Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №3.3

з дисципліни
«Інтелектуальні вбудовані системи»

на тему
«Дослідження генетичного алгоритму»

Виконала:

студентка

групи ІП-83

Гомілко Діана Володимирівна

Перевірив:

Регіда Павло Геннадійович

Київ 2021

Основні теоретичні відомості, необхідні для виконання лабораторної роботи

Генетичні алгоритми служать, головним чином, для пошуку рішень в багатовимірних просторах пошуку.

Можна виділити наступні етапи генетичного алгоритму:

 (Початок циклу)

 Розмноження (схрещування)

 Мутація

 Обчислити значення цільової функції для всіх особин

 Формування нового покоління (селекція)

 Якщо виконуються умови зупинки, то (кінець циклу), інакше (початок циклу).

Розглянемо приклад реалізації алгоритму для знаходження цілих коренів діофантового рівняння a+b+2c=15. Згенеруємо початкову популяцію випадковим чином, але з дотриманням умови – усі згенеровані значення знаходяться у проміжку від одиниці до y/2, тобто на відрізку [1;8] (узагалі, границі випадкового генерування можна вибирати на свій розсуд):

(1,1,5); (2,3,1); (3,4,1); (3,6,4)

Отриманий генотип оцінюється за допомогою функції пристосованості (fitness function). Згенеровані значення підставляються у рівняння, після чого обраховується різниця отриманої правої частини з початковим y. Після цього рахується ймовірність вибору генотипу для ставання батьком – зворотня дельта ділиться на сумму сумарних дельт усіх генотипів.

Наступний етап включає в себе схрещування генотипів по методу кросоверу – у якості дітей виступають генотипи, отримані змішуванням коренів – частина йде від одного з батьків, частина від іншого. Лінія кросоверу може бути поставлена в будь-якому місці, кількість потомків також може вибиратися. Після отримання нових генотипів вони перевіряються функцією пристосованості та створюють власних потомків, тобто виконуються дії, описані вище.

Ітерації алгоритму відбуваються, поки один з генотипів не отримає $\Delta=0$, тобто його значення будуть розв’язками рівняння.

Умови завдання для варіанту

Налаштувати генетичний алгоритм для знаходження цілих коренів діофантового рівняння ax1+bx2+cx3+dx4=y. Розробити відповідний мобільний додаток і вивести отримані значення. Провести аналіз витрат часу на розрахунки.

Лістинг програми із заданими умовами завдання

Equation.kt

```
package com.example.geneticalgorithm

data class Equation(
    val coeffs: List<Double>,
    var delta: Double = 0.0,
    var probability: Double = 0.0
)
```

WeightedRandom.kt

```
package com.example.geneticalgorithm

import java.util.*

class WeightedRandom(elements: List<Equation>) {
    private val map: NavigableMap<Double, Equation> = TreeMap()
    private val random = Random()
    private var total = 0.0

    init {
        elements.map{ e ->
            total += e.probability
            map[total] = e
        }
    }

    fun getRand(): Equation {
        val value = random.nextDouble() * total
        return map.higherEntry(value)!!.value
    }
}
```

GeneticAlgo.kt

```
package com.example.geneticalgorithm

import kotlin.math.abs
import kotlin.math.ceil
import kotlin.math.floor

class GeneticAlgo(private val coeffs: List<Double>, private val finalResult: Double) {
    private val variables = coeffs.size
    private val populationSize = 10
    private var population = mutableListOf<Equation>()

    private fun substitute(eq: Equation): Double {
        return coeffs.mapIndexed() { i, coeff -> coeff * eq.coeffs[i] }.sum()
    }

    private fun initPopulation() {
        val randLimit = ceil(finalResult / 2).toInt()
        for (i in 0 until populationSize) {
            val generatedCoeffs = (1..variables).map {
(1..randLimit).random().toDouble() }
            population.add(Equation(generatedCoeffs))
        }
    }
```

```
    private fun fitness(): Equation? {
        population.map { equation ->
            val result = substitute(equation)
            val delta = abs(finalResult - result)
            if (delta == 0.0) return equation
            equation.delta = delta
        }

        val sumReversedDeltas = population.map{ e -> 1 / e.delta }.sum()
        population.map { e -> e.probability = 1 / e.delta / sumReversedDeltas }
        return null
    }

    private fun crossbreeding() {
        val newGeneration = mutableListOf<Equation>()
        val crossover = floor(variables.toDouble() / 2).toInt()
        for (i in 0 until populationSize) {
            val curPopulation = population.toMutableList()
            val parent1 = WeightedRandom(curPopulation).getRand()
            curPopulation.remove(parent1)
            val parent2 = WeightedRandom(curPopulation).getRand()
            val parent1Part = parent1.coeffs.slice(0 until crossover)
            val parent2Part = parent2.coeffs.slice(crossover until variables)
            val coeffs = parent1Part.plus(parent2Part).toMutableList()

            if ((1..100).random() < 10) {
                val chromosome = (0 until variables).random()
                val diff = listOf(-1, 1).random()
                coeffs[chromosome] = coeffs[chromosome] + diff
            }
            newGeneration.add(Equation(coeffs))
        }
        population = newGeneration
    }

    fun runAlgorithm(): List<Double> {
        initPopulation()

        while(true) {
            val result = fitness()
            if (result != null) return result.coeffs
            crossbreeding()
        }
    }
}
```

MainActivity.kt

```
package com.example.geneticalgorithm

import android.os.Bundle
import android.os.SystemClock
import android.widget.Button
import android.widget.EditText
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {
    private var inputs = mutableListOf<EditText>()
    private lateinit var button: Button
    private lateinit var result: TextView
```

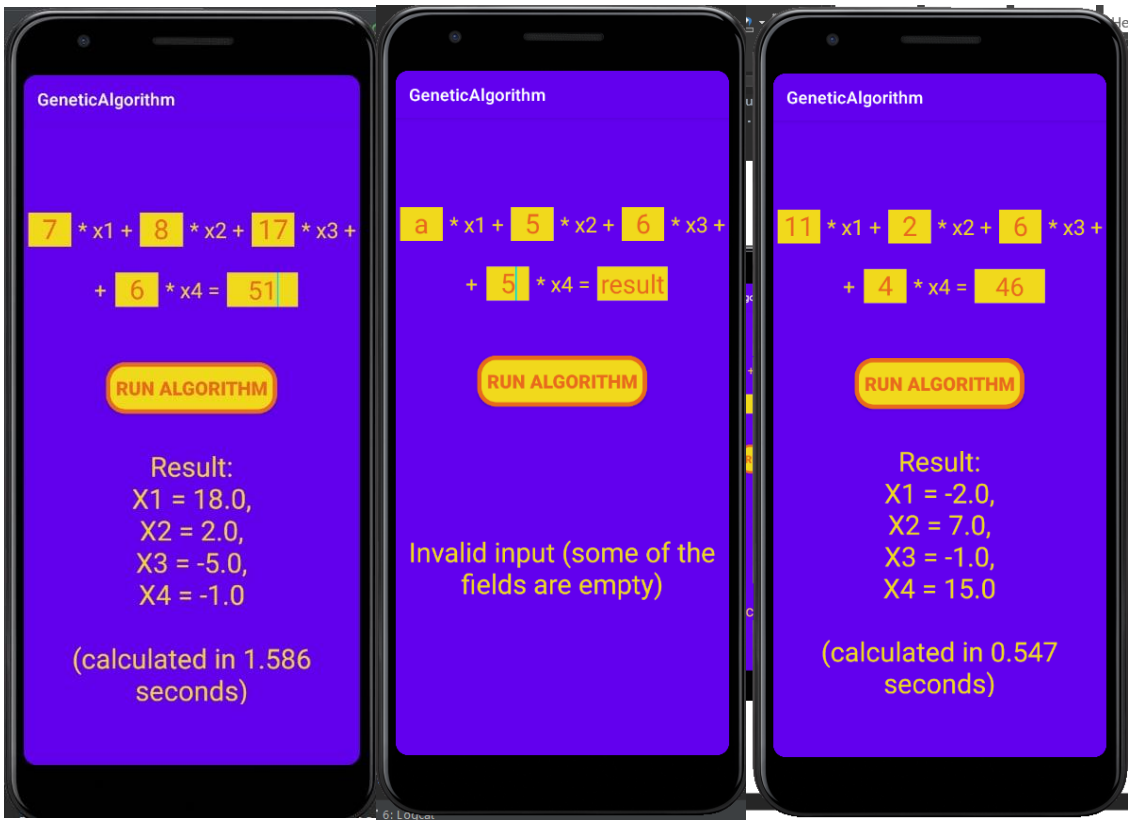
```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val ids = listOf(R.id.a, R.id.b, R.id.c, R.id.d, R.id.res)
        inputs = ids.map { id -> findViewById<EditText>(id) }.toMutableList()
        button = findViewById(R.id.btn)
        result = findViewById(R.id.resultArea)

        button.setOnClickListener { handleBtnClick() }
    }

    private fun handleBtnClick() {
        var output = ""
        if (inputs.none { input -> input.text.isEmpty() }) {
            val args = inputs.map { input -> input.text.toString().toDouble()
}.toMutableList()
            val expected = args.removeLast()
            val startTime = SystemClock.elapsedRealtime()
            val res = GeneticAlgo(args, expected).runAlgorithm()
            val endTime = SystemClock.elapsedRealtime()
            val elapsedMilliSeconds = endTime - startTime
            val elapsedSeconds = elapsedMilliSeconds / 1000.0
            output = "Result:\nX1 = ${res[0]},\nX2 = ${res[1]},\nX3 = ${res[2]},\nX4 =
${res[3]}\n\n" +
                    "(calculated in $elapsedSeconds seconds)"
        }
        else output = "Invalid input (some of the fields are empty)"
        result.text = output
    }
}
```

Результати виконання кожної програми



Висновки щодо виконання лабораторної роботи

Під час виконання лабораторної роботи ми ознайомилися з принципами реалізації генетичного алгоритму, вивчили та дослідили особливості даного алгоритму. Було створено мобільний застосунок для пошуку коренів діофантового рівняння з використанням генетичного алгоритму. При цьому розмір опуляції було обрано як 10, ймовірність мутації – 10%, а метод вибору батьків для схрещування – рулетка. Перевірка отриманих результатів показала, що програма працює коректно, отже, мету лабораторної роботи можна вважати виконаною.