**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»

Виконав:

студент 3 курсу

ФІОТ, групи ІП-84

Кришталь Дмитро Вікторович

Номер залікової - 8414

Перевірив:

Регіда П. Г.

Київ 2021

## Основні теоретичні відомості

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 $a+b+2c=15$.

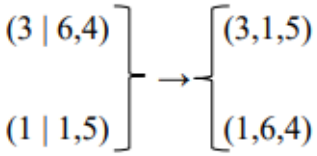
Згенеруємо початкову популяцію випадковим чином, але з дотриманням умови – усі згенеровані значення знаходяться у проміжку від одиниці до $y/2$, тобто на відрізку [1;8] (узагалі, границі випадкового генерування можна вибирати на свій розсуд):

(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.

| | | |
|---|---|---|
| 1+1+2·5=12 | Δ=3 | $\frac{\frac{1}{3}}{\frac{27}{24}} = 0{,}7$ |
| 2+3+2·1=7 | Δ=8 | $\frac{\frac{1}{8}}{\frac{27}{24}} = 0{,}11$ |
| 3+4+2·1=9 | Δ=6 | $\frac{\frac{1}{6}}{\frac{27}{24}} = 0{,}15$ |
| 3+6+2·4=17 | Δ=2 | $\frac{\frac{1}{2}}{\frac{27}{24}} = 0{,}44$ |

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, наприклад:




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 Δ=0, тобто його значення будуть розв'язками рівняння.

## Умови завдання

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

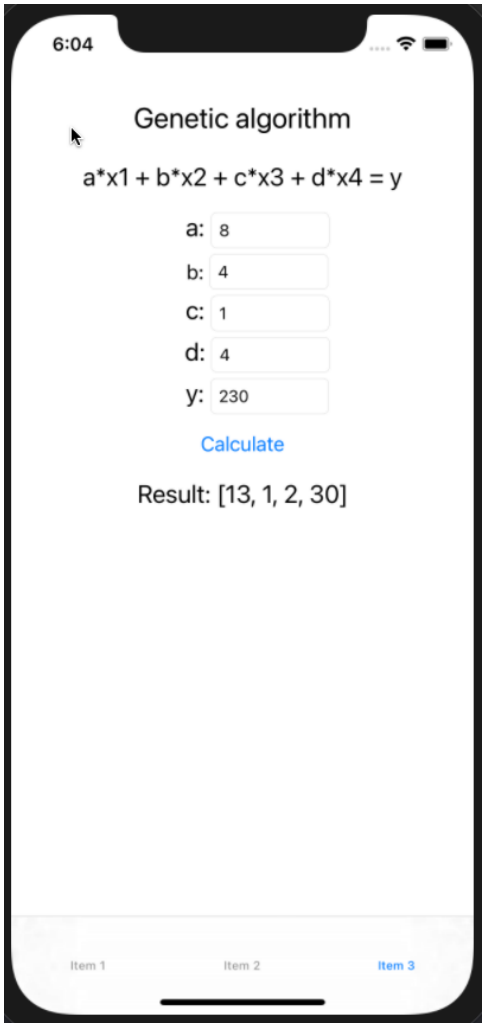
```
import UIKit

class ThirdViewController: UIViewController {
    @IBOutlet var a: UITextField!
    @IBOutlet var b: UITextField!
    @IBOutlet var c: UITextField!
    @IBOutlet var d: UITextField!
    @IBOutlet var y: UITextField!
    //@IBOutlet var calculate: UIButton!
    @IBOutlet var resultLabel: UILabel!
    @IBAction func buttonTapped(_ sender: UIButton) {
        a.resignFirstResponder()
        b.resignFirstResponder()
        c.resignFirstResponder()
        d.resignFirstResponder()
        y.resignFirstResponder()

        let gen = GeneticAlg(a: Int(a.text!)!, b: Int(b.text!)!, c: Int(c.text!)!, d: Int(d.text!)!, y:
Int(y.text!)!)
        gen.fitness()
        gen.calculatePercentage()
        gen.roulette()
        resultLabel.text = "Result: \(gen.crossAndMutation())"

    }

    override func viewDidLoad() {
        super.viewDidLoad()
```

```
    }

}

class GeneticAlg {
    var a: Int
    var b: Int
    var c: Int
    var d: Int
    var y: Int

    var population = [[Int]: Int]()
    var roulettePerc = [[Int]: Float]()
    var arrOfChosenOnes = [[Int]]()

    init(a: Int, b: Int, c: Int, d:Int, y:Int) {
        self.a = a
        self.b = b
        self.c = c
        self.d = d
        self.y = y
    }

    func calculateEquation(x1:Int, x2:Int, x3:Int, x4:Int) -> Int{
        return a*x1 + b*x2 + c*x3 + d*x4
    }

    func fitness() {
        for _ in 1...4 {
            let arr = [Int.random(in: 1...(y/2)),
                       Int.random(in: 1...(y/2)),
                       Int.random(in: 1...(y/2)),
                       Int.random(in: 1...(y/2))]

            let delta = abs(y - arr[0]*a + arr[1]*b + arr[2]*c + arr[3]*d)
            population[arr] = delta
        }

    }

    func calculatePercentage() {
        var rouletteParam:Double = 0
        for (_, delta) in population{
            rouletteParam += 1.0/Double(delta)
        }
        for (arr, delta) in population{
            roulettePerc[arr] = (Float(1.0/Double(delta)/rouletteParam) * 100).rounded() / 100
        }
    }

    func roulette(){
        for _ in 1...4 {
            var currentStartInterval:Float = 0
            var currentEndInterval:Float = 0
```

```
                let rand = (Float.random(in: 0...1) * 100).rounded() / 100
                for (arr, perc) in roulettePerc {
                    currentEndInterval = currentStartInterval + perc - 0.01


                    if(rand >= currentStartInterval && rand <= currentEndInterval) {
                        if(!arrOfChosenOnes.contains(arr)) {
                            arrOfChosenOnes.append(arr)
                        }
                        break
                    }
                    currentStartInterval += perc
                }

            }
            arrOfChosenOnes.sort {
                roulettePerc[$0]! > roulettePerc[$1]!
            }
        }

        func crossAndMutation() -> [Int] {
            let indexToCross = Int.random(in: 1..<arrOfChosenOnes[0].count)
            var arrWithCrossed = arrOfChosenOnes[0...1]
            var result = [Int]()
            for i in 0..<indexToCross {
                arrWithCrossed[0][i] = arrOfChosenOnes[1][i]
                arrWithCrossed[1][i] = arrOfChosenOnes[0][i]
            }

            for num in arrWithCrossed {
                if(calculateEquation(x1: num[0], x2: num[1], x3: num[2], x4: num[3]) == y) {
                    result = num
                    break
                }
            }
            if (result == []){
                let randIndex = Int.random(in: 0...1)
                var chosenGenotype = arrWithCrossed[randIndex]
                var calculated = calculateEquation(x1: chosenGenotype[0], x2: chosenGenotype[1], x3:
chosenGenotype[2], x4: chosenGenotype[3])
                while(calculated != y) {
                    let rand = Int.random(in: 0..<chosenGenotype.count)
                    if (calculated > y) {
                        if(chosenGenotype[rand] == 0) {
                            continue
                        }
                        chosenGenotype[rand] -= 1
                    } else {
                        chosenGenotype[rand] += 1
                    }
                    calculated = calculateEquation(x1: chosenGenotype[0], x2: chosenGenotype[1], x3:
chosenGenotype[2], x4: chosenGenotype[3])
                }
                result = chosenGenotype
```

```
        }

        return result
    }
}
```

## Результати виконання програми




## Висновок

Під час виконня лабораторної роботи №3.3 ознайомився з принципами реалізації генетичного алгоритму, вивчив та дослідив його особливості. Написав програму. Наведені скріншоти підтверджують правильність її роботи.