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

# ОТЧЕТ

**по учебной практике**

**Тема: Генетические алгоритмы**

Кузнецова С.Е.
Максимова Е.Д.
Студентки гр. 3341 _____ Чинаева М.Р.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2025

**Цель работы.**

Настройка связи генетического алгоритма с GUI, полная реализация функционала, тестирование

**Задание №18.**

Задача о клике.

В заданном графе G = (V,E) необходимо найти клику максимального размера.

**Распределение ролей в бригаде.**

Максимова Екатерина: Реализация GUI

Чинаева Маргарита: Реализация основного алгоритма и подготовка для связи с GUI

Кузнецова Светлана: Загрузка и выгрузка данных и вспомогательные функции для основного алгоритма (проверка смежности, прохождение по соседям, проверка подграфа, является ли он кликой и т.п.) + генерация случайных данных

**Описание текущей реализации генетического алгоритма:**

По итогам тестирования в алгоритм были внесены изменения:

1. Теперь при любом выборе случайного элемента из множества элементов с заданными весами, веса сначала масштабируются, аналогично выбору родителя. Например, ранее при генерации начальной популяции веса вершин были равны их степеням, что приводило к слишком частому выбору больших вершин и на тестах вершины с небольшой степенью почти никогда не выбирались

2. Изменился алгоритм отбора в новую популяцию, ранее после совместной сортировки потомков и родителей выбирался первый элемент из отсортированного списка, это приводило к тому, что, так как сортировка устойчивая, то всегда выбирался один и тот же элемент. Это вредило

алгоритму, ведь далее мы выбираем хромосому с наибольшим расстоянием хемминга от уже выбранных, а уже выбранная хромосома могла быть близка к тем, которые входят в наибольшую клику. Также теперь выбирается не просто последний элемент с наибольшим расстоянием от уже выбранных, а случайный элемент из множества элементов с равным друг другу наибольшим расстоянием до уже выбранных.

Также была произведена связь алгоритма и гуи, для этого в классе MainApp были добавлены или обновлены следующие методы:

*update_graph(self, adj_matrix: np.ndarray) -> None* – обновляет текущий граф приложения на основе переданной матрицы смежности. Передает граф в менеджер алгоритмов, автоматически сбрасывает состояние алгоритма и адаптирует параметры по умолчанию в соответствии с размером нового графа.

*run(self, n: int) -> None* – основной метод выполнения генетического алгоритма. Поддерживает три режима работы:

n=1: выполнение одного поколения

n>1: выполнение N поколений

n=-1: выполнение до завершения алгоритма

Перед запуском проверяет наличие графа и сбрасывает алгоритм при изменении параметров. Загружает параметры из интерфейса, выполняет алгоритм через AlgorithmManager, обновляет интерфейс (список решений, график фитнеса, визуализацию клики) и обрабатывает ошибки выполнения.

*run_algorithm(self) -> None* – запускает одно поколение генетического алгоритма. Вызывает основной метод выполнения с параметром n=1.

*step_algorithm(self)* -> None – выполняет пять последовательных поколений генетического алгоритма. Вызывает основной метод выполнения с параметром n=5.

*end_algorithm(self)* -> None – запускает выполнение генетического алгоритма до его полного завершения (до достижения критериев остановки).

Вызывает основной метод выполнения с параметром n=-1. Автоматически определяет момент завершения вычислений и фиксирует итоговый результат.

*reset_algorithm(self) -> None* – полностью сбрасывает состояние алгоритма и связанные элементы интерфейса.

Также были рассчитаны и добавлены значения по умолчанию, которые лучше остальных показали себя на тестах:

*population_size* – количество вершин, деленное на 10, но не менее 5 и не более 30

*max_generations* – количество вершин, умноженное на 3, но не менее 150 и не более 500

*stagnation_limit* –в 10 раз меньше, чем максимальное количество итераций, но не менее 20 и не более 50

*max_mutation_prob_gene* - 0,1

*max_mutation_prob_chrom* – 0,4, если в популяции меньше 8 особей и 0,3, если больше

*fitness_scaling_percent* –по умолчанию 40

*max_crossover_points* - количество вершин, деленное на 5, но не менее 3 и не больше количества вершин

*decrease_percent* – 20

*decrease_step* – 20

**Обработка и хранение данных, вспомогательные функции**

Были реализован класс, отвечающий за связь генетического алгоритма с GUI, между различными компонентами, хранение графа, параметров и промежуточных данных и работы с ними, а также некоторые вспомогательные функции

1. Класс *AlgoritmManager* – отвечает за координацию работы генетического алгоритма, управление состоянием, работу с графами, параметрами и промежуточными данными

Поля класса:

*Optional[Graph] graph* – граф, хранящийся во время работы алгоритма (объект класса Graph)

*Optional[Parameters] params* – параметры, использующиеся для работы алгоритма (объект класса Parameters)

*Optional[GeneticAlgorithm] algorithm* – объект класса GeneticAlgorithm, в котором реализуется работа генетического алгоритма

*Optional[History] history* – объект класса History, который хранит историю работы алгоритма (лучшую и среднюю приспособленность на каждом шаге)

*bool is_initialized* – флаг, указывающий на то, готов ли алгоритм к выполнению (введены ли данные: параметры и граф)

*bool is_completed* – флаг, указывающий на то, завершил ли алгоритм работу (достиг ли условия остановки (макс. поколений/застой))

Основные методы:

*load_graph_from_matrix(self, file_path: str) -> None* – загружает граф из JSON-файла с матрицей смежности

*generate_random_graph(self, n: int) -> None* – генерирует случайный неориентированный граф с n вершинами

*save_graph_as_matrix(self, file_path: str) -> None* – сохраняет текущий граф в JSON-файл как матрицу смежности

*set_graph(self, graph: Graph) -> None* – устанавливает граф для алгоритма

*set_parameters(self, params: Parameters) -> None* – устанавлиавет параметры алгоритма

*load_and_validate_parameters(self, data: dict) -> None* – проверяет параметры на корректность. Использует graph.n для проверки параметров

*_check_initialization(self) -> None* – проверяет, можно ли инициализировать алгоритм

*initialize_algorithm(self) -> None* – инициализирует алгоритм с текущими графом и параметрами

*_check_ready(self) -> None* – проверяет, готов ли алгоритм к выполнению

*step(self) -> Tuple[List[int], List[List[int]]]* – выполняет одну итерацию алгоритма

*step_n(self, n: int = 5) -> Tuple[List[int], List[List[int]]]* – выполняет N итераций алгоритма

*run_until_completion(self) -> Tuple[List[int], List[List[int]]]* – выполняет алгоритм до завершения

*_get_current_state(self) -> Tuple[List[int], List[List[int]]]* – возвращает текущее состояние алгоритма

*plot_history(self) -> plt.Figure* – строит график эволюции на основе истории

*reset_algorithm(self) -> None* – Сбрасывает алгоритм до начального состояния. Сохраняет текущие граф и параметры

Методы класса:

*record(self, generation: int, population: Population)* – Пересчитывает параметры популяции и сохраняет их в историю

*plot_fitness(self)* – Строит график динамики лучшей и средней приспособленности по поколениям

*save_to_json(self, path: str)* – Сохраняет историю работы алгоритма в результирующий json-файл

Были обновлены функции GUI для загрузки из JSON-файла графа, заданного в виде матрицы смежности и передаче его менеджеру для дальнейшей работы, и сохранения графа в JSON-файл в виде матрицы смежности

Были обновлены функции из класса *MainApp*:

*load_adjacency_matrix(self)* – загрузка матрицы смежности из файла

*save_graph(self)* – сохранение графа в файл

И проверка корректности матрицы, вводимой из файла с помощью функции *parse_matrix_from_file(filename)*

Также, была реализована установка параметров по умолчанию в GUI в зависимости от количества вершин графа с использованием класса *ParameterConfig* и функции *update_defaults_based_on_graph_size(cls, graph_size)*

**Разработка GUI**

Была проведена подготовка к связыванию GUI с алгоритмом. Предыдущие классы были разделены на другие классы, чтобы упростить расширение и изменение системы и ее связи алгоритмом.

Текущий набор классов:

1. GraphVisualizer — класс отрисовки графа.
2. LeftPanel — левая панель с параметрами, кнопками для вызова установки по умолчанию и вызова окна ввода матрицы соответственно.
3. ParameterConfig — класс для работы с параметрами по умолчанию.
4. FitnessPlotWidget — класс для отрисовки графика функции приспособленности.
5. SolutionListWidget — класс для отображения текущей популяции.
6. MatrixWindow — класс окна ввода матрицы.
7. ZoomableWidget — класс для создания способности приближать/отдалять виджеты.
8. FileManager — класс для работы с файлами.
9. Validator — класс проверок данных.
10. UIManager — класс унифицированного создания визуальных элементов интерфейса (вызовов окна ошибки, информации и т. д.)
11. Styles и Colors — стили и цвета для элементов интерфейса соответственно.

12. MainApp — класс основного окна, связывает остальные его визуальные блоки, получает и передает на визуализацию результаты работы алгоритма.

Описанные классы состоят из ранее реализованных методов. Изменены или добавлены методы получения данных, их проверки и изменения для получения от алгоритма или для передачи ему.

Изменена некоторая часть визуального отображения данных: убрано подсвечивание лучшего решения, оно всегда находится на вершине списка решений, при загрузке и сохранении по умолчанию установлено отображение файлов формата json.