МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра "Системи автоматизованого проектування"

Звіт

до розрахунково-графічної роботи

з курсу: «Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні»

на тему:

«jMetal framework»

Виконала:

студентка гр. КНМ-14

Ласка Галина

Перевірив:

асист. Кривий Р.З.

Львів - 2017

1. КОРОТКИЙ ОПИС JMETAL FRAMEWORK

jMetal являє собою об'єктно-орієнтований фреймворк для Java для багатокритеріальної оптимізації з використанням метаеврістичних методів.

Об'єктно-орієнтована архітектура фреймворку і включені функції дозволяють:

- експериментувати з різними техніками оптимізації,
- розробляти свої власні алгоритми,
- вирішувати свої проблеми оптимізації,
- інтегрувати jMetal в інші інструмени і т.д.

jMetal 5 є першою великою реорганізованою версією jMetal з моменту його створення. Архітектура була перероблена з нуля, щоб забезпечити більш просту конструкцію, зберігаючи ту ж функціональність. Поточна версія jMetal 5.1.

Проект jMetal був розпочатий в 2006 році через необхідність мати простий у використанні, гнучкий, модульний та портативної багатоцільовий засіб для основних методів оптимізації. З 2008 року він був розміщений в http://jmetal.sourceforge.net. З 2014 року розробка перебуває в https://github.com/jMetal/jMetal.

Через дев'ять років після першого випуску jMetal, був зроблений глибокий редизайн програмного забезпечення і втілено такі ідеї:

- Перебудова архітектури, що забезпечує більш просту конструкцію, зберігаючи ту ж функціональність.
- Використання Maven як інструмент для розробки, тестування, упакування і розгортання.
- Просування коду повторного використання шляхом надання шаблонів алгоритму
- Поліпшення якості коду:
- Застосування модульного тестування
- Більш ефективне використання можливостей Java (наприклад, генерики)
- Шаблони проектування (Singleton, будівельник, завод, спостерігач)
- Застосування принципів чистого коду
- підтримка паралельності
- Введені заходи, для отримання інформації про виконання алгоритму

Зараз також розробляються версії фреймворку jMetalPy, jMetalCpp та jMetalWeb для використаня в мовах Python, C++ та веб розробці.

2. ПІДКЛЮЧЕННЯ JMETAL ДО ПРОЕКТУ

Для Maven проектів, jMetal можна добавити як maven залежніть. В інших випадках можна просто завантажити вихідний код з офіційного репозиторія (https://github.com/jMetal/jMetal.).

jMetal 5.2 складається з таких підмодулів:

- jmetal-core - класи архітектури ядра плюс деякі комунальні послуги, в тому числі показники якості;
- Jmetal-algorithm - реалізації метаеврістичних алгоритмів;
- jmetal-problem - реалізації деяких проблем оптимізації (цільові функції);
- jmetal-exec - виконувані програми для настройки і запуску алгоритмів плюс деякі утиліти.

Якщо потрібно підключити модуль jmetal-core, достатньо додати до проекту залежність

Apache Maven

```
<dependency>
    <groupId>org.uma.jmetal</groupId>
    <artifactId>jmetal-core</artifactId>
    <version>5.1</version>
</dependency>
```

3. ЗАПУСК ПРОГРАМИ

Для запуску алгоритму в jMetal у є два варіанти: за допомогою IDE або за допомогою командного рядка. Тут буде розглянутий перший варіант: запуск алгоритму через IDE Eclipce.

Створимо проект Maven і підключимо необхідні залежності (рис. 1).

```
<dependencies>
        <dependency>
            <groupId>org.uma.jmetal</groupId>
            <artifactId>jmetal-core</artifactId>
            <version>5.2</version>
        </dependency>
        <dependency>
            <groupId>org.uma.jmetal</groupId>
            <artifactId>jmetal-exec</artifactId>
            <version>5.2</version>
        </dependency>
        <dependency>
            <groupId>org.uma.jmetal</groupId>
            <artifactId>jmetal-problem</artifactId>
            <version>5.2</version>
        </dependency>
        <dependency>
            <groupId>org.uma.jmetal</groupId>
            <artifactId>jmetal-algorithm</artifactId>
            <version>5.2</version>
        </dependency>
</dependencies>
```

Рис. 1. Залежності для jMetal

Для вирішення власної задачі, потрібно реалізовувати 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 уже існуючих.

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).setSelec
tionOperator(selectionOperator).build();

            /* Запускаемо алгоритм */
            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 результату у зовнішній файл */
            new SolutionListOutput(population).setSeparator("\n")
                        .setVarFileOutputContext(new
DefaultFileOutputContext("/home/laska/KNM2017/laska/laska_rhr/VAR.tsv"))
                        .setFunFileOutputContext(new
DefaultFileOutputContext("/home/laska/KNM2017/laska/laska_rhr/FUN.tsv")).pr
int();

            /* Вивід загальної інформації 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");
      }

}
```

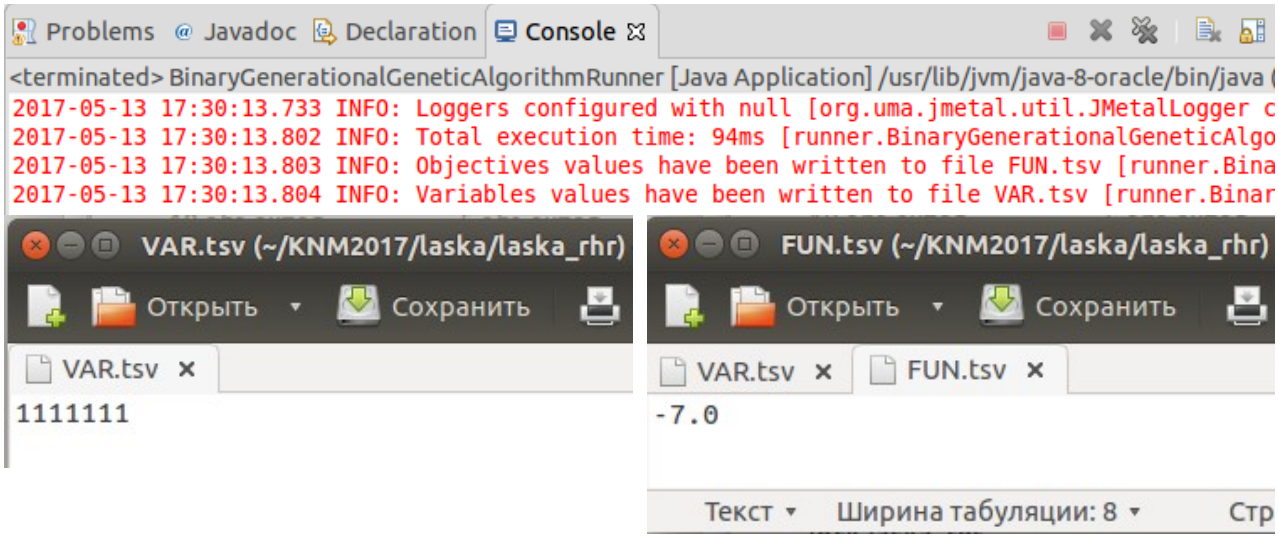

Рис. 2. Результати запуску

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

Реалізуємо власну проблему (MyOneMax()). Значенням функції буде не кількість одиниць, а десяткове представлення числа поділене на кількість одиниць в числі, якщо воно не рівне 0.

Та нові одиниці в популяції, будуть як і покращувати значення функції, до певного моменту, так і погіршувати одночасно.

MyOneMax (опис проблеми)

```
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);
   }
}
```

MyBinarySolution (опис цільової функції)

```
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;
 }

}
```

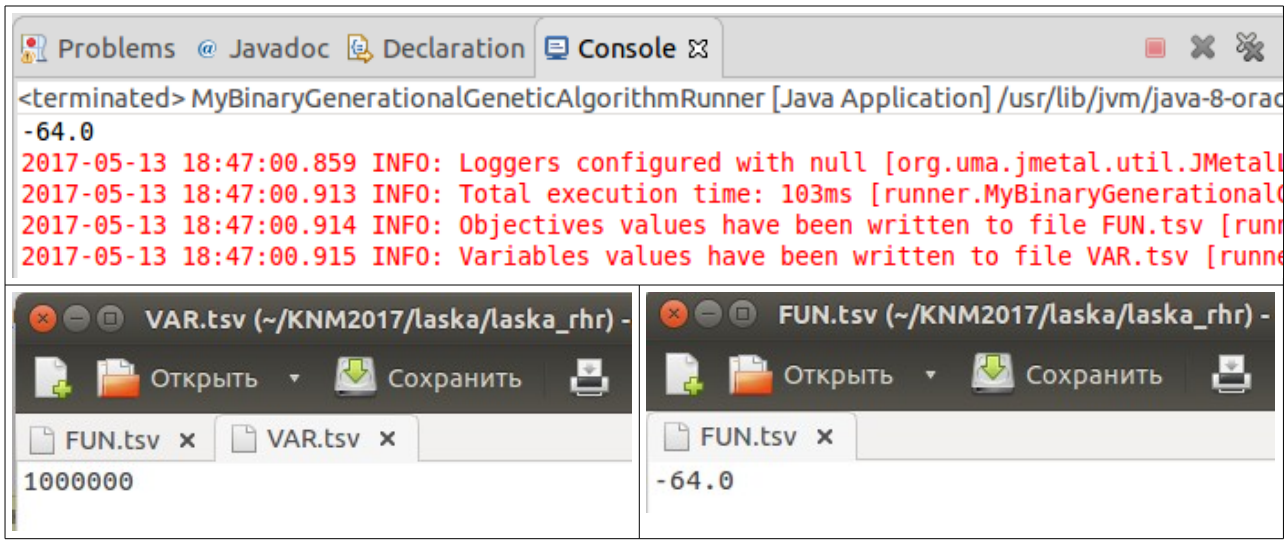

Рис. 3. Пошук оптимуму власної функції при 7 біт

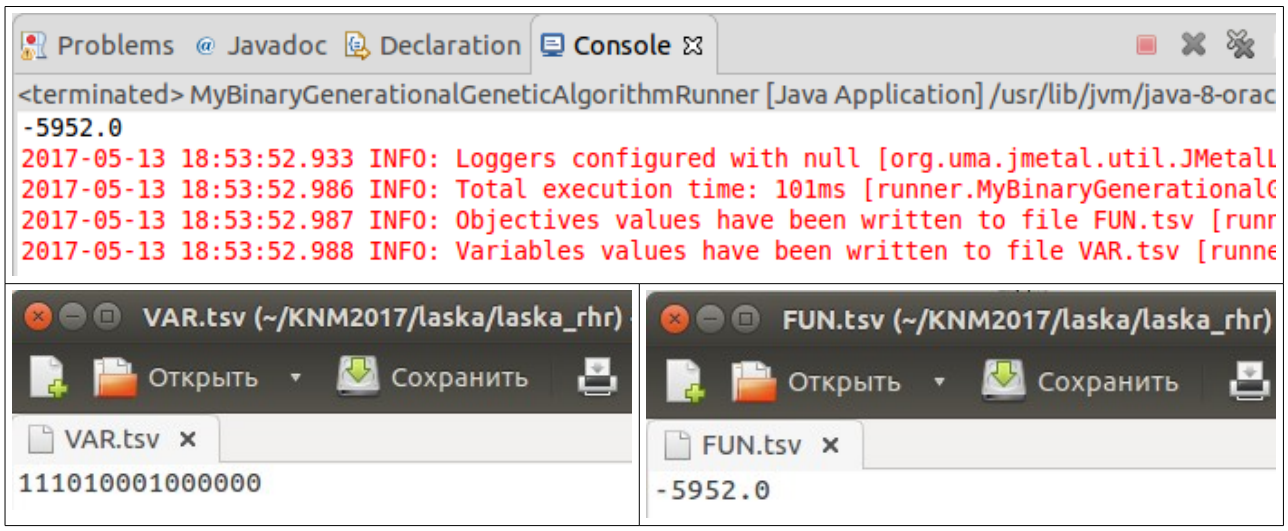

Рис. 4. Пошук оптимуму власної функції при 15 біт

ВИСНОВОК

Фреймворк jMetal активно розвивається і є потрібним продуктом, про що свідчить розробка нових версій. Проте документація з використання фреймоврку надто бідна, і не дозволяє оволодіти його основами початківцю. Громісткий синтаксис java тільки ускладнює сприйняття. 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).