COMP2001: Artificial Intelligence Methods – Lab Exercise 0

Recommended Timescales

This lab exercise is designed to take no longer than one week, and it is recommended that you start this exercise in the week commencing 29th January 2024.

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.

Objectives

The purpose of this lab exercise is to get set up with Java, IntelliJ IDE, and the COMP2001 Framework, and use the COMP2001 framework to experiment with pseudorandom number generators. You will be using these systems in the subsequent lab exercises which together aims to give you the knowledge, understanding, and practical skills required to meet the learning outcomes of the computing part of the module. These include designing and implementing low-level operators, heuristics, and meta-heuristics for solving combinatorial optimisation problems. These learning outcomes will be the focus of the in-lab assessment, so it is essential that you engage with these exercises.

The topics that will be covered in this exercise are:

- COMP2001 Framework.
- Maximum Satisfiability problem.
- Pseudo-random number generators.

Learning Outcomes

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

- Students should be able to understand the Maximum Satisfiability (MAX-SAT) Problem such that they can explain how heuristics modify the solutions to MAX-SAT problems.
- Students should gain a practical understanding through experimentation of how pseudorandom number generators work and their consequences for computational experiments.
- Students should gain a practical insight into the COMP2001 Framework API to enable them to implement higher-level heuristic optimisation methods in future exercises.

Task 1 – Getting set up with Java, IntelliJ, and the COMP2001 Framework

In this first task, we will guide you through setting up the software needed for completing the lab exercises. Depending on whether you plan on using the machines in A32, the computer science remote desktop, or your own machine, some of these may already be installed and you can skip those steps.

Java

Java should already be installed on university devices; you can check this by opening a command line and entering the command "java -version". Many of you should already have installed Java from previous modules, however if Java is new to you, follow the instructions below to download and install Java on your device. These instructions are given for Windows, your experience on other devices may differ.

Download and installation instructions

1. To download Java, go to the OpenJDK webpage OpenJDK JDK 21.0.2 GA Release (java.net) and download the archive that corresponds to your operating system.
2. Locate the archive and decompress it to get the "jdk-21.0.2" folder.
3. Move the "jdk-21.0.2" folder to a sensible location, for example "C:\Program Files\Java" if you are using Windows.
4. Now we need to tell the operating system where the Java binary resides by adding the folder to the system PATH.
    a. On Windows you can do this by opening the start menu and typing path. You should then see the option "Edit the system environment variables". Click on this and you should see a new window called "System Properties".
    b. Click on the "Environment Variables…" button (under the "Advanced" tab).
    c. In the new window look for the entry "Path" under "System variables", double click on the entry to open the "Edit environment variable" window.
    d. Press the "New" button" and enter the path to the directory containing the Java executable. If you put the JDK folder into "C:\Program Files\Java" then this should be "C:\Program Files\Java\jdk-21.0.2\bin".
    e. Press "OK" on all the open windows to close them.
    f. Check the operating system can find Java by opening a command line and entering "java -version". You should see the text OpenJDK version "21.0.2" 2024-01-16. If you do not, you may need to restart your device.

IntelliJ

Download instructions.

IntelliJ (IntelliJ IDEA – the Leading Java and Kotlin IDE (jetbrains.com)) has two versions, a free to use community version, and a paid-for ultimate version. The ultimate version is free for students and can be obtained by making an account with your university email address. You can do this at the following link Free Educational Licenses - Community Support (jetbrains.com). Once set up, you should download the version of IntelliJ IDEA Ultimate that corresponds to your operating system and device architecture.

Alternatively, if you do not wish to create an account with JetBrains, you can download the free community edition at the following link https://www.jetbrains.com/idea/download/. Scroll down to find the community edition section.

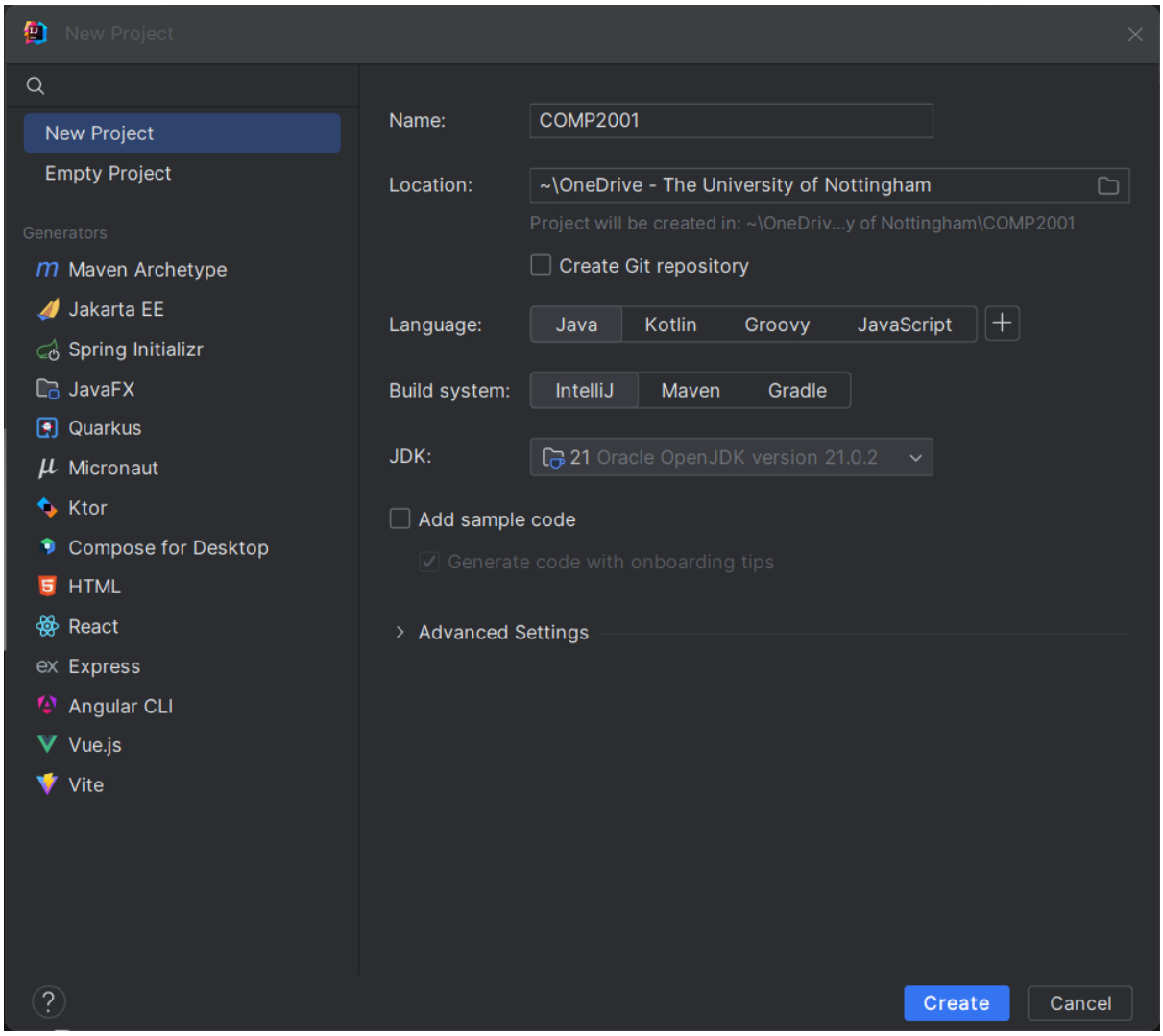
Installation instructions

Note that this process may be different for non-windows operating systems. To install IntelliJ and link the Java installation we installed in the previous step, run the installation package and install to your desired location. Choose "Next" on each page and finally press "Install". You should not need to change any options (unless you wish to).

COMP2001 Framework

The COMP2001 Framework consists of a JAR file which can be downloaded from the module's Moodle page Module: Artificial Intelligence Methods (COMP2001 UNUK SPR) (COMP2011 UNUK SPR) (23-24) (nottingham.ac.uk) along with the xchart dependency; search for and download "COMP2001 Framework JARs".

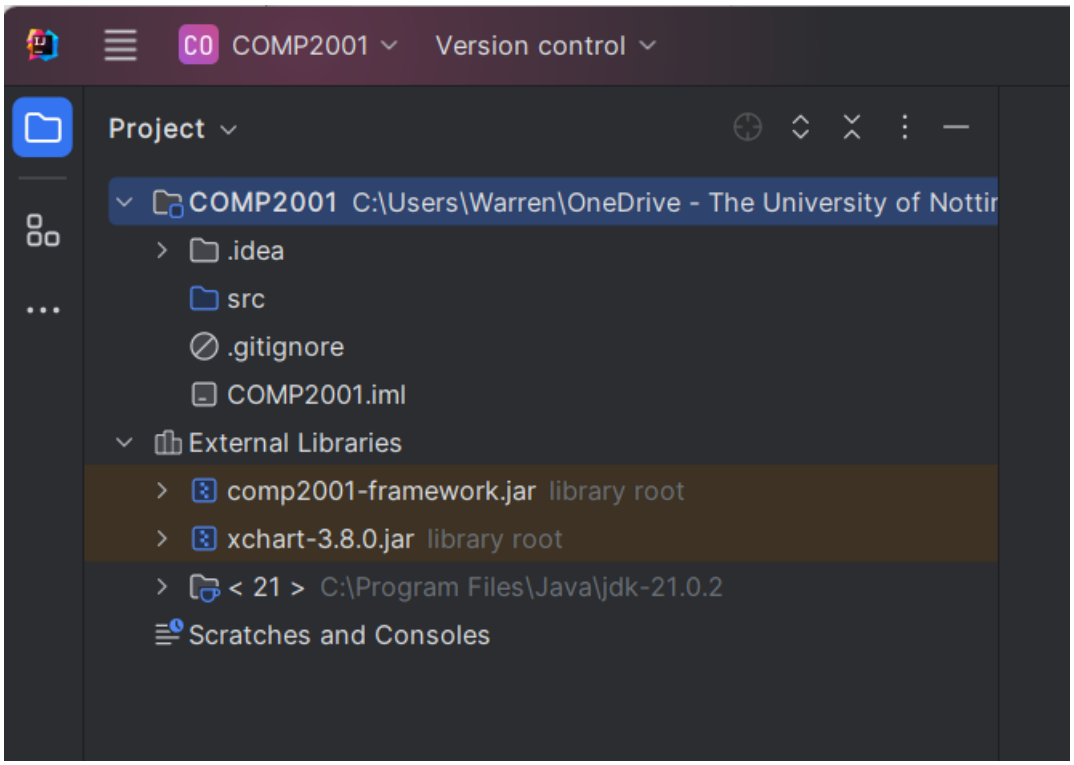
Open IntelliJ and choose to create a New Project. IntelliJ should automatically pick up the Java installation from the previous step. We can see this in the JDK dropdown. Name your project COMP2001 and make sure that it will be saved in a location which is persistent and backed up. We cannot be held responsible for loss of work if you have not backed it up using the university OneDrive, nor if you save it on a public device which is periodically wiped for security reasons (e.g. the lab machine C: drive). Uncheck "Add sample code" and press "Create".



Now to include the COMP2001 Framework, we need to configure the project to locate the JAR files downloaded from Moodle. To do this, press on the button that has four horizontal lines in the top left of the window and choose “project structure”.

In the new window, select “Modules” on the left of the window, highlight the project (e.g. COMP2001), select "Dependencies”, and press on the + icon. Select “JARs or Directories…” and locate the folder containing the JAR files downloaded from Moodle. Select OK to get back to the main interface.

You can verify that this was successful by expanding “External Libraries” to see comp2001-framework.jar and xchart-3.8.0.jar.

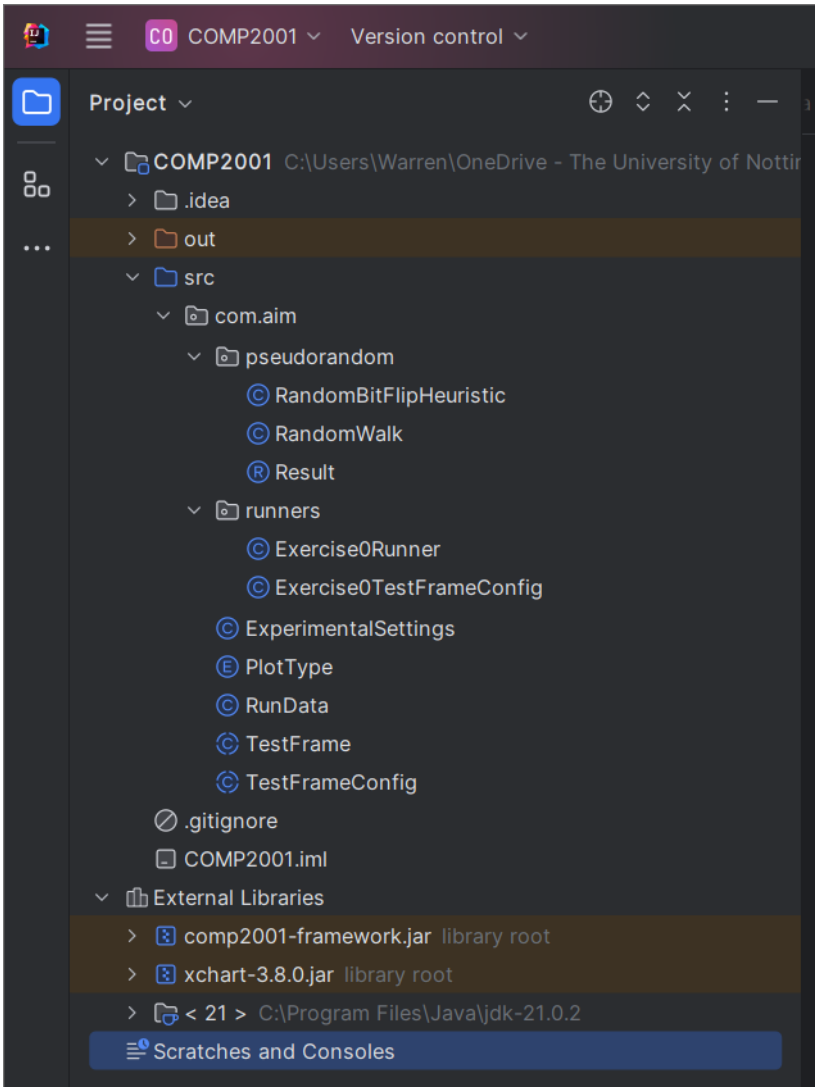


Task 2 – Importing lab exercise files

Each lab exercise will introduce new functionality and requires different experiment runners, source files, and code templates. Each lab exercise code will depend on some of the files from this lab exercise, so it is important that you complete this exercise. Go ahead and download “Lab Exercise 0 Source Files”.

Decompress the archive and locate the “com” folder inside the folder “src”. Drag the “com” folder onto the “src” folder within IntelliJ to move the files into your project. IntelliJ might ask you to confirm the directory to move the files to; you can press on “Refactor” to proceed with the move. Your project structure should now look the same as the below screenshot.

You will need to follow the same set of steps for future lab exercises so you should remember what you did in this step.



Task 3 – The Maximum Satisfiability Problem

The maximum satisfiability problem (MAX-SAT) is an NP-hard optimisation problem. A MAX-SAT problem instance is defined in terms of a number of Boolean variables and a number of disjunctive clauses connected by conjunction. This is known as conjunctive normal form (CNF). The objective in a MAX-SAT problem is to maximise the number of satisfied clauses.

Here is an example of a MAX-SAT problem:

$$(A \vee \neg B) \wedge (B \vee C) \wedge (\neg A \vee \neg C)$$

Here, A, B, and C are the variables and can be either true or false. There are three clauses, $(A \vee \neg B)$, $(B \vee C)$, and $(\neg A \vee \neg C)$ which are all disjunctions of variables. If a variable has a truth assignment in one clause, then that same variable has the same truth assignment across all clauses. In the MAX-SAT problem, the goal is to find a truth assignment to A, B, and C which maximises the number of satisfied clauses. In this example, we can assign $A = true$ so satisfy the first clause, $B = true$ to satisfy the second clause, and for the third clause, we cannot reassign A to be false otherwise the first clause becomes UNSAT. We therefore need to assign $C = false$ to satisfy the third clause.

Typically, and within the COMP2001 framework, a MAX-SAT problem can be represented as a binary string (a sequence of bits which are true or false) where each bit represents the truth value of each variable. In this example, an optimal solution can be represented as "110".

Question 1: What would be the objective value of a solution to the above example problem instance with the representation "000"? Remember to discuss your answers with your peers either in the lab or via the Moodle discussion forum, alternatively you can ask one of the lab helpers if you are unsure.

In the context of the COMP2001 Framework, and other frameworks which we will use later in the module, it is common to formulate optimisation problems as minimisation problems where the goal is to minimise some cost or penalty as defined by the objective function. Within the COMP2001 framework we will be using a minimisation objective function for the MAX-SAT problem where the aim is to **minimise** the total number of **unsatisfied clauses**.

Question 2: What would be the objective value of a solution to the above example problem instance with the representation "110" if we were using the minimisation objective function?

Task 4 – The Bitflip Operator and Pseudorandom Number Generators

Please ensure that you have completed task 3 before attempting this task as it is important that you understand the problem we are solving and its representation.

Familiarisation with the source codes

Going back to IntelliJ (or your preferred IDE), you should see that there are a number of source codes in the package com.aim.pseudorandom. The practical aspect of this lab exercise is centred around these source files, and an exercise specific exercise runner class.

Each lab exercise will have its own runner class(es) and will be numbered corresponding to the lab exercise number. You will receive new runner classes for each exercise and you can find these files in the package com.aim.runners. Each exercise will contain a pre-defined runnable test frame class (in this case Exercise0Runner.java) and a configuration (in this case Exercise0TestFrameConfig) which allows you to change some settings within the experimentation; you will be instructed which settings you should change in the corresponding exercise sheet.

Randomised search

The test frame in this exercise runs a random walk search method which iteratively randomly modifies the solution-in-hand. Open the RandomWalk class and let's see what is happening. Within the run method, there is some code to keep track of actual time taken. This is not very interesting but the code that the timing encapsulates is the important part. Here we are initialising a new solution and iteratively flipping random bits in the bitstring that represents the

solution to the problem being solved. Look at each line of code and the corresponding comment. Refer to the COMP2001 API document to understand what each line of code is doing.

Experiment runners and termination criterion

Now open the experiment runner Exercise0Runner and try running the test frame for this lab exercise – what happens? You should observe that the program never terminates. This is due to the termination criterion of the experiments, and because we have not implemented the randomised bitflip low-level operator. Open the RandomBitFlipHeuristic.java file. You will see a method with the signature "public void applyHeuristic(SAT problem)" which implements an abstract method in its parent SATHeuristic class. At the moment this does nothing which means the solution-in-hand is never modified. For fairness across different devices, the termination criterion within the COMP2001 framework is implemented as a ratio between the number of operations achieved in 10 nominal minutes on a reference machine and the number of evaluations of solutions that have been modified. Since we didn't modify the solution before evaluating the objective value of the solution, we considered this to not contribute to the search and hence we never reach the limit of the termination criterion.

Random bit flip operator

To implement a randomised bit flip operator, we need to understand a few methods in the API. Go ahead and look up the following in the API document before continuing:

- getNumberOfVariables() in the SAT class.
- bitflip(int index) in the SAT class.

Within this operator, we want to be able to choose one of the variables in the solution at random and change its truth value. Try to implement this yourself using the API methods above as a hint. You can use Java's Random class to generate a random integer, and a seeded random number generator is already defined in the SATHeuristic class with the variable name random.

After you have completed the implementation, try running the test frame again. If you implemented it correctly (and haven't modified any configurations) you should get an output with shows the results of running three different trials with the same seed value to get a solution which has an objective value of 358. If your solution is different, you should discuss your implementation with your peers or you can attend the in-person lab and ask one of the lab assistants for help.

Pseudorandom Number Generators

At the end of the previous exercise, you should have run the test frame which executes an experiment consisting of three independent trials using the seed 02022024. This is the value that is used to seed the random number generator Random (Java SE 21 & JDK 21) (oracle.com).

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.

Question 3: What was the objective value of the best solution obtained in each of the three different trials? What do you notice about these and why might this happen?

You can change which seeds are used in the configuration file Lab00TestFrameConfig. Try experimenting with using different seed values to observe the effects and importance of using seeds correctly.

Question 4: What happens if you change m_seeds to contain a single seed value? Why might this be a bad idea for experimental design?

Question 5: What happens if you use different seed values for all three trials? Referring to the RandomWalk search method, what effect does using a different seed value have on the search across each of the trials?