ASSESSED

G52AIM Lab 2 – Iterated Local Search

This is the second 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 Iterated Local Search (ILS) embedding "intensity of mutation" and "depth of search" parameters with a single-perturbative random mutation heuristic.
- Note that the local search heuristic used by ILS will be that implemented during lab 01 (DBHC); if you did not complete this, you should do so before the lab!

1.2 Report

- To be announced at the start of the lab.

2 Implementation [50 marks]

2.1 Tasks

You are given two Classes; `IteratedLocalSearch`, and `RandomMutation`, within which you will need to complete the implementation of Iterated Local Search embedding *intensity of mutation* and *depth of search* parameters, and a single-perturbative random mutation heuristic. Remember your implementation should perform a single iteration of the respective heuristic method, hence no `while(terminationCriterionNotMet)` loop is required. Running of your solutions are handled by the `Lab_02_Runner` Class that is provided for you and the experimental parameters can be configured in `Lab02TestFrameConfig`. Note also that `Lab_02_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 Iterated Local Search

ILS was covered in the lecture on metaheuristics. Below is the pseudocode from the lecture adapted to embed intensity of mutation (IOM) and depth of search (DOS) parameters. Remember that the test framework handles solution initialisation and the outer loop of the search method. Hence only lines 4 to 7 need to be implemented in the `runMainLoop()` method. For the acceptance criterion, you can use either; accept only improving moves, or accept improving or equal moves. We suggest that you think about how ILS works and think about whether strict acceptance is necessary or not when deciding which criterion to use.

```
1 | s_0 = generateInitialSolution();
```

```
2 | s* = localSearch(s0);              // optional - not used in this case
3 | REPEAT
4 |     s' ← s*;                      // syntactic step for below pseudocode –
                                        this does not represent an actual operation
5 |     for( 0 -> iom - 1 ) s' ← perturbation(s');
6 |     for( 0 -> dos - 1 ) s' ← localSearch(s');
7 |     s* ← acceptanceCriterion(s*, s', memory);
8 | UNTIL( termination conditions are satisfied );
9 | return s*;
```

2.1.2 Single Perturbation Random Mutation

Each solution to a SAT problem is represented by a bit string, for example 0011101010. A single-perturbative random mutation heuristic should flip **one** bit/variable **selected randomly** using the in-built experimental random number generator.

```
1 | i ← random ∈ [0, N];
2 | s'_i ← flip(i, s_i);
```

2.1.3 Importing the required files

Within the Lab 02 source files archive is one folder named after the package which you must copy the containing source files to; within "`com.g52aim.lab02`" are 4 classes. `IteratedLocalSearch.java`, `RandomMutation.java`, `Lab_02_Runner.java`, and `Lab02TestFrameConfig.java`. Copy these files into the package "`com.g52aim.lab02`".

2.1.4 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 Iterated Local Search. **[40 marks]**.
2. A correct implementation of Random 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

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 does not modify the backup solution in the `BACKUP_SOLUTION_INDEX` within the intermediate solution memory state. The move should then

either be accepted ($s_{i+1} \leftarrow s_i'$) or rejected ($s_{i+1} \leftarrow s_i$) using the respective move acceptance criterion.

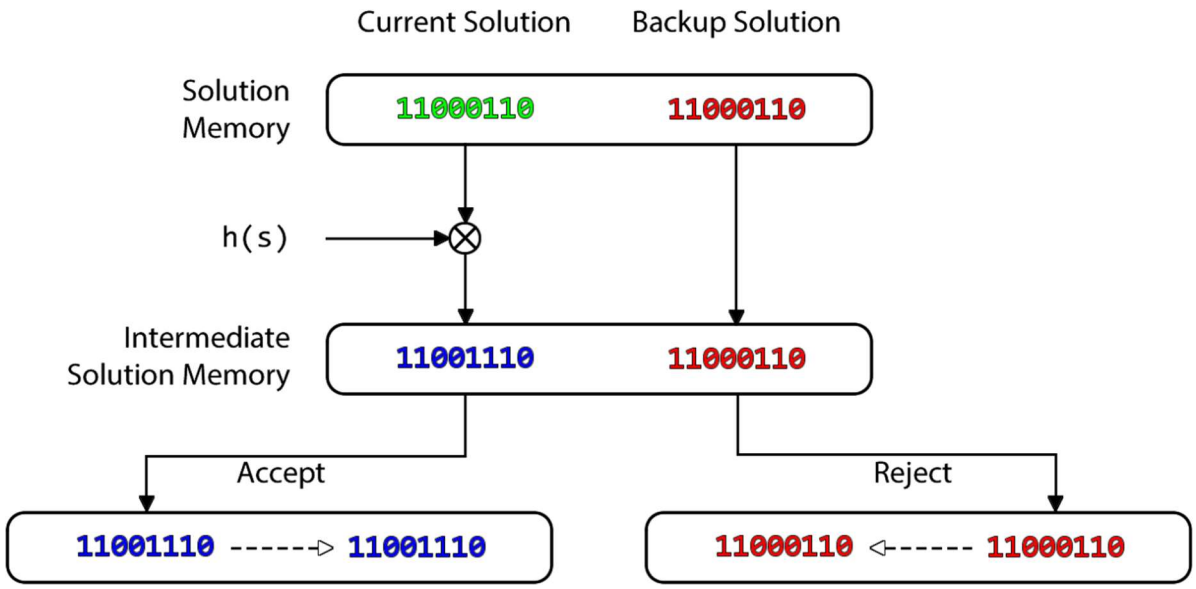

Figure 1 - Single Point-based Search in the G52AIM Framework. The dashed arrow represents a solution being copied between solution memory indices using the copySolution(int source_memory_index, int destination_memory_index) method of the SAT problem.

5 Implementation Hints

Below are some answers to questions which may arise from the implementation of the hill climbing heuristics.

Where do I find the mutation and local search heuristics?

Both mutation and local search heuristics are created within `Lab_02_Runner` and are passed into the constructor of the `IteratedLocalSearch` Class. They are then stored as member variables named `mtn` and `ls` respectively.

How do I use different/multiple heuristics within a single search method?

A heuristic h is applied to a solution by calling $h.applyHeuristic(problem)$. Remember `SATHeuristics` within this framework modify the solution in the `CURRENT_SOLUTION_INDEX` **only**. Hence, multiple `applyHeuristic` calls can be made one after another with each heuristic receiving the candidate solution generated by the previous heuristic.

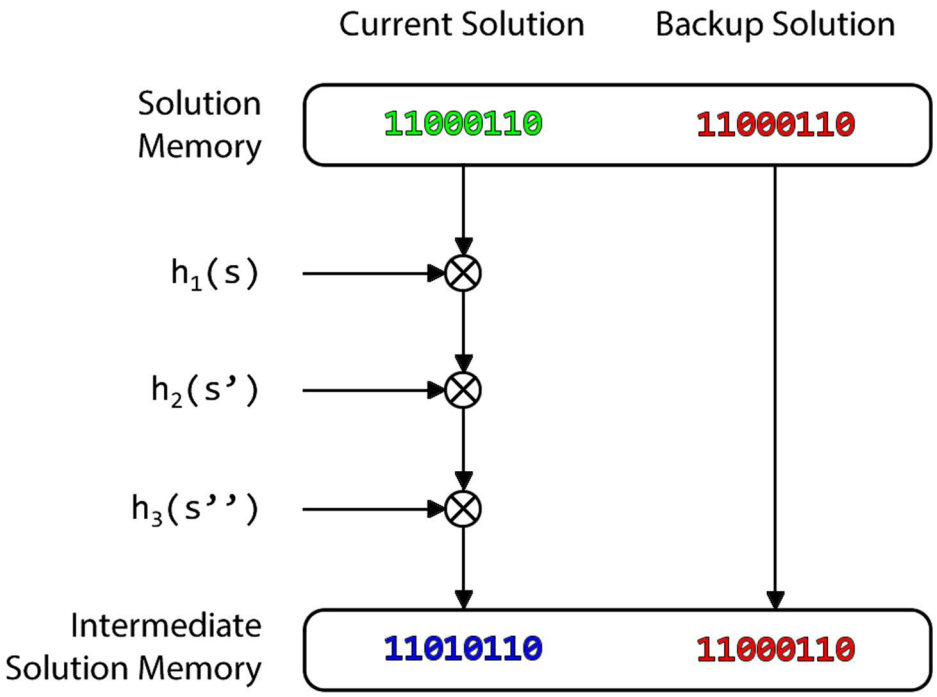

Figure 2 - Illustration of chaining multiple heuristics in the G52AIM Framework.

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 Tuesday, 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: 15/02/2018 – 17:00

You should submit a single **ZIP folder** called **[username]-lab02-implementation.zip** including `IteratedLocalSearch.java` and `RandomMutation.java` to Moodle under Submission area CW2a. **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 20/02/2018 – 15:00

You should submit a single PDF file called **[username]-lab02-report.pdf** to Moodle under Submission area CW2b.

7 Outcomes

1. You should know how to implement Iterated Local Search.
2. You should understand the effects that varying the intensity of mutation and depth of search settings have on the search space.
3. You should have an idea how other single point-based metaheuristics can be implemented within the G52AIM Framework.

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.