ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

профессор		Ю.А. Скобцов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №8

Эволюционные алгоритмы оценки стоимости проектов в программной инженерии

По дисциплине: Эволюционные методы проектирования программно-информационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР.	4134к		Столяров Н.С.
		подпись, дата	инициалы, фамилия

Санкт-Петербург
2024

Цель работы:
разработка эволюционного алгоритма оценки стоимости программных проектов. Графическое отображение результатов.

Вариант:

7	ГП	Дерево	RMS	поддеревьев	Усекающ. Растущ.	ранговый

Задание:
1. Разобраться в теоретическом описании математического метода оценки стоимости программного проекта – модели COCOMO.
2. Из приведенной выше табл. 8.1 (или табл. 8.2) экспериментальных данных (программных проектов НАСА) отобрать из 18 проектов в качестве обучающего множества 13 (40) проектов.
3. В соответствии с вариантом лабораторной работы, заданного табл. 8.3 определить тип используемого эволюционного алгоритма (генетический или роевой алгоритм, генетическое программирование), кодирование потенциального решения, вид ошибки в целевой функции, вид генетических операторов кроссовера, мутации и репродукции
4. Отработать алгоритм решения задачи с помощью заданного метода на обучающем множестве.
5. Разработать программу на языке Python, включающую в себя реализацию пользовательского интерфейса в виде диалогового меню, реализацию алгоритма решения поставленной задачи заданным методом.
6. Протестировать разработанную программу: вычислить заданный тип ошибки на тестовом множестве – оставшихся 5 (из 18) проектов табл. 8.1 (или табл. 8.2).
5. Выполнить вывод полученного решения в виде текста и графиков.

Выполнение:
Ядром модели является следующая формула Ef=aL^b, где L – длина кода ПО в килостроках; Ef – оценка сложности проекта в человеко-месяцах; a и b – коэффициенты (параметры) модели, которые для различных типов ПО имеют различные значения.

Экспериментальные данные проектов НАСА

Номер проекта	L	Me	Ef	Efm	Efm2
1	90,2000	30,0000	115,8000	124,8585	134,0202
2	46,2000	20,0000	96,0000	74,8467	84,1616
3	46,5000	19,0000	79,0000	75,4852	85,0112
4	54,5000	20,0000	909,8000	85,4349	94,9828
5	31,1000	35,0000	39,6000	50,5815	56,6580
6	67,5000	29,000	98,4000	99,0504	107,2609
7	12,8000	26,000	18,9000	24,1480	32,6461
8	10,5000	34,0000	10,3000	18,0105	25,0755
9	21,5000	31,0000	28,5000	37,2724	44,3086
10	3,1000	26,000	7,0000	4,5849	14,4563
11	4,2000	19,0000	9,0000	8,9384	19,9759
12	7,8000	31,0000	7,3000	13,5926	21,5763
13	2,1000	28,0000	5,0000	1,5100	11,2703
14	5,0000	29,0000	8,4000	8,2544	17,0887
15	78,6000	35,0000	98,7000	110,5249	118,0378
16	9,7000	27,0000	15,6000	18,2559	26,8312
17	12,5000	27,0000	23,9000	23,3690	31,6864
18	100,8000	34,0000	138,3000	135,4825	144,4587

13 первых проектов — обучающее множество, 5 последних — тестовое множество. Отработать алгоритм решения задачи с помощью заданного метода на обучающем множестве и применить на тестовом. Тип эволюционного алгоритма – Генетическое программирование (ГП). Кодирование решения – Деревья операций (генетическое дерево). Фитнесс-функция (тип ошибки) – RMS (Root Mean Square Error). Оператор кроссовера – Кроссовер поддеревьев. Оператор мутации – Замена узлов и поддеревьев. Оператор репродукции – Ранговая селекция. Выполнить вывод полученного решения в виде текста и графиков. Графики: 1. С фактическими и предсказанными значениями для обучающего множества. 2. С фактическими и предсказанными значениями для тестового множества.

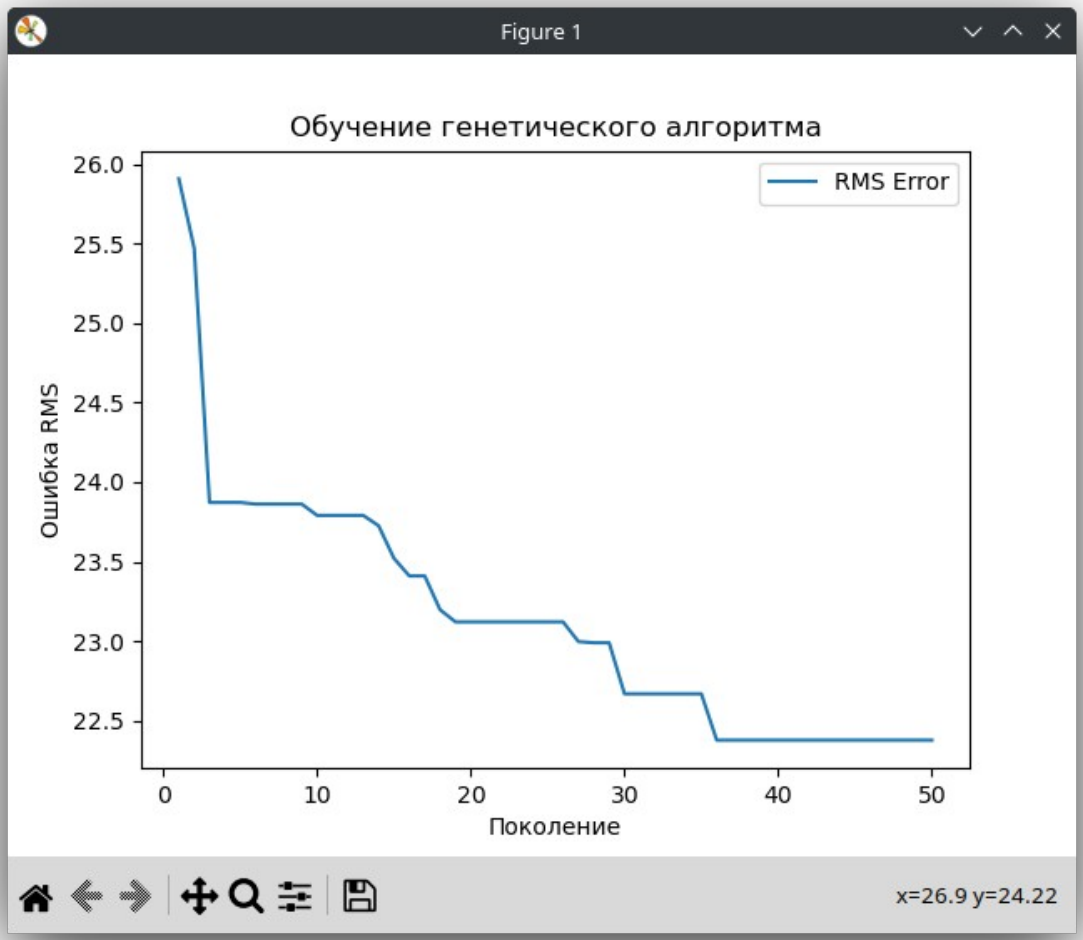
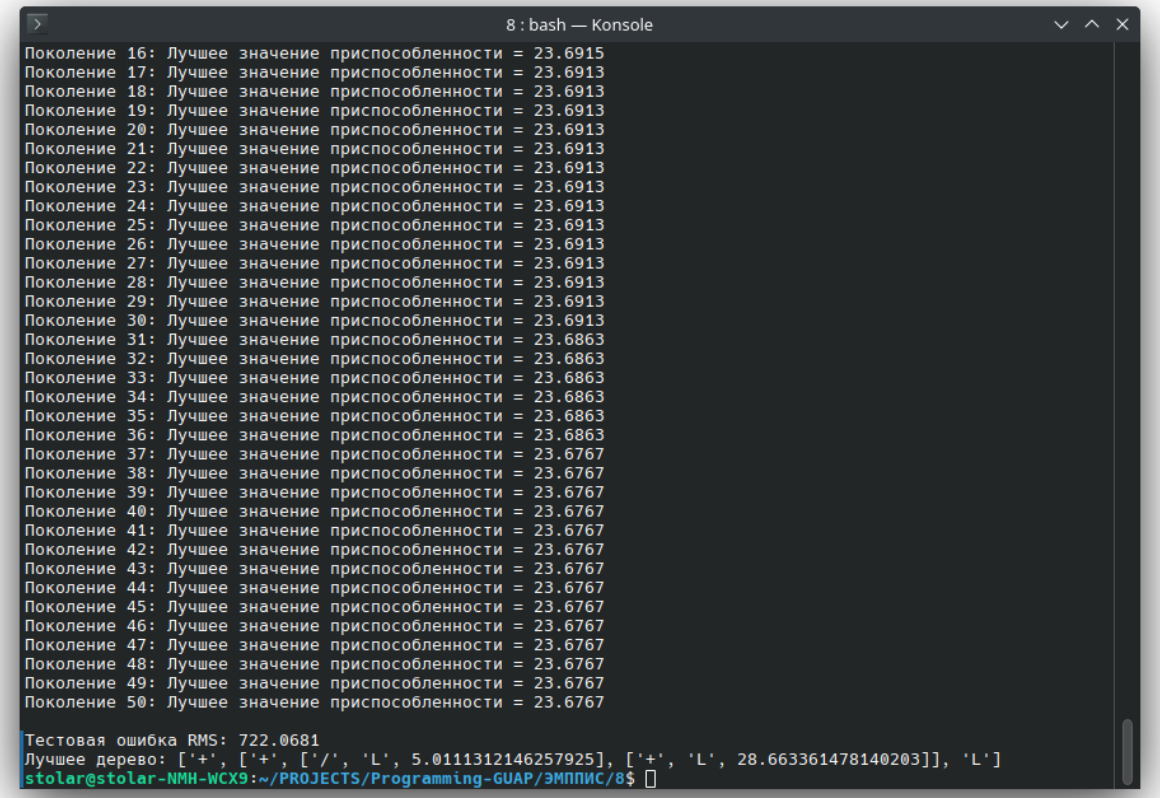


Figure 1
Обучение генетического алгоритма
RMS Error
Ошибка RMS
Поколение
26.0
25.5
25.0
24.5
24.0
23.5
23.0
22.5
0
10
20
30
40
50
x=26.9 y=24.22


8 : bash — Konsole
Поколение 16: Лучшее значение приспособленности = 23.6915
Поколение 17: Лучшее значение приспособленности = 23.6913
Поколение 18: Лучшее значение приспособленности = 23.6913
Поколение 19: Лучшее значение приспособленности = 23.6913
Поколение 20: Лучшее значение приспособленности = 23.6913
Поколение 21: Лучшее значение приспособленности = 23.6913
Поколение 22: Лучшее значение приспособленности = 23.6913
Поколение 23: Лучшее значение приспособленности = 23.6913
Поколение 24: Лучшее значение приспособленности = 23.6913
Поколение 25: Лучшее значение приспособленности = 23.6913
Поколение 26: Лучшее значение приспособленности = 23.6913
Поколение 27: Лучшее значение приспособленности = 23.6913
Поколение 28: Лучшее значение приспособленности = 23.6913
Поколение 29: Лучшее значение приспособленности = 23.6913
Поколение 30: Лучшее значение приспособленности = 23.6913
Поколение 31: Лучшее значение приспособленности = 23.6863
Поколение 32: Лучшее значение приспособленности = 23.6863
Поколение 33: Лучшее значение приспособленности = 23.6863
Поколение 34: Лучшее значение приспособленности = 23.6863
Поколение 35: Лучшее значение приспособленности = 23.6863
Поколение 36: Лучшее значение приспособленности = 23.6863
Поколение 37: Лучшее значение приспособленности = 23.6767
Поколение 38: Лучшее значение приспособленности = 23.6767
Поколение 39: Лучшее значение приспособленности = 23.6767
Поколение 40: Лучшее значение приспособленности = 23.6767
Поколение 41: Лучшее значение приспособленности = 23.6767
Поколение 42: Лучшее значение приспособленности = 23.6767
Поколение 43: Лучшее значение приспособленности = 23.6767
Поколение 44: Лучшее значение приспособленности = 23.6767
Поколение 45: Лучшее значение приспособленности = 23.6767
Поколение 46: Лучшее значение приспособленности = 23.6767
Поколение 47: Лучшее значение приспособленности = 23.6767
Поколение 48: Лучшее значение приспособленности = 23.6767
Поколение 49: Лучшее значение приспособленности = 23.6767
Поколение 50: Лучшее значение приспособленности = 23.6767
Тестовая ошибка RMS: 722.0681
Лучшее дерево: ['+', ['+', ['/', 'L', 5.0111312146257925], ['+', 'L', 28.663361478140203]], 'L']
stolar@stolar-NMH-WCX9:~/PROJECTS/Programming-GUAP/ЭМППИС/8$

Выводы:

В результате выполнения лабораторной работы был разработан эволюционный алгоритм на основе генетического алгоритма для оценки стоимости программных проектов, реализованный на языке Python. Проведенное тестирование на основе модели COCOMO показало приемлемую точность предсказаний, подтверждая эффективность применения эволюционных методов для решения задач оценки стоимости в программной инженерии.

Листинг

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random

data = pd.read_csv('data.csv')

# Параметры алгоритма
# population_size = 300
# generations = 50
# mutation_rate = 0.3
# train_percentage = 0.5
population_size = 100
generations = 30
mutation_rate = 0.1
train_percentage = 0.8

# Разделение данных на обучающее и тестовое множество на основе train_percentage
train_size = int(len(data) * train_percentage)
# train_size = 13
train_data = data.iloc[:train_size]
test_data = data.iloc[train_size:]

# Генерация случайного дерева
def generate_tree(depth=3):
    if depth == 0 or (depth > 1 and random.random() > 0.7):
        return random.choice(["L", "Me", random.uniform(1, 100)])
    operation = random.choice(["+", "-", "*", "/"])
    return [operation, generate_tree(depth - 1), generate_tree(depth - 1)]

# Оценка дерева
def evaluate_tree(tree, data_row):
    if isinstance(tree, (int, float)):
        return tree
    if isinstance(tree, str):
        return data_row[tree]
    operation, left, right = tree
    left_val = evaluate_tree(left, data_row)
    right_val = evaluate_tree(right, data_row)
    try:
        if operation == "+":
            return left_val + right_val
        elif operation == "-":
            return left_val - right_val
        elif operation == "*":
            return left_val * right_val
        elif operation == "/":
            return left_val / right_val if right_val != 0 else 1
    except:
        return 1
```

```
# RMS ошибка
# def fitness(tree, data_subset):
#     predictions = data_subset.apply(lambda row: evaluate_tree(tree, row), axis=1)
#     error = np.sqrt(np.mean((predictions - data_subset["Ef"]) ** 2))
#     return -error
def fitness(tree, data_subset):
    predictions = data_subset.apply(lambda row: evaluate_tree(tree, row), axis=1)
    error = np.sqrt(np.mean((predictions - data_subset["Ef"]) ** 2))
    complexity_penalty = count_tree_nodes(tree) * 0.01  # Штраф за сложность
    return -(error + complexity_penalty)

def count_tree_nodes(tree):
    if isinstance(tree, (int, float, str)):
        return 1
    return 1 + count_tree_nodes(tree[1]) + count_tree_nodes(tree[2])

# Кроссовер
def crossover(tree1, tree2):
    if isinstance(tree1, (int, float, str)) or isinstance(tree2, (int, float, str)):
        return tree1 if random.random() < 0.5 else tree2
    if not isinstance(tree1, list) or not isinstance(tree2, list):
        return tree1 if random.random() < 0.5 else tree2
    return [tree1[0] if random.random() < 0.5 else tree2[0],
            crossover(tree1[1], tree2[1]),
            crossover(tree1[2], tree2[2])]

# Мутация
def mutate(tree, depth=3):
    if isinstance(tree, (int, float, str)) or depth == 0:
        return generate_tree(depth) if random.random() < mutation_rate else tree
    if random.random() < mutation_rate:
        return generate_tree(depth)
    return [tree[0], mutate(tree[1], depth - 1), mutate(tree[2], depth - 1)]

# Ранговая селекция
def rank_selection(population, fitness_values):
      sorted_population = [x for _, x in sorted(zip(fitness_values, population), key=lambda pair: pair[0])]
    return random.choices(sorted_population[-10:], k=1)[0]
# Генетический алгоритм
def genetic_algorithm(train_data, test_data):
    population = [generate_tree() for _ in range(population_size)]
    best_tree = None
    best_fitness = -np.inf
    fitness_history_train = []
    fitness_history_test = []

    for gen in range(generations):
        # Оценка на обучающем множестве
        fitness_values = [fitness(tree, train_data) for tree in population]

        # Обновление лучшего решения
```

```
        if max(fitness_values) > best_fitness:
            best_fitness = max(fitness_values)
            best_tree = population[np.argmax(fitness_values)]

        # Сохранение ошибки для обучающего и тестового множеств
        fitness_history_train.append(-best_fitness)
        test_rms_error = -fitness(best_tree, test_data)
        fitness_history_test.append(test_rms_error)

        print(f"Поколение {gen + 1}: Ошибка на обучении = {-best_fitness:.4f}, Ошибка на тесте
= {test_rms_error:.4f}")

        # Формирование нового поколения
        new_population = []
        while len(new_population) < population_size:
            parent1 = rank_selection(population, fitness_values)
            parent2 = rank_selection(population, fitness_values)
            child = crossover(parent1, parent2)
            child = mutate(child)
            new_population.append(child)

        population = new_population

    # График обучения
    # plt.plot(range(1, generations + 1), fitness_history, label="RMS Error")
    # plt.xlabel("Поколение")
    # plt.ylabel("Ошибка RMS")
    # plt.title("Обучение генетического алгоритма")
    # plt.legend()

    plt.figure(figsize=(10, 6))
    plt.plot(range(1, generations + 1), fitness_history_train, label="Ошибка на обучении (RMS)",
color='b')
        plt.plot(range(1, generations + 1), fitness_history_test, label="Ошибка на тесте (RMS)",
color='r')
    plt.xlabel("Поколение")
    plt.ylabel("Ошибка RMS")
    plt.title("Динамика ошибок на обучающем и тестовом множестве")
    plt.legend()
    plt.grid()

    # Тестирование на тестовом множестве
    test_error = -fitness(best_tree, test_data)
    print(f"\nТестовая ошибка RMS: {test_error:.4f}")
    print(f"Лучшее дерево: {best_tree}")

    plt.show()

# Запуск алгоритма
genetic_algorithm(train_data, test_data)
```