ASSESSED

G52AIM Lab 5 – Multimeme Memetic Algorithms

This is the fifth assessed lab session and will account for 1/5th (including the report) of the module's total coursework mark (5% of the entire module).

1 Objectives

This lab exercise builds on the last week's lab my extending the memetic algorithm to a multimeme memetic algorithm. It is highly recommended **before** the lab to revisit last week's exercise, especially the implementation of the memetic algorithm.

1.1 Lab Exercise

- Implement a Multimeme Memetic Algorithm:
  - Adapting the local search operator,
  - Adapting the intensity of mutation, and
  - Using the simple inheritance method.

The implementations of the local search operators, mutation, crossover and replacement will be given to you in the form of a JAR file.

1.2 Report

- To be announced at the start of the lab.

2 Implementation [50 marks]

2.1 Tasks

You are given 2 Classes, `MultiMeme`, and `SimpleInheritanceMethod`, which you will need to complete to implement the Multimeme Memetic Algorithm.

Remember your implementation should perform a single iteration/generation of the respective heuristic method, hence no `while(terminationCriterionNotMet)` loop is required. Running of your solutions is handled by the `Lab_05_Runner` Class that is provided for you and the experimental parameters can be configured in `Lab05TestFrameConfig`.

Within `MutiMeme.java`, we have specified three methods which we ask that you implement and call from `runMainLoop()` to perform the respective operations. This has the advantage of making your code neater and easier to implement, and enables us to test your code more easily! These methods are as follows:

```
1. applyMutationForChildDependentOnMeme( int childIndex, int memeIndex );
2. applyLocalSearchForChildDependentOnMeme( int childIndex, int memeIndex );
```

3. `performMutationOfMemeplex( int solutionIndex );`

The configuration which you are asked to use is as follows:

- Population size = 16
- Parent selection = tournament selection with tournament size = 3 for both parents.
- Memeplex:
  - Meme in index 0 is for the intensity of mutation setting
    - Intensity of mutation ∈ $\{0,1,2,3,4\}$.
  - Meme index 1 is for the local search operator
    - Local search operator ∈ $\{DBHC\_OI, DBHC\_IE, SDHC\_OI, SDHC\_IE\}$.
- Crossover operator = 1PTX.
- Replacement scheme = trans-generational replacement with elitism.

2.1.1 Multi-meme Memetic Algorithms

Multi-meme Memetic Algorithms (MMA's) were covered in the lecture "Evolutionary Algorithms II". Below is the pseudocode for a Multimeme Memetic Algorithm embedding memeplexes for deciding which local search operator to apply and the intensity of mutation as described in section 2.1.2.

```
1  | INPUT: PopulationSize, MaxGenerations, InnovationRate
2  | generateInitialPopulation();
3  | FOR 0 -> MaxGenerations
4  |     FOR 0 -> PopulationSize / 2
5  |         select parents using tournament selection
6  |         apply crossover to generate offspring
7  |         inherit memeplex using simple inheritance method
8  |         mutate the memes within each memeplex of each child with
           ⏎       probability dependent on the innovation rate

9  |         apply mutation to offspring with intensity of mutation set for
           ⏎       each solution dependent on its meme option
10 |         apply local search to offspring with choice of operator
           ⏎        dependent on each solution's meme option
11 |     ENDFOR
12 |     do population replacement
13 | ENDFOR
14 | return s_best;
```

Code 1 - Pseudocode for the Multimeme Memetic Algorithm.

Mutation of each meme within each memeplex should be performed separately such that it is possible to mutate the meme in meme index 0 but not the meme in meme index 1 even though they are from the same memeplex.

Within the MMA that you are asked to implement, there will be two memes in each memeplex. The first meme (meme index 0) will represent the intensity of mutation setting and has 5 possible options which are mapped to intensity of mutation settings as $iom \leftarrow option$. That is, the intensity of mutation setting equals the meme option. The second meme (meme index 1) will represent the local search operator to use. The option values should be mapped as:

- `map | 0 -> DBHC accepting only improving moves.`
- `    | 1 -> DBHC accepting non-worsening moves.`

- `| 2 -> SDHC accepting only improving moves.`
- `| 3 -> SDHC accepting non-worsening moves.`

2.1.2 Intensity of Mutation

The mutation operator will be the same as that used in last week's lab. Within bit mutation, the MUTATION_RATE was set as $1/chromosome\_length$. In the implementation provided, the MUTATION_RATE is set as $x/chromsome\_length$ where x is the integer intensity of mutation setting. There is a method `setMutationRate(int intensityOfMutation)` which should be called on the mutation heuristic to set the aforementioned parameter setting.

2.1.3 Simple Inheritance Method

Within the simple inheritance method (SIM), all memes (the memeplex) of each offspring are inherited from the best parent where the best parent is the parent with the best objective value. In the event of a tie, the parent from which the memes are inherited is chosen randomly. Below is the **pseudocode** for SIM. Information relating to the actual methods can be found both in the API document and in the sections "Framework Background" and "Implementation Hints".

```
1  | INPUT: parent1, parent2, child1, child2
2  | IF f(parent1) == f(parent2) THEN
3  |     inherit = random ∈ {parent1, parent2}
4  |     child1.memeplex <- inherit.memeplex
5  |     child2.memeplex <- inherit.memeplex
6  | ELSEIF f(parent1) < f(parent2) THEN
7  |     child1.memeplex <- parent1.memeplex
8  |     child2.memeplex <- parent1.memeplex
9  | ELSE
10 |     child1.memeplex <- parent2.memeplex
11 |     child2.memeplex <- parent2.memeplex
12 | ENDIF
13 | return;
```

Code 2 - Pseudocode for Simple Inheritance Method.

2.1.4 Importing the required files

Within the Lab 05 source files archive is one folder named after the package which you must copy the containing source files to. Within "`com.g52aim.lab05`" are 4 classes. `MultiMeme.java`, `Lab_05_Runner.java`, `Lab05TestFrameConfig.java`, `SimpleInheritanceMethod.java`. Copy these files into the package "`com.g52aim.lab05`".

There is also a JAR file which you should import and contains the implementation of the mutation operator, 1-point crossover, tournament selection parent selection, and four local search heuristics.

2.1.5 Problem with program hanging

On some student's computers, the program hangs before displaying boxplots/progress plots. Within the `ExperimentalSettings` Class, there are two variables ***`ENABLE_GRAPHS`***, and ***`ENABLE_PARALLEL_EXECUTION`***; Try setting one or both to `false` if you are experiencing these problems. They can be changed back to `true` on the lab machines if you wish to analyse these algorithms later. **Note**: ***`ENABLE_PARALLEL_EXECUTION`*** is not used in this lab's test framework.

2.2 Marking Criterion

1. Correctness of the Multimeme Memetic Algorithm including:

a. The overall correctness of MMA **[10 marks]**.
b. A correct implementation of the simple inheritance method **[10 marks]**.
c. A correct implementation of mutation of memes using innovation rate **[10 marks]**.
d. A correct use of memes to apply local search **[10 marks]**.
e. A correct use of memes to decide intensity of mutation **[10 marks]**.

3 REPORT [50 MARKS]

To be announced at the start of the lab via a "report exercise sheet" on Moodle.

4 FRAMEWORK BACKGROUND

A multi-meme memetic algorithm extends the standard memetic algorithm by introducing memeplexes to each solution in the population. Within the framework, each memeplex is referenced within each solution as illustrated below with each memeplex containing a set of memes. These memes are stored in an array and can be referenced by their meme index. In the below example meme m_1 is in meme index 0, and meme m_2 is in meme index 1.

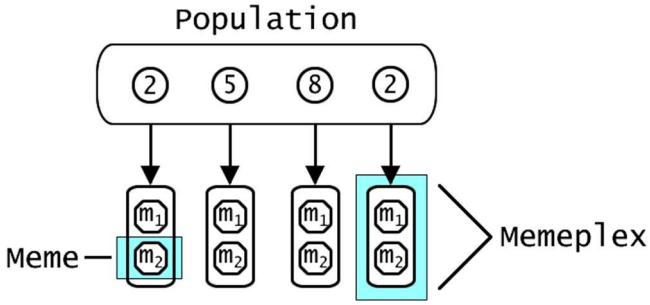

Figure 1 - Representation of the solution memory with memes and memeplexes.

Each meme points to a specific setting/option which symbolises its genetic code. Each individual meme has a specific number of different options, which in this case is 5. These are integer values starting at 0 and ending at the number of options **exclusive**. You may want to create a mapping of these integer values to each option's characteristic, for example local search operator.

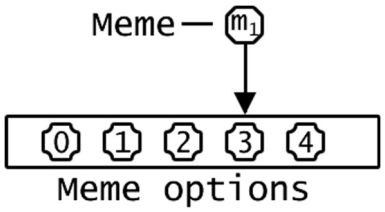

Figure 2 - Association of memes with meme options.

5 Implementation Hints

How do I get the number of meme's in the memeplex?

You can configure this in `Lab05TestFrameConfig` by modifying `MEMES` however this is already set for you. You can retrieve this value by calling `problem.getNumberOfMemes()`.

How do I get a specific meme from a solution?

`problem.getMeme( solutionIndex, memeIndex )` is used to retrieve the meme at index *`memeIndex`* of the solution in memory index *`solutionIndex`*.

How do I find out the number of possible options for each meme?

`meme.getTotalOptions()` is used to get the total number of possible options for the specified meme.

How do I get the current option of a meme?

`meme.getMemeOption()` is used to get the current option of the specified meme. This method returns an integer relating to the current option and should be mapped to its respective operator/value to be meaningful.

How do I set the current option of a meme?

`meme.setMemeOption( memeOption )` is used set the current option of the specified meme.

How do I mutate the current option of a meme?

It is left to the implementer to generate a different option (integer) which you should then set using the `setMemeOption` method explained above.

How do I copy a memeplex from one solution to another?

You should use a combination of `problem.getMeme( solutionIndex, memeIndex )`, `meme.getMemeOption()`, and `meme.setMemeOption( memeOption )` to get and set each meme of the memeplexes.

How do I: perform a bit flip; get the number of variables; evaluate the solution; etc.

See the G52AIM Framework API for framework specific questions on Moodle!

6 Submission

As with all future labs, each coursework will comprise a two-part submission:

(i) [50%] your implementation (code) which needs to be submitted within the computing hours at the lab which will be enforced by attendance sheets. If any extenuating circumstances are preventing you from attending the lab, then please contact me.

(ii) [50%] a brief report which needs to be submitted by Monday, 3pm following each computing session.

IMPORTANT: No late submissions are allowed. Hence any late submission will receive a mark of 0. It is fine just to return the (i) code and not the report for a partial mark. However, if (i) code is not returned within the computing hours, then the report will not be marked yielding a mark of 0 for that computing exercise.

6.1 Implementation Submission

Deadline: 08/03/2018 – 17:00

You should submit a single **ZIP folder** called **[username]-lab05-implementation.zip** including:

1. `MultiMeme.java`
2. `SimpleInheritanceMethod.java`

…to Moodle under Submission area CW5a. **Reminder** late submissions or solutions completed outside of the lab will receive a mark of 0 and you will not be able to submit the report section of the coursework.

6.2 Report Submission

Deadline: Tuesday 13/03/2018 – 15:00

You should submit a single PDF file called **[username]-lab05-report.pdf** to Moodle under Submission area CW5b.

7 Outcomes

1. You should now understand what a Multimeme Memetic Algorithm (MMA) is.
2. You should be able to implement a Multimeme Memetic Algorithm.
3. You should have gained insight into the advantages of using an MMA over a standard MA.

IMPORTANT INFORMATION

The labs are assessed exam style sessions. Hence, you are not allowed to complete the coursework outside of the lab session or discuss your solutions with other students during the lab. (You can, of course, ask questions to the lab helpers though!). You should not prepare code in advance of the labs and use it as part of your answer(s). Template code will be provided before the lab including the test framework to run your implementations, and you will be given part completed files where you should provide your answers.