Прізвище: Якубенко

Ім’я: Віталій

Група: КНМ-14

Дата прийняття роботи

у системі Git:

Дисципліна: Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні

Перевірив: Кривий Р.З.

**Розрахунково-графічна робота**

**«EvoJ framework»**

**МЕТА РОБОТИ**

Ознайомитися з основними теоретичними відомостями про фреймворк EvoJ для рішення задач, використовуючи генетичні алгоритми у мові програмування Java.

**Опис фреймворка**

EvoJ є основою Java, яка дозволяє програмістам і вченим легко вирішити проблеми з використанням генетичного алгоритму. Основна особливість структури є те, що не потрібно будувати хромосоми вручну - все, що користувачу потрібно зробити - це оголосити Java інтерфейс, який тримає всі змінні, необхідні для вирішення ваших проблем.

Ключові особливості структури є:

• простота - сама EvoJ вимагає всього кілька рядків коду, що дозволяє

зосередитися на проблемі, яка вирішується;

• відсутність зовнішніх залежностей - все, що потрібно - це evoj-3.0.jar;

• розширюваність – є можливість налаштувати будь-який етап еволюційного алгоритму;

• вбудована підтримка штучних нейронних мереж;

• вбудована підтримка багатопоточності для підвищення продуктивності;

• декларативного управління мутацією - використовувати Java анотації або власність, як файли для управління як ваші змінні мутувати.

**Основні принципи**

**Автоматичне відображення хромосоми**

Якщо потрібно вирішити проблему за допомогою генетичного алгоритму, то зазвичай маємо деякий набір змінних на увазі. Генетичний підхід вимагає, щоб організувати ці змінні в лінійну хромосому, а потім виконати безліч еволюційних операцій над такими хромосоми.

Щоб спростити процес відображення і дозволити програмістам зручно використовувати можливості сучасних Java IDEA, EvoJ може ставитися до Java-інтерфейсів як до хромосом.

Припустимо, що ми вирішуємо задачу про знаходження оптимального багатокутник в полярних координатах, давайте забудемо про фітнес-функції в даний час. У цьому випадку змінні, якими ми повинні були б описувати рішення, можуть бути вираженні як Java інтерфейси.

EvoJ сканує інтерфейс і виділяє масив Object, щоб тримати всі змінні. Потім масив використовується в якості хромосоми в кросинговеру і мутації операцій.

**Підтримуванні типи змінних**

Користувач може оголосити будь-яку кількість властивостей в інтерфейсі. Властивості можуть бути примітивними типами об'єктів і списками. Правила оголошення інтерфейсів рішень є:

1. Імена методів повинні починатися з «get» або «set», навіть якщо властивість логічна;

2. Геттера не повинні мати ніяких параметрів і повинні повертати якесь значення.

3. Сетери повинні мати один параметр, який має точно такий же тип, що повертається методом “get”;

4. Властивостями можуть бути:

а) будь-який примітивний тип або відповідний тип об'єкта (буде згадуватися як «простий тип »);

б) клас Enum. Властивості Enum розглядаються як прості типи

с) інтерфейс відповідно до цих правил (буде згадуватися як «складний тип»)

г) Список простого або складного типу

**Базові класи фреймворка**

**Factories, Individuals, GenePools**

Популяція рішень в EvoJ представлена класом генофонду, який тримає список примірників об'єктів, що реалізують певний інтерфейс рішення. В той самий час ці об'єкти реалізовують індивідуальний інтерфейс, який використовується класами стратегії для доступу до низькорівневого представлення - ДНК, рейтинг і властивості віку.

Генофонд практично розділений на елітні і не елітні частини. Елітна частина є перша особа в списку. Кількість елітних особин визначається властивість генофонду eliteCount. Там повинна бути принаймні один елітний індивід. Елітні особини потім схрещують для утворення наступного покоління рішень, нові народжені люди.

Насправді ніякі нові окремі об'єкти не породжуються, замість того ДНК з існуючих перезаписуються, вік скидається в 0, і рейтинг також. Це робиться для підвищення продуктивності і зменшення кількості породженого об'єкта на кожній ітерації, так як число ітерацій, необхідних для вирішення проблеми, як правило, враховуються в тисячі або навіть мільйони.

Оскільки рішення в EvoJ представлені у вигляді Java-інтерфейсів ми не можемо створити екземпляр безпосередньо. Для цих цілей існують PoolFactories і Individual factories.

Існує тільки один клас, який реалізує інтерфейс PoolFactory – DefaultPoolFactory. Він повинен, зазвичай, використовується програмістами для конкретизації генофондів.

**Handlers**

Обробники є втіленням самого генетичного алгоритму. Інтерфейс Handlerвизначається як:

public interface PoolHandler<T> {

void iterate(GenePool genePool);

void iterate(GenePool genePool, int count);

boolean iterate(GenePool genePool, SuccessCondition success, long maxIter);

}

Типова ітерація, яка виконується обробниками, складається з наступних кроків:

1. Переконайтеся, що генофонд сортується відповідно до фітнес-функції (краще рішення приходять в першу чергу)

2. Утворіть нове покоління рішень в не-елітній частині генофонду використовуючи елітні особи, як батьківські.

3. мутувати деякі новонароджені осіби.

4. Збільшіть вік елітних осіб.

5. Сортування генофонду згідно придатності.

Як ви можете бачити, після ітерації генофонд завжди сортується.

Є два класи, які реалізують інтерфейс Handler:

* DefaultHandler
* MultithreadedHandler.

Ці два інтерфейси роблять ті ж речі з точки зору генетичного алгоритму,

єдина відмінність полягає в тому, що останні використовують переваги сучасних багатоядерних процесорів і обчислюють поняття в потрібній темі.

Кожна операція генетичного алгоритму, що виконується обробниками ховається за відповідним інтерфейсом стратегії. Там існують стандартні стратегії вибору, ДНК рекомбінації і мутації, які можуть бути налаштовані програмістами перевизначенням відповідних інтерфейсів. Однак це не є обов'язковим.

Єдиною стратегією, яку програміст повинен реалізувати - це інтерфейс PoolSorter, так як він є специфічним для розв'язуваної задачі. Там існують допоміжні класи, такі як AbstractSimpleRating, які спрощують реалізацію цієї стратегії.

Стратегії повинні бути приведені в якості параметрів конструкторів оброблювачів. Якщо потрібно використовувати версію стратегії за замовчуванням, то просто дати нуль для відповідного параметра.

**Strategies** **(Стратегії)**

***Fitness***

У EvoJ пристосованість індивіда представлено положення, яке він займає в генофонді(GenePool). Чим ближче до початку - тим вище fitness має індивід. Обробники використовують інтерфейс PoolSorter щоразу, коли потрібні індивіди в генофонді повинні бути впорядковані за їх придатністю.

Тоді положення індивіда використовується в реалізаціях SelectionStrategy щоб вибрати батька індивіда наступного покоління.

Незважаючи на те, що інтерфейс PoolSorter є обов'язковим для реалізації програмістом, немає необхідності реалізовувати його безпосередньо.

Існує безліч допоміжних класів, які спрощують цей процес в загальних випадках.

Існують два основні підходи до визначення придатності:

• на основі оцінки - коли ви можете думати про функцію, яка визначає

фітнес розчину в якості порівнянної величини;

• на основі порівняння - коли складно виразити пристосованість рішення як значення, але легко порівняти два рішення разом і сказати, яке з цих двох краще.

***Мутація***

Друга важлива стратегія, яку можна налаштувати це MutationStrategy.

Інтерфейс визначається як:

public interface MutationStrategy {

void mutate(Individual i);

}

Ця стратегія застосовується до кожної новонародженої особи.

Зазвичай немає необхідності реалізовувати цей інтерфейс, так як найбільш поширені випадки використання покриті класом DefaultMutationStrategy. Він виконує мутації відповідно до конфігурації мутації, зазначеної в анотації і контекстних властивостях. Більш того, можна вказати, скільки новонароджені особини будуть мутувати, і скільки елементів ДНК мутуватимуть через параметри конструктора.

Екземпляр DefaultMutationStrategy створюється наступним чином:

DefaultMutationStrategy m = new DefaultMutationStrategy(0.1, 0.3, 0.8);

Перший параметр конструктора визначає ймовірність того, що новонароджена особа буде мутувати взагалі. Другий параметр визначає, скільки змінних будуть порушені в результаті мутації, якщо число менше 1, то воно розглядається як частина всіх змінюваних елементів. Останній параметр вказує ймовірність того, що замість того, щоб мутувати відповідно до анотації @MutationRange обраної змінної буде дано випадкове значення в межах його діапазону, тобто буде скидати значення.

***Селекція***

Метою інтерфейсу SelectionStrategy є вибрав пару батьків, що будуть

використовується для утворення новонародженої особи. Нова пара вибирається для кожного новонародженої особини.

Інтерфейс оголошується наступним чином:

public interface SelectionStrategy {

void performSelection(List pool, int[] pair);

}

Тут параметр pool є елітною частиною генофонду в стадії обробки.

Реалізація повинна вибрати дві особи, і вказати їх індекси в масиві пар.

Якщо eliteCount з генофонду рівний 1 або 2, то стратегія вибору не використовується, тому що в цих випадках батьки очевидні.

Є дві вбудовані стратегії вибору:

* CompetitiveSelectionStrategy
* EqualChanceSelectionStrategy.

CompetitiveSelectionStrategy обирає індивіди з точки зору їх місця в «басейні»(pool), тим краще ті, які вибирають більш часто. Ця стратегія встановлюється за замовчуванням.

EqualChanceSelectionStrategy вибирає батьків з однаковою ймовірністю

незалежно від їх положення в «басейні».

***Реплікація***

Інтерфейс ReplicationStrategy оголошений як:

public interface ReplicationStrategy {

void setChildDNA(Individual child, Individual parent1, Individual parent2);

}

Інтерфейс повинен встановити ДНК дитини шляхом змішання ДНК обох батьківських особин.

Вбудованою реалізацією стратегії є RandomInterleaveCrossingover і MultisplitCrossingover.

MultisplitCrossingover розщеплює ДНК на секції в потрібних точках розділення (заданих через параметр конструктора) і виробляє нову ДНК

чергуванням ділянок ДНК від обох батьків.

На малюнку нижче показано, як працює клас MultisplitCrossingover.

Стрілки позначають довільно вибрані спліт-точки.

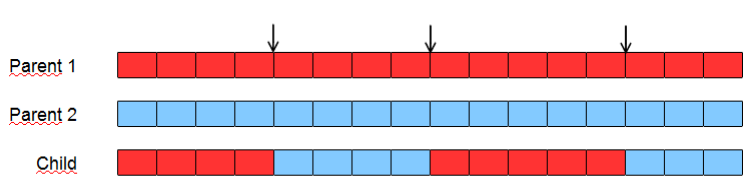
****

Рис. 1.Схематичне зображення роботи

методу MultisplitCrossingover

RandomInterleaveCrossingover просто проходить через елементи з

ДНК і випадковим чином вирішує, від якого батька приймати відповідний елемент. Схема наведена нижче:

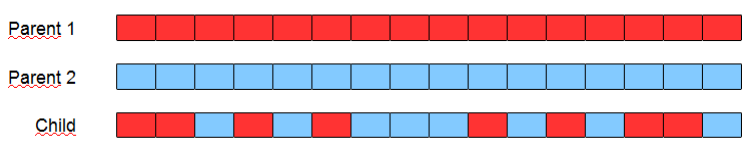


Рис. 2. Схематичне зображення роботи методу

RandomInterleaveCrossingover

Стратегія реплікації за замовчуванням є MultisplitCrossingover з однією точкою розподілу. Рідко виникає необхідність реалізації користувальницької стратегії реплікації.

**Встановлення фреймворка**

При дослідженні даного фреймворка, я знайшов два способи підключення його до проекту:

* Завантажити архів із усіма функціональними класами фреймворка із офіційного сайту;
* Встановити фреймворк, використовуючи Maven.

Завантаження із офіційного сайту:

1. Зайти на офіційний сайт, використавши наступне посилання <http://evoj-frmw.appspot.com/index.html>
2. Вибрати пункт Download EvoJ в колонці, яка зображена на рис. 3.

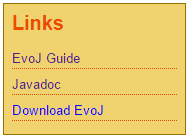
****

Рис. 3. Завантаження фреймворка з

офіційного сайту

1. Зберегти архів на персональному комп’ютері.
2. Розпакувати завантажений файл.
3. При розробці програми, використовуючи генетичні алгоритми, щоразу вручну підключати необхідні файли із завантаженого файлу.

Встановлення фреймворка, використовуючи Maven:

1. Запустити середовище розробки програмного забезпечення(для цього я обрав середовище Intellij IDEA)
2. Створити новий проект

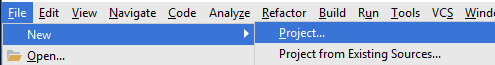


Рис. 4. Створення нового проекту в середовищі Intellij IDEA

1. Вибрати проект Maven та версію SDK із випадаючого списку, або використовуючи кнопку New… та натиснути Next.

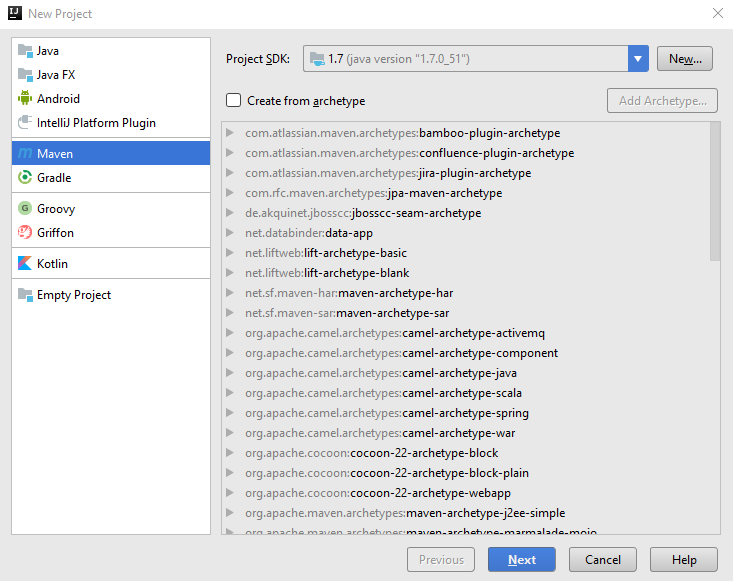


Рис. 5. Вибір параметрів проекту

1. Вказати назву майбутнього проекту та директорію програми та натиснути Finish.
2. Після створення проекту у файлі pom.xml необхідно прописати репозиторій та залежності.

Після виконання даних кроків, середовище підгрузить усі бібліотеки, які містяться у фреймворку і користувач може приступати до розв’язання поставленої задачі.

Лістинг файлу pom.xml в кінцевому варіанті:

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**groupId**>test</**groupId**>  
 <**artifactId**>testProj</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 <**repositories**>  
 <**repository**>  
 <**id**>EvojHome</**id**>  
 <**name**>EvoJ</**name**>  
 <**url**>http://evoj-frmw.appspot.com/repository</**url**>  
 </**repository**>  
 </**repositories**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>net.sourceforge.evoj</**groupId**>  
 <**artifactId**>evoj</**artifactId**>  
 <**version**>2.1</**version**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

**Приклад 1**

За допомогою засобів мови програмування Java я розробив програмне забезпечення для задачі знаходження мінімального значення функції:

**Фрагменти коду програми**

***Main.java***

**import** net.sourceforge.evoj.GenePool;  
**import** net.sourceforge.evoj.core.DefaultPoolFactory;  
**import** net.sourceforge.evoj.handlers.DefaultHandler;  
**import** net.sourceforge.evoj.strategies.mutation.SimpleBitMutation;  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** main {  
 **public static void** main(String[] args) {  
 DefaultPoolFactory pf = **new** DefaultPoolFactory();  
 GenePool<Solution> pool = pf.createPool(200, Solution.**class**, **null**);  
 DefaultHandler handler = **new** DefaultHandler(**new** Rating(), **null**, **null**, **null**);  
  
 handler.iterate(pool, 100);  
 Solution solution = pool.getBestSolution();  
 pool.getBestIndividual().getRating();  
 System.***out***.println(**"X ="** + solution.getX());  
 System.***out***.println(**"Y ="** + solution.getY());  
 System.***out***.println(**"Rating ="** + pool.getBestIndividual().getRating());  
 }  
}

***Rating.java***

**import** net.sourceforge.evoj.Individual;  
**import** net.sourceforge.evoj.strategies.sorting.AbstractSimpleRating;  
  
**public class** Rating **extends** AbstractSimpleRating<Solution> {  
 **public static double** calcFunction(Solution solution) {  
 **double** x = solution.getX();  
 **double** y = solution.getY();  
 **return** 12 \* x \* x + 8 \* x + 9 \* y \* y;  
 }  
  
 @Override  
 **public** Double doCalcRating(Solution solution) {  
 **double** fn = *calcFunction*(solution);  
 **if** (Double.*isNaN*(fn)) {  
 **return null**;  
 } **else** {  
 **return** -fn;  
 }  
 }  
  
 @Override  
 **public boolean** equals(Object obj) {  
 **if** (obj == **null**) {  
 **return false**;  
 }  
 **if** (getClass() != obj.getClass()) {  
 **return false**;  
 }  
  
 **return true**;  
 }  
  
 @Override  
 **public int** hashCode() {  
 **int** hash = 5;  
 **return** hash;  
 }  
}

***Solution.java***

**import** net.sourceforge.evoj.core.annotation.Range;  
**public interface** Solution {  
 @Range(min = **"-10"**, max = **"10"**)  
 **double** getX();  
  
 @Range(min = **"-10"**, max = **"10"**)  
 **double** getY();  
}

**Результати виконання завдання**

При першому запуску, я задав розмір популяцій в 200 особин та кількість ітерацій(генерацій) – 100.

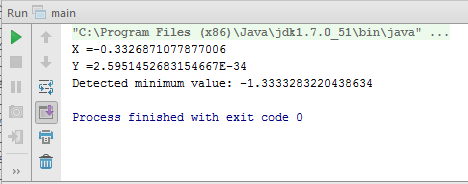


Рис. 6. Розв`язок задачі при початкових умовах

При другому запуску я змінив параметри мутації, збільшив розмір популяції на кількість генерацій до 200.

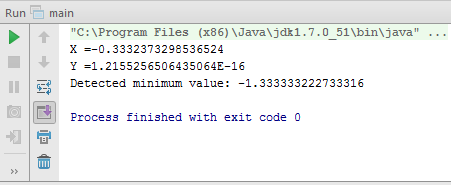


Рис. 7. Розв’язок задачі при зміні параметрів

При третьому запуску я змінив ще раз змінив параметри мутації та зменшив розмір популяції до 320 особин.

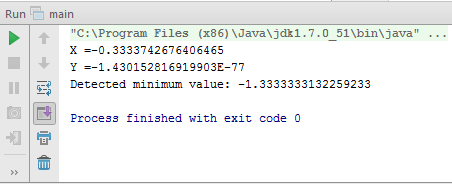


Рис. 8. Розв’язок задачі при змінні параметрів

**Приклад 2**

Реалізація прикладу, який показує можливий підхід до проблеми упаковки деталей в контейнери.

**Фрагменти коду програми**

***Main.java***

**package** net.sourceforge.evoj.examples.binpacking;  
  
**import** net.sourceforge.evoj.GenePool;  
**import** net.sourceforge.evoj.Individual;  
**import** net.sourceforge.evoj.multithreading.IntermixType;  
**import** net.sourceforge.evoj.multithreading.MultiThreadedGenePool;  
**import** net.sourceforge.evoj.mutation.FlipMutation;  
  
**public class** Main {  
  
 **public static void** main(String[] args) {  
  
 **int**[] logs = {10, 9, 12, 12, 14, 12, 12, 9, 9, 11, 10, 12, 12, 9, 9,  
 12, 12, 12, 12, 12, 12, 12, 12};*//lengths of available logs* **int**[] details = {3, 3, 4, 5, 3, 7, 7, 5, 4, 4, 4, 8, 8, 5, 5, 5, 5,  
 3, 3, 8, 8, 8, 8, 9, 5, 4, 5, 5, 6};*//lengths of required details*

LumberMill lm = **new** LumberMill(logs, details);

GenePool gp1 = **new** GenePool(400, 0, lm, **null**, **null**, lm, lm);  
 GenePool gp2 = **new** GenePool(300, 0, lm, **null**, **null**, lm, lm);  
 GenePool gp3 = **new** GenePool(200, 0, lm, **null**, **null**,  
 **new** FlipMutation(0.3, lm), lm);  
 GenePool gp4 = **new** GenePool(100, 0, lm, **null**, **null**,  
 **new** FlipMutation(0.8, lm), lm);  
 gp3.setEliteCount(15);  
  
 GenePool[] pools = {gp1, gp2, gp3, gp4};  
  
 MultiThreadedGenePool mt = **new** MultiThreadedGenePool(pools);  
  
**for** (**int** i = 1; i <= 50; i++) {  
 mt.iterate(200);  
  
 **if** (i % 10 == 0) {  
 GenePool minPool = gp1;  
 Individual min = gp1.getChampion();  
 **for** (GenePool p : pools) {  
 **if** (p.getChampion().compareTo(min)<0) {  
 minPool = p;  
 min = p.getChampion();  
 }  
 }  
mt.intermix(IntermixType.***ELITE***);  
minPool.reset();  
 }  
  
 *//print out current results* System.***out***.print(lm.asString(mt.getChampion()));  
 **for** (GenePool gp : pools) {  
 System.***out***.print(**"Rating:"** + (Double)gp.getChampion().getRating());  
 System.***out***.println(**" Age:"** + gp.getChampion().getAge());  
 }  
 }mt.dispose();  
 }  
}

***LumberMill.java***

**package** net.sourceforge.evoj.examples.binpacking;  
  
**import** net.sourceforge.evoj.Individual;  
**import** net.sourceforge.evoj.BirthStrategy;  
**import** net.sourceforge.evoj.MutationStrategy;  
**import** net.sourceforge.evoj.RatingStrategy;  
**import** java.util.ArrayList;  
**import** java.util.Arrays;  
**import** java.util.Random;  
  
**public class** LumberMill **implements** BirthStrategy, MutationStrategy,  
 RatingStrategy {  
  
 **int**[] **logs**;  
 **int**[] **details**;  
ThreadLocal **tl2** = **new** ThreadLocal() {  
  
 @Override  
 **protected** Object initialValue() {  
 **return new int**[**logs**.**length**];  
 }  
 };  
 **protected** ThreadLocal<Random> **tl** = **new** ThreadLocal<Random>() {  
  
 @Override  
 **protected** Random initialValue() {  
 **return new** Random();  
 }  
 };  
  
 **public** LumberMill(**int**[] logs, **int**[] details) {  
 **this**.**logs** = Arrays.*copyOf*(logs, logs.**length**);  
 **this**.**details** = Arrays.*copyOf*(details, details.**length**);  
 }  
  
 **public** Individual getNewIndividual() {  
 Individual result = **new** Individual(**details**.**length** \* 2);  
 **for** (**int** i = 0; i < **details**.**length**; i++) {  
 result.setShort(i << 1, (**short**) **tl**.get().nextInt(**logs**.**length**));  
 }  
 **return** result;  
 }  
**public void** mutate(Individual i) {  
 Random rnd = **tl**.get();  
 **if** (rnd.nextDouble() < 0.4) {*//mutate 40% of Individuals* **for** (**int** k = 0; k < **details**.**length**; k++) {  
 **if** (rnd.nextDouble() < 0.2) {*//Mutate 20% of DNA* i.setShort(k << 1, (**short**) rnd.nextInt(**logs**.**length**));  
 }  
 }  
 }  
 }  
  
 **public void** calcRating(Individual[] pool, **int** eliteCount) {  
 **for** (**int** i = eliteCount; i < pool.**length**; i++) {  
 setRating(pool[i]);  
 }  
eliminateTwins(pool, eliteCount);  
 }  
  
 **private void** setRating(Individual ind) {  
 **int**[] tmp = (**int**[]) **tl2**.get();  
 **for** (**int** i = 0; i < tmp.**length**; i++) {  
 tmp[i] = 0;  
 }  
  
 **for** (**int** i = 0; i < **details**.**length**; i++) {  
**short** lg = ind.getShort(i << 1);  
 **if** ((lg >= **logs**.**length**) || (lg < 0)) {  
ind.setRating(**null**);  
 **return**;  
 }  
tmp[lg] += **details**[i];  
 }  
 **double** positive = 0;*//positive accumulator* **double** negative = 0;*//negative accumulator* **for** (**int** i = 0; i < tmp.**length**; i++) {  
 **int** dl = **logs**[i] - tmp[i];*//calculate difference* **if** (dl < 0) {  
 negative += dl \* dl;  
 } **else** {  
 positive += dl \* dl;  
 }  
 }  
 **if** (negative > 0) {  
 ind.setRating(-negative);  
 } **else** {  
 ind.setRating(positive);  
 }  
 }  
  
 **public** String asString(Individual ind) {  
 **int**[] tmp = (**int**[]) **tl2**.get();  
 **for** (**int** i = 0; i < tmp.**length**; i++) {  
 tmp[i] = 0;  
 }  
 ArrayList<Integer>[] logDet = **new** ArrayList[**logs**.**length**];  
 **for** (**int** i = 0; i < **logs**.**length**; i++) {  
 logDet[i] = **new** ArrayList<Integer>();  
 }  
 **for** (**int** i = 0; i < **details**.**length**; i++) {  
**short** lg = ind.getShort(i << 1);  
tmp[lg] += **details**[i];  
 logDet[lg].add(**details**[i]);  
 }  
 StringBuilder sb = **new** StringBuilder();  
  
 **for** (**int** i = 0; i < **logs**.**length**; i++) {  
 sb.append(**"Log "** + i + **" length:"** + **logs**[i]);  
 sb.append(**" details length:"**);  
 sb.append(tmp[i] + **" "**);  
 sb.append(logDet[i].toString() + **"\n"**);  
 }  
  
 **return** sb.toString();  
 }  
  
 @Override  
 **public int** getDNASize() {  
 **return details**.**length** \* 2;  
 }  
  
 **public void** eliminateTwins(Individual[] pool, **int** eliteCount) {  
 **for** (**int** i = 0; i < eliteCount - 1; i++) {  
 **if** (pool[i].compareTo(pool[i + 1])==0) {  
 **if** (pool[i].equals(pool[i + 1])) {  
pool[i].setRating(**null**);  
 }  
 }  
 }  
 }  
}

***MainProfiling.java***

**package** net.sourceforge.evoj.examples.binpacking;  
  
**import** net.sourceforge.evoj.GenePool;  
**import** net.sourceforge.evoj.mutation.FlipMutation;  
**import** net.sourceforge.evoj.mutation.SimpleBitMutation;  
**public class** MainProfiling {  
  
 **public static void** main(String[] args){  
 **int**[] logs = {10, 9, 12, 12, 14, 12, 12, 9, 9, 11, 10, 12, 12, 9, 9, 12, 12, 12, 12, 12, 12, 12, 12};*//lengths of available logs* **int**[] details = {3, 3, 4, 5, 3, 7, 7, 5, 4, 4, 4, 8, 8, 5, 5, 5, 5, 3, 3, 8, 8, 8, 8, 9, 5, 4, 5, 5, 6};*// lengths of required details* LumberMill lm = **new** LumberMill(logs, details);  
GenePool gp1 = **new** GenePool(500, 0, lm, **null**, **null**, lm, lm);  
 GenePool gp2 = **new** GenePool(500, 0, lm, **null**, **null**, **new** SimpleBitMutation(0.3, 0.5), lm);GenePool gp3 = **new** GenePool(500, 0, lm, **null**, **null**, **new** FlipMutation(0.1, lm), lm);  
 gp3.setEliteCount(250);  
  
 gp1.iterate(1000);  
 }  
}

**Результат виконання завдання**

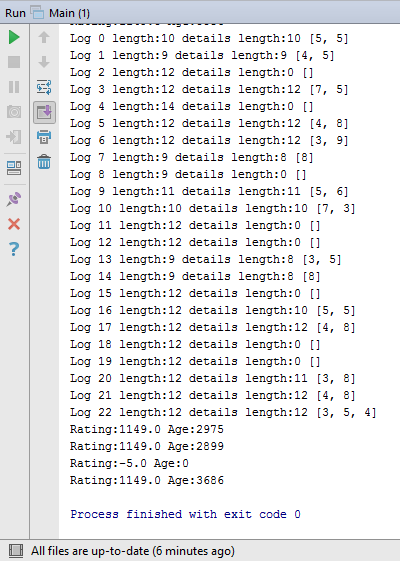


Рис. 9. Результат розв’язку задачі при

початкових умовах

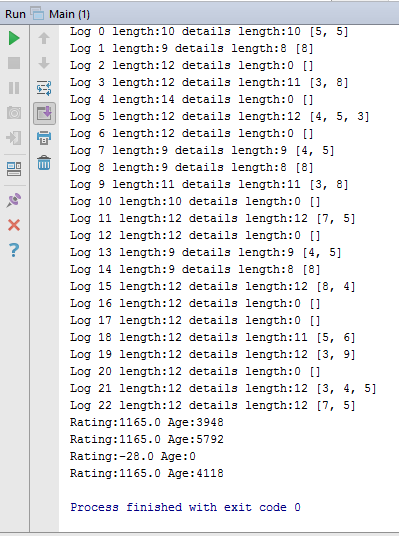


Рис. 10. Результат розв’язку задачі після зміни параметрів

**Приклад 3**

Використовуючи засоби фреймворка Evoj реалізувати програму для визначення значень коефіцієнтів площини.

**Фрагменти коду програми**

***Main.java***

**package** net.sourceforge.evoj.examples.surfaceapprox;  
  
**import** net.sourceforge.evoj.GenePool;  
**import** net.sourceforge.evoj.mutation.SimpleBitMutation;  
**import** net.sourceforge.evoj.params.ParameterMapping;  
**import** net.sourceforge.evoj.params.ParameterType;  
**import** java.util.ArrayList;  
**import** java.util.List;  
**public class** Main {  
  
 **public static void** main(String[] args) {  
 List<Point> pset = **new** ArrayList<Point>();  
 pset.add(**new** Point(1, 2, -4.481));  
 pset.add(**new** Point(0.6, 5.69, 35.5));  
 pset.add(**new** Point(4, 3.56, 44.75));  
 pset.add(**new** Point(2.92, 8, 127.06));  
 pset.add(**new** Point(4.94, 7, 118.35));  
 pset.add(**new** Point(2, 8, 83));  
 pset.add(**new** Point(2, 1.93, 9.22));  
 pset.add(**new** Point(5, 2, 74.12));  
 pset.add(**new** Point(4, 6, 76.15));  
 pset.add(**new** Point(3, 4.56, 33.38));  
  
 ParameterMapping pm = **new** ParameterMapping();  
 pm.addParam(**"a"**, ParameterType.***DOUBLE***);  
 pm.addParam(**"b"**, ParameterType.***DOUBLE***);  
 pm.addParam(**"c"**, ParameterType.***DOUBLE***);  
 pm.addParam(**"d"**, ParameterType.***DOUBLE***);  
 pm.addParam(**"e"**, ParameterType.***DOUBLE***);  
 pm.addParam(**"f"**, ParameterType.***DOUBLE***);  
  
 Point[] points = **new** Point[pset.size()];  
  
 Surface surf = **new** Surface(pset.toArray(points), pm);  
  
 GenePool gp = **new** GenePool(300, 0, surf, **null**, **null**, **new** SimpleBitMutation(0.1, 0.7), surf);  
 gp.setEliteCount(50);  
  
 System.***out***.println(**"Before"**);  
 System.***out***.println(pm.asString(gp.getChampion()));  
 System.***out***.println(**"Rating="** + gp.getChampion().getRating());  
 System.***out***.println(**"Age="** + gp.getChampion().getAge());  
 gp.iterate(1000);  
 System.***out***.println(**"After"**);  
 System.***out***.println(pm.asString(gp.getChampion()));  
 System.***out***.println(**"Rating="** + gp.getChampion().getRating());  
 System.***out***.println(**"Age="** + gp.getChampion().getAge());  
 }  
}

**Результат виконання завдання**

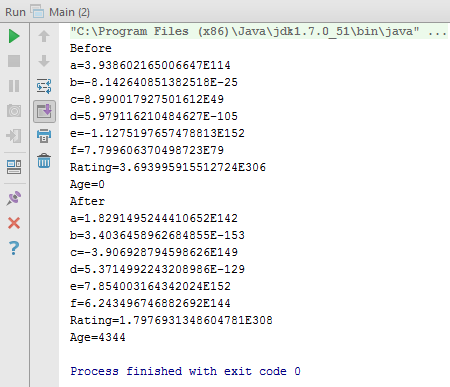


Рис. 11. Результат розв’язку задачі при початкових умовах

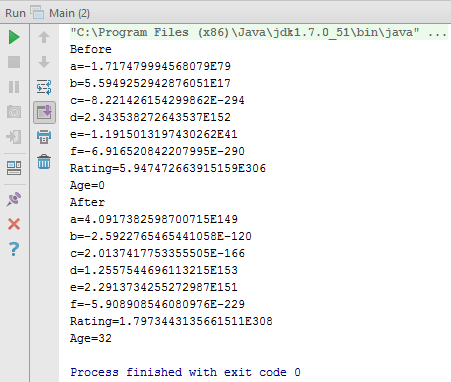


Рис. 12. Результат розв’язку задачі при зміні параметрів

**Висновок:** виконуючи розрахунково-графічну роботу я ознайомився із теоретичними відомостями про фреймворк EvoJ для мови програмування Java. Даний фреймворк надає зручний спосіб для розв’язання задач, використовуючи генетичні алгоритми. Використовуючи середовище програмування Intellij IDEA я встановив усі бібліотеки фреймворка для майбутнього проекту та запустив на виконання ряд стандартних прикладів, які були запропоновані розробниками фреймворка.

Ознайомлення із засобом EvoJ показав, що розв'язувати задачі, використовуючи генетичні алгоритми можна просто і не прописуючи багато коду. Для отримання результату достатньо оформити у потрібному форматі вхідні дані та передати їх у відповідні методи класів фреймворка. Якщо користувачу необхідно певним чином модифікувати деякі алгоритми він може це зробити, здійснивши операцію @override потрібного методу і використати його для рішення задачі.

**Список використаної літератури**

1. EvoJ [Електронний ресурс] – Режим доступу: <http://evoj-frmw.appspot.com/index.html>
2. IntelliJ IDEA 2017.1 Help: Meet IntelliJ IDEA [Електронний ресурс] – Режим доступу: <https://www.jetbrains.com/help/idea/2017.1/meet-intellij-idea.html>