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 групи ІП-84

Валигні Андрій Олексвндрович
номер залікової книжки: 8503

Перевірив:

ас. Регіда П. Г.

Київ 2020

Мета роботи - ознайомлення з принципами реалізації генетичного алгоритму, вивчення та дослідження особливостей даного алгоритму з використанням засобів моделювання і сучасних програмних оболонок.

Основні теоретичні відомості

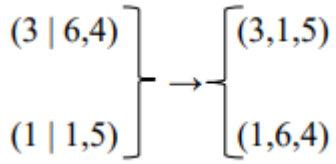
Генетичні алгоритми служать, головним чином, для пошуку рішень в багатовимірних просторах пошуку. Можна виділити наступні етапи генетичного алгоритму: (Початок циклу) Розмножения (схрещування) Мутація Обчислити значення цільової функції для всіх особин Формування нового покоління (селекція) Якщо виконуються 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.

Код програми

```
import 'dart:math';
List<double> fitnessFunction(List<int> deltas) {
  List<double> reverseDeltas = deltas.map((delta) => 1 / delta).toList();
  double reversedDeltaSum =
      reverseDeltas.reduce((value, element) => value += element);
  List<double> fitnesses =
      reverseDeltas.map((deltas) => (deltas / reversedDeltaSum) *
100).toList();
  return fitnesses;
}


List<List<int>> diophant({a, b, c, d, ans, iter}) {
  int maxRange = (ans / 2).round();
  List<List<int>> initVals = [];
  for (int i = 0; i < 5; i++) {
    initVals.add([
```

```
    Random().nextInt(maxRange) + 1,
    Random().nextInt(maxRange) + 1,
    Random().nextInt(maxRange) + 1,
    Random().nextInt(maxRange) + 1
  ]);
}
var successRes = 0;
var iterations = 1;
while (successRes != 1) {
  List<int> deltas = initVals
      .map<int>((gen) =>
          (ans - (a * gen[0] + b * gen[1] + c * gen[2] + d * gen[3])).abs())
      .toList();
  var resultIndex = deltas.indexOf(0);
  if (resultIndex > -1) {
    successRes = 1;
    return [
      initVals[resultIndex],
      [iterations]
    ];
  }
  List<double> fitnesses = fitnessFunction(deltas);


  var bestParentArr = [];
  for (int i = 0; i < fitnesses.length; i++) {
    for (int j = 0; j < fitnesses[i].round(); j++) {
      bestParentArr.add(i);
    }
  }
  List<List<int>> parents = [];
  List<List<int>> newInitVals = [];
```

```
for (int i = 0; i < 5; i++) {
  var firstParent;
  var secondParent;
  do {
    firstParent = bestParentArr[Random().nextInt(bestParentArr.length)];
    secondParent = bestParentArr[Random().nextInt(bestParentArr.length)];
  } while (firstParent == secondParent ||
      parents
              .where((parent) =>
                  parent[0] == firstParent && parent[1] == secondParent)
              .length >
          0);
  parents.add([firstParent, secondParent]);


  var devider = Random().nextInt(3);
  var mutant = Random().nextInt(2);
  List<int> child = [];
  List<int> firstGen = [];
  List<int> secondGen = [];
  switch (devider) {
    case 0:
      if (mutant == 0) {
        firstGen = [0, 1];
        secondGen = [1, 4];
      } else {
        secondGen = [0, 1];
        firstGen = [1, 4];
      }
      break;
```

```
        case 1:
          if (mutant == 0) {
            firstGen = [0, 2];
            secondGen = [2, 4];
          } else {
            secondGen = [0, 2];
            firstGen = [2, 4];
          }
          break;
        case 2:
          if (mutant == 0) {
            firstGen = [0, 3];
            secondGen = [3, 4];
          } else {
            secondGen = [0, 3];
            firstGen = [3, 4];
          }
          break;
        default:
      }


      child.addAll(initVals[firstParent].getRange(firstGen[0], firstGen[1]));
      child.addAll(initVals[secondParent].getRange(secondGen[0],
secondGen[1]));
      newInitVals.add(child);
    }
    List<int> newInitValsDelta = newInitVals
        .map<int>((gen) =>
            (ans - (a * gen[0] + b * gen[1] + c * gen[2] + d * gen[3])).abs())
        .toList();
    double avgDelta =
```

```
        deltas.reduce((value, element) => value + element) / deltas.length;
    double avgNewValDelta =
        newInitValsDelta.reduce((value, element) => value + element) /
            newInitValsDelta.length;
    if (avgDelta > avgNewValDelta) {
      for (int i = 0; i < newInitVals.length; i++) {
        for (int j = 0; j < 4; j++) {
          var randomInt = Random().nextInt(10);
          if (randomInt > 4) {
            newInitVals[i][j] = Random().nextInt((ans / 2).round());
          }
        }
      }
    }
    initVals = newInitVals.getRange(0, 5).toList();
    iterations++;
    if (iterations > iter - 1) {
      return [
        initVals[0],
        [iterations]
      ];
    }
  }
}
```

Скріншоти

Висновок

Під час виконання лабораторної роботи я ознойомився з основами ревлізації генетичного алгоритму. Розробив відповідну програму за допомогою Flutter Dart.