Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут ім. Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

ЗВІТ

до лабораторної роботи №3.3

з дисципліни «Інтелектуальні вбудовані системи»

на тему «Дослідження генетичного алгоритму»

Виконав:

студент групи ІП-83

Черевач А.М.

Перевірив:

асистент 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.

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.

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.

Завдання

Налаштувати генетичний алгоритм для знаходження цілих коренів діофантового рівняння $ax_1+bx_2+cx_3+dx_4=y$. Розробити відповідний мобільний додаток і вивести отримані значення. Провести аналіз витрат часу на розрахунки.

Лістинг програми

Файл з классом для знаходження коренів діофантового рівняння

```
const random = (min, max) => ~~(Math.random() * (max - min) + min)

class Chromosome {

   genes = [];

   fitness = Infinity;

   task = [];

   target = 0;

   calc = () =>

       this.genes.reduce((a, gene, i) => a + (gene * this.task[i]))

   constructor({ genes, task, target }) {

       Object.assign(this, { genes, task, target });

       this.calcFitness();

   }

   crossover(partner) {

       const child = this.clone();

       const fromParent = child.genes.length / 2;

       child.genes = [...child.genes.slice(0, fromParent),
...partner.genes.slice(child.genes.length - fromParent)];

       child.calcFitness();
```

```
        return child
    }
    calcFitness() {
        this.fitness = Math.abs(this.target - this.calc());
    }
    clone() {
        return Object.assign(Object.create(Object.getPrototypeOf(this)), this);
    }
}
class Genetic {
    population = []
    constructor(task, target) {
        const { length } = task
        this.population =
            Array.from(
                { length },
                () => new Chromosome({
                    genes: Array.from({ length }, () => random(1, target / 2)),
                    task: task,
                    target: target,
                })
            )
    }
    solve() {
        while (true) {
            const chromosome = this.crossover()
```

```
            if (chromosome)

                return chromosome.genes

        }

    }

    crossover() {

        const children = []

        for (let i = 0; i < this.population.length; i++) {

            const parents = this.population

                .map((chromosome) => ({ chromosome, probability: Math.random() *
(chromosome.fitness * 1000) }))

                .sort((a, b) => a.probability - b.probability)

            const parent = parents[0].chromosome

            const partner = parents[1].chromosome

            const child = parent.crossover(partner)

            if (child.fitness === 0)

                return child



            children.push(child)

        }

        this.population = children

    }

}

export default Genetic;
```

Основний файл програми з мобільним інтерфейсом

```
import React, { useState } from 'react';

import { StyleSheet, Text, View, SafeAreaView, TextInput, Button } from
'react-native';

import Genetic from './src/Genetic'

export default function App() {

 const [a, setA] = useState(null);

 const [b, setB] = useState(null);

 const [c, setC] = useState(null);

 const [d, setD] = useState(null);

 const [y, setY] = useState(null);

 const [result, setResult] = useState('[]');

 return (

   <SafeAreaView>

     <View style={styles.container}>

       <TextInput style={styles.expression}

         onChangeText={setA}

         value={a}

         placeholder="a"

         keyboardType="numeric"

       />

       <Text style={styles.expression}>{'*x1 + '}</Text>

       <TextInput style={styles.expression}

         onChangeText={setB}

         value={b}
```

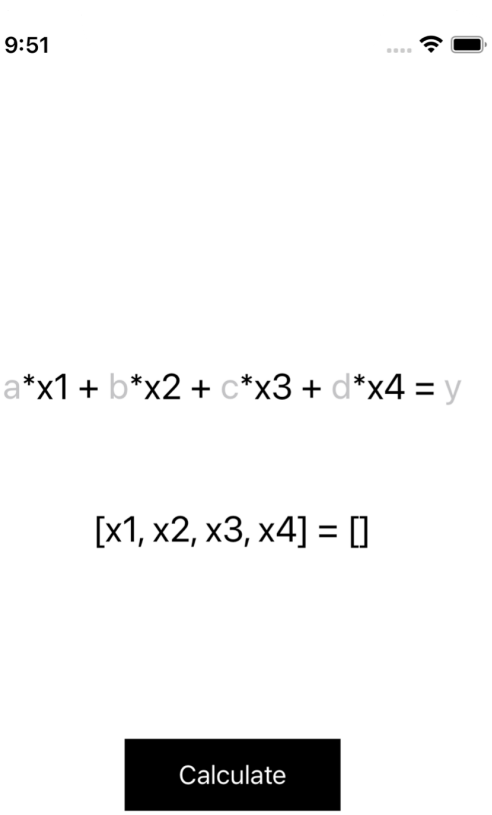
```
        placeholder="b"
        keyboardType="numeric"
      />
      <Text style={styles.expression}>{'*x2 + '}</Text>
      <TextInput style={styles.expression}
        onChangeText={setC}
        value={c}
        placeholder="c"
        keyboardType="numeric"
      />
      <Text style={styles.expression}>{'*x3 + '}</Text>
      <TextInput style={styles.expression}
        onChangeText={setD}
        value={d}
        placeholder="d"
        keyboardType="numeric"
      />
      <Text style={styles.expression}>{'*x4 = '}</Text>
      <TextInput style={styles.expression}
        onChangeText={setY}
        value={y}
        placeholder="y"
        keyboardType="numeric"
      />
    </View>
    <Text style={styles.result}>
```

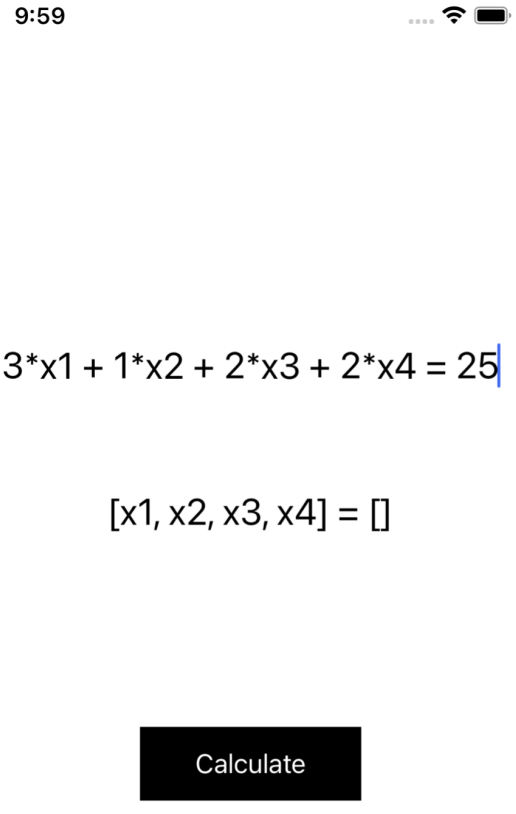
```
      {`[x1, x2, x3, x4] = ${result}`}
    </Text>
    <View style={styles.btn}>
      <Button
        title="Calculate"
        color="#fff"
        onPress={() => setResult(new Genetic([a, b, c, d], y).solve())}
      />
    </View>

  </SafeAreaView>
 );
};
const styles = StyleSheet.create({
 container: {
   width: '90%',
   top: 200,
   flexDirection: 'row',
   alignSelf: 'center',
   alignItems: 'center',
   justifyContent: 'center',
 },
 expression: {
   fontSize: 25
 },
 result: {
```

```
    alignSelf: 'center',
    top: 269,
    fontSize: 25
  },
  time: {
    alignSelf: 'center',
    top: 320,
    fontSize: 22
  },
  btn: {
    justifyContent: 'center',
    alignItems: 'center',
    alignSelf: 'center',
    top: 400,
    height: 50,
    width: 150,
    backgroundColor: 'black',
  },
});
```

Результат роботи програми


9:51
a*x1 + b*x2 + c*x3 + d*x4 = y
[x1, x2, x3, x4] = []
Calculate


9:59
3*x1 + 1*x2 + 2*x3 + 2*x4 = 25
[x1, x2, x3, x4] = []
Calculate

$$3*x1 + 1*x2 + 2*x3 + 2*x4 = 25$$

$$[x1, x2, x3, x4] = [5, 2, 3, 1]$$

Calculate

Висновки

Під час виконання лабораторної роботи було досліджено генетичний алгоритм для розв’язку діофантового рівняння. Було реалізовано програму для знаходження коренів у вигляді мобільного додатку за допомогою фреймворку React Native та Expo. Програма реалізує генетичний алгоритм та за його допомогою розв’язує діофантове рівняння з вказаними користувачем коефіцієнтами.