МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І.І. МЕЧНИКОВА
Факультет математики, фізики та інформаційних технологій

Звіт
з дисципліни: «Комп’ютерні системи штучного інтелекту»
Лабораторна робота № 2
за темою: «Генетичні алгоритми»

Виконав: студент денної форми навчання
Лавров Владислав
спеціальності: Комп’ютерна інженерія
кафедри: КСТ
курс: 4
Викладач:
Гожий О.П.

Одеса – 2024

Лабораторна робота №2. Генетичні алгоритми

Завдання: Скласти програму для знаходження цілих невід'ємних коренів рівняння.

$$c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4 + c_5x_5 = d,$$

де $c_1, c_2, \ldots, c_n, d$ – цілі додатні числа.

Реалізувати генетичний алгоритм, для вирішення задачі для різної кількості змінних рівняння, для різних значень коефіцієнтів c; передбачити можливість зміни кількості хромосом у популяції та кількості мутованих хромосом; навести кількість епох, необхідних для вирішення задачі.

Варіант № 7

$$2x_1 + 8x_2 + 5x_3 + 7x_4 - 6x_5 = 7$$

Теоретичні відомості:

Генетичний алгоритм - це інноваційна методика, яка відтворює природний процес еволюції з метою вирішення складних завдань. Під час еволюції природи, екземпляри виду, найбільш пристосовані до змін у навколишньому середовищі, мають кращі шанси на виживання та передачу своїх генетичних характеристик нащадкам. Рівень адаптації визначається генетичним кодом, який складається з хромосом, успадкованих від попередніх поколінь. Генетичний алгоритм працює з групою можливих рішень, кожне з яких представлене у вигляді хромосоми. Кожен ген хромосоми кодує певну характеристику проблеми, що вирішується, і може бути представлений різними структурами даних, що робить його ефективним і універсальним інструментом для розв'язання різноманітних завдань оптимізації.

Хід роботи:

Алгоритм, що реалізований у цьому коді - це генетичний алгоритм для пошуку розв'язку задачі оптимізації. Алгоритм складається з наступних етапів:

1.Ініціалізація: Початкові значення (initialValues) та цільове значення (target) передаються як параметри конструктора. Кількість хромосом також може бути вказана при створенні екземпляру класу.

2.Генерація початкових хромосом: Початкові хромосоми створюються випадковим чином у межах діапазону від 0 до цільового значення. Кількість хромосом задається параметром amountOfChromosomes.

3.Запуск алгоритму: Метод run() викликає алгоритм генетичного пошуку (alg()).

4.Головний цикл алгоритму: Виконується поки не буде знайдено розв'язок або поки не буде досягнуто максимальної кількості ітерацій (10000 у моєму випадку).

5.Оцінка функції пристосованості: Обчислюється значення функції пристосованості для кожної хромосоми, яке є сумою елементів поелементного множення початкових значень і значень хромосоми.

6.Перевірка цільового значення: Якщо цільове значення знаходиться серед значень функції пристосованості, алгоритм завершується з виведенням результату.

7.Мутація хромосом: Якщо середнє значення функції пристосованості збільшується порівняно з попереднім кроком, виконується мутація хромосоми.

8.Обчислення коефіцієнтів: Обчислюється коефіцієнт для кожної хромосоми, який використовується для вибору хромосом для розмноження.

9.Отримання нащадків: Вибираються хромосоми для розмноження з урахуванням їх коефіцієнтів. Потім відбувається кросовер (перемішування генів батьків) для створення нових хромосом.

10.Повторення алгоритму: Алгоритм рекурсивно викликає себе з отриманими нащадками, середнім значенням функції пристосованості та індексом кроку, збільшуючи його на одиницю.

Головна ідея полягає в тому, що через кілька ітерацій, за допомогою комбінації випадковості і відбору, алгоритм знаходить оптимальний розв'язок або принаймні наближається до нього.

У консолі можемо побачити етапи вирішення задачі (рисунок 1 та рисунок 2).

```
Generating start chromosomes
[6 6 3 0 4]
[5 7 7 3 5]
[5 5 3 3 0]
[5 4 5 2 1]
[1 6 4 1 5]
Step 0

Values of function: [51 92 86 75 47]
Average function value is: 70.2

Coefficients: [0.26166585 0.13545056 0.14573794 0.1693132  0.28783244]

Childrens:
[5 7 7 1 5]
[5 4 4 1 5]
[5 4 5 2 1]
[5 5 4 1 5]
[5 5 3 3 0]
Step 1

Values of function: [78 39 75 47 86]
Average function value is: 65.0

Coefficients: [0.14416281 0.31986123 0.15052293 0.25588898 0.12956404]
```

Рисунок 1 – Початок роботи генетичного алгоритму

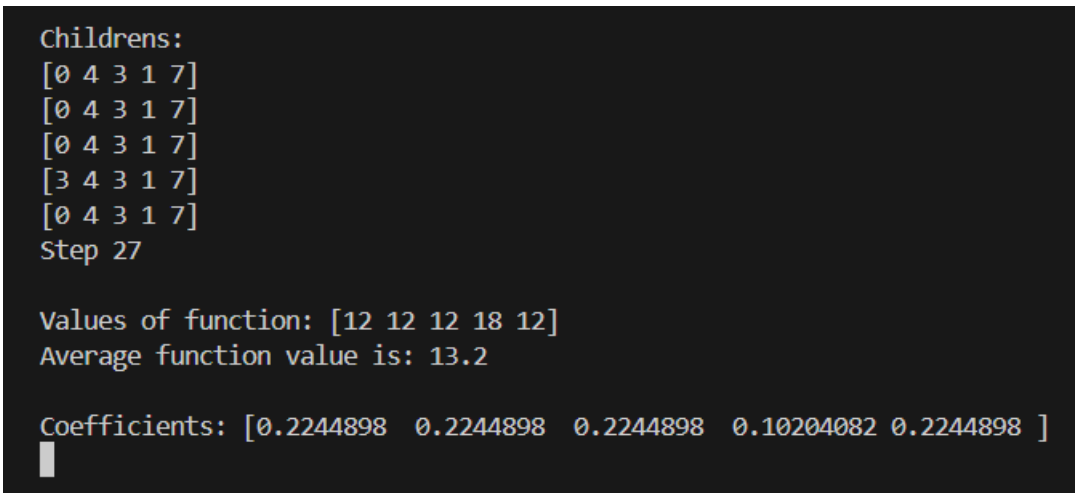

Рисунок 2 – Кінець роботи генетичного алгоритму

Можемо побачити, що при популяції з 5 хромосом, з’являється домінуюча хромосома яка не є рішенням рівняння. Спробуємо збільшити популяцію та проаналізуємо новий результат (рисунок 3 та рисунок 4).

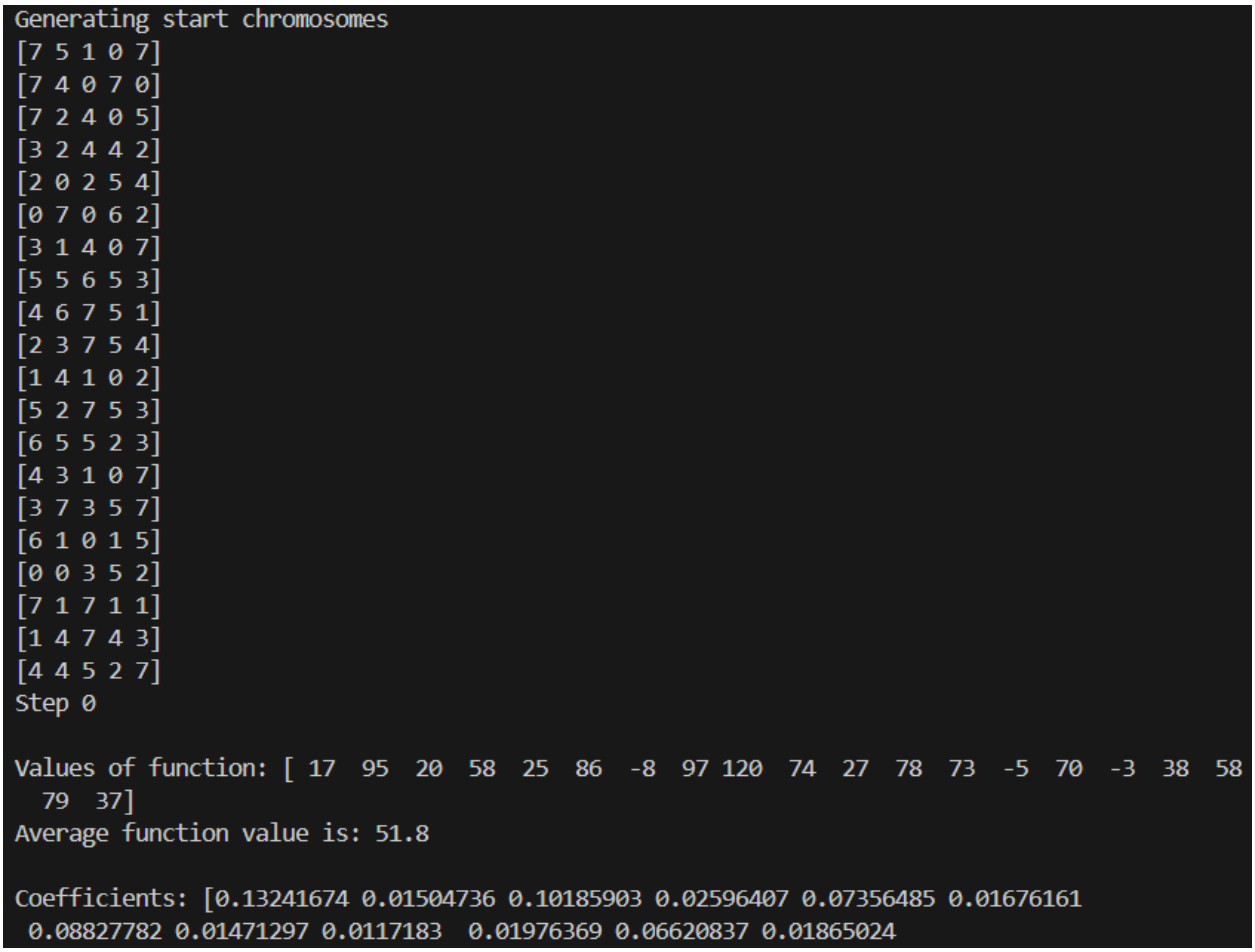

Рисунок 3 – Початок роботи алгоритму зі збільшеною популяцією

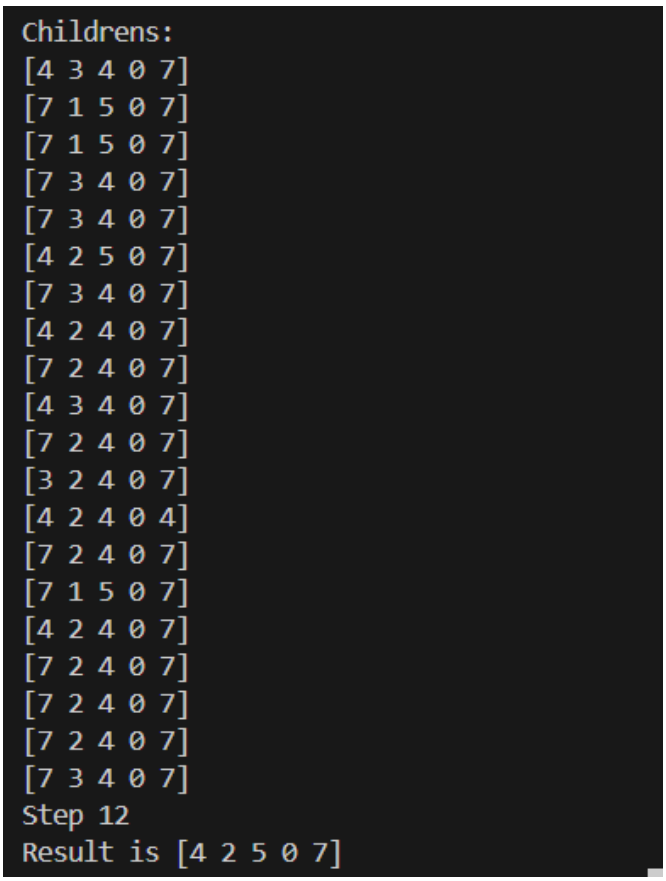

Рисунок 4 – Кінець роботи алгоритму зі збільшеною популяцією

Нижче наведено лістинг програми на мові програмування Python:

Лістинг коду:

```
import numpy as np


class GenAlg:
    def __init__(self, initialValues, target, amountOfChromosomes):
        # Ініціалізуємо початкові значення класу
        self.initialValues = initialValues  # Початкові значення хромосом
        self.target = target  # Цільове значення, яке ми шукаємо
        self.amountOfChromosomes = amountOfChromosomes  # Кількість хромосом
у популяції

    def run(self):
        # Запускаємо головний алгоритм
        print("Generating start chromosomes")
        # Генеруємо початкову популяцію хромосом
        chromosomes = self.generate_start_chromosomes()
        for arr in chromosomes:
            print(arr)
```

```
        result = self.alg(chromosomes, float('inf'), 0)
        print("Result is", result)
        return result

    def alg(self, chromosomes, prevAvg, step):
        # Головний алгоритм генетичного пошуку
        print("Step", step)
        if step > 10000:
            print("No solution was found")
            return None

        # Оцінка значень функції для кожної хромосоми
        valuesOfFunction = np.array(
            [self.calc_function(arr) for arr in chromosomes])
        if self.target in valuesOfFunction:
            return chromosomes[np.where(valuesOfFunction ==
self.target)[0][0]]

        # Обчислюємо середнє значення функції
        avgFunc = np.mean(valuesOfFunction)
        print("\nValues of function:", valuesOfFunction)
        print("Average function value is:", avgFunc)
        # Проводимо мутації, якщо середнє значення функції зростає
        if avgFunc > prevAvg:
            print("Avg > previous avg. Some mutation")
            chromosomes = self.mutation(chromosomes)
            valuesOfFunction = np.array(
                [self.calc_function(arr) for arr in chromosomes])
            if self.target in valuesOfFunction:
                return chromosomes[np.where(valuesOfFunction ==
self.target)[0][0]]

        # Обчислюємо коефіцієнти для вибору батьків
        coef = self.calc_coef(valuesOfFunction)
        print("\nCoefficients:", coef)

        # Генеруємо нових нащадків
        childrens = self.get_children(chromosomes, coef)
        print("\nChildrens:")
        for arr in childrens:
            print(arr)
        # Рекурсивно запускаємо головний алгоритм для нової популяції
        return self.alg(childrens, avgFunc, step + 1)

    def generate_start_chromosomes(self):
        # Генерація випадкових початкових хромосом
        chromosomes = np.random.randint(
            0, self.target + 1, size=(self.amountOfChromosomes,
len(self.initialValues)))
        return chromosomes
```

```
    def calc_function(self, values):
        # Обчислення значення функції для заданих значень хромосом
        return np.sum(np.array(self.initialValues) * np.array(values))

    def calc_coef(self, values):
        # Обчислення коефіцієнтів для вибору батьків
        r = 1.0 / np.abs(self.target - values)
        rSum = np.sum(r)
        coef = r / rSum
        return coef

    def get_children(self, chromosomes, coefficients):
        # Генерація нових нащадків
        sortedCoef = np.sort(coefficients)[::-1]
        sortedIndices = np.argsort(coefficients)[::-1]

        chromosomes = chromosomes[sortedIndices]
        uniqParent = set()
        while len(uniqParent) != len(sortedCoef):
            parent1Index = np.random.choice(
                range(len(sortedCoef)), p=sortedCoef)
            p_without_parent1 = np.delete(sortedCoef, parent1Index)
            # Нормалізуємо ймовірності
            p_without_parent1 /= np.sum(p_without_parent1)
            parent2Index = np.random.choice(
                range(len(sortedCoef) - 1), p=p_without_parent1)

            if parent1Index == parent2Index:
                continue
            parent1 = tuple(chromosomes[parent1Index])
            parent2 = tuple(chromosomes[parent2Index])
            uniqParent.add((parent1, parent2))

        children = np.array([self.crossover(parents)
                             for parents in uniqParent])
        return children

    def crossover(self, chromosomesParents):
        # Схрещування батьків для створення нащадка
        crossoverPoint = np.random.randint(1, len(chromosomesParents[0]) + 1)

        child = np.zeros(len(chromosomesParents[0]), dtype=int)
        child[:crossoverPoint] = chromosomesParents[0][:crossoverPoint]
        child[crossoverPoint:] = chromosomesParents[1][crossoverPoint:]

        return child

    def mutation(self, chromosomes):
        # Мутація хромосоми
        numOfChanges = np.random.randint(1, len(chromosomes) + 1)
        for _ in range(numOfChanges):
```

```
            i = np.random.randint(len(chromosomes))
            j = np.random.randint(len(chromosomes[i]))
            chromosomes[i][j] = np.random.randint(0, self.target + 1)
        return chromosomes

# Приклад використання для мого варіанту:
initialValues = [2, 8, 5, 7, -6]
target = 7
amountOfChromosomes = 20
genAlg = GenAlg(initialValues, target, amountOfChromosomes)
result = genAlg.run()
```

Висновок

Під час виконання цієї лабораторної роботи я ознайомився з принципами роботи генетичного алгоритму та використав їх для розробки програми, здатної знаходити розв'язки Діофантових рівнянь.