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:

Класс *GeneticAlgorithm* - реализует генетический алгоритм для поиска максимальной клики в графе. Использует эволюционные методы для нахождения оптимального решения задачи о максимальной клике.

Поля класса:

Graph graph - Граф, в котором ищется максимальная клика

Parameters params - Параметры работы генетического алгоритма

float current_mutation_prob_chrom - Текущая вероятность мутации хромосомы

float current_mutation_prob_gene - Текущая вероятность мутации отдельного гена

int current_crossover_points - Текущее количество точек кроссовера

int generation - Номер текущего поколения

int stagnation_count - Счетчик поколений без улучшения лучшего решения

int best_fitness - Найденный максимальный размер клики

list[int] best_chromosome - Лучшая найденная хромосома

int n - Количество вершин в графе

int max_degree_original - Максимальная степень вершины в исходном графе

Основные методы:

__init__(self, graph: Graph, params: Parameters) – Инициализирует алгоритм с заданным графом и параметрами, выполняет преобразование графа и генерацию начальной популяции.

generate_chromosome(self) -> List[int] – Генерирует случайную хромосому, представляющую клику, использует жадный алгоритм с случайным выбором вершин среди вершин одной степени.

generate_initial_population(self) -> List[Individual] – Создает начальную популяцию заданного размера.

select_parents(self) -> List[Individual] – Выбирает родителей для скрещивания методом рулетки с масштабированием приспособленности.

crossover(parent1, parent2) -> Tuple[List[int], List[int]] - Выполняет кроссовер с несколькими точками разрыва, попеременно копируя сегменты хромосом родителей.

mutate_and_repair(chromosome) -> List[int] – Применяет мутации и восстанавливает хромосому до валидной клики.

select_new_population(current_pop, offspring) -> List[Individual] – Формирует новую популяцию с сохранением разнообразия, выбирая особи, максимально отличающиеся от уже выбранных.

_update_best_solution() – Обновляет лучшее решение при обнаружении улучшения.

_reduce_parameters() – Уменьшает параметры алгоритма на заданный процент от начальных значений.

should_stop() -> bool – Проверяет условия остановки алгоритма.

next_generation() – Выполняет одну полную итерацию генетического алгоритма:

1. Выбор родителей
2. Генерация потомков
3. Формирование новой популяции
4. Обновление лучшего решения и списка поколений
5. Уменьшение параметров (при необходимости)

get_best_solution() -> Tuple[int, List[int]] – Возвращает лучшее решение в исходной нумерации вершин.

_hamming_distance() – Вычисляет нормализованное расстояние Хэмминга

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 – устанавлиает 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.

ПРИЛОЖЕНИЕ А

ТЕСТИРОВАНИЕ АЛГОРИТМА

graph1.json:


```
{
    "0": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    "1": [0, 2, 3, 4, 5, 6, 7, 11, 12],
    "2": [0, 1, 3, 4, 5, 6, 7, 13, 14],
    "3": [0, 1, 2, 4, 5, 6, 7, 15],
    "4": [0, 1, 2, 3, 5, 6, 7, 16],
    "5": [0, 1, 2, 3, 4, 6, 7, 17],
    "6": [0, 1, 2, 3, 4, 5, 7, 18],
    "7": [0, 1, 2, 3, 4, 5, 6, 19],
    "8": [0, 9, 10, 11, 20, 21],
    "9": [0, 8, 12, 13, 20, 21],
    "10": [0, 8, 14, 15, 20],
    "11": [1, 8, 16, 17, 20],
    "12": [1, 9, 18, 19, 20],
    "13": [2, 9, 16, 17, 20],
    "14": [2, 10, 18, 19, 20],
    "15": [3, 10, 16, 18, 20],
    "16": [4, 11, 13, 15, 20],
    "17": [5, 11, 13, 19, 20],
    "18": [6, 12, 14, 15, 20],
    "19": [7, 12, 14, 17, 20],
    "20": [8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
    "21": [8, 9, 22],
    "22": [21, 23],
    "23": [22, 24],
    "24": [23, 25],
    "25": [24, 26],
    "26": [25]
}
```


Вывод при тесте:

```
Начало работы генетического алгоритма
Размер графа: 27 вершин
Параметры алгоритма:
  Размер популяции: 5
  Макс. поколений: 100
  Лимит застоя: 20
  Начальные точки кроссовера: 5

Начальная популяция:
  Особа 1: 01111111100000000000000000 (размер: 8)
  Особа 2: 01111111100000000000000000 (размер: 8)
  Особа 3: 10000000000010000010000000 (размер: 3)
  Особа 4: 01111111100000000000000000 (размер: 8)
  Особа 5: 10000000001001000000000000 (размер: 3)

Поколение 1:
  Лучший размер клики: 8
  Застой: 1 / 20
  Параметры: точки=5, мутация хромосомы=0.300, мутация гена=0.100
  Текущая популяция:
```

```
    Особа 1: 01111111100000000000000000 (размер: 8)
    Особа 2: 10000000000010000010000000 (размер: 3)
    Особа 3: 00000101100000000000000000 (размер: 3)
    Особа 4: 10000000001001000000000000 (размер: 3)
    Особа 5: 10000000000000100010000000 (размер: 3)
  Лучшая клика: вершины [0, 1, 2, 3, 4, 5, 6, 7]

Поколение 2:
  Лучший размер клики: 8
  Застой: 2 / 20
  Параметры: точки=5, мутация хромосомы=0.300, мутация гена=0.100
  Текущая популяция:
    Особа 1: 01111111100000000000000000 (размер: 8)
    Особа 2: 10000000000010000010000000 (размер: 3)
    Особа 3: 00000101100000000000000000 (размер: 3)
    Особа 4: 10000000001001000000000000 (размер: 3)
    Особа 5: 10000000000000100010000000 (размер: 3)
  Лучшая клика: вершины [0, 1, 2, 3, 4, 5, 6, 7]

Поколение 3:
  Лучший размер клики: 8
  Застой: 3 / 20
  Параметры: точки=5, мутация хромосомы=0.300, мутация гена=0.100
  Текущая популяция:
    Особа 1: 01111111100000000000000000 (размер: 8)
    Особа 2: 10000000000010000010000000 (размер: 3)
    Особа 3: 00000101100000000000000000 (размер: 3)
    Особа 4: 10000000001001000000000000 (размер: 3)
    Особа 5: 00000000000000000000000010 (размер: 1)
  Лучшая клика: вершины [0, 1, 2, 3, 4, 5, 6, 7]

…
Сокращение вывода
…

Поколение 17:
  Лучший размер клики: 8
  Застой: 17 / 20
  Параметры: точки=4, мутация хромосомы=0.270, мутация гена=0.090
  Текущая популяция:
    Особа 1: 01111111100000000000000000 (размер: 8)
    Особа 2: 10000000000010000010000000 (размер: 3)
    Особа 3: 00000101100000000000000000 (размер: 3)
    Особа 4: 01000000000100000000000000 (размер: 2)
    Особа 5: 10000000001001000000000000 (размер: 3)
  Лучшая клика: вершины [0, 1, 2, 3, 4, 5, 6, 7]

Поколение 18:
  Лучший размер клики: 8
  Застой: 18 / 20
  Параметры: точки=4, мутация хромосомы=0.270, мутация гена=0.090
  Текущая популяция:
    Особа 1: 01111111100000000000000000 (размер: 8)
    Особа 2: 10000000000010000010000000 (размер: 3)
    Особа 3: 00000101100000000000000000 (размер: 3)
    Особа 4: 01000000000100000000000000 (размер: 2)
    Особа 5: 10000000001001000000000000 (размер: 3)
  Лучшая клика: вершины [0, 1, 2, 3, 4, 5, 6, 7]
```

```
Поколение 19:
  Лучший размер клики: 8
  Застой: 19 / 20
  Параметры: точки=4, мутация хромосомы=0.270, мутация гена=0.090
  Текущая популяция:
    Особа 1: 01111111100000000000000000 (размер: 8)
    Особа 2: 10000000000010000100000000 (размер: 3)
    Особа 3: 00000101100000000000000000 (размер: 3)
    Особа 4: 01000000000100000000000000 (размер: 2)
    Особа 5: 10000000001001000000000000 (размер: 3)
  Лучшая клика: вершины [0, 1, 2, 3, 4, 5, 6, 7]

Поколение 20:
  Лучший размер клики: 8
  Застой: 20 / 20
  Параметры: точки=3, мутация хромосомы=0.240, мутация гена=0.080
  Текущая популяция:
    Особа 1: 01111111100000000000000000 (размер: 8)
    Особа 2: 10000000000010000100000000 (размер: 3)
    Особа 3: 00000101100000000000000000 (размер: 3)
    Особа 4: 01000000000100000000000000 (размер: 2)
    Особа 5: 10000000001001000000000000 (размер: 3)
  Лучшая клика: вершины [0, 1, 2, 3, 4, 5, 6, 7]


Результат работы алгоритма:
Найдена клика размера 8
Вершины в клике: [0, 1, 2, 3, 4, 5, 6, 7]
Всего поколений: 20
Клика валидна
Причина остановки: Превышен лимит застоя
```