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

ОТЧЕТ

по летней учебной практике

по дисциплине «Генетические алгоритмы»

Студенты гр. 3341 _____ Пчёлкин Н. И.,
Романов А. К.,
Шаповаленко Е. В.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2025

(I итерация)

Распределение обязанностей

Обязанности в бригаде были распределены следующим образом:

- Пчёлкин Н. И. — лидер, координация графической составляющей и алгоритмической части.

- Романов А. К. — разработка графической составляющей (GUI).

- Шаповаленко Е. В. — разработка алгоритмической составляющей (генетический алгоритм).

По мере занятости участники бригады могут помогать товарищам с их частью работы.

(II итерация)

Порядок перемножения матриц

Для перемножения *N* матриц требуется *N-1* операция матричного умножения. Очевидным образом каждую из операций умножения можно пронумеровать *N-1* числами (на рис. 1а представлен пример для 4 матриц). Порядок перемножения матриц определяется порядком этих чисел. Таким образом индивид будет представлять собой порядок перемножения матриц, а геном будет номер операции (на рис. 1б представлен пример индивида для примера на рис. 1а, на рис. 1в представлен порядок перемножения матриц для индивида на рис. 1б).

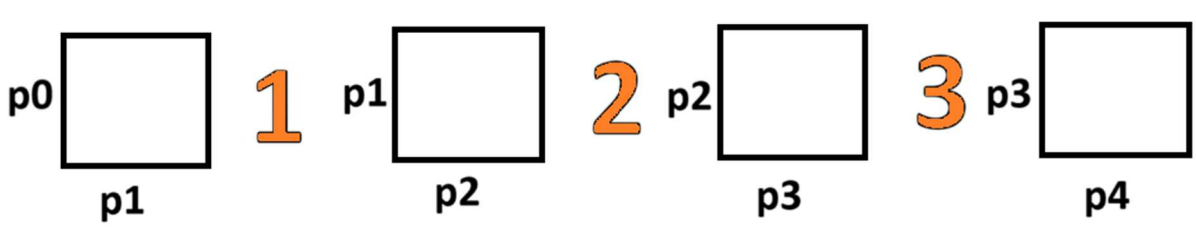


рис. 1а, нумерация операций матричного умножения

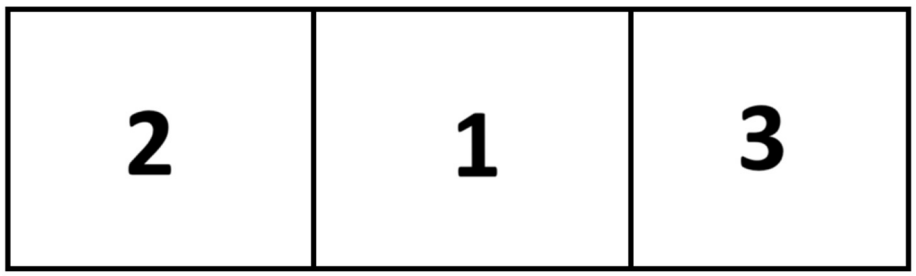


рис. 1б, пример индивида

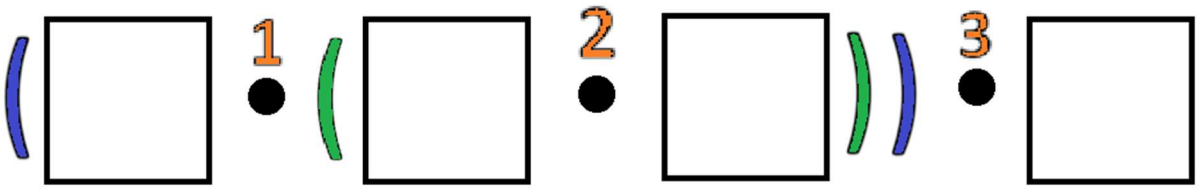


рис. 1в, порядок перемножения матриц

Возможны случаи, когда 2 различных хромосомы могут соответствовать одному и тому же индивиду, например, *1243* и *1423* для 5 матриц, или *132* и *312* для 4 матриц, но это допустимо, так как на количество затраченных операций для непосредственного вычисления произведения матриц это не влияет.

Отбор будет производиться правилом рулетки.

Функция приспособленности будет вычисляться следующим образом: для перемножения двух матриц, с размерами (p_i, p_{i+1}) и (p_{i+1}, p_{i+2}) соответственно, требуется $p_i * p_{i+1} * p_{i+2}$ операций умножения (стоимость). В результате получится матрица с размерами (p_i, p_{i+2}). Выполнив все операции матричного умножения в определенном порядке, получим *N-1* стоимости, которые необходимо сложить. Полученный результат — суммарное количество операций умножения для данного порядка перемножения матриц, которое и необходимо минимизировать.

Будет использоваться упорядоченное скрещивание, т. к. не допускается повтор генов, а также требуется сохранение их некого относительного порядка.

Метод мутации будет параметром алгоритма. Будут применяться мутации обменом, обращением и перетасовкой.

Все ограничения исключительно жесткие, т. к. условия задачи не подразумевают возможности каких-либо нарушений.

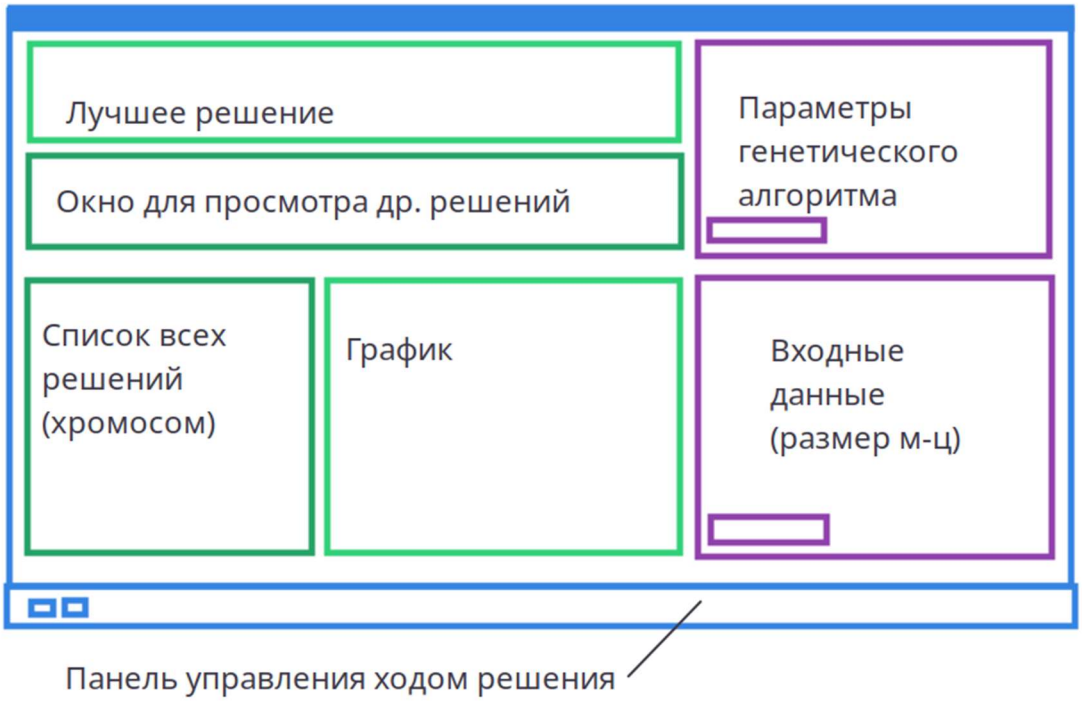
Прототип GUI

Предварительно опишем общий концепт организации работы приложения. Предполагается, что оно будет состоять из трех ключевых компонент: бэкэнда (части, отвечающей за реализацию генетического алгоритма), фронтэнда (графического интерфейса для взаимодействия с приложением) и движка, связывающего работу двух этих компонент (в его задачи будет входить обмен данными между ними и т.п.).

Для разработки графического интерфейса было решено использовать библиотеку tkinter, поскольку имеется опыт работы с ней, а также с ней знакома часть участников бригады.

Графический интерфейс должен реализовывать следующие функции: ввод данных и параметров его работы генетического алгоритма; вывод на экран результата работы алгоритма или его промежуточных шагов; управление ходом работы алгоритма.

На этапе планирования была разработана схема организации графического интерфейса приложения:



Согласно техническому заданию должна иметься возможность генерировать случайные входные данные, а также читать их из файла. Для этих целей на виджетах для ввода планируется добавить кнопки для соответствующих операций, а также кнопку для очистки введенных данных. Для того, чтобы отображать результат работы алгоритма и промежуточные данные, планируется добавить несколько виджетов: основные выделены на рисунке светло-зеленым (их наличие обязательно по техническому заданию). Прочие виджеты, выделенные темно-зеленым, будут реализовываться позже. Внизу экрана будет размещена панель контроля хода решения.

Создание прототипа.

Все блоки представляют из себя Frame с заголовком (Label)

1. *ParameterFrame(InputFrame)* — Блок для ввода параметров генетического алгоритма хранит набор именованных полей для ввода (Label + Text). Внизу него размещены кнопки (Button) для генерации произвольных данных и чтения данных из файла, а также кнопка очистки полей ввода.

2. *MatrixFrame(InputFrame)* — Блок для ввода размера матриц, в нем хранится другой Frame, имеющий возможность прокрутки, на нем размещаются

именованные поля для ввода размер матриц. Для того, чтобы решать задачу для разного числа матриц, на виджете ниже размещены кнопки «+» и «-» для увеличения и уменьшения числа этих полей соответственно. (Число таких полей не может быть уменьшено ниже трех). При добавлении новых полей, они помещаются на прокручиваемом фрейме, таким образом общий размер виджета не изменяется. Внизу виджета также размещены кнопки для случайно генерации данных, чтения из файла и очистки ввода.

3. *AnswerFrame(DisplayClass)* — Блок для вывода лучшего ответа хранит поле ввода текста (Text) с отключенной возможностью печати для пользователя. (Ввод в него осуществляется программно). Поле текста также имеет свойство прокрутки, для того, чтобы возможно было посмотреть всю строку, если она длинная. Выводимая строка представляет из себя изображение лучшей расстановки скобок для перемножения матриц. (В квадратных скобках пишутся размеры матриц, круглые скобки определяют порядок операций). Предполагается выделение круглых скобок цветом для улучшения читаемости.

4. *GraphFrame(DisplayClass)* — Блок для отображения графика хранит объект класса FigureCanvasTkAgg, позволяющего интегрировать графики matplotlib в приложение tkinter. На нем будет показываться график изменения приспособленности. (На момент прототипирования добавлена кнопка для генерации случайных данных).

5. *SolutionsFrame(DisplayClass)* — Был также создан блок для отображения всех имеющихся решений (хромосом). Из-за ограничений библиотеки, он был реализован не самым стандартным способом: имеющиеся решения представляют из себя лейблы, к которым добавлены функции при нажатии (click) и наведении курсора (hover). Эти лейблы размещены на фрейме с возможностью прокрутки. При нажатии на решение предполагается, что оно будет отображаться в окне для просмотра решений (не было реализовано к данному моменту).

6. *ControlPanel* — Панель управления представляет из себя фрейм с несколькими кнопками. Предполагается, что кнопки будут давать возможность запустить выполнение алгоритмаЮ сделать шаг алгоритма вперед (и, возможно,

назад), запустить алгоритм без прерываний до конца, прервать выполнение алгоритма.

Объекты 1-2 унаследованы от класса *InputFrame.* Он имеет следующие поля:

- *self.buttons* — массив кнопок.

- *self.parameters* — словарь полей ввода для доступа к параметрам.

И следующие методы:

def init_layout(self, buttons=None) — задает внешний вид виджета (заголовок, тело, кнопки).

def init_contents(self, parameters=None) — задает содержимое тела виджета. Абстрактный метод.

def generate_data(self, generator=None) — используется для заполнения полей случайными значениями.

def read_data(self) — читает данные из файла, заполняет ими поля.

def clear_data(self) — очищает поля ввода.

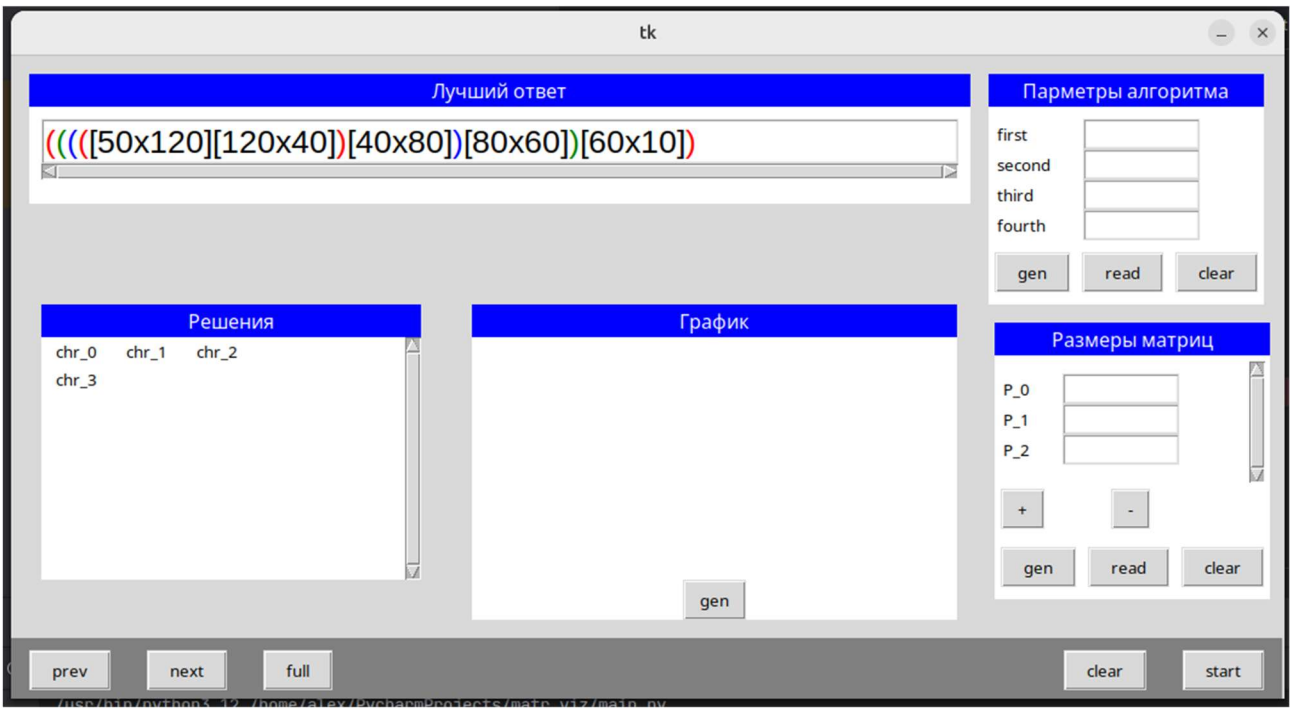
def get_parameter(self, parameter_name="first") — позволяет получить данные параметра из поля ввода по имени параметра.

Объекты 3-5 унаследованы от класса *DisplayFrame*. Он имеет следующие методы:

def init_layout(self, buttons=None) — задает внешний вид виджета (заголовок, тело).

def init_contents(self, parameters=None) — задает содержимое тела виджета. Абстрактный метод.

Классы *DisplayFrame* и *InputFrame* унаследованы от класса *BaseFrame,* который задает основную структуру виджета — заголовок и тело.



(III итерация)

Организация GUI.

Структурно графический интерфейс разделен на несколько логических уровней. Внизу иерархии расположены вспомогательные классы *...Utils*, выполняющие повторяющиеся действия, такие как

- создание одинаковых лейблов, кнопок и т. п. — класс *GUIUtils*,

- модификация состояний элементов: подсветка, смена цвет элемента — класс *ConfigUtils*,

- запись данных в поле ввода, осуществляемая на уровне программа (не пользовательский ввод) — класс *ConfigUtils*,

- инициализация содержимого виджетов — класс *InitUtils*,

- чтение данных из файла — класс *FileUtils*.

Все вышеперечисленные классы имеют один или несколько статических методов для выполнения необходимых задач.

Также на нижнем логическом уровне находятся примитивные композитные виджеты, являющиеся комбинацией базовых классов библиотеки tkinter. К ним относятся

- *classs EntryBox* — комбинация из лейбла и поля ввода.

- *class InfoBox* — комбинация двух лейблов, один из которых содержит фиксированный текст, а содержимое второго может изменяться для отображения информации.

- *class ScrollableFrame* — прокручиваемый фрейм, необходимый для хранения не фиксированного количества других виджетов.

- *class TabMenu* — комбинация из лейблов, позволяющих имитировать переключение между вкладками.

- *class SolutionBox* — комбинация из поля для отображения текста и двух объектов типа *InfoBox*. Данный композитный виджет предназначен для отображения решений в виде произведения матриц.

На более высоком логическом уровне находятся виджеты описанные выше в разделе «Создание прототипа». В них используются как композитные виджеты, так и методы классов *...Utils*.

Для того, чтобы координировать создание и работу виджетов, описанных выше, был создан класс Visualizer, отвечающий за их инициализацию и хранение, а также предоставляющий функционал для ввода и вывода данных. Для того, чтобы избежать непосредственного обращения к индивидуальным виджетам, а также для того, чтобы унифицировать имеющиеся возможности ввода и вывода, над классом Visualizer было создано два класса-фасада: InputFacade и OutputFacade. Их методы позволяют осуществить все необходимые действия, касающиеся ввода и вывода.

Для удобного и простого доступа к данным задачи был создан класс DataStorage. В нем хранится набор размеров матриц, а также последние три популяции, полученные в ходе действия алгоритма.

Организация генетического алгоритма.

Для создания *популяции-0* из случайных индивидов созданы функции *generate_individual*, принимающая на вход размер индивида, и *generate_population*, принимающая на вход размер индивида и их количество.

Для оценки приспособленности индивидов создана функция *evaluate*, принимающая на вход индивида. Функция оценивает индивида (вычисляет количество операций умножения при определенном им порядке операций матричного умножения) на основании массива *dimensions*, хранящего размеры матриц.

Для выборки индивидов для скрещивания создана функция *roulette_selection*, принимающая на вход популяцию, значения функции приспособленности для каждого из индивидов и размер выборки (в данной работе размер выборки равняется размеру популяции).

Для упорядоченного скрещивания индивидов создана функция *ordered_crossover*, принимающая на вход двух скрещиваемых индивидов.

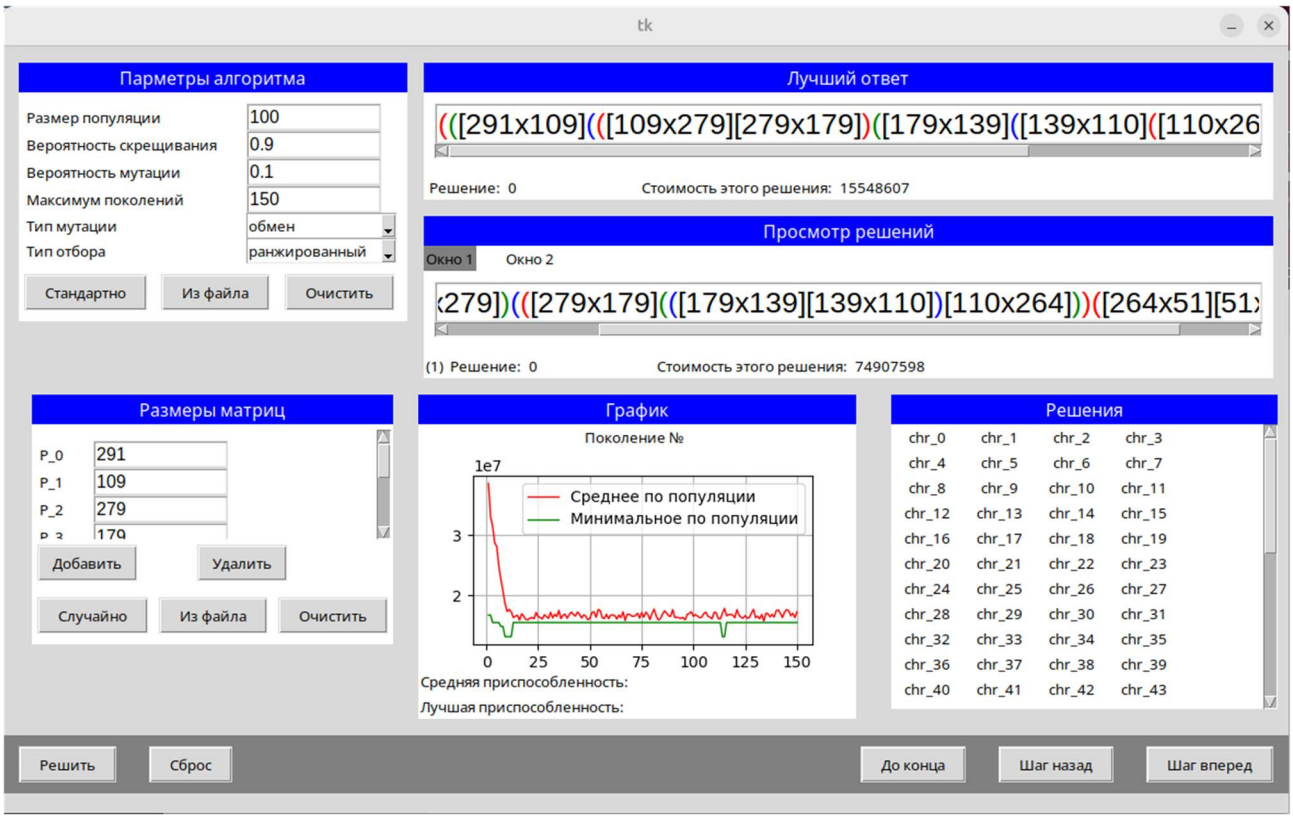
Созданы 3 функции для мутации индивидов: *mutate_swap*, *mutate_reverse* и *mutate_shuffle* для мутаций обменом, обращением и перетасовкой соответственно. Для выбора типа мутации используется словарь *mutations*.

Реализован основной цикл генетического алгоритма, программа возвращает лучшее решение.

(IV итерация)

Финальная версия GUI.

При создании финальной версии графического интерфейса были учтены комментарии от преподавателя: в частности виджеты для ввода данных были перенесены в левую часть окна, кнопки запуска решений и контроля шагов алгоритма были поменяны местами, были изменены некоторые подписи других кнопок и добавлены новые поля для отображения информации (например, средней и лучшей приспособленности поколения). Также в финальную версию GUI был добавлен класс для независимого просмотра имеющихся решений, отличных от лучшего. Новое окно имеет два вьюпорта для того, чтобы можно было сравнить два решения.



Финальная версия вычислительного блока.

Были внесены изменения в программе для возможности управления извне, произведена подготовка для интеграции с графическим интерфейсом.

Все функции и переменные были обернуты в один класс *Solver*. Класс в конструкторе принимает параметры генетического алгоритма.

Добавлена возможность работы алгоритма в пошаговом режиме – метод *advance()*.

Реализованы методы для вычисления средней и лучшей приспособленности – *get_avg()* и *get_best()* соответственно. В методе *advance()* сохраняются значения средней и лучшей приспособленности в *avg_all_gens* и *best_all_gens* соответственно. Для легкого доступа к текущему лучшему решению создан метод *get_best_index()*.

Выборка методом рулетки заменена на ранжированную выборку, т. к. с ней проще минимизировать функцию приспособленности. Сама выборка была оптимизирована бинарным поиском (до этого применялся линейный).

Для отката к предыдущим поколениям создан метод *set_gen(gen_number, population, avg_all_gens, best_all_gens)*, где *gen_number* – номер поколения, к которому происходит откат, *population* – популяция в этом поколении, *avg_all_gens, best_all_gens* – данные для графиков.

Объединение алгоритмической части и интерфейса

Был добавлен класс *Application* – класс приложения. Класс имеет следующие поля:

- *self.data_storage* - объект класса *DataStroage*, хранящий информацию о текущей популяции решений,

- *self.viz* — объект класса *Visualizer*, отвечающий за работу GUI,

- *self.input_f* — объект класса *InputFacade*, отвечающий за настройку пользовательского ввода,

- *self.output_f* — объект класса *OutputFacade*, отвечающий за настройку вывода информации,

- *self.solver* — объект класса *Solver*, отвечающего за реализацию нужной задачи с помощью генетического алгоритма.

Запускается приложение методом *start(),* после чего инициализируется визуализатор и дальнейшее поступление данных в программу будет производиться именно через пользовательский интерфейс.

Данные, полученные интерфейсом, такие как параметры алгоритма и размеры матриц, проходят валидацию в приложении и преобразуются для вида, подходящего для части с алгоритмом. После нажатия кнопки «решить», вся информация с полей собирается и, после валидации, поступает в алгоритм. Если какие-то данные не прошли валидацию, программа не запускает решение дальше и предупреждает пользователя, показывая соответствующее уведомление. Начинается решение задачи с полученными данными. Ввод данных для алгоритма после этого отключается на время решения.

Всем кнопкам на GUI отвечает соответствующий метод приложения, который дает сигнал алгоритму сделать определенное действие, а полученный

результат передает обратно в графический интерфейс (обновляет информацию, показывающуюся пользователю).

Были реализованы блоки с исключениями для валидации вводимых данных.

Таким образом в классе приложения были были объединены все компоненты, написанные ранее. Все полученные от пользователя данные передаются в алгоритм, проходя предварительную валидацию, а из алгоритма полученные данные передаются обратно и отображаются в удобном для пользователя виде.