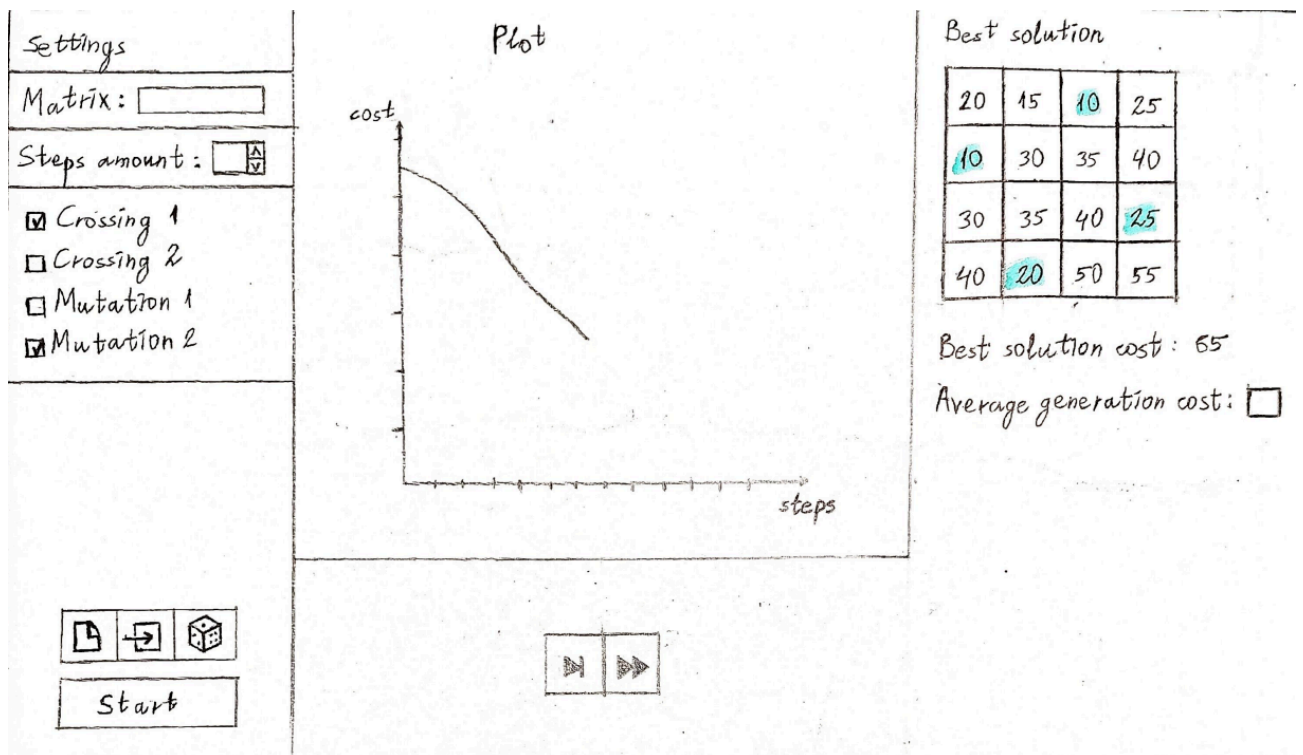
Для начала будет создано главное окно приложения (main_window). Оно создает другие виджеты и управляет ими, а также обрабатывает основные события.

Input_w предоставляет интерфейс для выбора источника входных данных (из файла, ввод через GUI, случайная генерация) и для их ввода/настройки. Обрабатывает ввод пользовательских данных и предоставляет их в структурированном формате.

Matrix_w отображает матрицу затрат.

Plot_w отображает график изменения приспособленности (лучшей и средней) в зависимости от поколения в процессе работы генетического алгоритма.

Control_w предоставляет элементы управления для управления выполнением генетического алгоритма.



1.5. Порядок реализации, роли и сроки

- Первоначально будут реализованы модуль хранения данных и работа ГА. Параллельно разработан прототип GUI. (02.07)
- Затем ГА будет доработан, а интерфейс связан с данными алгоритма. (04.07)
- Тестирование и демонстрация полного проекта (06.07)

За создание графического интерфейса, взаимодействие с пользователем, визуализацию данных и управление пользовательским вводом отвечает Якимов Ю.А.

За реализацию генетического алгоритма, обработку данных, расчет целевой функции отвечает Клюкин А.В.

2. РЕАЛИЗАЦИЯ

2.1. Реализация генетического алгоритма

Код состоит из трёх модулей.

- `genetic_solver.py` — содержит класс `GeneticSolver`, реализующий генетический алгоритм.
- `utils.py` — вспомогательные функции для работы с матрицей затрат, сохранения и загрузки состояния алгоритма, а также вывода результатов.
- `main.py` — основной скрипт для демонстрации работы алгоритма.

Класс `GeneticSolver` содержит логику генетического алгоритма: инициализацию популяции, отбор, скрещивание, мутацию и управление состоянием. На вход подаются параметры:

- `cost_matrix` — матрица затрат, где элемент `cost_matrix[i][j]` представляет стоимость назначения кандидата i на работу j .
- `population_size` — размер популяции.
- `mutation_rate` — вероятность мутации особи.
- `crossover_rate` — вероятность скрещивания родителей.
- `max_generations` — максимальное количество поколений.
- `early_stop` — критерий ранней остановки, если улучшение не наблюдается в течение заданного числа поколений.

Содержит следующие методы:

Инициализация популяции (`_initialize_population`) - создает начальную популяцию случайных перестановок, где каждая перестановка представляет возможное распределение работ.

Расчет приспособленности (`_calculate_fitness`) - вычисляет стоимость решения как сумму затрат для данного распределения работ.

Отбор (`_do_selection`) - использует метод рулетки для выбора особей с вероятностью, пропорциональной их приспособленности. Для этого берутся обратные числа от приспособленности, тк нам нужно минимальное. И нормируются по их сумме.

Скрещивание (`_ordered_crossover`) - реализует упорядоченное скрещивание, создавая потомков на основе частей родительских хромосом. Создаются два потомка. От родителей берется произвольная часть и записывается в потомка. Затем пустые места дополняются оставшимися значениями.

Мутация (`_apply_mutation`) - применяет два типа мутаций: обмен двух генов и инверсия подпоследовательности.

Сохранение и откат состояния (`_save_state, rollback`): сохраняет текущее состояние алгоритма (популяцию, выбранных особей, потомков и т.д.) в историю для возможного отката.

Пошаговое выполнение (`step, step_clear`) - выполняет одну фазу алгоритма (инициализация, отбор, скрещивание или мутация) с подробным выводом информации или без неё, соответственно.

Запуск алгоритма (`run`) - выполняет алгоритм до достижения критериев остановки.

Вспомогательные функции содержатся в файле `utils.py`. В их числе:

- Генерация матрицы затрат (`generate_cost_matrix`)
- Сохранение и загрузка состояния из файла (`save_solver_state, load_solver_state`)
- Вывод результатов (`print_solution, print_cost_matrix`)

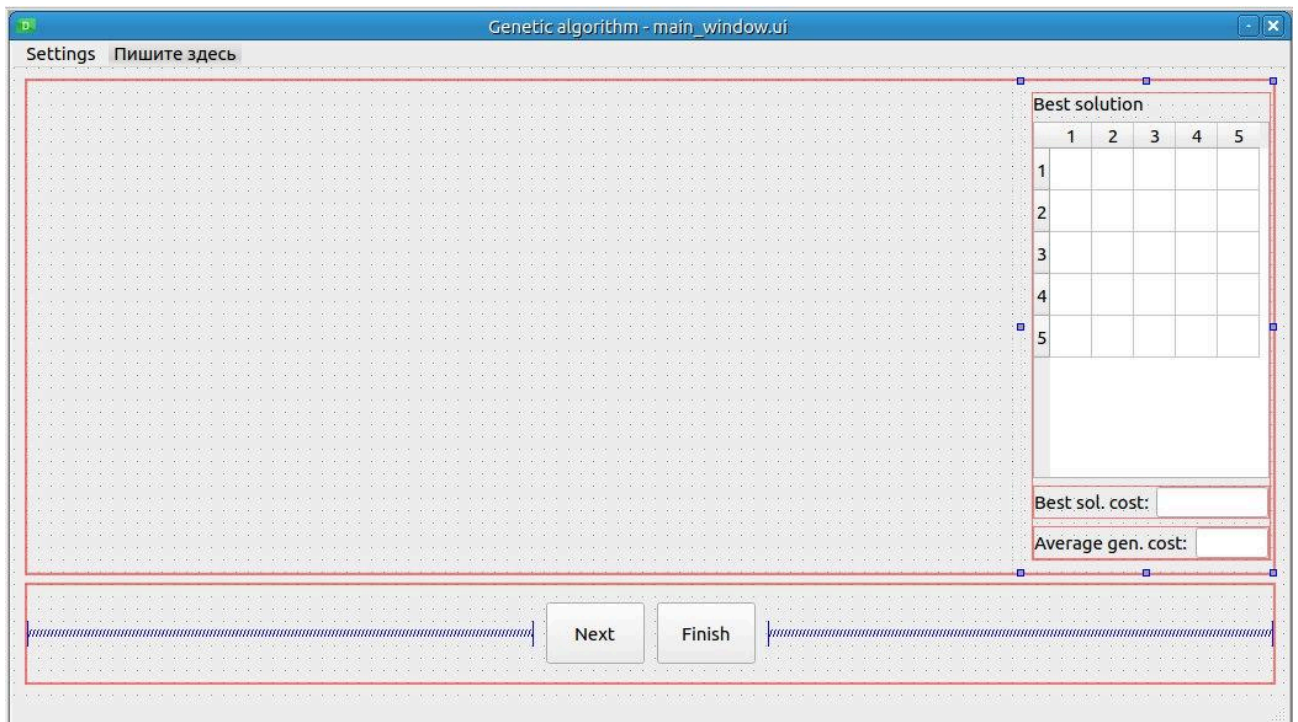
Пример работы - `main.py`

1. Генерируется матрица затрат размером 5x5.
2. Создается экземпляр `GeneticSolver` с заданными параметрами
3. Выполняется 5 шагов алгоритма с выводом информации о каждой фазе.
4. Состояние алгоритма сохраняется в файл `solver_state.json`.
5. Производится откат на 2 шага и вывод текущего состояния.
6. Загружается сохраненное состояние, и алгоритм продолжает работу до завершения.

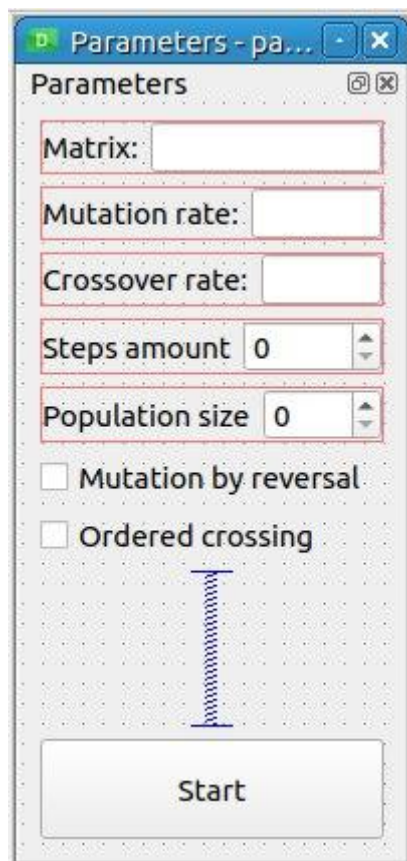
2.2. Реализация GUI (визуальная часть)

Для начала было создано главное окно приложения (`main_window.py` и `main_window_ui.py`).

Большую часть окна занимает виджет `canvas`, в котором будет отображаться `Figure`, создаваемая в `matplotlib`. Ниже располагаются кнопки перехода к следующему шагу и завершения работы алгоритма. Справа расположен информационный виджет с таблицей лучшего решения (клетки решения будут выделены), ценой лучшего решения и средней ценой поколения.



Для задания пользовательских параметров был создан `DockWidget` `ParametersDock` (`parameters_dock.py` и `parameters_dock_ui.py`). Этот виджет прикрепляется слева от основного окна.

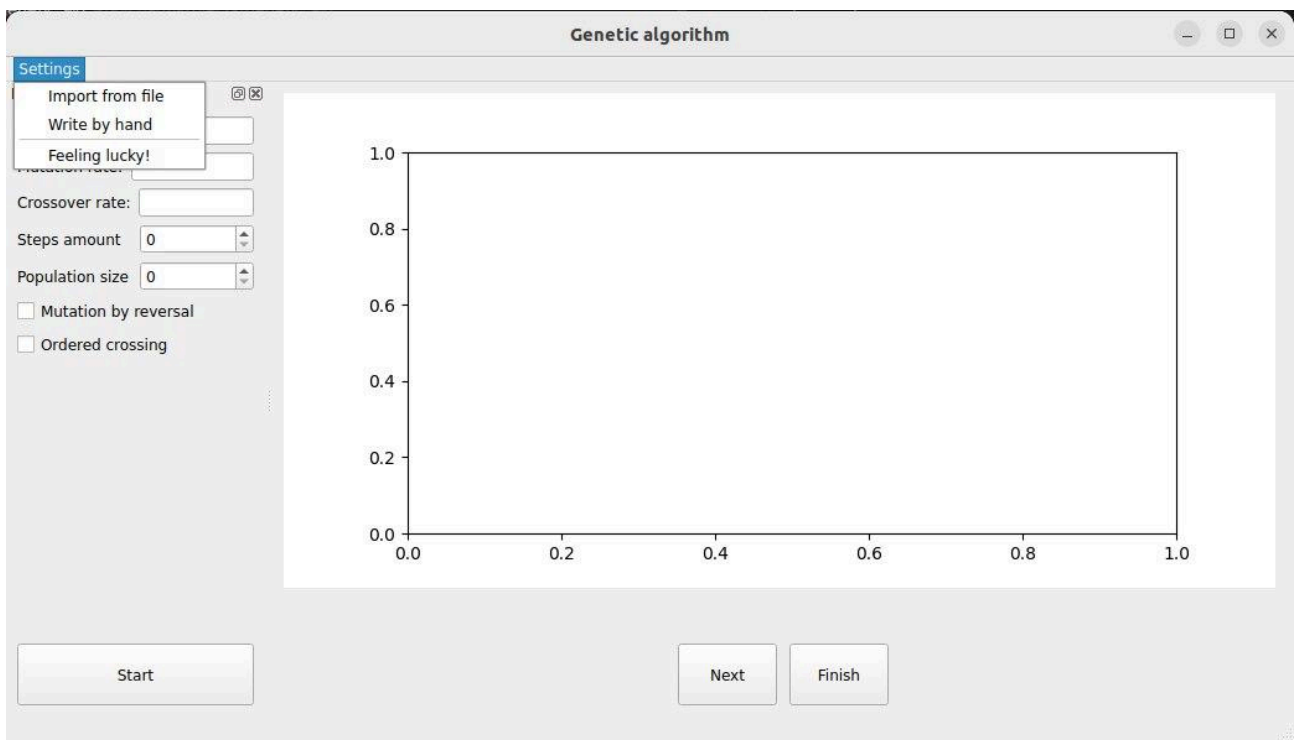


Виджет содержит следующие поля для заполнения:

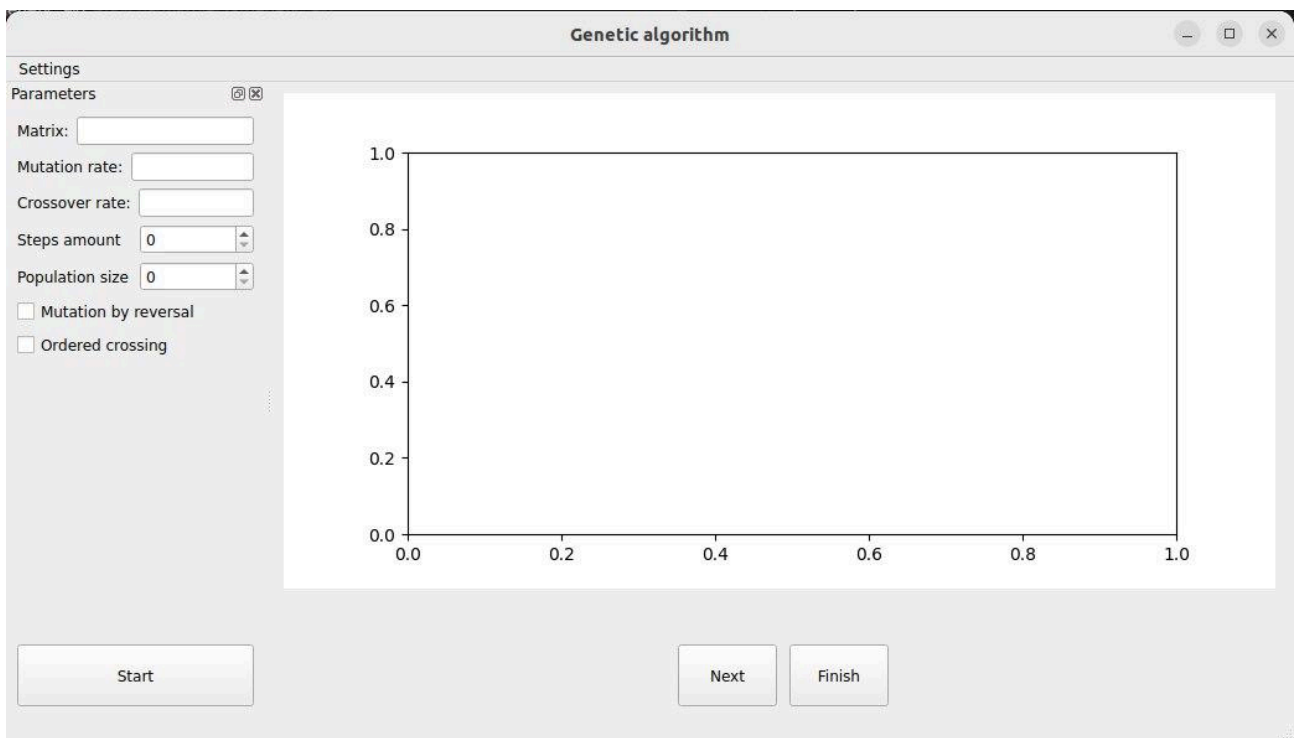
1. Matrix - входные данные согласно задаче (массив чисел)
2. Mutation rate - уровень мутации (от 0 до 1), дробное число
3. Crossover rate - уровень скрещивания (от 0 до 1), дробное число
4. Steps amount - количество шагов (поколений), целое число
5. Population size - численность популяции, целое число

Также реализовано 2 чекбокса для выбора методов мутации и скрещивания.

При нажатии на Settings можно задать данные из разных источников:



Финальная версия:



3. СВЯЗЬ МЕЖДУ GUI И АЛГОРИТМОМ

3.1. Дополнение функциональности GUI

Были дополнены некоторые поля для ввода данных (метод отбора и число остановки для генетического алгоритма), а также для более подробного вывода информации (номер текущего поколения, фаза мутации, выбранные методы мутации и отбора).

Были реализованы кнопки “Back”, “Play”, “Stop”, чтобы добавить возможность вернуться на шаг назад или запустить выполнение алгоритма в автономном режиме.

Было реализовано графическое отображение текущего лучшего решения в виде подсвеченных клеток таблицы, содержащей матрицу.

Также были добавлены обработчики ошибок для входных данных в parameters_dock.py. После проверки на корректность данных начинает работу алгоритм.



Была добавлена возможность задать входные данные из разных источников (файл, ввод через GUI, случайная генерация).

3.2. Классы и их взаимодействие.

Для управления взаимодействием между генетическим алгоритмом и графическим интерфейсом был реализован класс GeneticController.

Основные методы:

- initialize_solver() - инициализирует GeneticSolver с заданными параметрами.
- step() - вызывает step_clear() у GeneticSolver и отправляет данные для отображения через сигнал update_display.
- run_full() - запускает автоматическое выполнение алгоритма с использованием QTimer.
- rollback() - откатывает состояние алгоритма и обновляет интерфейс.
- set_mutation_method()/set_selection_method() - методы для изменения способа мутации или отбора соответственно.

Содержит в себе объект решения генетического алгоритма и управляет им. Передает данные для графического интерфейса через update_display.

Основное окно, которое управляет всеми компонентами GUI - MainWindow. Содержит в себе методы для обработки событий, например start_algorithm, next_step, prev_step, run_full. Имеет метод для обновления информации на дисплее включая промежуточные данные и график - update_display(). Так же содержит методы для генерации для импорта входных данных из файла - import_matrix_from_file и для генерации случайной матрицы заданного размера - generate_random_matrix.

Класс ParametersDock отвечает за размещение и обработку полей для ввода параметров генетического алгоритма.

Класс InfoWidget отвечает за размещение промежуточных результатов. Содержит метод для обновления отображаемой информации - update_info.

Класс `MplCanvas` отвечает за размещение области графика и отрисовку.
Содержит `plot_points` для отрисовки.

ЗАКЛЮЧЕНИЕ

В ходе учебной практики были изучены принципы работы генетического алгоритма, затем был разработан алгоритм для решения задачи на оптимизацию с использованием вышеупомянутых принципов. Был реализован графический интерфейс, позволяющий задать данные из разных источников и демонстрирующий пошаговую работу алгоритма. Далее графический интерфейс был доработан и интегрирован с основным алгоритмом.

Программа была протестирована на различных входных данных, корректность работы установлена. Полученный результат соответствует поставленной цели.