Algorithmic complexity and graphs : project

NICOLAS LE HIR
nicolaslehir@gmail.com

TABLE DES MATIÈRES

INTRODUCTION

All processing should be made with python3.

A pdf report is expected in order to present your work. There is no length constraint on the report, you do not need to write more than necessary. The goal of writing a report is that you understand what you did with the project (an also that I understand more easily too and can give you some useful feedback).

The 4 parts of the project are independent.

1 PART 1 : THE SNOWPLOW PROBLEM

1.1 Introduction

A snowplow must clean the snow in front of n houses, $n \in \mathbb{N}$. The position of each house is represented by a float number (not necessary positive), hence we assume that all houses are on a given one dimensional road.

Initially, the snowplow is in 0. The traveling time of the snowplow is equivalent to the distance that it went through since the beginning of the day. If there are several houses at the same point, it does not take a longer time to clean the snow (the snow is cleaned as soon as the snowplow has passed). However, since the positions of the houses are represented as floats, this won't happen (with a very high probability).

The snowplow must minimize the **average waiting time of the houses**, before the snow is cleaned in front of their doorstep. **This minimization is from the point of view of the houses** (and not from the point of view of the snowplow). The total

distance traveled by the snowplow is not relevant itself.

1.2 Exercise

1) Propose a $n \in \mathbb{N}$ (it does not need to be large $n = 10$ is fine) and a configuration of the houses where the optimal order of cleaning is **not** obtained by either :

— sorting the houses positions and cleaning them in that order.

— going to the closest house at each time step.

2) Propose and implement a polynomial-time algorithm (in terms of the number of houses) based on the positions of the houses, in order to clean the snow. If it is not polynomial (if it is exponential for example) we saw in class that it would not be able to be run in a reasonable time.

Conventions : You must write a function that takes has input the positions of the n houses, and outputs the positions of the houses, sorted in the same order as they are cleaned.

Name of the function : parcours(list)

returns : list of sorted houses in the relevant order.

test : I generate random initial positions of houses with a normal distribution centered in 0, with $n = 1000$, with **numpy.random.normal(0,1000,1000).tolist()** , and then I statistically evaluate the performance of your algorithm. So **parcours()** receives a **python list** as an argument.

Hence you can prototype you algorithm with small n and then check the behavior with larger n.

You are strongly encouraged to use python3. If you really prefer to use another language, you need to ensure that you program can be used from python. I am not sure if it is convenient to do so, but some projects seem to exist `https://cpp.developpez.com/tutoriels/interfacer-cpp-python/`.

I test your solution and compare it to several benchmarks. Your solution should give a waiting time that is smaller to 90% of the time that we would obtain with a greedy solution, that consists in going to the closest house at each step. You are encouraged to try to obtain even lower values (which is possible with some algorithms).

Hence, please note that you can test your solution yourself in order to make sure that it meets the requirements, by also implementing the greedy solution, and comparing the cleaning time to that returned by your solution.

3) In your pdf report, prove that your solution runs in polynomial time. You don't have to be extremely formal, but clear enough to show that you understand why your solution is polynomial.

2 PART 2 : CHOOSING BINOMS IN A COMPANY (MATCHING)

2.1 Introduction

A manager must choose binoms among a set of employees in a company. However, while some employees may work together, others may not. This information is represented in a graph G. Each employee is represented by a node in G. If two employees are represented by the nodes A and B, there is an edge between these two nodes if and only if they may work together. Choosing binoms corresponds to perform a matching in the graph G.

The manager would like to form a number of binoms that is as large as possible. Hence, The goal of this exercise if to perform a matching of large size. This does not necessary mean that you have to find an optimal matching in the strict sense of the term, but the larger the size of the matching, the better it is.

2.2 Random graphs

In this exercise, you will work with random graphs, in order to statistically evaluate the performance of different matching methods. The graphs must be Erdös-Renyi graphs, also called G_{np} graph, with 120 nodes, and a probability $p = 0.04$ for the presence of each edge in the graph. In order to generate this kind of graphs, you can use the method **gnp _random _graph()** from **networkx**, with $n = 120$ and $p = 0.04$.

https://networkx.org/documentation/stable/reference/generated/networkx.generators.random_graphs.gnp_random_graph.html

https://en.wikipedia.org/wiki/Erd%C5%91s%E2%80%93R%C3%A9nyi_model

Figure 1 presents an example of such graph.

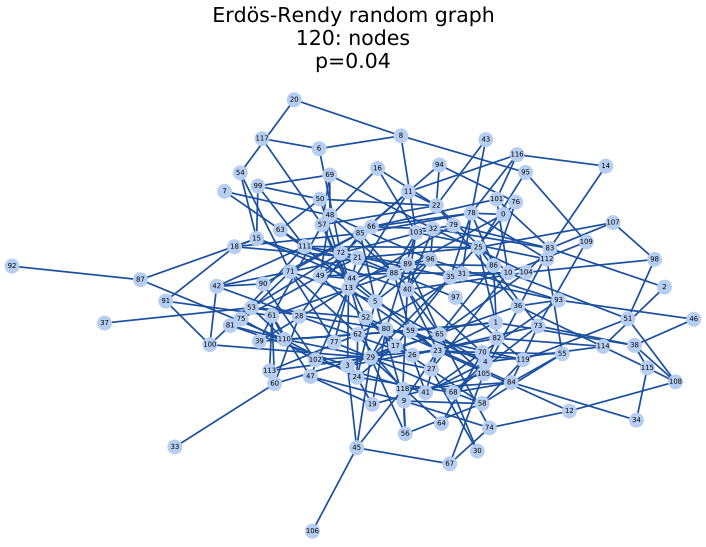

FIGURE 1 – Example 1 of a G_{np} graph.

To visualize the graphs, you may use and adapt the code from the repo.

2.3 Exercise

1) Pick **two** different degree-based heuristics of your choice and perform matchings on your generated graphs. Both heuristics must run in polynomial time. You can take inspiration from the document in the **documents/** folder in the repo.

2) Verify that your heuristics indeed return matchings by adding a test file **test_matching.py**. This file may use functions from the **networkx** library (or another library).

3) Compare the two heuristics statistically, in terms of :
— the size of the returned matching
— the measured computation time

Figure 2 presents the statistical result of the matching size obtained by applying the builtin method **maximal _matching.py** from **networkx**.

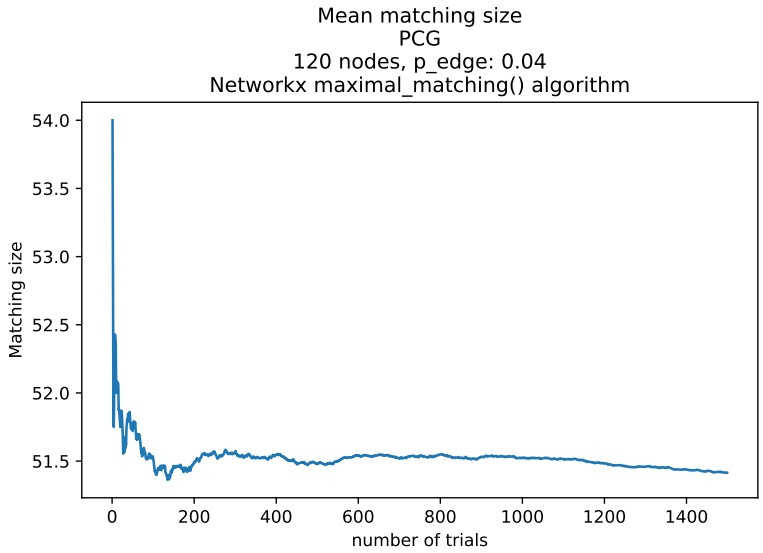

FIGURE 2 – Convergence of the size of the matching returned by the buitin method from networkx.

You should do enough trials to observe a convergence, like in figure 2. It is not mandatory that your heuristics perform better than this method !

4) Prove that both heuristics run in polynomial time (same instructions as in Exercise 1 for this proof).

3 PART 3 : COMPLEXITIES

For each of the two functions **function_1()** and **function_2()** that you can find in **./exercise _3 _functions.py** :

— compute an upper bound of their complexity (the smaller the better, it is better to have a $\mathcal{O}(n^2)$ bound than a $\mathcal{O}(n^4)$ bound), as a function of their argument n.

— verify the bound is correct by fitting a polynomial of the found degree to a curve representing the measured elapsed computation time as a function of n. For instance in figure 3, the function as a $\mathcal{O}(n^3)$ complexity, so a fitted polynomial of degree 2 can not represent well the curve. However, a polynomial of degree 3 can, as in figure 4.

You can use for instance **numpy.polyfit**

https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html

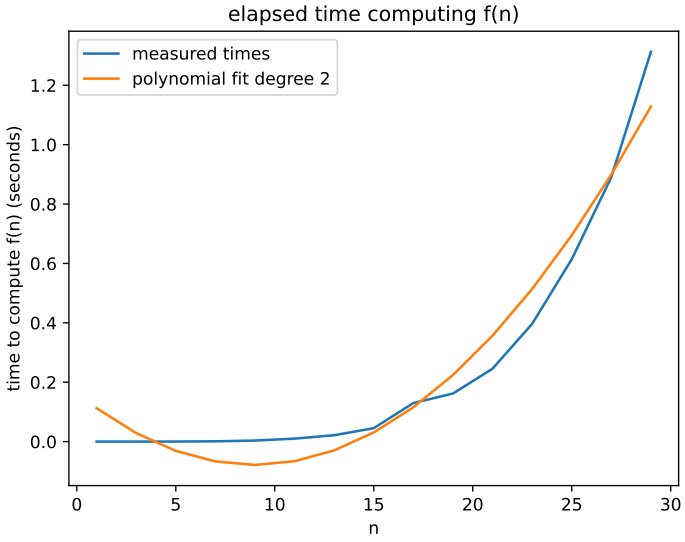

FIGURE 3 – Here, the degree of the polynomial used to fit the curve is too small.

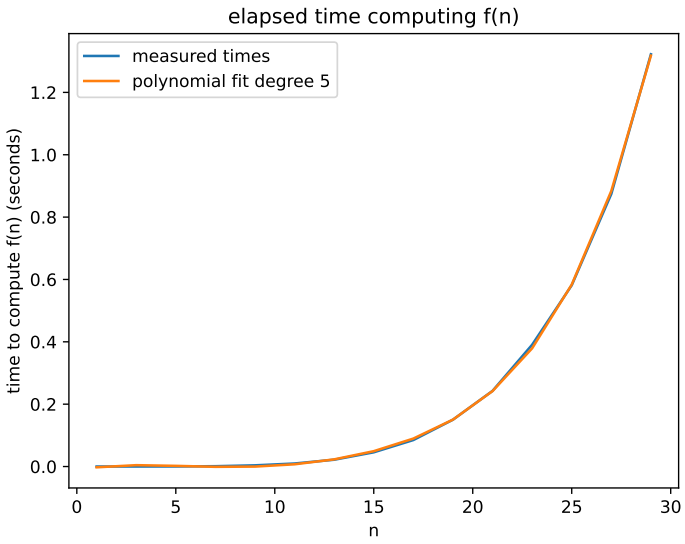

FIGURE 4 – Here, the degree of the polynomial used to fit the curve is correct.

4 PART 4 : LINE GRAPHS, HAMILTONIAN PATHS AND EULERIAN PATHS

In this exercise, you may use any method from networkx, the goal is to think a little bit more theoretically.

1) After doing some research, present with your own words the two following problems, and then the connection between them :

— existence of an eulerian path in a graph

— existence of a hamiltonian path in a graph

To do so, you will need to understand what the **line graph** (graphe adjoint) of a graph is.

2) Illustrate this connection by building small random graphs with networkx and finding eulerian or hamiltonian paths in the obtained graphs / line graphs.

Useful methods :

— `https://networkx.org/documentation/stable/reference/generators.html`

— `https://networkx.org/documentation/stable/reference/generated/networkx.generators.line.line_graph.html`

— `https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.tournament.hamiltonian_path.html`

— etc

3) Which problem should be considered as "harder" from an algorithmic point of view and in what sense ?

5 THIRD-PARTY LIBRARIES

You may use libraries such as networkx or graphviz, for instance for visualisations of the graph, but not for the algorithmic part that is the subject of the corresponding exercise, unless specified.

6 CODE STYLE

Add a short docstring at the top of each file and in functions, if relevant.
You are encouraged to use **type hints**.
https://docs.python.org/3/library/typing.html
http://mypy-lang.org/

7 ORGANISATION

Number of students per group : 3.

Deadline for submitting the project :
— 1st session (September 29th, 30th, October 1st) : October 30th.
— 2nd session (November 3rd, 4th, 5th) : December 4th.

The project should be shared through a github repo with contributions from all students. Please briefly indicate how work was divided between students (each student must have contributions to the repository).

Each exercise should be in its own folder.

If you used third-party libraries, please include a **requirements.txt** file in order to facilitate installations for my tests.

https://pip.pypa.io/en/stable/user_guide/#requirements-files

You can reach me by email if you have questions.