*дата :*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САП** | | Тема | оцінка | підпис |
| КНC-13 | РГР | **Фреймворк jMetal варіант 17 (3)** |  |  |
| Селезень А.Ю. | |
| Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні | | Викладач: | |
| Кривий Р. З. | |

**1. Загальні відомості про фреймворк jMetal**

jMetal - об'єктно-орієнтований фреймворк, написаний на мові програмування Java, який створено для мультиоб’єктної багатокритеріальної оптимізації, яка базується на використанні і впровадженні метаеврістичних методів.

Проект jMetal був розпочатий в 2006 році через необхідність мати простий у використанні, гнучкий, модульний та портативної багатоцільовий засіб для основних методів оптимізації.

jMetal 5 є першою великою реорганізованою версією jMetal з моменту його створення. Архітектура була перероблена з нуля, щоб забезпечити простішу конструкцію, при цьому зберігегти ту ж функціональність. Поточна версія фреймворку: jMetal 5.1.

Наразі jMetal - проект Maven, який розміщується на GitHub, де зацікавлені особи можуть отримати доступ до поточного стану проекту і можуть вільно зробити свій внесок у проект. Всі попередні версії jMetal все ще доступні на SourceForge

Об'єктно-орієнтована архітектура jMetal і включені функції дають наступні можливості:

* експериментувати з різними техніками оптимізації,
* розробляти власні алгоритми,
* вирішувати власні проблеми оптимізації,
* інтегрувати jMetal в інші інструмени тощо

Через дев'ять років після того, як світ побачив перший реліз jMetal, вирішено було провести глибоку реструктуризацію програмного забезпечення, цим самим було втілено наступні ідеї:

* Перебудова архітектури, що забезпечує більш просту конструкцію, зберігаючи ту ж функціональність.
* Використання Maven як інструменту для розробки, тестування, упакування і розгортання.
* Просування коду повторного використання шляхом надання шаблонів алгоритму
* Поліпшення якості коду
* Застосування модульного тестування
* Більш ефективніше, порівняно з попередником, використання можливостей Java
* Шаблони проектування (Singleton, будівельник, завод, спостерігач)
* Застосування принципів чистого коду
* Підтримка паралельності

Наразі відомо про розробку версій фреймворку jMetalPy, jMetalCpp та jMetalWeb для використання в мовах Python, С++ та веб-розробці.

**Загальна специфікація можливостей (англ.мова):**

* Multi-objective algoritms: NSGA-II, SPEA2, PAES, PESA-II, OMOPSO, MOCell, AbYSS, MOEA/D, GDE3, IBEA, SMPSO, SMPSOhv, SMS-EMOA, MOEA/D-STM, MOCHC, MOMBI, MOMBI-II, NSGA-III, WASF-GA, GWASF-GA
* Single-objective algoritms: genetic algorithm (variants: generational, steady-state), evolution strategy (variants: elitist or mu+lambda, non-elitist or mu, lambda), DE, CMA-ES, PSO (Stantard 2007, Standard 2011), Coral reef optimization.
* Algorithms that can be executed in parallel: NSGA-II, SMPSO, GDE3, SPEA2, PESA-II
* Included problems:
  + Problem families: ZDT, DTLZ, WFG, CEC2009, LZ09, GLT, MOP
  + Classical problems: Kursawe, Fonseca, Schaffer, Viennet2, Viennet3
  + Constrained problems: Srinivas, Tanaka, Osyczka2, Constr\_Ex, Golinski, Water, Viennet4
  + Combinatorial problems: multi-objective TSP
  + Academic problems: OneMax, OneZeroMax
* Quality indicators: hypervolume, spread, generational distance, inverted generational distance, inverted generational distance plus, additive epsilon.
* Variable representations: binary, real, integer, permutation, mixed encoding (real+binary, int+real).

**2. Підключення jMetal до проекту**

Для Maven проектів на основі Maven фреймвор можна додати як maven-залежніть. В інших випадках потрібно завантажити вихідний код з офіційного репозиторія (https://github.com/jMetal/jMetal.).

jMetal 5.2 складається з наступних підмодулів:

• **jmetal-core** - класи архітектури ядра

• **Jmetal-algorithm** - реалізації метаеврістичних алгоритмів;

• **jmetal-problem** - реалізації деяких проблем оптимізації (цільові функції);

• **jmetal-exec** - виконувані програми для настройки і запуску алгоритмів, а також деякі утиліти.

Підключення модуль jmetal-core: додаємо до проекту наступний код, що виражатиме залежність:

****

Рис. 1 Підключення модуля jmetal-core

**3. Створення проекту та запуск програми**

Для запуску алгоритму в jMetal у є два варіанти:

* за допомогою IDE
* за допомогою командного рядка

Надалі буде розглянуто запуск алгоритму через IDE, а саме через IDE Eclipse (версія Luna).

Створюємо Maven –проект і під’єднуємо залежності (рис. 2-3).

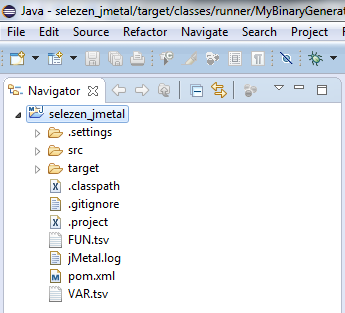


Рис. 2 Створений проект selezen\_jmetal



Рис. 3 Під’єднання залежностей

Для вирішення власної задачі потрібно реалізувати власну функцію оптимізації та метод для запуску. Кожен окремий метод оптимізації, з конкретними параметрами, реалізуються у вигляді окремого класу Runner. Власну задачу оптимізації можна реалізувати на прикладі вже існуючих.

**Код файлу BinaryGenerationalGeneticAlgorithmRunner**

public class BinaryGenerationalGeneticAlgorithmRunner {

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

Algorithm<BinarySolution> algorithm;

/\* Задача оптимізації \*/

BinaryProblem problem = new OneMax(7);

/\* Параметри генетичного алгоритму \*/

CrossoverOperator<BinarySolution> crossoverOperator = new SinglePointCrossover(0.9);

MutationOperator<BinarySolution> mutationOperator = new BitFlipMutation(1.0 / problem.getNumberOfBits(0));

SelectionOperator<List<BinarySolution>, BinarySolution> selectionOperator = new BinaryTournamentSelection<BinarySolution>();

algorithm = new GeneticAlgorithmBuilder<BinarySolution>(problem, crossoverOperator, mutationOperator)

.setPopulationSize(6).setMaxEvaluations(50).setSelectionOperator(selectionOperator).build();

/\* Запуск алгоритму \*/

AlgorithmRunner algorithmRunner = new AlgorithmRunner.Executor(algorithm).execute();

BinarySolution solution = algorithm.getResult();

List<BinarySolution> population = new ArrayList<BinarySolution>(1);

population.add(solution);

long computingTime = algorithmRunner.getComputingTime();

/\* Запис результатів до файлів \*/

new SolutionListOutput(population).setSeparator("\n")

.setVarFileOutputContext(new DefaultFileOutputContext("D:/VAR.tsv"))

.setFunFileOutputContext(new DefaultFileOutputContext("D:/FUN.tsv")).print();

/\* Вивід інф-ї про виконання алгоритму \*/

JMetalLogger.logger.info("Total execution time: " + computingTime + "ms");

JMetalLogger.logger.info("Objectives values have been written to file FUN.tsv");

JMetalLogger.logger.info("Variables values have been written to file VAR.tsv");

}}







Рис. 4 Результати запуску програми

Обрана проблема OneMax() має на меті збільшення одиниць (VAR.tsv). В файлі FUN.tsv записано значення функції. Оскільки оптимізація зводиться до пошуку мінімуму, значення функції (кількість одиниць) виводиться з від'ємним знаком.

Реалізуємо власну проблему (MyOneMax()). Значенням функції в цьому випадку буде десяткове представлення числа, поділене на кількість одиниць в числі, якщо воно не рівне 0. В той же час нові одиниці в популяції будуть як і покращувати значення функції до певного моменту, так і погіршувати її.

**Код файлу MyOneMax :**

public class MyOneMax extends AbstractBinaryProblem{

private int bits ;

/\*\* Constructor \*/

public MyOneMax() {

this(256);

}

/\*\* Constructor \*/

public MyOneMax(Integer numberOfBits) {

setNumberOfVariables(1);

setNumberOfObjectives(1);

setName("MyOneMax");

bits = numberOfBits ;

}

@Override

protected int getBitsPerVariable(int index) {

if (index != 0) {

throw new JMetalException("Problem MyOneMax has only a variable. Index = " + index) ;

}

return bits ;

}

@Override

public BinarySolution createSolution() {

return new MyBinarySolution(this) ;

}

/\*\* Evaluate() method \*/

public void evaluate(BinarySolution solution) {

int counterOnes;

counterOnes = 0;

BitSet bitset = solution.getVariableValue(0) ;

for (int i = 0; i < bitset.length(); i++) {

if (bitset.get(i)) {

counterOnes++;

}

}

solution.setObjective(0, -1.0 \* counterOnes);

}

}

**Код файлу** **MyBinarySolution**

public class MyBinarySolution extends AbstractGenericSolution<BinarySet, BinaryProblem> implements BinarySolution {

private static final long serialVersionUID = 1L;

/\*\* Constructor \*/

public MyBinarySolution(BinaryProblem problem) {

super(problem) ;

initializeBinaryVariables();

initializeObjectiveValues();

}

/\*\* Copy constructor \*/

public MyBinarySolution(MyBinarySolution solution) {

super(solution.problem);

for (int i = 0; i < problem.getNumberOfVariables(); i++) {

setVariableValue(i, (BinarySet) solution.getVariableValue(i).clone());

}

for (int i = 0; i < problem.getNumberOfObjectives(); i++) {

setObjective(i, solution.getObjective(i)) ;

}

attributes = new HashMap<Object, Object>(solution.attributes) ;

}

private BinarySet createNewBitSet(int numberOfBits) {

BinarySet bitSet = new BinarySet(numberOfBits);

for (int i = 0; i < numberOfBits; i++) {

double rnd = randomGenerator.nextDouble();

if (rnd < 0.5) {

bitSet.set(i);

} else {

bitSet.clear(i);

}

}

return bitSet;

}

public int getNumberOfBits(int index) {

return getVariableValue(index).getBinarySetLength();

}

public MyBinarySolution copy() {

return new MyBinarySolution(this);

}

public int getTotalNumberOfBits() {

int sum = 0;

for (int i = 0; i < getNumberOfVariables(); i++) {

sum += getVariableValue(i).getBinarySetLength();

}

return sum;

}

public String getVariableValueString(int index) {

String result = "";

for (int i = 0; i < getVariableValue(index).getBinarySetLength(); i++) {

if (getVariableValue(index).get(i)) {

result += "1";

} else {

result += "0";

}

}

return result;

}

private void initializeBinaryVariables() {

for (int i = 0; i < problem.getNumberOfVariables(); i++) {

setVariableValue(i, createNewBitSet(problem.getNumberOfBits(i)));

}

}

/\* Значення функції \*/

public double getObjective(int i) {

int n = 0;

String str = "";

for (int j = 0; j < getVariableValue(i).getBinarySetLength(); j++) {

if (getVariableValue(i).get(j)) {

n++;

str+="1";

} else {

str+="0";

}

}

int val = Integer.parseInt(str, 2);

if (n == 0) n=1;

return -val/n;

}

}







Рис. 5 Пошук оптимуму функції (7 біт)







Рис. 6 Пошук оптимуму функції (15 біт)

**5. Висновок:**

Фреймворк jMetal активно розвивається і є актуальним продуктом, про що свідчить і розробка нових версій, і підтримка повередніх. Його недоліками є не надто детальна документація, що ускладнює оволодіння фреймворком початківцями. Також значним недоліком можна вважати громісткий і часто незрозумілий синтаксис, що ускладнює сприйняття і розуміння коду. Щоб реалізувати оптимізацію власної задачі, навіть найпростішої, потрібно як мінімум написати власні реалізації функції запуску (runner), описати клас задачі (problem) та сам клас рішення (solution). В цілому, цей фреймворк потребує доопрацювання і оптимізації, наразі через свої недоліки залишаючись не надто популярним рішенням на ринку.