МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет “Львівська політехніка”

Кафедра САПР

Розрахункова робота

На тему:

“Ознайомлення із фреймворком JGAP ”

з курсу “Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні”

Варіант 19(5)

Виконав:

ст. гр. КНC-13

Сологуб С.І.

Прийняв:

Кривий Р.З

Львів – 2017

**Мета**

Ознайомитись із зданим фреймворкоми та навести приклади його використання.

**Індивідуальне завдання (Варіант 5)**

Ознайомитись із фреймворком JGAP**.** Навести теоретичні відомості по роботі із даним фреймворком. Навести конкретні приклади його використання та результати виконання даних прикладів.

**Теоретичні відомості**

JGAP - Генетичноне програмування і Генетичні алгоритми написані на Java. Він забезпечує основні генетичні механізми, які можуть бути легко використані для застосування еволюційних принципів до вирішення завдань. Генетичний алгоритм є потужним засобом для вирішення проблем, з величезним простором розчину, що має обмежений час і потужність процесора.

JGAP був розроблений, щоб бути дуже простим у використанні «з коробки», а також призначений для високим ступенем модульності, так що більш підприємливі користувачі можуть легко вставляються призначені для користувача генетичні оператори, такі як мутації або кросинговеру та інших допоміжних компонентів.

.**Встановлення**

Щоб встановити JGAP, спочатку необхідно переконатись, щопринаймні Java 5 Runtime Environment (JRE) встановлено у вашій системі. (Для цього можна використати команду java --version в командній стрічці (консолі) що покаже вашу версію JRE.). Якщо у вас відсутнє JRE, скачайте та встановіть останню версію для вашої операційної системи, з http://java.sun.com.

Далі вам необхідно завантажити JGAP http://jgap.sourceforge.net/Завантажте останню версію (зараз це, ZIP файл jgap\_3.6.3\_full). Та включити набір бібліотек в проект на java.

**Configurator та Viewer**

Після огляду фреймворку, було виявлено що фреймворк безінтерфейсний. Щоб запустити певний алгоритм на операційній системві Windows потрібно скористатися бат файлом. Для зміни параметрів при розвязуванні задачі потрібно відкрити відповідний java файл і змінити необхідні парамети. Після чого запустити бат файл для запуску алгоритму вирішення задачі.

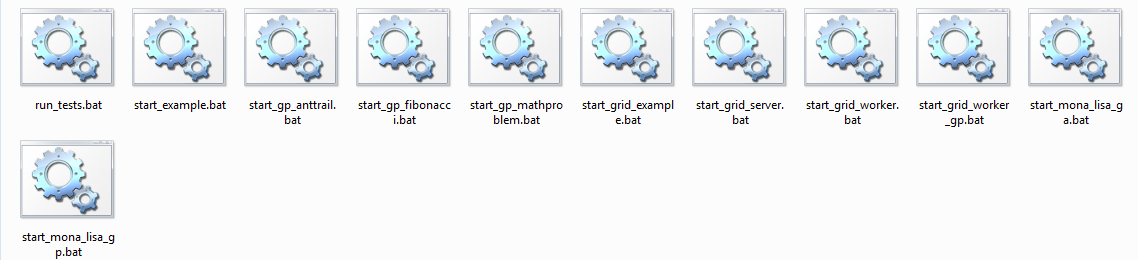


Рис.1 Свисок бат файлів

Для всіх алгоритмів відсутні бат файли для запуску. Тому потрібно написати їх самому або запускати через консоль.

**Розуміння** **та використання JGAP**

Щоб використовувати JGAP в програмах, є п'ять основних речей, які потрібно зробити:

1. Спроектувати хромосому
2. Реалізувати фітнес функції
3. Налаштувати об’єкт конфігурації
4. Створити популяцію потенційних рішень
5. Еволюціонувати популяцію

Далі буде покроково створено та представлено простий приклад програми, що використовує JGAP.

## Крок 1. Проектування хромосоми

В основі генетичного алгоритму є Хромосома. Хромосома являє собою потенційне рішення і розділена на кілька генів. Гени в JGAP представляють різні аспекти вирішення в цілому, так само, як людські гени являють собою різні аспекти окремих людей, наприклад, статі або кольору очей. В процесі еволюції JGAP, хромосоми піддаються кільком генетичних операторам, які представляють собою схрещування, мутацію і т.д., а потім вибираються для наступного покоління під час природної фази відбору на основі їх «придатності», яка є мірою того, наскільки оптимальним рішення є відносно інших можливих рішень. Вся мета генетичного алгоритму, полягає у тому щоб імітувати природний процес еволюції з метою отримання кращих рішень.

Крок 1 полягає у прийнятті рішення про склад хромосом, який включає в себе, скільки генів потрібно і що ці гени будуть представляти. Програма, що розглядається полягає в пошуку найменшої кількості монет різного номіналу сума яких буде рівною числу, що задасть користувач. Таким чином хромосомою є набір монет, а генами різні номінали монет, наприклад, номінали американських монет (рис.1).

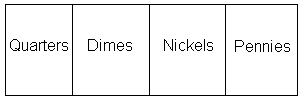


Рис.3 Хромосома з чотирьома генами

## Крок 2. Реалізація фітнес функції

JGAP розроблений, щоб зробити майже всю еволюційну роботу для користувача у відносно загальному вигляді. Тим не менш, немає загального шляху вирішення, для будь якої проблеми. Ось де фітнес функція є необхідною: це єдиний метод, який необхідно реалізувати, який приймає потенційне рішення задачі і повертає ціле значення, яке вказує, наскільки кращим є рішення щодо інших можливих рішень. Чим вище число, тим краще рішення. Чим менше число (1 є найнижчим допустимим значенням придатності), тим гірше рішення.

Оскільки мета цієї програми, щоб знайти найменший набір американських монет при якому сума рівна заданій, зрозуміло, що міра придатності для конкретного рішення буде являти собою поєднання суми грошей та загальної кількості монет.

**package** examples;

**import** org.jgap.Chromosome;

**import** org.jgap.FitnessFunction;

**public** **class** MinimizingMakeChangeFitnessFunction **extends** FitnessFunction

{

**private** **final** **int** m\_targetAmount;

**public** MinimizingMakeChangeFitnessFunction( **int** a\_targetAmount )

{

**if**( a\_targetAmount < 1 || a\_targetAmount > 99 )

{

**throw** **new** IllegalArgumentException(

"Change amount must be between 1 and 99 cents." );

}

m\_targetAmount = a\_targetAmount;

}

**public** **double** evaluate( IChromosome a\_subject )

{

**int** changeAmount = *amountOfChange*( a\_subject );

**int** totalCoins = *getTotalNumberOfCoins*( a\_subject );

**int** changeDifference = Math.*abs*( m\_targetAmount - changeAmount );

**double** fitness = ( 99 - changeDifference );

**if**( changeAmount == m\_targetAmount )

{

fitness += 100 - ( 10 \* totalCoins );

}

**return** fitness;

}

**public** **static** **int** amountOfChange( IChromosome a\_potentialSolution )

{

**int** numQuarters = *getNumberOfCoinsAtGene*( a\_potentialSolution, 0 );

**int** numDimes = *getNumberOfCoinsAtGene*( a\_potentialSolution, 1 );

**int** numNickels = *getNumberOfCoinsAtGene*( a\_potentialSolution, 2 );

**int** numPennies = *getNumberOfCoinsAtGene*( a\_potentialSolution, 3 );

**return** ( numQuarters \* 25 ) + ( numDimes \* 10 ) + ( numNickels \* 5 ) +

numPennies;

}

**public** **static** **int** getNumberOfCoinsAtGene( IChromosome a\_potentialSolution,

**int** a\_position )

{

Integer numCoins =

(Integer) a\_potentialSolution.getGene(a\_position).getAllele();

**return** numCoins.intValue();

}

**public** **static** **int** getTotalNumberOfCoins( IChromosome a\_potentialsolution )

{

**int** totalCoins = 0;

**int** numberOfGenes = a\_potentialsolution.size();

**for**( **int** i = 0; i < numberOfGenes; i++ )

{

totalCoins += *getNumberOfCoinsAtGene*( a\_potentialsolution, i );

}

**return** totalCoins;

}

}

Для початку, визначаємо власний клас і наслідуємо клас org.jgap.FitnessFunction. Всі фітнес-функції повинні наслідувати клас FitnessFunction. Потім визначаємо конструктор і метод оцінки. Метод оцінки evaluate() є стандартним методом, що всі фітнес функції повинні виконувати. Тобто метод, який буде викликатися генетичним двигуном, коли він повинен знайти, фітнес значення хромосоми.

Конструктор є не дуже цікавим: він просто приймає величину зміни цільового значення, що бажає користувач, перевіряє, чи сума відповідає обмеженню від 1 до 99 центів, а потім зберігає введену суму в змінному екземплярі для подальшого використання.

Цікава частина цілого класу є метод оцінки evaluate(), в якому і відбувається основна робота. Метод оцінки завжди передається в хромосомі, яка представляє потенційне рішення. Хромосома складається з генів, кожен з яких представляє собою відповідну частину рішення. У прикладі, Хромосома являє собою суму монет, в той час як гени являють собою конкретні види монет: 25 копійок для першого гена, 10 для другого, 5 для третього, і 1 для четвертого.

Зрештою, ми хочемо повернути високі значення придатності для рішень, які відповідають цільовій сумі з дуже малою кількістю монет, і повернути більш низькі значення придатності для рішень, які знаходяться далеко очікуваної суми або представляють велику кількість монет.

## Крок 3. Налаштування конфігурації об’єкта

JGAP призначений бути дуже гнучким і легко налаштовуваним. Він дозволяє створювати свої власні генетичні оператори, генератори випадкових чисел, натуральні селектори і так далі. Для того, щоб підтримати все це, JGAP використовує об'єкт конфігурації, який повинен бути встановлений з усіма параметрами, які користувач хоче застосовувати. Для зручності наявний стандартний клас DefaultConfiguration, який поставляється вже налаштований з найбільш поширеними параметрами. Потрібно лише надати три додаткові частини інформації: які фітнес функції використовувати, як Хромосоми мають бути налаштовані, і скільки Хромосом має бути у популяції. Розглянемо приклад:

**public** **static** **void** main(String[] args) **throws** Exception {

Configuration conf = **new** DefaultConfiguration();

**int** targetAmount = Integer.parseInt(args[0]);

FitnessFunction myFunc =

**new** MinimizingMakeChangeFitnessFunction( targetAmount );

conf.setFitnessFunction( myFunc );

Gene[] sampleGenes = **new** Gene[ 4 ];

sampleGenes[0] = **new** IntegerGene(conf, 0, 3 ); // Quarters

sampleGenes[1] = **new** IntegerGene(conf, 0, 2 ); // Dimes

sampleGenes[2] = **new** IntegerGene(conf, 0, 1 ); // Nickels

sampleGenes[3] = **new** IntegerGene(conf, 0, 4 ); // Pennies

Chromosome sampleChromosome = **new** Chromosome(conf, sampleGenes );

conf.setSampleChromosome( sampleChromosome );

conf.setPopulationSize( 500 );

}

Для початку створюється об’єкт класу Configuration як DefaultConfiguration, тобто конфігурація по замовчуванню. Тоді оголошується фітнес функція, розроблена раніше, з вхідним параметром – сумою грошей.

Після цього за допомогою метода setFitnessFunction() оголошена фітнес функція підключається до конфігурації. Тоді оголошується масив генів, який є одним з вхідних параметрів Хромосоми, що оголошується після.

Останнім кроком налаштування є додавання до конфігурації оголошеної Хромосоми та розміру популяції.

## Крок 4. Створення популяції

Популяція хромосом називається генотипом, оголошення об’єкта цього класу і є наступним пунктом налаштування ГА. Можна побудувати кожну хромосому окремо, а потім створити з них генотип, так як створювалась хромосома з генів у кроці 3. Але в JGAP є простіший та легший спосіб для створення випадкової популяції:

Genotype population = Genotype.randomInitialGenotype( conf );

Метод randomInitialGenotype () приймає об’єкт конфігурації (який був налаштований на кроці 3) і повертає генотип з правильною кількістю хромосом, кожна з яких має свої випадкові гени. Іншими словами, він генерує випадкову популяцію. Для більшості програм, це все, що необхідно для створення початкової популяції потенційних рішень.

## Крок 5. Еволюція

Наступним кроком після всіх налаштувань є еволюція. Найпростішим способом зробити один цикл еволюції є:

population.evolve();

Для визначення найкращої хромосоми з популяції застосовується метод:

IChromosome bestSolutionSoFar = population.getFittestChromosome();

Якщо отримане рішення не влаштовує можна провести еволюцію знову та знову, або задати кількість еволюцій та провести їх у циклі.

IChromosome bestSolutionSoFar;

**for**( **int** i = 0; i < MAX\_ALLOWED\_EVOLUTIONS; i++ )

{

population.evolve();

}

System.out.println( "The best solution contained the following: " );

System.out.println(

MinimizingMakeChangeFitnessFunction.getNumberOfCoinsAtGene(

bestSolutionSoFar, 0 ) + " quarters." );

System.out.println(

MinimizingMakeChangeFitnessFunction.getNumberOfCoinsAtGene(

bestSolutionSoFar, 1 ) + " dimes." );

System.out.println(

MinimizingMakeChangeFitnessFunction.getNumberOfCoinsAtGene(

bestSolutionSoFar, 2 ) + " nickels." );

System.out.println(

MinimizingMakeChangeFitnessFunction.getNumberOfCoinsAtGene(

bestSolutionSoFar, 3 ) + " pennies." );

System.out.println( "For a total of " +

MinimizingMakeChangeFitnessFunction.amountOfChange(

bestSolutionSoFar ) + " cents in " +

MinimizingMakeChangeFitnessFunction.getTotalNumberOfCoins(

bestSolutionSoFar ) + " coins." );

**Приклади та результати**

**Приклад 1. Monalisa**



Рис. 3 Робота алгоритму після ~2500 популяцій

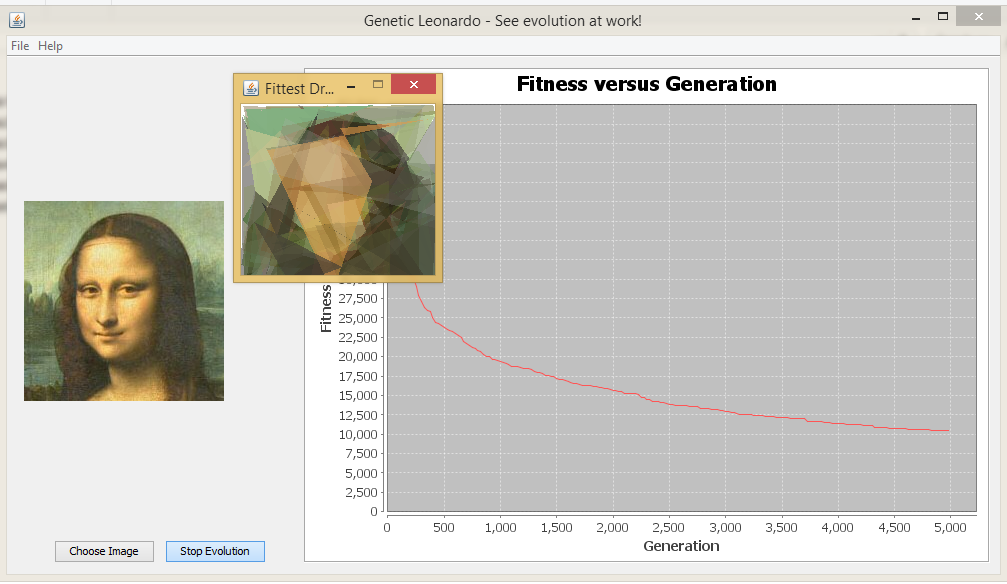


Рис. 4 Робота алгоритму після ~5000 популяцій

**Приклад 2. Gp anttrail**

**Демонструє, як рухатися мураха, вибираючи стільки їжі, скільки можливо**

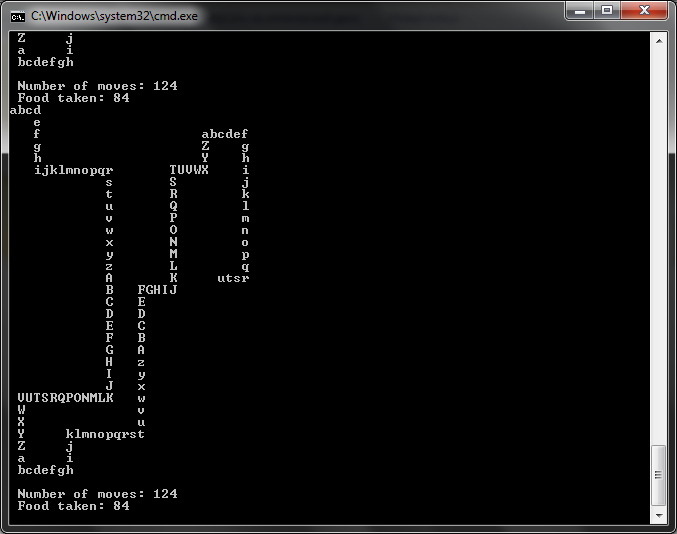
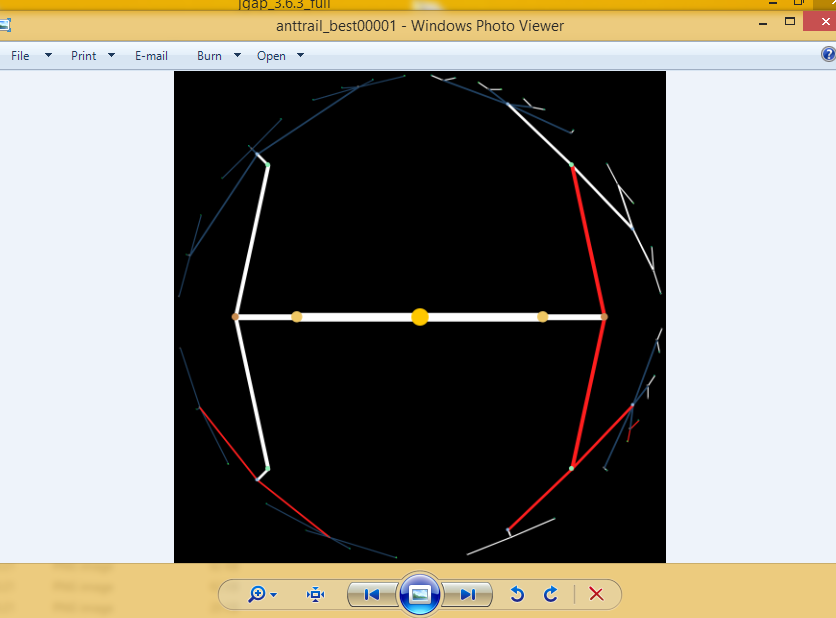
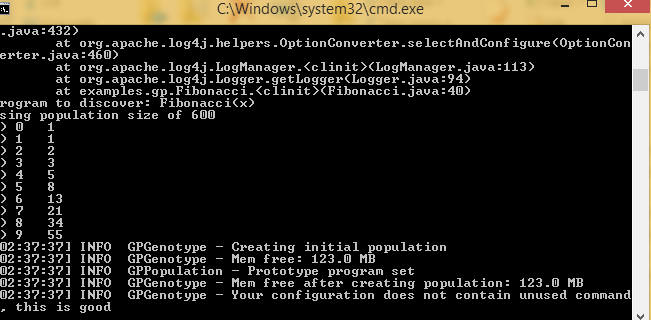
 

Рис.5 Результат виконання

**Приклад 3. Fibonacchi**



**Висновок**

В результаті виконання розрахункової роботи озайомився з фреймворком для еволюційних обчислень JGAP. Навів короткі теоретичні відомості, щодо структури та роботи. Навів приклади роботи даного фреймворку з результатами виконання.