**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)**

Факультет «Информатика и системы управления»

Кафедра «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 12

по курсу « Методы оптимизации »

**ГЕНЕТИЧЕСКИЙ АЛГОРИТМ ДЛЯ МАТРИЦ 3х3**

Студент: Яровикова А. С.

Группа: ИУ9-81Б

Преподаватель: Посевин Д. П.

Москва, 2024

СОДЕРЖАНИЕ

[ЦЕЛЬ И ПОСТАНОВКА ЗАДАЧИ 3](#_Toc158799284)

[ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ 3](#_Toc158799285)

[РЕЗУЛЬТАТЫ 6](#_Toc158799286)

[ВЫВОДЫ 7](#_Toc158799287)

# **ЦЕЛЬ И ПОСТАНОВКА ЗАДАЧИ**

**Цель работы:**

Реализовать генетический алгоритм и проверить его работоспособность на матрицах размером 3x3.

**Постановка задачи:**

1. Исследовать генетический алгоритм.
2. Реализовать метод на языке программирования Python, включая функции для вычисления фитнеса, методов кроссовера и мутации.

# **ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ**

Импорт библиотек:

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**import** **random**

Функция отрисовки матрицы:

**def** **draw\_image**(image\_mat):

plt.imshow(image\_mat, cmap='gray')

plt.axis('on')

plt.show()

Генерация случайной матрицы:

target\_image = np.random.randint(**2**, size=(**3**, **3**))

print(target\_image)

draw\_image(target\_image)

Инициализация параметров генетического алгоритма:

population\_size = **100** # Размер популяции

mutation\_rate = **0.01** # Вероятность мутации

number\_of\_generations = **200** # Количество поколений

tournament\_size = **50** # Размер турнира для селекции

elite\_size = **20** # Размер группы лучших особей

Функция вычисления фитнес-оценки (приспособленности):

**def** **calculate\_fitness**(individual):

difference = np.sum(np.abs(individual - target\_image))

fitness = **1** / (difference + **1**) # Добавляем 1, чтобы избежать деления на ноль

**return** fitness

Функция селекции:

**def** **tournament\_selection**(population, tournament\_size):

tournament = random.sample(population, tournament\_size)

fitness\_scores = [calculate\_fitness(individual) **for** individual **in** tournament]

winner\_index = np.argmax(fitness\_scores)

**return** tournament[winner\_index]

Функция кроссовера:

**def** **single\_point\_crossover**(parent1, parent2):

**if** random.random() < **0.5**: # Горизонтальный кроссовер

crossover\_point = random.randint(**1**, parent1.shape[**0**]-**1**)

offspring1 = np.vstack((parent1[:crossover\_point, :], parent2[crossover\_point:, :]))

offspring2 = np.vstack((parent2[:crossover\_point, :], parent1[crossover\_point:, :]))

**else**: # Вертикальный кроссовер

crossover\_point = random.randint(**1**, parent1.shape[**1**]-**1**)

offspring1 = np.hstack((parent1[:, :crossover\_point], parent2[:, crossover\_point:]))

offspring2 = np.hstack((parent2[:, :crossover\_point], parent1[:, crossover\_point:]))

**return** offspring1, offspring2

Функция мутации:

**def** **mutate**(individual, mutation\_rate):

**for** i **in** range(individual.shape[**0**]):

**for** j **in** range(individual.shape[**1**]):

**if** random.random() < mutation\_rate:

individual[i, j] = **1** - individual[i, j]

**return** individual

Генетический алгоритм:

# Инициализация популяции случайными матрицами

population = [np.random.randint(**2**, size=target\_image.shape) **for** \_ **in** range(population\_size)]

best\_individuals = []

**for** generation **in** range(number\_of\_generations):

# Сортировка популяции по фитнес-оценке

population.sort(key=calculate\_fitness, reverse=**True**)

best\_individual = population[**0**]

best\_individuals.append(best\_individual)

# Сохранение лучших особей

next\_generation = population[:elite\_size]

# Выбор и кроссовер особей для заполнения следующего поколения

**while** len(next\_generation) < population\_size:

parent1 = tournament\_selection(population, tournament\_size)

parent2 = tournament\_selection(population, tournament\_size)

offspring1, offspring2 = single\_point\_crossover(parent1, parent2)

next\_generation.append(mutate(offspring1, mutation\_rate))

**if** len(next\_generation) < population\_size:

next\_generation.append(mutate(offspring2, mutation\_rate))

population = next\_generation

# Отслеживание лучшей особи

**if** calculate\_fitness(population[**0**]) > calculate\_fitness(best\_individual):

best\_individual = population[**0**]

# Вывод прогресса

print(f'Поколение {generation+1}/{number\_of\_generations}, Лучшая приспособленность: {calculate\_fitness(best\_individual):.4f}')

Отображение наилучшей особи генетического алгоритма:

plt.imshow(best\_individual, cmap='gray')

plt.show()

# **РЕЗУЛЬТАТЫ**

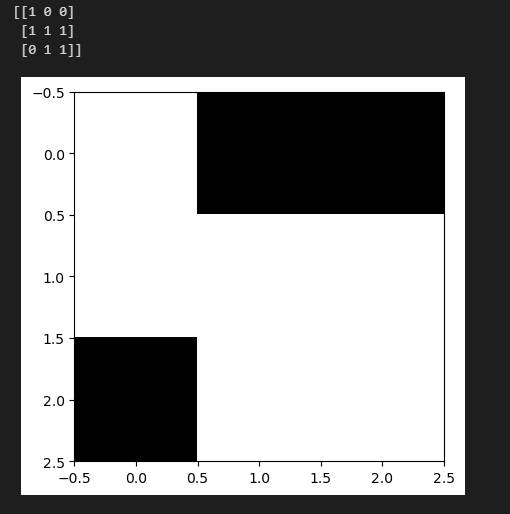


Рисунок 1 – Сгенерированная начальная случайная матрица

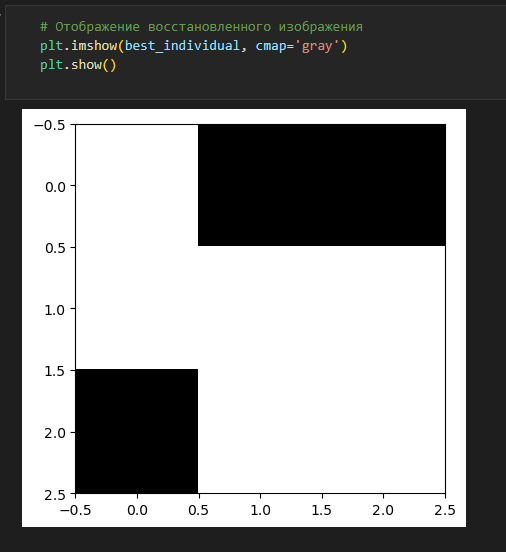


Рисунок 2 – Наилучшая особь последнего поколения

# **ВЫВОДЫ**

В ходе выполнения лабораторной работы был генетический алгоритм. Работоспособность реализации проверена на случайных матрицах размером 3x3. Результаты тестирования подтверждают высокую точность алгоритма.