ASSESSED

G52AIM Lab 1 – Hill Climbing

This is the first 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

- Implementation of **two** hill climbing heuristics covered in the lecture "Components of Heuristics and Hill Climbing", namely Steepest Descent Hill Climbing and Davis's Bit Hill Climbing for **minimising** the number of unsatisfied clauses in the MAX-SAT problem.

1.2 Report

- To be announced at the start of the lab.

2 Implementation [50 marks]

2.1 Tasks

You are given two Classes, `DavissBitHC` and `SteepestDescentHC`. Within these Classes, you will need to complete the implementation of Davis's Bit Hill Climbing, and Steepest Descent Hill Climbing respectively. Within each Class, you will find the pseudocode for reference. Remember these should perform a single iteration of the respective heuristic method, hence no `while(terminationCriterionNotMet)` loop is required. Running of your solutions is handled by the `Lab_01_Runner` Class that is provided for you and the experimental parameters can be configured in `Lab01TestFrameConfig`. Note also that the Lab_01_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's responsiveness may slow down during the experiments due to the high CPU usage and parallelisation. This is normal and is nothing to worry about.

When implementing both heuristics, we would like you to perform the hill climbing step as accepting only improving moves. That is, the candidate solution/bit flip is accepted if and only if the objective value of the process results in an improved solution, otherwise the previous solution state should be reverted to. This has the added benefit of preventing the search from becoming stuck on plateau's/neutral spaces/shoulder regions in the search landscape.

$$f(s_{i+1}) \leftarrow \begin{cases} f(s_i') & f(s_i') < f(s_i) \\ f(s_i) & otherwise \end{cases}$$

2.1.1 Importing the required files

Within the Lab 01 source files archive are two folders. These are named after the packages which you must copy the containing source files to. There are four java files included in the "`com.g52aim.lab01`" folder. `DavissBitHC.java`, `SteepestDescentHC.java`, `Lab_01_Runner.java`, and `Lab01TestFrameConfig.java`. Copy these files into the package "`com.g52aim.lab01`".

2.1.2 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 Davis's Bit Hill Climbing **[25 marks]**.
2. A correct implementation of Steepest Descent Hill Climbing **[25 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 should modify the solution in the current solution index (`CURRENT_SOLUTION_INDEX`).

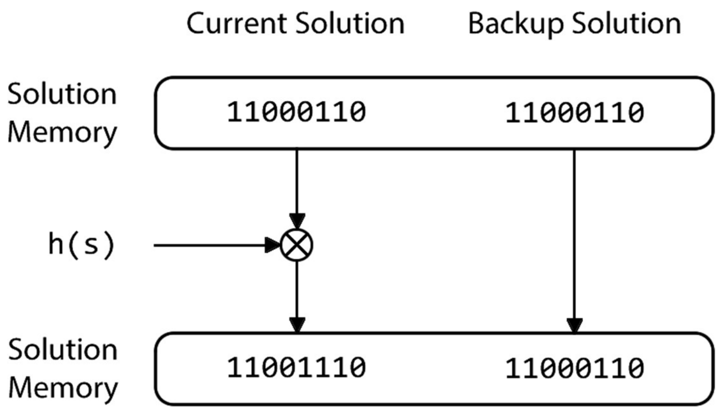

Figure 1 - Modification of the solution in memory when applying a SATHeuristic for single point-based search.

When invoking `bitFlip(int bitIndex)` on the problem, the bit that is flipped is that of the solution in the current solution index **only**.

The `Lab_01_Runner` Class that will run your solutions employs <u>all moves acceptance</u> as its move acceptance criterion since the hill climbing heuristics themselves should return a solution in the current solution index that is either better than the original solution, or the original solution itself.

5 Implementation Hints

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

How can I generate a random number using the experimental seed?

The framework has been designed such that a single random number generator is created using the experimental seed value. This can be accessed from within the `SATHeuristic` class using the variable name `random`.

The random number generator within Java is a Random Object, more information can be found here: link. More specifically, you should be looking at one of the `nextInt` methods.

How can I create a permutation of numbers?

There are two helper functions which you could use to do this:

Java includes a utility function `Collections.shuffle(list, rnd)` (link) where `list` is a `Collection` (link), and `rnd` is a specified random number generator.

The G52AIM Framework has an inbuilt utility `ArrayMethods.shuffle(array, random)` where `array` is the array to be shuffled, and `random` is a specified random number generator. (see the API Documentation for more details).

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: 08/02/2018 – 17:00

You should submit a single **ZIP folder** called **[username]-lab01-implementation.zip** including `DavissBitHC.java` and `SteepestDescentHC.java` to Moodle under Submission area CW1a.

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 13/02/2018 – 15:00

You should submit a single PDF file called **[username]-lab01-report.pdf** to Moodle under Submission area CW1b.

7 Outcomes

1. You should be able to implement some simple hill climbing heuristics for solving the MAX-SAT problem. (In fact, the logic is the same for all binary encoded problems!).
2. You should have found that different hill climbing heuristics have quite different characteristics and understand that there is no "best" heuristic.
3. You should be able to identify the strengths and weaknesses of each hill climbing heuristic and potentially identify a minor modification leading to significant performance improvement.

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.