Experimental Methods in Computer Science

DEI-FCTUC, 2022/2023

This assignment is a first (and simple) exercise of designing an experiment. There are many types of experiments and the term "design of experiments" (often referred to as *experimental design* as well) is used for the process of defining and planning experiments in such a way that the data obtained can be analyzed to draw valid and objective conclusions.

The topic of designing experiments of different types and for different purposes is addressed in detail in the lectures (T classes). In any case, the next paragraphs present a brief overview before presenting the assignment purpose.

A classic experiment includes the following elements (you can take them as basic steps for the purpose of this assignment):

1. Problem statement (or research question)
2. Identify variables
3. Generate hypotheses
4. Define the experimental setup/scenario
5. Develop the necessary tools and procedures for the experiment
6. Run the experiments and collect the data/measurements
7. Perform data analysis
8. Draw conclusions (and often go back to the beginning and reformulate the problem statement or test a different hypothesis)

The formulation of sound problem statements (or research questions) is not easy and this is particularly true for the computer science field. A good (i.e., relevant problem statement) should be focused enough to allow the clear identification of the variables of the problem but, at the same time, should be sufficiently open to allow different hypotheses to answer the problem/question.

An example of formulation of a possible problem statement is:

How does the setting (noise, temperature, music, open space, etc.) of the room used by programmers affect the number of bugs found by test suits in the modules produced by such programmers?

The effect we want to measure (dependent variable) is the number of bugs found and the factors (independent variables) are the different room settings and situations. It is assumed that there is a set of conditions that remain stable (e.g., type of programming language, complexity of the modules, etc.). Even for the independent variables, the basic type of experiments only changes one factor at the time (more complex experiments use a factorial approach in which several factors varied together to make the experiment more efficient and allow the study of possible interactions among factors. But, for the time being, we consider only one factor at the time).

A possible hypothesis for the problem statement mentioned above is that programmers tend to make fewer bugs if they listen to classic music at a comfortable volume while programming. In this case, the hypothesis is directional (i.e., establishes a direction for the dependency between the dependent variable and the independent variable) but the hypothesis could simply state that music influences the number of bugs (non-directional hypothesis).

Even in a simple example like this, it is easy to understand the difficulties associated to the correct validation of an hypothesis. For example, it is necessary to perform experiments with a representative group of programmers, use a variety of software under development, assure that the conditions of the different experiment runs remain stable, etc. The analysis of the results if often quite difficult as well, often requiring complex statistic testing techniques to assure that the conclusions are statistically valid.

1. Goals, assumptions and recommendations

The purpose of the assignment is to design the experiment and execute all the steps to evaluate the following general problem statement:

Given a directed graph, the *Maximum Flow Problem* consists of finding the maximum flow that can sent from a source vertex to a target vertex, without exceeding the capacity of each arc. We assume that a capacity of each arc is a positive integer. This problem arises in many real-life applications, for instance, to find a set of disjoint paths in a communications network taking into account the maximum bandwidth between every two network nodes.

There are several algorithms to solve this problem for a given input. For this project, we will consider three algorithms for the maximum flow problem: *Edmond-Karp* (EK), *Dinic*, and *Malhotra, Pramodh-Kumar and Maheshwari* (MPM). For a given graph $G=(V, A)$, where V is the vertex set and A is the arc set, EK Algorithm has a time complexity of $O(|V||A|^2)$, whereas Dinic has $O(|A||V|^2)$ and MPM has $O(|V|^3)$. Therefore, the first is better suited for graphs with few arcs, while the last two are better for graphs with fewer vertices. However, these are theoretical results that only apply to the worst case and it is hard to extrapolate them for a given particular problem input.

Your goal is to investigate the runtime performance of these three algorithms in practice. You believe that several problem features may play a role on performance, such as, the number of vertices, the number of arcs, the graph topology, the capacity at each arc, etc.. Although this is a somewhat artificial problem statement (the main goal is pedagogic and not to represent a realistic research question), it is worth noting that it requires all the steps of a real experiment. Furthermore, the type of measurements required by this

experiment (runtime performance) is highly representative of experiments with computers.

The following assumptions and recommendations should be taken into account:

- There are three programs written in C++ (EK.cpp, Dinic.cpp, MPM.cpp) that implement the two algorithms above, which are available at UCStudent. You can decide to use these three implementations or to use others that are publicly available (the source must be referenced in the assignment report), or even to implement your own.
- To compile the provided programs you should use a C++ compiler that supports the C++11 standard (e.g., GCC, Clang, or MSVC). Furthermore, all three programs should be compiled with the same optimization flags, e.g., -O3 -DNDEBUG for GCC and Clang, and /O2 /DNDEBUG for MSVC. A Makefile assuming the GCC compiler is provided along with the code for the programs to help with compilation.
- All implementations require two arguments: the maximum CPU-time in seconds to run the program and the name of the file that contains the input graph. All programs output the following information for each input file: the value of the maximum flow found, or value -1 in case the program was not able to find the solution within the maximum time defined by the user, and the CPU time taken (in both cases). Since some experiments may take too much time, define the maximum CPU-time at your choice.
- There is an input data generator in python (gen.py) that is also available at UCStudent. This generator creates randomly generated input data (a file) that can be read by the two implementations mentioned above. You need to define the number of vertices (n), the probability of generating an arc between two vertices (p), the maximum capacity (r) that can be assigned to an arc, a random seed and the name of the file that will contain the input data. The code will generate a random graph and assign a capacity to each arc that is taken randomly from the interval [1,c] (according to an uniform distribution). Each input file that is generated contains two integers (n and m) in the first row, and each of the next m lines contains three integers, the indices of two vertices that are connected by an arc and the maximum capacity of this arc. The vertices are numbered from 1 to n. We assume that the source vertex has index *1* and the target vertex has index n. You may consider other generators with different features. Note that if p is too small, the graph may not be feasible for the problem, that is, there exists no path between vertex *1* and vertex n. In that case, the file will only contain value -1.
- Two example inputs generated with the data generator (gen.py) are available in the code folder (input01.txt and input02.txt), which you can use to see if the programs are working as expected.
- Note that if the size of the graph is too small, you may have difficulties in measuring the time accurately, but if the graph is too large you may have other problems such as the time taken by the experiments or the programs may reach some limits of the machine (e.g., memory) that may complicate the experiments.
- The experiments can be repeated using slightly different conditions in order to understand better the situation and allow the generalization of the results. It is open to you to explore this possibility.

2. Outcome

The outcome of the assignment is a written report (PDF file). The report should describe all the steps of the design of the experiment with enough detail to allow others to reproduce the experiment and, of course, should provide clear conclusions about the problem statement.

Given that the sole outcome of the assignment is a report, the quality of the document (i.e., structure of the report, precision of the results reported, quality of writing) is of paramount importance. There is no suggested template for the report structure. The goal is to avoid the rather passive situation in which the students just fill in a given report structure. On the contrary, defining the best structure for the report is part of the goals of the assignment. The teacher is fully available to discuss the proposed structure with the students, as well as any other aspect of the report. Some of the "PL classes" are specifically devoted to provide such support to students.

In addition to the report, the students must keep a folder with all the programs and tools used in the work that allow the complete results to be reproduced.

3. Milestones

There are three milestones, each of which covers particular techniques that are discussed throughout the course:

1. *Exploratory Data Analysis and Linear Regression*
2. *Pre-registration of hypotheses*
3. *Hypothesis testing*

The goal of each milestone is to design the experiment, collect the data, and use the corresponding data analysis technique to draw conclusions about the problem statement. The outcome of the first and the third milestone is a written report (PDF file, maximum 8 single-column pages, including cover, references and appendices). The report should describe the steps of the design of the experiments with enough detail to allow others to reproduce the experiments and should provide clear and sound conclusions. The outcome of the second milestone is a single written page (PDF file) for the pre-registration of hypotheses to test in the third milestone. The goals for each milestone will be discussed in the classes.

4. Resources

Students are supposed to use their own computers for running the experiments, but virtual machines can also be used. See more information about virtual machines at DEI in the following link: https://helpdesk.dei.uc.pt/configuration-instructions/cloud2-dei.

5. Calendar and miscellaneous

The assignments must be done by groups of up to 3 students according to the following calendar:

- September 26th – Submit a single PDF file with the names, student IDs and emails of the students (only @student.dei.uc.pt or @student.uc.pt) that constitute each group. Note that the group of students must remain the same for all the assignments.
- November 4th – Submit the assignment report for the first milestone.
- November 25th – Submit the assignment report for the second milestone.
- December 23rd – Submit the assignment report and code for the third milestone.
- Oral defense from 5 to 7 January (to be defined later on for each group).

Submissions of the reports should be uploaded at Inforestudante. Submissions after the deadline (at 23h59) are not possible.

Plagiarism means mandatory fail in the course and internal (UC) disciplinary procedure. Please, cite adequately all text and material you take from the Internet. All parts of the report must be written by the students and not copied & pasted & changed from the Internet.