Assignment 2

Rubén Pérez Mercado

November 23, 2022

1 Introduction

In this assignment, we will solve the Exam Scheduling Problem by transforming it into a Graph Colouring Problem. We will discuss the running time of this problem in terms of the number of subjects.

1.1 Definitions

1.1.1 Exam Scheduling Problem

Given a list of subjects and students enrolled in each of them, determine a schedule for conducting their exams in the minimum number of days, bearing in mind that a student cannot sit more than one exam on the same day.

1.1.2 Graph Colouring Problem (GCP)

Given a graph, colour its vertices such that no two adjacent vertices are of the same colour.

1.2 Questions

1. *What does the chromatic number mean in this context?*
   The chromatic number is the minimum number of colours used in a graph to solve the GCP. If we think about the Exam Scheduling Problem, it could be seen as the number of days we need to do all the exams.

2. *What does the clique number mean in this context?*
   The size of the maximum clique that can be done with these nodes.

3. *What is the relation between both numbers?*
   The clique number and the chromatic number will be the same.

4. What is their relation to the maximum vertex degree?
   The maximum vertex degree means the maximum number of conflicts a subject could have. If the maximum vertex degree is high, the probability of the chromatic number (and therefore, the clique number) being a high number increases.

2 Methodology

For this assignment, we have created four different files. In Section 2.1 we will see the generator. In Section 2.2, the translator is explained. Section 2.3 shows the auxiliary class we created to represent our graph. Finally, Section 2.4 explains the solver function for both approaches (i.e. Receiving the nodes in any order and sorting the nodes by their vertex degree), along with the main function.

2.1 Generator

We have implemented our generator in "generator.hpp".

Our generator creates our instance of the Exam Scheduling Problems. Given an integer $n = \{5, 10, 15, 20, 25, 30\}$, it will create n subjects, and some students enrolled on those subjects. The global number of students is set to 500, and the number of students per subject is limited to a certain value (for this assignment, we have changed the conditions of the problem to avoid cliques, so the maximum number of students per subject is 30 instead of 250). In fact, the possible numbers of students for a subject are always a submultiple of the latter number. This is due to the fact that we generate a random number $k, 1 \leq k \leq n$, and with this number we calculate $p(k) = \lceil k/10 \rceil^{-1}$. For instance, the value of $p(k)$ for $n = 30$ could be 1, $\frac{1}{2}$ or $\frac{1}{3}$.

Once we know the number of students for a subject, we choose random students from the group of 500 to fill this subject. We store those students and the subject on a file whose name is given as an input.

2.2 Translator

We have implemented our translator in "translator.hpp".

The translator is going to create a graph colouring instance out of an exam scheduling instance (previously created by the generator), and is going to create a file to store this graph in DIMACS format. This format follows this convention:

- One problem line: *p edge <number of nodes> <number of edges>*
- One edge line per edge: *e <node> <node>*
- Optional comment line: *c text*. We have added a comment line at the beginning of the file to indicate the source (the exam scheduling instance related).

In this function, we are going to use a map, in which the key is a string (the name of the subject) and the value is a set (the students enrolled in a subject). We have used this structure because in the algorithm we are going to check for each student of a subject if he/she is enrolled in another subject. By using this map, we can check that efficiently.

If we identify a conflict between two subjects (i.e., a student enrolled in these two subjects), we create an edge.

2.3 Graph Class

We have implemented our graph class in "graph.hpp".

In order to make it easier for us to compute the GCP, we have created this auxiliary class to represent the graph. The graph is modelled as an adjacency matrix(a matrix of booleans. $matrix[i][j]$ represents whether nodes i and j are connected or not (with true or false values respectively).

This class has some interesting methods:

- Constructor: Creates a graph instance out of a file that contains a graph in DIMACS format. We are going to use the graphs created by the translator.
- Size: Returns the number of nodes of the graph.
- areConnected: Returns true if the nodes given as input are connected, otherwise it returns false.
- addEdge and removeEdge: Creates and removes edges of the graph respectively. We only needed the first method, and it is only used in the constructor.

For the second approach of this assignment, we needed to select the nodes by their vertex degree. We have implemented this in this class by using an auxiliary vector that is going to map the nth element with the most connections to the real vertex. This vector is sorted_vertex. This vector has the nodes sorted in descending order by their vertex degree. Each node is represented by a pair: the first value is the vertex degree and the second value is the node. The number of connections is necessary in order to sort the vertexes. Regarding this attribute, we have two more methods:

- sortedVertex: Given an integer n, returns the nth vertex with more connections of the graph.
- getSortedVertexes: Returns the nodes in descending order by their vertex degree. This method is going to be useful to know the actual solution after solving the problem with the second approach.

2.4 Solver

The code of both solver functions, the main function, and some auxiliary functions are in "solver.cpp".

In the solver function, we have implemented the algorithm to solve the GCP. We have made some modifications regarding the algorithm given in the guide of this assignment.

- In the original algorithm, C represents the list of colours available for each node. In our algorithm, C represents the number of conflicts on each colour for each node. With this modification, we provide more information, and we also know if a colour is available for a node (if the value of $C[node][colour] == 0$).
- We have added a variable vertex, which represents the selected vertex. At the beginning of the algorithm the selected vertex should be 0.
- We have two additional vectors S and A. S represents the best solution so far (and at the end of the algorithm will be the best solution), and A represents the current solution (i.e. the colours we have used in the nodes we have seen so far).
- In the original algorithm, the function no_colour_under_bound has to check if, for a given vertex, we have a colour available whose value is under the bound B. As we have changed C, now we have to check that there is a colour such that the number of conflicts with that vertex is zero.

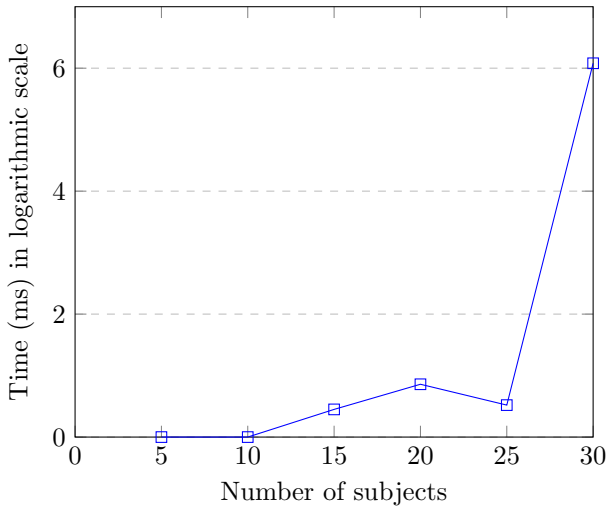

Figure 1: Logarithm of the execution time over the number of subjects when solving the GCP with "backGCP".

The first approach is computed in the function "backGCP", and the second approach is implemented in "backGCPv2". The implementation of the latter algorithm is the same as in the first one, but we are using the function *sortedVertex* of the class Graph. With this modification, we are selecting the nodes by the number of connections in descending order without changing the algorithm that much.

In the main function, we have a loop to test both algorithms. We are going to solve instances of the problem of length $n = \{5, 10, 15, 20, 25, 30\}$, and we are going to see the times that the algorithms take during their computation.

3 Results

3.1 First Approach (Nodes in any order)

As we can see in Figure 1, the time escalated quickly when the number of subjects increased (Note that the time is represented in a logarithmic scale). In fact, we were lucky, because we had really good times for $n = 25$ (3.34 seconds) and $n = 30$ (20 minutes and 40 seconds approximately), but sometimes our algorithm took more time to compute a solution when the number of subjects was greater than 20. This can be seen by executing our code a few times. In some cases, it will compute the algorithm immediately. In other cases, we had the algorithm running for hours and we could not find the optimal solution. We will discuss our guesses in the next section.

3.2 Second Approach (Vertexes sorted in descending order by vertex degree)

In Figure 2, we can see that the time execution over the number of subjects is very much like an exponential function. We stress that Figure 1 represents the time in a logarithmic scale, so the times in that figure a way bigger than the times we got using this approach. As we can see, we get the solutions in less than 0.1 seconds for $n = 30$. In some cases, the algorithm took some seconds to find the optimal solution, but if we compare these times with the ones we got in the first approach, they are smaller.

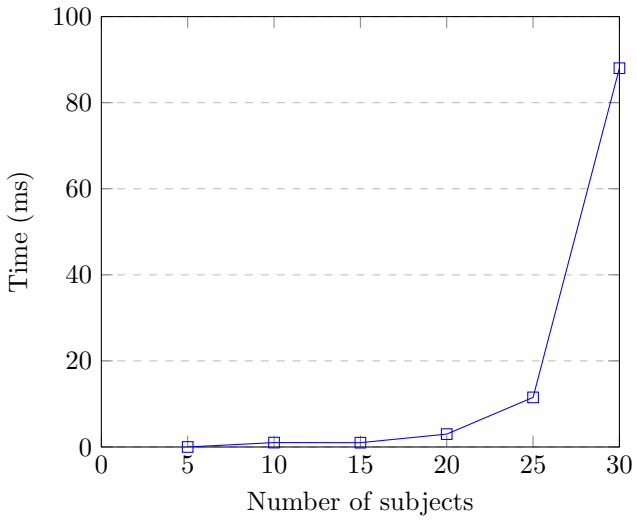

Figure 2: Execution time over the number of subjects when solving the GCP with "backGCPv2".

4 Conclusions

In this assignment, we solved the Exam Scheduling Problem by translating it into a Graph Colouring Problem. We have set the equivalences between the concepts needed in both problems so we could find an optimal solution for the first problem solving the second problem. However, the algorithm that solved the GCP usually needed a lot of time to compute a solution when we had a big number of subjects in the instance of our problem. By debugging the algorithm, I noticed that the algorithm returns a solution faster if it is able to find a solution that uses just a few colours at the beginning of the computation. By doing this, we updated the value of the variable B (the bound), so whenever we had to use more than B colours to complete a solution, we "stopped" looking for a solution with that configuration of colours. On the other hand, if we do not find a solution at the beginning of the computation that uses a small number of colours, the algorithm took more time. For $n = 30$, if we were not able to find a solution that used less than 9 colours approximately, they spend more time finding another possible solution than if the solution they found used just 7 colours.

Furthermore, we have to stress that we chose fewer students than in the guide of the assignment. If we had $250 * p(k)$ students instead of $30 * p(k)$ students, the graph was usually fully connected (or almost fully connected). In that case, as the clique number was high, the chromatic number was also high, and if we recall what we have said in the last paragraph, we would need even more time to compute a solution (if the algorithm needed hours to compute a solution which used 10 colours, if for another instance of the problem the optimal solution uses 25 colours we will need days or even weeks to compute that solution).

That said, when we sorted the vertexes in descending order by the vertex degree, we were able to compute faster a solution. The reason behind this is that the vertexes with fewer connections are less restrictive than the vertexes with a lot of connections, so if we set the colour of the latter vertexes first, we will find fast a solution that uses just a few colours.