ASSESSED

G52AIM Lab 3 – Simulated Annealing and Cooling Schedules

This is the third 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

1.1 Lab Exercise

- Implement Simulated Annealing with different cooling schedules covered in the lecture on "Move Acceptance in Metaheuristics and Parameter Setting Issues":
  - Geometric Cooling
  - Lundy and Mees's Cooling

1.2 Report

- To be announced at the start of the lab.

2 Implementation [50 marks]

2.1 Tasks

You are given three Classes, `SimulatedAnnealing`, `GeometricCooling`, and `LundyAndMeessCooling`, within which you will need to complete the implementation of Simulated Annealing, and the two respective cooling schedules. Remember your implementation should perform a <u>single iteration</u> of the respective heuristic method, hence no `while(terminationCriterionNotMet)` loop is required. Running of your solutions is handled by the `Lab_03_Runner` Class that is provided for you and the experimental parameters can be configured in `Lab03TestFrameConfig`. Note also that `Lab_03_Runner` executes the experiments in parallel (which is ok to do given an iteration based termination criterion such as that used within this framework) to reduce the time you must wait. Do not be concerned that experiments seem to complete quicker than expected. Your computer may slow down during the experiments due to the high CPU usage and parallelisation. This is normal and is nothing to worry about.

2.1.1 Simulated Annealing

Simulated Annealing (SA) was covered in the lecture on Move Acceptance in Metaheuristics and Parameter Setting Issues. Below is the pseudocode from the lecture. Note that updating of the cooling schedule is purposely abstract. This allows us to use different cooling schedules without having to reimplement the main body of SA. Remember that the test framework handles solution initialisation and the outer loop of the search method. Hence only lines 7 to 14 need to be implemented in the `runMainLoop()` method. For the acceptance criterion, you <u>can use either accept only improving</u>

moves, or accept improving or equal moves however unlike previous algorithms, this will not make a difference for Simulated Annealing using the Boltzmann distribution equation.

```
 1 | INPUT: $T_0$ and any other parameters of the cooling schedule
 2 | $s_0$ = generateInitialSolution();
 3 | $Temp \leftarrow T_0$;                // initialise temperature to initial.
 4 | $s_{best} \leftarrow s_0$;            // best and current solutions.
 5 | $s^* \leftarrow s_0$;                 // equal to initial solution.
 6 | REPEAT
 7 |     $s' \leftarrow perturbation(s^*)$;     // choose a neighbouring solution of s.
 8 |     $\Delta = f(s') - f(s^*)$;
 9 |     $r \leftarrow random \in [0,1]$;       // generate a uniform random number in
                                       // the range [0,1].
10 |     if( $\Delta < 0 || r < P(\Delta, Temp)$ ) { // accept candidate solution if it is non-
                                       // worsening or with the Boltzmann probability.
11 |         $s^* \leftarrow s'$;
12 |     }
13 |     $s_{best} \leftarrow updateBest()$;    // note this step is handled by the framework .
14 |     $Temp \leftarrow coolTemperature()$; // decrease temperature according to schedule.
15 | UNTIL( termination conditions are satisfied );
16 | return $s_{best}$;
```

You should use the random bit flip heuristic as the move operator within the implementation of Simulated Annealing. In contrast to the previous lab where ILS was implemented, this means that we do not need to keep making a backup of the solution each iteration because we can easily "reject" the move by remembering which bit was flipped and reverting it in the same way we did for DBHC and SDHC. This will have the advantage of eliminating the computationally expensive cloning of the solutions whenever problem.copySolution() is executed and will allow your implementations to run faster!

When generating a random number (pseudocode line 9), whilst there are multiple methods to generate random numbers in the range $[0,1]$, please **only use the nextDouble() method of the random number generator**.

2.1.2 Geometric Cooling

Below is the pseudocode for the geometric cooling schedule:

```
1 | INPUTS: $T_i$, $\alpha$.
2 | $T_{i+1} = T_i \times \alpha$
```

2.1.3 Lundy and Mees's Cooling

Below is the pseudocode for Lundy and Mees's cooling schedule:

```
1 | INPUTS: $T_i$, $\beta$.
2 | $T_{i+1} = \frac{T_i}{1+\beta \times T_i}$
```

2.1.4 Importing the required files

Within the Lab 03 source files archive is one folder named after the package which you must copy the containing source files to. The heuristic that will be used in this exercise will be the same `RandomMutation` heuristic that was used in the previous lab. Within "`com.g52aim.lab03`" are 6 classes. `SimulatedAnnealing.java`, `Lab_03_Runner.java`, `Lab03TestFrameConfig.java`, `CoolingSchedule.java`, `GeometricCooling.java`, and `LundyAndMeesCooling.java`. Copy these files into the package "`com.g52aim.lab03`".

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.

2.2 Marking Criterion

1. A correct implementation of Simulated Annealing **[30 marks]**.
2. A correct implementation of Geometric Cooling **[10 marks]**.
3. A correct implementation of Lundy and Mees's Cooling **[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

No additional knowledge of the framework is required for Lab 03; this section is the same as that given in the previous lab exercise sheet.

For a single point based search method, the G52AIM Framework maintains two solutions in memory, a current solution, and a backup solution. When a `SATHeuristic` is invoked on a problem, it modifies the solution in the current solution index (`CURRENT_SOLUTION_INDEX`) leaving the candidate solution (s') in the `CURRENT_SOLUTION_INDEX`, and a backup of the original solution (s) in the `BACKUP_SOLUTION_INDEX` within the intermediate solution memory state. The move is then either accepted $(s_{i+1} \leftarrow s_i')$ or rejected $(s_{i+1} \leftarrow s_i)$ using the respective move acceptance criterion.

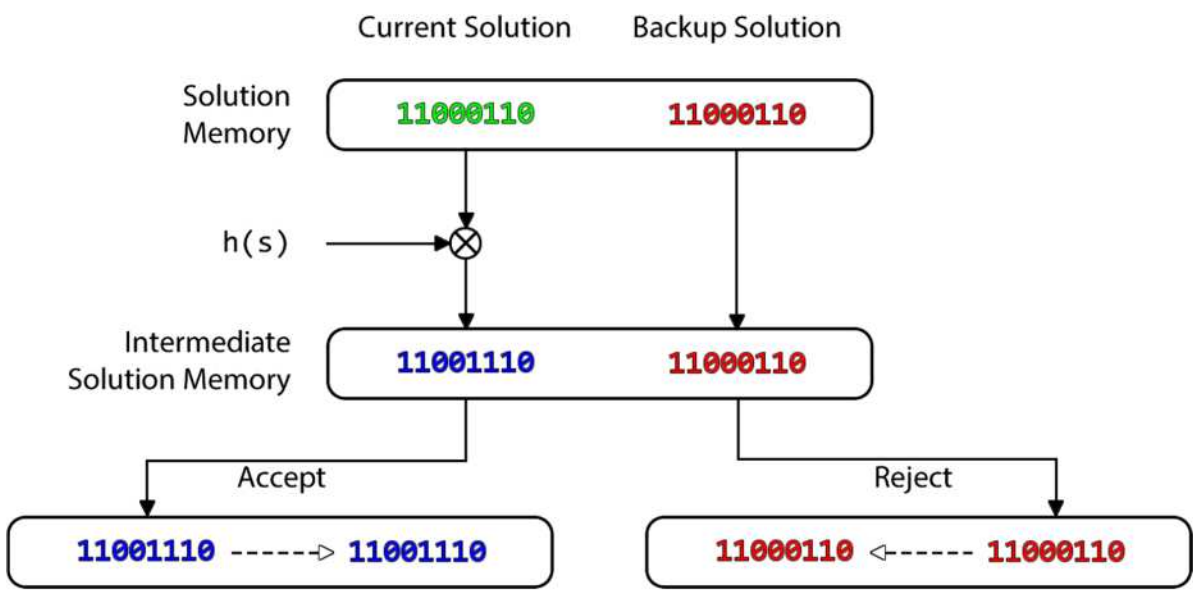

Figure 1 - Single Point-based Search in the G52AIM Framework. The dashed arrow represents a solution being copied between solution memory indices using the copySolution(int source_memory_index, int destination_memory_index) method of the SAT problem.

5 Implementation Hints

How do I use Euler's number in Java?

Java's Math API includes a constant `E` which is Euler's number. The Math API actually contains a function `exp(double a)` which allows you to calculate e to the power of a.

E.g. `Math.exp(0) = 1`.

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

Ensure that when submitting your work that you accept the plagiarism statement and mark it as final; otherwise, your submission will be a *draft* and cannot be accepted as a final 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: 22/02/2018 – 17:00

You should submit a single **ZIP folder** called **[username]-lab03-implementation.zip** including `SimulatedAnnealing.java`, `GeometricCooling.java`, and `LundyAndMeesCooling.java` to Moodle under Submission area CW3a. **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.

Make sure that you sign the attendance register before leaving the lab!

6.2 Report Submission

Deadline: Tuesday 27/02/2018 – 15:00

You should submit a single PDF file called **[username]-lab03-report.pdf** to Moodle under Submission area CW3b.

7 Outcomes

1. You should know how to implement Simulated Annealing.
2. You should be able to implement different cooling schedules and understand the differences between them.
3. You should appreciate the difficulty of parameter tuning, even for solving problems of the same type (MAX-SAT).

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.