Міністерство освіти і науки України

національний університет “Львівська політехніка”

****

Звіт

до розрахунково графічної роботи

На тему:

«jMetal фреймворк»

З курсу:

«Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні»

Виконав:

ст. гр. СПКм-12

Баландюх О.А.

Перевірив:

Кривий Р.З

Львів 2016

**Мета роботи:**

Ознайомитись зі можливостями jMetal. Навчитись використовувати цей програмний продукт для вирішення певних задач.

**Огляд програмного продукту jMetal**

jMetal використовується для метаеврістічних алгоритмів в Java, це об'єктно-орієнтоване середовище на основі Java спрямоване на багатоцільовоу оптимізацію за допомогою метаеврістики. jMetal надає багатий набір класів які можуть бути використані в якості будівельних блоків багатоцільових методів; Таким чином, користуючись повторним використанням коду, jMetal надає можливість на одних й тих самих базових компонентах створювати алгоритми, такі як реалізація генетичних операторів і оцінок щільності, тим самим сприяючи не лише розвитку нових багатоцільовий техніки, а й проводити різні види експериментів. Наявність ряду класичних алгоритмів та алгоритмів, що використовують доволі “творчий” підхід до вирішення задачі надають новачкам не тільки можливість вивчити основні принципи багатоцільової оптимізації в метаеврістиці, а й дозволяють, без значних зусиль, застосувати їх для вирішення реальних проблем. Проект jMetal постійно розвивається. Так як він розробляється окремою групою розробників, а не компанією з виготовлення програмного забезпечення, нові версії випускаються тоді, коли розробники вважають, що потрібно додати новий функцій, які буде потрібним і актуальним для проведення дослідницької діяльності.

**Переваги jMetal**

До переваг фреймворку jMetal можна віднести наступне:

* Простота і зручність у використанні. Це ключові цілі: якщо вони не будуть виконані, мало хто буде використовувати програмне забезпечення. Класи, що надаються jMetal дотримується принципу, що кожен компонент повинен робити тільки одну річ, і робити це добре. Таким чином, базові класи (SolutionSet, Solution, Variable і т.д.) і їх операції інтуїтивно зрозумілі і, як наслідок, легко зрозуміти і використовувати. Крім того, каркас включає в себе реалізацію багатьох метаеврістичних класів, які можуть бути використані в якості шаблонів для розробки нових методів.
* Гнучкість. Це спільна мета. З одного боку, програмне забезпечення повинно включати простий механізм для виконання алгоритмів при різних налаштувань параметрів, включаючи специфічні параметри, а також ті параметри, які тісно пов'язані з проблемою, що потрібно вирішити. З іншого боку, такі питання, як вибір реального або двійкового представлення коду, відповідно до обраного типу представлення конкретні оператори для використання, повинні вимагати мінімальних змін в програмній реалізації.
* Переносимість. Фреймворк і алгоритми, розроблені з його використанням повинні виконуватись на машинах з різною архітектурою і / або бути готовими до запуску під різними операційними системами. Використання Java як мови програмування дозволяє з легкістю виконати це завдання; Крім того, дане програмне забезпечення не повинне бути повторно зкомпільоване для роботи в іншому середовищі.
* Можливість розширення. Нові алгоритми, оператори і вирішення проблем повинні бути легко додані в фреймворк. Ця мета досягається за рахунок використання деяких механізмів Java, таких як успадкування і пізнє зв'язування. Наприклад, всі MOPи успадковуються від класу Problem, тому нова проблема може бути створена написанням методу що зазначений в цьому класі; після того, як клас, який визначає нову проблему компілюється, більше нічого не повинно бути зроблено: механізм пізнього зв’язування дозволяє завантажувати код MOPа тільки тоді, коли цього потребує той чи інший алгоритм. Таким чином, jMetal дозволяє відокремити опис реалізації алгоритму від конкретної реалізації алгоритму.

**Інсталяція jMetal та виконання тестового алгоритму**

Фреймворк jMetal написано на Java, це значить, що він не вимагає будь-якого додаткового програмного забезпечення. Все що необхідно, це мати інстальованим Java JDK 1.5 або більш пізню версію. Вихідний код поставляється в пакеті tar.gz, який можна завантажити з [jmetal.sourceforge.net](http://jmetal.sourceforge.net).

Нижче буде представлено спосіб інсталяції jMetal використовуючи Eclipse IDE.

1. Розпакування вихідного коду фреймворка

Для подальшого використання фреймворку, нам необхідно видобути з архіву jmetal4.5.2.src.tar.gz весь його вміст в папку з назвою JMETALHOME.

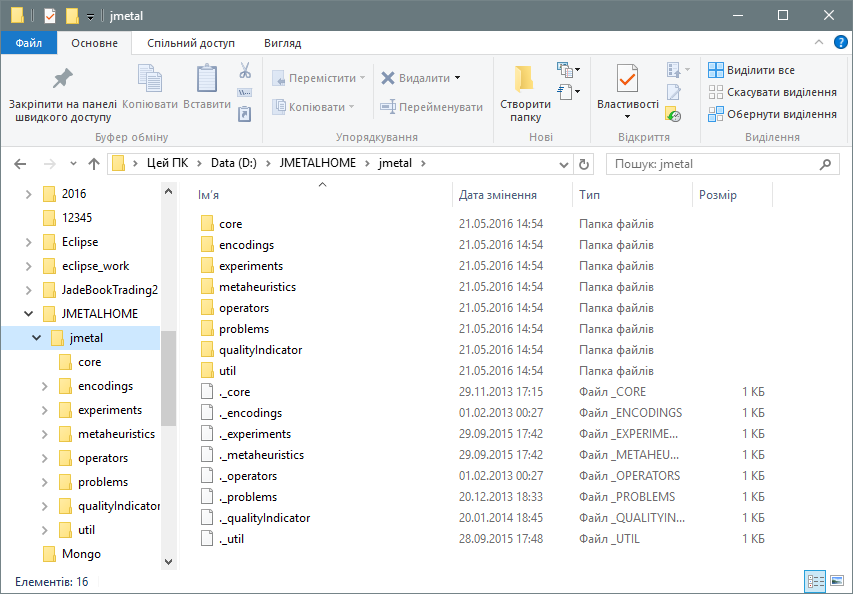


Рис. 1. Структура файлів вихідного коду jMetal

1. Підключення jMetal в Eclipse IDE.

Створення проекту:

* Виберіть *File* → *Create* → *New Java Project*.
* Введіть ім'я проекту (наприклад, jMetal) і натисніть на кнопку *Next*.
* Виберіть *Link additional source* і *Browse* щоб вказати розташування папки JMETALHOME (Рис. 2-3).
* Натисніть кнопку *Finish*.

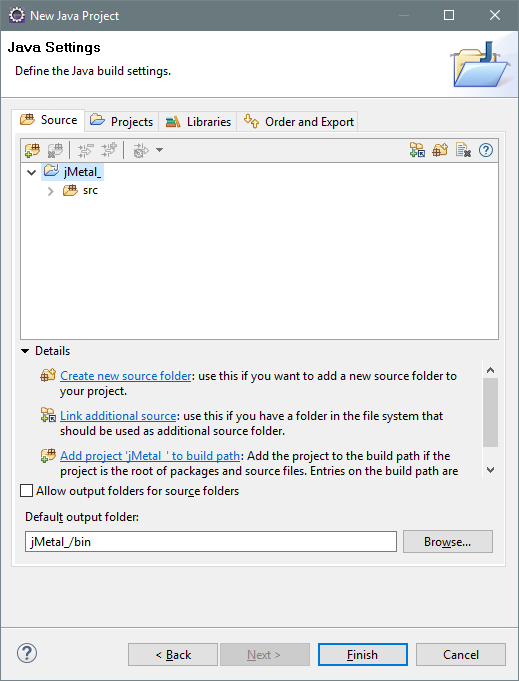


Рис.2. Вікно *New Java Project*

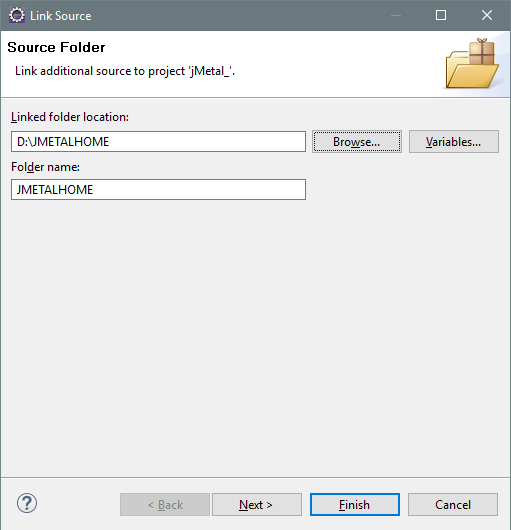


Рис.3. Вікно *Link Source*

1. Налагодження та виконання тестового алгоритму NSGA-II.

Ми використовуємо NSGA-II (Non-dominated Sorting Genetic Algorithm-II) в якості прикладу. Щоб налаштувати алгоритм, відкрийте файл NSGAII Main.java

вибравши його з пакета jmetal.metaheuristics.nsgaII (Рис.4). Після цього ви можете внести будь які зміни, чи удосконалення, в залежності від поставленої задачі.

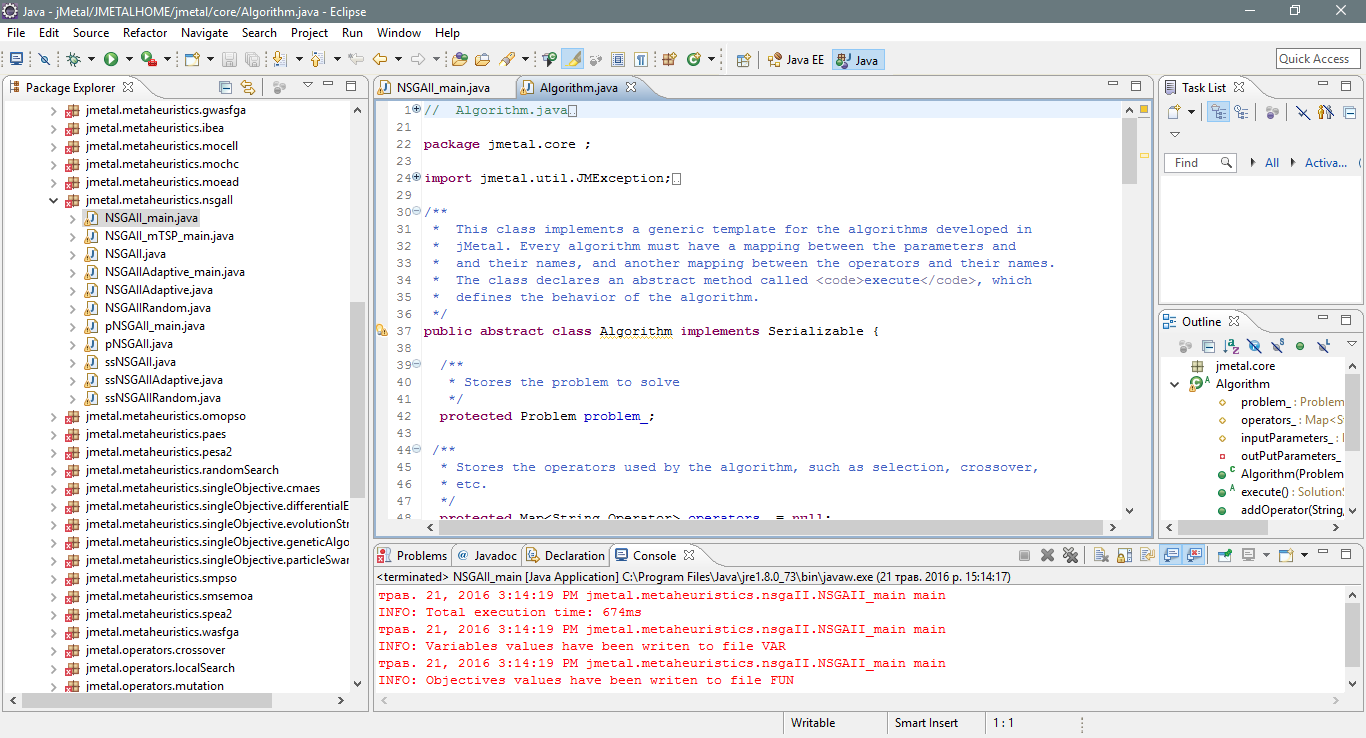


Рис.4. Представлення реалізації алгоритму NSGA-II в jMetal

Для запуску алгоритму, клацніть правою кнопкою миші на NSGA-II Main.java в дереві проекту. Виберіть *Run As* → *Java Application*. В результаті ви отримаєте два файли один з яких містить Парето-оптимальні рішення, а інший - фронт Парето, знайдені алгоритмом. За замовчуванням, ці файли називаються VAR і FUN, відповідно. Вони розташовані в каталозі *workspace* (наприклад, C:\Users\Oleg\workspace\jMetal). Вигляд *workspace* з файлами – результатом виконання програми зображенні на Рис.4. Код програмної реалізації NSGA-II знаходиться в додатку А.

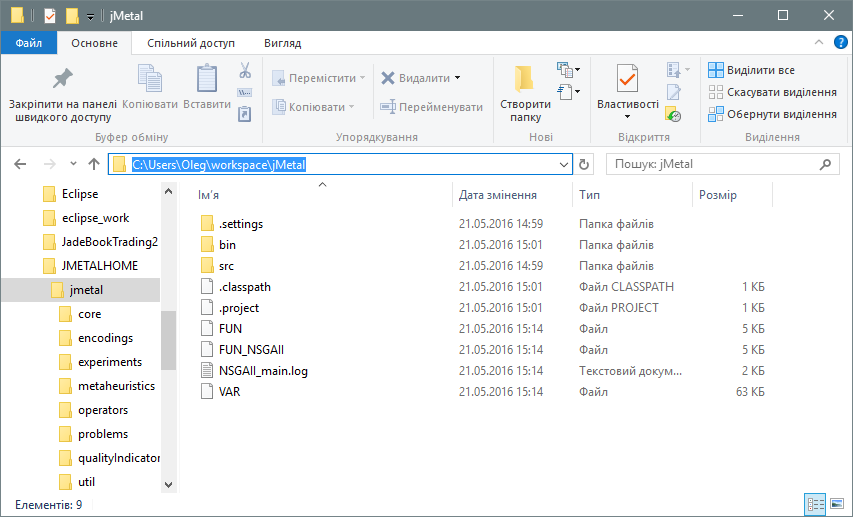


Рис.5. Вигляд *workspace* з файлами – результатом виконання алгоритму

**Висновок:** під час виконання розрахунково-графічної роботи, я ознайомилась з програмним продуктом jMetal, якй є надзвичайно зручним та ефективним інструментом для розв’язання різного роду задач, використовуючи евристичні та генетичні алгоритми. Фреймворк містить велику кількість вже реалізованих алгоритмів, а також набір задач з стандартними налаштуваннями.

Перевага використання цієї програми перш за все в тому, що для розв’язання задач немає необхідності реалізовувати певний алгоритм самостійно, а на томість можна використати чи модифікувати уже реалізований. Для цього фреймворк надає відкритий код своїх реалізацій алгоритмів.

Додаток А.

// Algorithm.java

//

// Authors:

// Antonio J. Nebro <antonio@lcc.uma.es>

// Juan J. Durillo <durillo@lcc.uma.es>

////

// Copyright (c) 2011 Antonio J. Nebro, Juan J. Durillo

//

// This program is free software: you can redistribute it and/or modify

// it under the terms of the GNU Lesser General Public License as published by

// the Free Software Foundation, either version 3 of the License, or

// (at your option) any later version.

//

// This program is distributed in the hope that it will be useful,

// but WITHOUT ANY WARRANTY; without even the implied warranty of

// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

// GNU Lesser General Public License for more details.

//

// You should have received a copy of the GNU Lesser General Public License

// along with this program. If not, see <http://www.gnu.org/licenses/>.

package jmetal.core ;

import jmetal.util.JMException;

import java.io.Serializable;

import java.util.HashMap;

import java.util.Map;

/\*\*

\* This class implements a generic template for the algorithms developed in

\* jMetal. Every algorithm must have a mapping between the parameters and

\* and their names, and another mapping between the operators and their names.

\* The class declares an abstract method called <code>execute</code>, which

\* defines the behavior of the algorithm.

\*/

public abstract class Algorithm implements Serializable {

/\*\*

\* Stores the problem to solve

\*/

protected Problem problem\_;

/\*\*

\* Stores the operators used by the algorithm, such as selection, crossover,

\* etc.

\*/

protected Map<String,Operator> operators\_ = null;

/\*\*

\* Stores algorithm specific parameters. For example, in NSGA-II these

\* parameters include the population size and the maximum number of function

\* evaluations.

\*/

protected Map<String,Object> inputParameters\_ = null;

/\*\*

\* Stores output parameters, which are retrieved by Main object to

\* obtain information from an algorithm.

\*/

private Map<String,Object> outPutParameters\_ = null;

/\*\*

\* Constructor

\* @param problem The problem to be solved

\*/

public Algorithm(Problem problem) {

problem\_ = problem ;

}

/\*\*

\* Launches the execution of an specific algorithm.

\* @return a <code>SolutionSet</code> that is a set of non dominated solutions

\* as a result of the algorithm execution

\*/

public abstract SolutionSet execute() throws JMException, ClassNotFoundException ;

/\*\*

\* Offers facilities for add new operators for the algorithm. To use an

\* operator, an algorithm has to obtain it through the

\* <code>getOperator</code> method.

\* @param name The operator name

\* @param operator The operator

\*/

public void addOperator(String name, Operator operator){

if (operators\_ == null) {

operators\_ = new HashMap<String,Operator>();

}

operators\_.put(name,operator);

} // addOperator

/\*\*

\* Gets an operator through his name. If the operator doesn't exist or the name

\* is wrong this method returns null. The client of this method have to check

\* the result of the method.

\* @param name The operator name

\* @return The operator if exists, null in another case.

\*/

public Operator getOperator(String name){

return operators\_.get(name);

} // getOperator

/\*\*

\* Sets an input parameter to an algorithm. Typically,

\* the method is invoked by a Main object before running an algorithm.

\* The parameters have to been inserted using their name to access them through

\* the <code>getInputParameter</code> method.

\* @param name The parameter name

\* @param object Object that represent a parameter for the

\* algorithm.

\*/

public void setInputParameter(String name, Object object){

if (inputParameters\_ == null) {

inputParameters\_ = new HashMap<String,Object>();

}

inputParameters\_.put(name,object);

} // setInputParameter

/\*\*

\* Gets an input parameter through its name. Typically,

\* the method is invoked by an object representing an algorithm

\* @param name The parameter name

\* @return Object representing the parameter or null if the parameter doesn't

\* exist or the name is wrong

\*/

public Object getInputParameter(String name){

return inputParameters\_.get(name);

} // getInputParameter

/\*\*

\* Sets an output parameter that can be obtained by invoking

\* <code>getOutputParame</code>. Typically this algorithm is invoked by an

\* algorithm at the end of the <code>execute</code> to retrieve output

\* information

\* @param name The output parameter name

\* @param object Object representing the output parameter

\*/

public void setOutputParameter(String name, Object object) {

if (outPutParameters\_ == null) {

outPutParameters\_ = new HashMap<String,Object>();

}

outPutParameters\_.put(name,object);

} // setOutputParameter

/\*\*

\* Gets an output parameter through its name. Typically,

\* the method is invoked by a Main object after the execution of an algorithm.

\* @param name The output parameter name

\* @return Object representing the output parameter, or null if the parameter

\* doesn't exist or the name is wrong.

\*/

public Object getOutputParameter(String name) {

if (outPutParameters\_ != null)

return outPutParameters\_.get(name);

else

return null ;

} // getOutputParameter

/\*\*

\* Returns the problem to solve

\* @return Problem The problem to solve

\*/

public Problem getProblem() {

return problem\_ ;

}

} // Algorithm