Міністерство освіти і науки України

Черкаський державний технологічний університет

Кафедра програмного забезпечення автоматизованих систем

Звіт

З лабораторної роботи №6

З предмету «Аналіз великих даних»

Перевірив

доцент к. ПЗАС

Рідкокаша А. А.

Виконав

Студент IV курсу

Група ПЗ-154

Кравченко А. О.

Черкаси, 2019

**Тема:** Генетичний алгоритм.

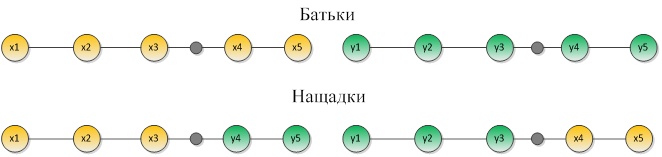
**Мета:** Вивчити основні методи генетичного пошуку. Навчитися використовувати генетичні методи для розв’язку оптимізаційних задач

**Постановка завдання:** Реалізувати генетичний алгоритм, застосувавши мови програмування.

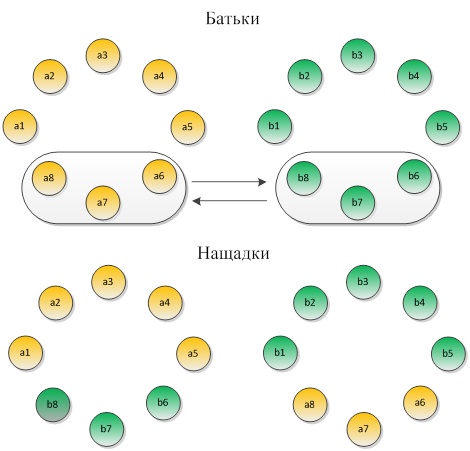
**Теоретичні відомості:**

Кросинговер - це один із видів оператора рекомбінації [генетичного алгоритму](https://uk.wikipedia.org/wiki/%D0%93%D0%B5%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC" \o "Генетичний алгоритм). Застосовується на хромосомах з бінарними генами.

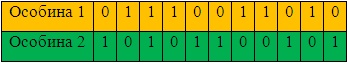
Одноточковий кросинговер (Single-point crossover) моделюється наступним чином. Нехай є дві батьківські особини з хромосомами X={x_i,i \in [0,L]} і Y={y_i,i \in [0,L]}. Випадковим чином визначається точка всередині [хромосоми](https://uk.wikipedia.org/wiki/%D0%A5%D1%80%D0%BE%D0%BC%D0%BE%D1%81%D0%BE%D0%BC%D0%B0_(%D0%B3%D0%B5%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC)" \o "Хромосома (генетичний алгоритм)) (точка розриву), в якій обидві хромосоми діляться на дві частини і обмінюються ними. Такий тип кросинговеру називаються одноточковим, так як при ньому батьківські хромосоми розділяються тільки в одній випадкової точці.



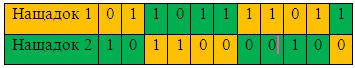
У двоточковому кросинговері (і в багатоточковому кросинговері також) хромосоми розглядаються як цикли, які формуються з'єднанням кінців лінійної хромосоми разом. Для заміни сегменту одного циклу сегментом іншого циклу потрібно вибрати дві точки розрізу. З цієї точки зору, одноточковий кросинговер може бути розглянутий як кросинговер з двома точками, але з однією точкою розриву, зафіксованій на початку рядка.



Для багатоточкового кросинговеру (Multi-point crossover), вибираємо m точок розриву k_i \in [1,Nvar], i=(1,...,m), Nvar- кількість змінних (генів) у особині. Точки розриву вибираються випадково без повторень і сортуються в порядку зростання. При кросинговері походить обмін ділянками хромосом, обмеженими точками розрізу і таким чином отримують двох нащадків. Ділянка особини з першим геном до першої точки розрізу в обміні не бере участь.



Оберемо такі точки розриву кросинговеру: 2, 6 і 10. Отримаємо таких нащадків:



Хромосомні мутації — структурні зміни хромосом, хромосомні перебудови, які відбуваються синхронно в обох хроматидах. Перебудови можуть здійснюватися як в межах однієї хромосоми – *внутрішньохромосомні мутації* (делеція, інверсія, дуплікація), так і між хромосомами - *міжхромосомні мутації* (транслокація). Деякі типи хромосомних мутацій змінюють дію деяких генів і чинять на фенотип більш значний вплив, ніж генні мутації.

Основні типи хромосомних мутацій:

1. *Делеції* - втрата ділянки хромосоми або послідовність ДНК. При делеції виникає втрата генетичного матеріалу. При цьому може бути вилучена будь-яке число нуклеотидів, від одної основи до великого шматка хромосоми.
2. *Дуплікація* — подвоєння генів у певній ділянці хромосоми. Дуплікація генів може відбуватися в результаті помилки при гомологічній рекомбінації, в результаті ретротранспозиції або дуплікації всієї хромосоми.
3. *Інверсія*- хромосомна мутація, коли після двох розривів в одній хромосомі сегмент хромосоми, розташований між розривами, повертається на 180 ° і займає інвертоване положення.
4. *Транслокація* — перенесення частини хромосом на іншу негомологічну хромосому, як результат — зміна групи зчеплення генів. Загальне число генів не змінюється.

**Виконання лабораторної роботи:**

Код програми, що реалізує генетичний алгоритм.

ChildKindWrapper.class – клас потомства.

package ark.ailab;  
  
public class ChildKindWrapper {  
 int individual;  
 double geneticValue;  
 boolean isMutant;  
}

KindWrapper.class – клас що уособлює вид.

package ark.ailab;  
  
public class KindWrapper extends ChildKindWrapper{  
 double qualityValue;  
}

PopulationManager.class – допоміжний клас, що реалізує головну логіку генетичного алгоритму та контролює покоління особин.

package ark.ailab;  
  
import java.util.ArrayList;  
import java.util.Random;  
  
public class PopulationManager {  
 final double CROSSING\_POSSIBILITY = 0.75;  
 final double MUTATON\_POSSIBILITY = 0.1;  
  
 private ArrayList<KindWrapper> population;  
  
 public PopulationManager(ArrayList<KindWrapper> population) {  
 this.population = population;  
 }  
  
 public PopulationManager(int populationNum) {  
 this.population = new ArrayList<>();  
 for (int i = 0; i < populationNum; i++){  
 population.add( createKind() );  
 }  
  
 for (KindWrapper kind : population){  
 kind.qualityValue = calculateQualityValue(kind.geneticValue);  
 }  
 }  
  
 public ArrayList<KindWrapper> getPopulation() {  
 return population;  
 }  
  
 public double calculateGeneticvalue(int x){  
 return ( -1 \* Math.*pow*( (17\*Math.*pow*(x, 2) - 14\*x + 15), 2) ) / 256;  
 }  
  
 public double calculateQualityValue(double individualGeneticValue){  
 return individualGeneticValue / calculatePopulationGeneticValue();  
 }  
  
 public KindWrapper createKind(){  
 int minRange = 0;  
 int maxRange = 255;  
 Random random = new Random();  
  
 return createKind( random.nextInt(maxRange - minRange + 1) + minRange );  
 }  
  
 public KindWrapper createKind(int individual){  
 KindWrapper kind = new KindWrapper();  
 kind.individual = individual;  
 kind.geneticValue = calculateGeneticvalue(kind.individual);  
  
 return kind;  
 }  
  
 public KindWrapper getKindByQualityRange(double quality){  
 double qualityRangeStepsValue = 0;  
 for (KindWrapper kind : population){  
 if (quality >= qualityRangeStepsValue && quality <= qualityRangeStepsValue + kind.qualityValue){  
 return kind;  
 }  
  
 qualityRangeStepsValue += kind.qualityValue;  
 }  
  
 return null;  
 }  
  
 public ChildKindWrapper speciesCrossesResult(KindWrapper parentOne, KindWrapper parentTwo){  
 if (Math.*random*() <= CROSSING\_POSSIBILITY){  
 int childIndividual = parentOne.individual | parentTwo.individual;  
 return kindMutation(childIndividual);  
 }else {  
 return null;  
 }  
 }  
  
 public ChildKindWrapper kindMutation(int individual){  
 ChildKindWrapper child = new ChildKindWrapper();  
  
 if (Math.*random*() <= MUTATON\_POSSIBILITY){  
 child.individual = ~individual;  
 child.isMutant = true;  
 }else {  
 child.individual = individual;  
 child.isMutant = false;  
 }  
  
 child.geneticValue = calculateGeneticvalue(child.individual);  
 return child;  
 }  
  
 */\*\*  
 \* Helper Methods \*/* private double calculatePopulationGeneticValue(){  
 double populationGeneticValue = 0;  
 for (KindWrapper kind : population){  
 populationGeneticValue += kind.geneticValue;  
 }  
  
 return populationGeneticValue;  
 }  
}

MainAppActivity.class – головний клас, що дає початок процесу еволюції та виконує спостереження і логування.

**Результат роботи програми:**

Програма генерує початковий набір особин, що будуть давати потомство. Для даної лабораторної роботи початковий набір зображено на рисунку 1.

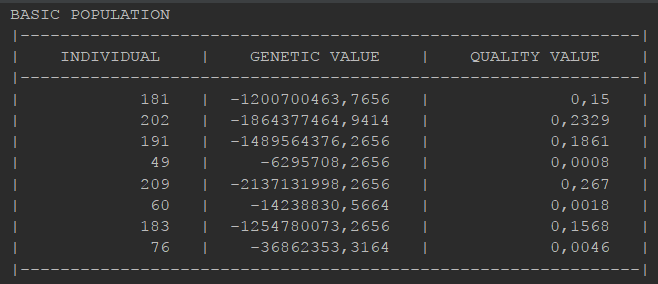


Рисунок 1 – початковий набір особин популяції.

Випадковим чином обираєтсья набір характеристик, що відбере особин здатних відтворити нащадків.

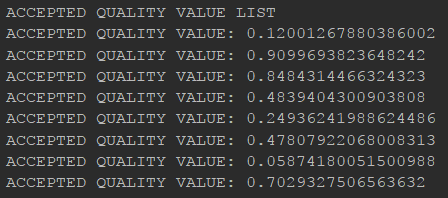


Рисунок 2 – характеристики, яким мають відповідати особини, щоб дати нащадків.

Потім серед усієї популяції відбираються особини, що підходять під визначені раніше характеристики. Список цих особин відображено на рисунку 3.

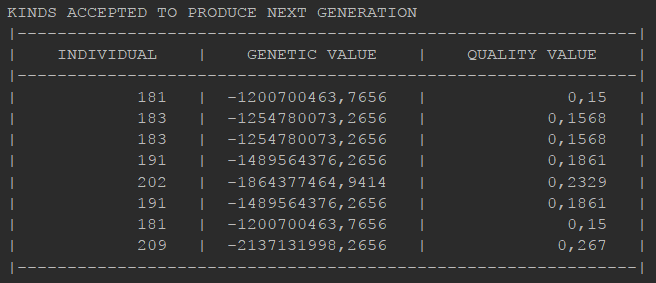


Рисунок 3 – особини, що здатні дати потомство.

Особини, що здатні дати потомство групуються у пари. Отримані пари проходять перевірку чи дали воно потомство чи ні. Якщо пара створила нащадка, ця особина проходять перевірку на мутації. На рисунку 4 зображено список нащадків та інформація про мутації.

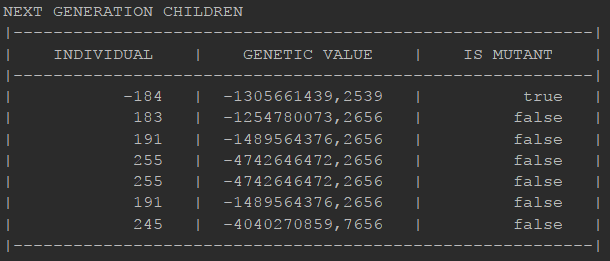


Рисунок 4 – список нащадків та інформації про мутації нащадків.

**Висновок:** під час виконання лабораторної роботи, я розібрався із генетичним алгоритмом. Навчився реалізовувати його на мові програмування Java. Опрацював кожний крок алгоритму та детально прослідкував усі етапи генетичного алгоритму та потомства.