МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №  43

ОТЧЁТ

ЗАЩИЩЁН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

### профессор                                   Скобцов Ю.А.

должность, уч. Степень, звание   подпись, дата           инициалы, фамилия

ОТЧЁТ О ЛАБОРАТОРНОЙ РАБОТЕ №8.

Эволюционные алгоритмы оценки стоимости проектов в программной инженерии.

по курсу: Эволюционные методы проектирования программно-информационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. 4136                                                                                Бобрович Н. С.

                                                                         подпись, дата                      инициалы, фамилия

Санкт-Петербург 2024

1. **Постановка задачи.**
2. Разобраться в теоретическом описании математического метода оценки стоимости программного проекта – модели СОСОМО.
3. Из приведённой табл. экспериментальных данных (программных проектов НАСА) отобрать из 18 проектов в качестве обучающего множества 13 (40) проектов.
4. В соответствии с вариантом лабораторной работы, заданного табл. 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.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № варианта | Тип эволюционного алгоритма | Кодирование решения | Фитнес-функция | Оператор кроссовера | Оператор мутации | Оператор репродукции |
| 3 | ГА | Двоичн.стр . | RMS | 2-точечный | Классическая | Турнир |

1. Отработать алгоритм решения задачи с помощью заданного метода на обучающем множестве.
2. Разработать программу на любом доступном вам языке программирования, включающую в себя реализацию пользовательского интерфейса в виде диалогового меню, реализацию алгоритма решения поставленной задачи заданным методом.
3. Протестировать разработанную программу: вычислить заданный тип ошибки на тестовом множестве – оставшихся 5 (из 18) проектов табл.
4. Выполнить вывод полученного решения в виде текста и графиков.
5. **Теоретические сведения.**

Одной из самых популярных моделей, используемых для оценки сложности проектируемого программного обеспечения (ПО) , является модель COCOMO (COnstructive COst Model), предложенная Boehm. Эта модель разработана на основе фактически статистики 63 проектов ПО (НАСА). Модель позволяет определить математическую зависимость между сложностью ПО, выраженную в килостроках кода, и затратами на его разработку, которые оцениваются в человеко-месяцах. 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. Основная проблема модель COCOMO заключается в том, что она не обеспечивает реальных оценок на затраты при проектировании ПО в современных условиях. Т.е. оценка программного обеспечения на основе существующих параметров не всегда дает точный результат; из-за этого часто требуется настройка параметров для получения более точных результатов. Поэтому в настоящее время идет активный поиск новых моделей (или развития и модификаций существующих). Это ограничение модели COCOMO можно преодолеть путем применения методов искусственного интеллекта, таких как искусственные нейронные сети, генетические алгоритмы и другие метаэвристики.

В данной лабораторной работе для определения значений коэффициентов a и b используются генетический или роевой алгоритм в соответствии с заданным вариантом. Фактически задача сводится к машинному обучению на заданной обучающей выборке. В этом случае обучающая выборка строится на основе следующей таблицы, которая дает реальные данные для 18 проектов НАСА, на основе которых мы ищем зависимость между L и Ef.

1. **Листинг программы.**

import numpy as np

import matplotlib.pyplot as plt

all\_projects = [

{"L": 90.2, "Ef": 115.8},

{"L": 46.2, "Ef": 96.0},

{"L": 46.5, "Ef": 79.0},

{"L": 54.5, "Ef": 909.8},

{"L": 31.1, "Ef": 39.6},

{"L": 67.5, "Ef": 98.4},

{"L": 12.8, "Ef": 18.9},

{"L": 10.5, "Ef": 10.3},

{"L": 21.5, "Ef": 28.5},

{"L": 3.1, "Ef": 7.0},

{"L": 4.2, "Ef": 9.0},

{"L": 7.8, "Ef": 7.3},

{"L": 2.1, "Ef": 5.0},

{"L": 5.0, "Ef": 8.4},

{"L": 78.6, "Ef": 98.7},

{"L": 9.7, "Ef": 15.6},

{"L": 12.5, "Ef": 23.9},

{"L": 100.8, "Ef": 138.3}

]

#Ошибку считаем MMRE среднее значение относительной погрешности

def fitness(a, b, projects):

errors = []

for project in projects:

errors.append(abs(project["Ef"] - a\*project["L"]\*\*b) / project["Ef"])

mmre = 1/len(projects) \* sum(errors)

return mmre

def correction\_speed(speed, position, personal\_best\_positions, global\_best\_position, c1=1.5, c2=1.5):

n = len(speed)

for i in range(n):

speed[i] += c1 \* np.random.rand() \* (personal\_best\_positions[i] - position[i]) + \

c2 \* np.random.rand() \* (global\_best\_position[i] - position[i])

if speed[i] < -3:

speed[i] = -3

if speed[i] > 3:

speed[i] = 3

return speed

def correction\_position(position, speed):

n = len(position)

for i in range(n):

position[i] += speed[i]

if position[i] < -5:

position[i] = -5

# if position[i] > 20:

# position[i] = 20

return position

def swarm\_alg():

projects = all\_projects[0:13]

num\_particles = 100 # Количество частиц

num\_iterations = 1000 # Количество итераций

c1 = 1 # Коэффициент личного опыта

c2 = 2 # Коэффициент опыта группы

# Инициализация частиц

positions = np.random.uniform(0, 10, (num\_particles, 2)) # Позиции частиц

velocities = np.random.uniform(-1, 1, (num\_particles, 2)) # Начальные скорости частиц

personal\_best\_positions = positions.copy() # Личные лучшие позиции

personal\_best\_scores = np.array([fitness(p[0], p[1], projects) for p in positions])

global\_best\_position = personal\_best\_positions[np.argmin(personal\_best\_scores)] # Глобальная лучшая позиция

print('Лучшая позиция в начале алгоритма:')

print(global\_best\_position)

for i in range(num\_iterations):

for j in range(num\_particles):

if fitness(positions[j][0], positions[j][1], projects) < fitness(personal\_best\_positions[j][0], personal\_best\_positions[j][1], projects):

personal\_best\_positions[j] = positions[j].copy()

if fitness(personal\_best\_positions[j][0], personal\_best\_positions[j][1], projects) < fitness(global\_best\_position[0], global\_best\_position[0], projects):

global\_best\_position = personal\_best\_positions[j].copy()

velocities[j] = correction\_speed(velocities[j], positions[j], personal\_best\_positions[j], global\_best\_position, c1, c2)

positions[j] = correction\_position(positions[j], velocities[j])

print('Лучшая позиция в конце алгоритма:')

print(global\_best\_position)

print\_plot(projects, global\_best\_position)

print('\n')

return global\_best\_position

def print\_plot(projects, global\_best\_positions):

# Prepare data

L\_values = [proj["L"] for proj in projects]

Ef\_values = [proj["Ef"] for proj in projects]

# Calculate Efm values

Efm\_values = [global\_best\_positions[0] \* L \*\* global\_best\_positions[1] for L in L\_values]

# Create indices for uniform steps on x-axis

x\_indices = list(range(1, len(L\_values) + 1))

# Plot the data

plt.figure(figsize=(10, 6))

plt.plot(x\_indices, Ef\_values, label="Ef (actual)", marker="o", color="blue", linestyle="-")

plt.plot(x\_indices, Efm\_values, label=f"Efm", marker="x", color="red", linestyle="-")

# Customize x-axis ticks

plt.xticks(ticks=x\_indices, labels=L\_values)

# Add labels, legend, and title

plt.xlabel("Projects (L values)")

plt.ylabel("Ef / Efm")

plt.title("Graph of Ef and Efm values")

plt.legend()

plt.grid(True)

# Show the plot

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

swarm\_alg()

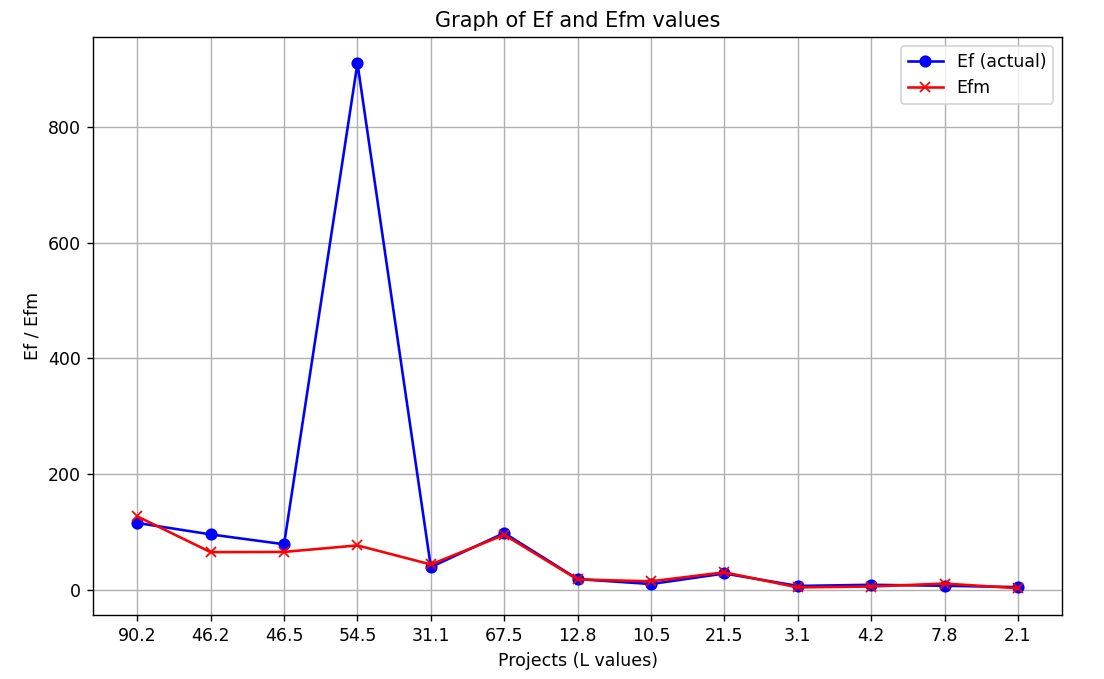
1. **Результат выполнения программы:**

Лучшая позиция в начале алгоритма:

[6.85666474 0.01230697]

Лучшая позиция в конце алгоритма:

[1.44539504 0.99437177]



1. **Выводы по полученным результатам.**

Генетический алгоритм успешно нашел оптимальное сочетание коэффициентов для модели COCOMO, минимизируя ошибку на обучающих данных. Тестовые данные также показали хорошую точность прогнозирования, что подтверждает эффективность предложенного подхода.