COMP2001: Artificial Intelligence Methods – Lab Exercise 1

Recommended Timescales

This lab exercise is designed to take no longer two hours and it is recommended that you start and complete this exercise within the week commencing 5th February 2024. The contents of this exercise will be/was covered in lecture 2.

Getting Support

The computing exercises have been designed to facilitate flexible studying and can be worked through in your own time or during timetabled lab hours. It is recommended that you attend the lab even if you have completed the exercises in your own time to discuss your solutions and understanding with one of the lab support team and/or your peers. It is entirely possible that you might make a mistake in your solution which leads to a false observation and understanding.

Learning Outcomes

By completing this lab exercise, you should meet the following learning outcomes:

- Students should gain experience implementing some local search heuristics for solving combinatorial optimisation problems.
- Students should understand that no single heuristic performs better than any other given heuristic through experimentation and analysis.
- Students should be able to analyse the behaviour of different heuristics and critique their individual strengths and weaknesses.

Objectives

The purpose of this lab exercise is to gain experience in implementing some "hill-climbing" local search heuristic that were covered in the lecture "Components of Heuristic Search Methods and Hill Climbing".

This lab exercise, in addition to importing the files, is split into three sections with the last being optional for eager students. Note that completion of the optional tasks may take more than two hours.

- COMP2001 framework background including solution memory. **[REQUIRED].**
- Implementation and evaluation of Steepest Descent (SDHC) and Davis's Bit (DBHC) Hill Climbing heuristics. **[REQUIRED].**
- Implementation and evaluation of two variants of the previous heuristics. **[OPTIONAL].**

Task 0 – Importing lab exercise files

Download the source code for this lab exercise from the Moodle page. You should find various files related to the hill climbing heuristics, and the exercise runner configuration, and the exercise runner itself. Drag the **heuristics** folder into your IDE so that it is a direct subfolder of "com.aim". The "Exercise1TestFrameConfig" and "Exercise1Runner" classes should be placed within the "runner" package alongside the ones from exercise 0.

You will also see the source code on Moodle for "Lab Exercise 1 Source Files (Optional)", you should download and import these files if you want to complete the optional exercise.

Task 1 – Framework Background

For a single point-based search method, the COMP2001 Framework maintains two solutions in memory, a current solution, and a backup solution. When a `SATHeuristic` is invoked on a problem, it should only modify the solution in the current solution index (`CURRENT_SOLUTION_INDEX`).

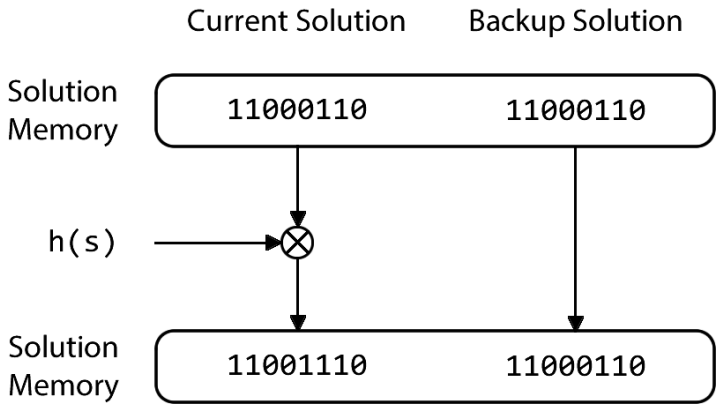


Figure 1 - Modification of the solution in memory when applying a SATHeuristic for single point-based search.

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 **all moves acceptance** 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.

Question 1: What would be the issue if a heuristic was implemented to modify a solution that is stored in a non-current solution memory index? **Hint**: think about the experimental framework and any assumptions that it regarding the layout of solutions. You might also find it useful to read and understand the code for the method "`public RunData runExperiment(int run, int heuristicId`" in the `Exercise1Runner` class.

Task 3 – Steepest Descent and Davis's Bit Hill Climbing Heuristics

Implementation

Template code is provided for you in DavissBitHC.java and SteepestDescentHC.java. Your task is to implement these hill climbing heuristics by writing the code to **perform a single pass** of the heuristic within the method "`public void applyHeuristic(SAT problem)`".

Pseudocode is supplied as the header to the apply heuristic method in the respective class defining the heuristic. Pseudocode is not code and you will need to use the appropriate methods within the COMP2001 API to implement the heuristics. Some of these methods do not

exist, for example `createRandomPermutation()`, and you should implement these yourself.

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 can have the added benefit of preventing the search from becoming stuck on plateau's/neutral spaces/shoulder regions in the search landscape.

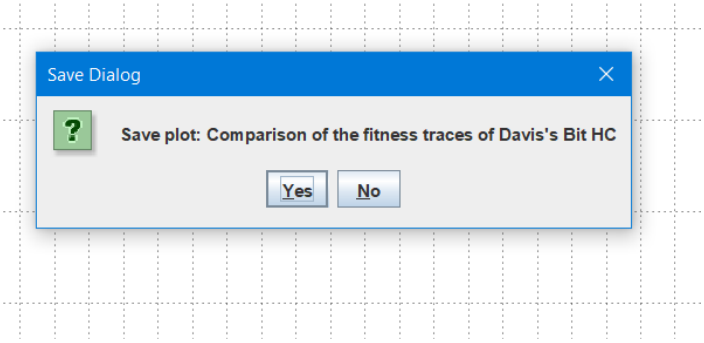
$$f(s_{i+1}) \leftarrow \begin{cases} f(s_i') & f(s_i') < f(s_i) \\ f(s_i) & otherwise \end{cases}$$

Experimentation

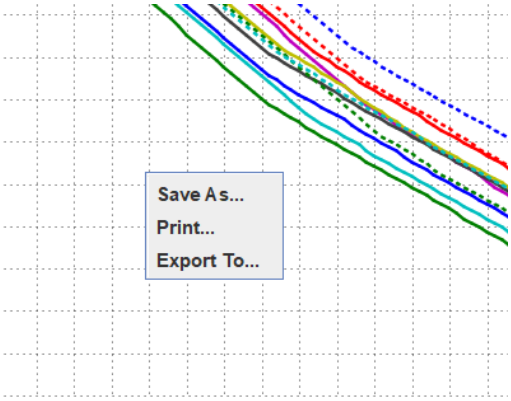
Within the test frame configuration for this lab, there are two parameters that you should change:

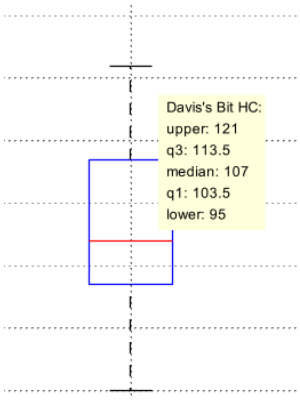
1. The MAX-SAT instance to be solved (referenced by instance ID).
    a. Instance "1", and
    b. Instance "4".
2. The total execution time.
    a. 1 second,
    b. 5 seconds,
    c. 10 seconds, and
    d. 20 seconds.

Note that once your experiments have finished, you can save the images of the plots and boxplot as a PDF document by clicking on the window itself. (For the top-most window, you should click off of it and select it again for the save dialog to appear).



You may also save the plot and data by right clicking on the plots and choosing "Save As..." to save the plot, or "Export To..." to save the data as CSV files. Note that this saves each series as a separate file so you will need to merge these files and name them appropriately before exporting the data of another plot.





The boxplot does not support saving data however you can hover over each box to read off the data.

For each of the following questions, remember to discuss your findings with your peers or with any of the lab assistants to get feedback on your understanding. When discussing findings on Moodle, it would be helpful to use a sensible name for the topic so that everyone knows which exercise you are referring to. I.e. "Lab#1: Question#2", "Lab#1: Question#3", and "Lab#1: Question#4". You will need to modify the `INSTANCE_ID` and `RUN_TIME` variables within `Exercise1TestFrameConfig` to ensure that you are running the correct experiments for the following questions.

Question 2

Run both heuristics for 1 second on SAT instance #4. Which of these heuristics performs better for solving this instance given this run time?

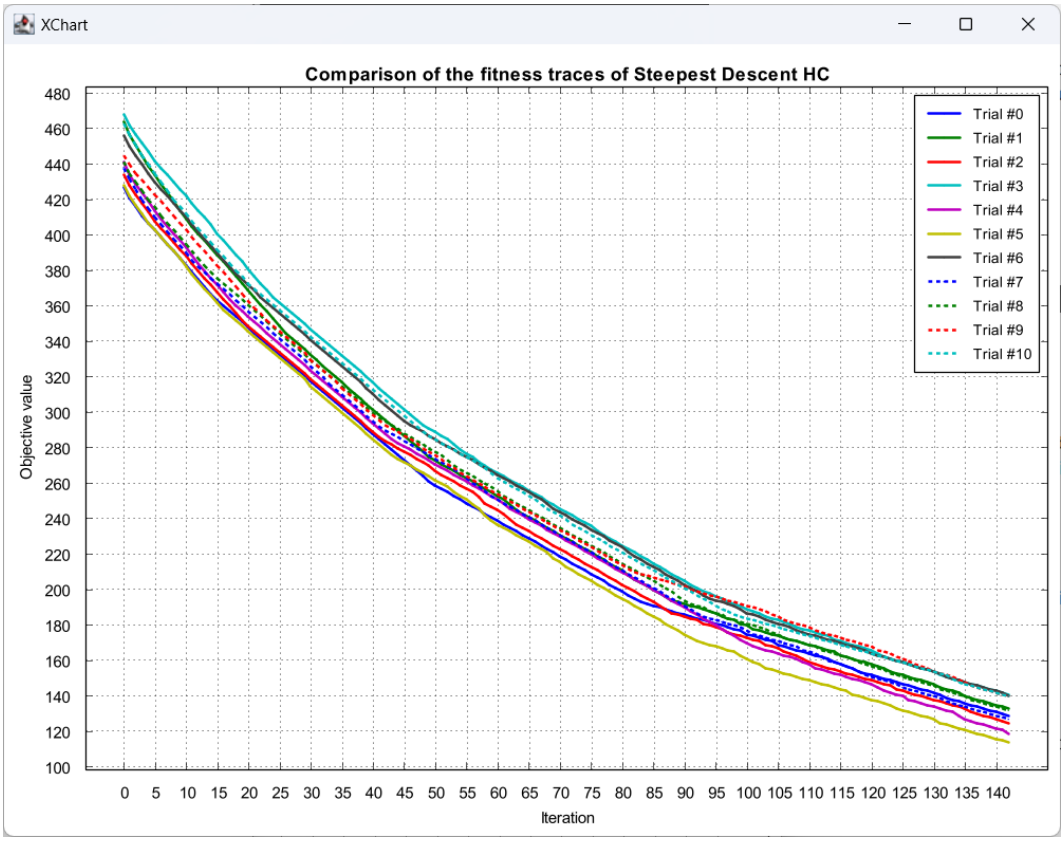
Question 3

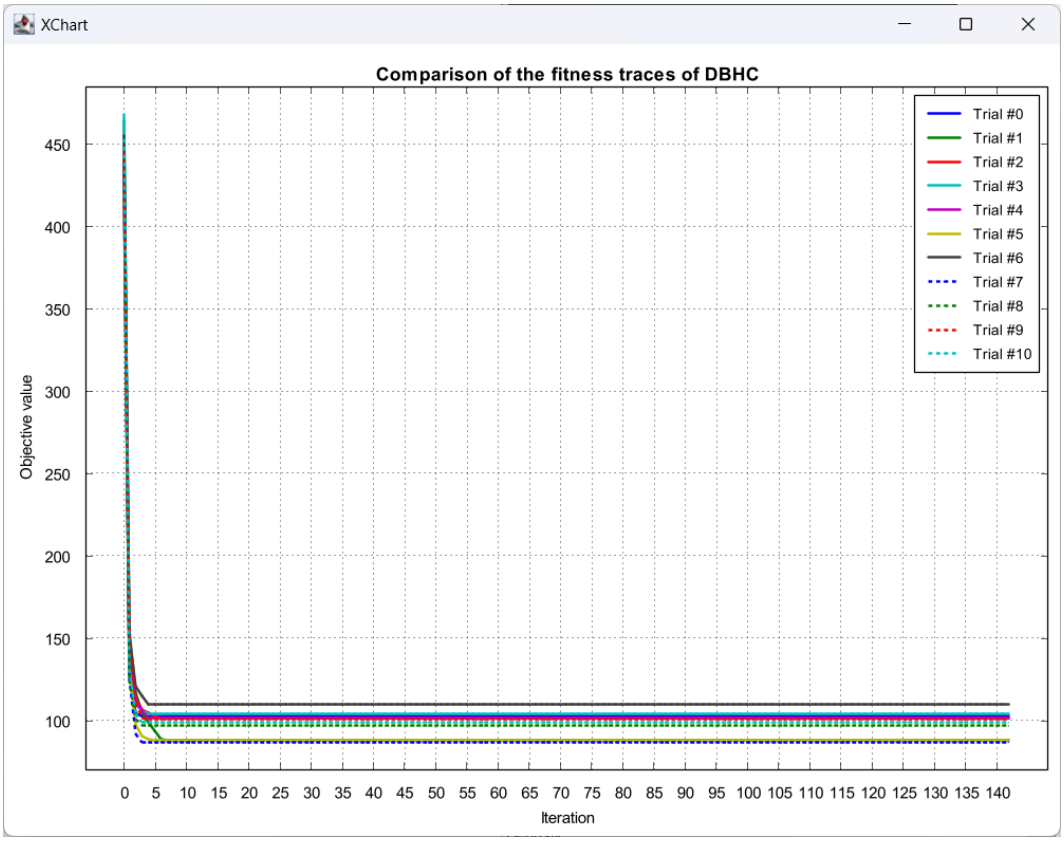
Run both heuristics for an extended run time for solving SAT instance #4. Which heuristic performs better now? Why do you think this is the case? Evidence your claim(s) using the plots provided after running your experiment.

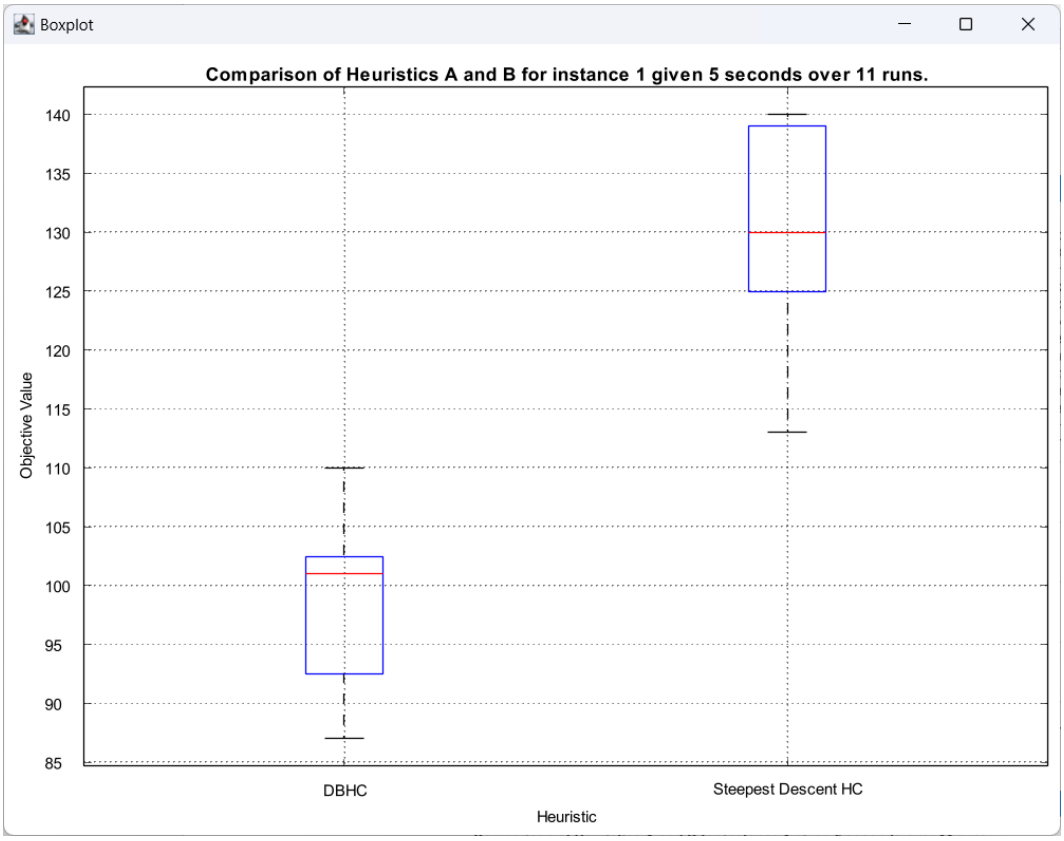
Question 4

Run both heuristics for some amount of time for solving SAT instance #9. Which heuristic performs the best for solving this problem instance using your chosen run time?

Expected output for a valid implementation of SDHC and DBHC when used to solve MAX-SAT instance #1 given a runtime of 5 nominal seconds.







Task 4 [OPTIONAL] – Shallowest Descent and k-Bit Davis's

Note that this task is optional but may be interesting if you are eager to go further.

Implementation

Template code is provided for you in ShallowestDescentHC.java and KBitDavissHC.java. Your task is to implement these hill climbing heuristics by writing the code to **perform a single pass** of the heuristic within the method "`public void applyHeuristic(SAT problem)`".

A description of these heuristics is provided in the header to the apply heuristic method in the respective class defining the heuristic. It should not be too difficult to implement each of these as modification of the existing Steepest Descent and Davis's Bit Hill Climbing heuristic classes.

Experimentation

In addition to the test frame configuration parameters introduced in task 3, you should familiarise yourself with the method "public static SATHeuristic getSATHeuristic(int heuristicID, Random random)" within the `Exercise1TestFrameConfig` class, and modify the variable NUMBER_OF_HEURISTICS_TO_TEST as appropriate for running the following experiments.

Question 5

Run some experimentation to analyse the performance of all four heuristics given 1 second on SAT instance #1. Which of these heuristics performs better for solving this instance given this run time?

Question 6

Following on from question 5, extend the runtime for as long as required to check if your observation is still valid given a larger computational budget. Which heuristic performs the best and why did you settle on the runtime used in your experiment?

Question 7

Do the results of steepest versus shallowest descent surprise you? With relation to the search landscape, why might one outperform the other given variable computational budgets?

Question 8

Re-run the experiments but this time using instance 4. What do you observe with relation to the performance difference between steepest and shallowest descent compared to instance 1?