0   Index

1 Overview

This document presents all the important information regarding the *Classroom management at the School of Computer Engineering using Artificial Intelligence methods* end-of-degree thesis.

It is important to note that the structure of the contents for this document is done following the criteria and recommendations of the template document for Degree's and Master's Thesis of the School of Computing Engineering of Oviedo (version 1.4) by Redondo [Red]. However, some additional chapters were introduced in order to capture the particularity of the work carried out, inspired by the research of de la Cruz [dlC18].

Introduction. Here we explain in a simple way the problem we want to solve, what reasons are behind the development of the project and give a description of the current situation of the School with regard to this and other similar problems. The scope and goals of the project are also discussed.

Theoretical aspects. The first chapter delving into the theory supporting the developed system. One example of an assignment problem is presented and solved using a greedy algorithm and a genetic algorithm, which helps to better internalize the concepts.

Problem definition. Here the formulation of the problem as an assignment problem is elaborated.

Proposed solution. This chapter lays down in detail the proposed solution to the previously defined assignment problem by means of a genetic and greedy algorithms.

Project planning and budget overview. For the planning, the Gantt chart of the project is shown, as well as the work breakdown structure. The internal and client budgets are presented, but not how they were calculated. That goes in the budget section.

Analysis. An analysis of the system, with the system requirements, draft diagrams, use cases and the test plan specification.

System design. The technical details of the system analysed in the previous section. Here we include the finalised diagrams, as well as the arquitecture of the system and the in-depth test plan.

System implementation. Details of the development of the software. The programming languages, standards and tools used to code the system, and all the relevant information gathered in the process of creating the system.

Test development. A rundown of all the testing done for the system, with explanations for every test and the obtained results.

Experimental results. The conclusions reached after experimenting with different input data and the optimal configuration for the genetic algorithm's parameters.

System manuals. All the manuals for the system, with screenshots and guided steps meant to help the target audience for each document.

Conclusions and future work. The conclusions after the implementation of the project are given, as well as a list of possible improvements and new functionalities for the prototype.

Budget. Here we give the full details for the elaboration of the internal and client budgets, with all the intermediate steps that led us to that result.

Annexes. The glossary of definitions and abbreviations and a small commentary on the submitted files.

Source code. The *javadoc* of the software developed.

2   Introduction

The School of Computing Engineering of the University of Oviedo has more than twenty classrooms, including theory classrooms and laboratory classrooms. Each semester there are over three hundred groups, each with their type (theory, seminar or laboratory), subject and schedule. The timetable of the groups varies on a weekly basis, this means that not all groups have to attend classes all weeks, and some of them do not even have repeating patterns.

This makes assigning classrooms to groups a complicated task, since there can be no temporal collisions. When various other constraints enter the equation, such as minimising the number of labs used by a subject or assigning classrooms to Spanish groups that are different from English groups, things become much more complex.

All this assignments are done *manually* by one person. The number of enrolled students can only be *guessed* when this process is done. This means that groups can be created, modified or removed once the semester has already started, so more assignments are usually made, checking once again all the restrictions. These new assignments are difficult to manage as there is not much room for flexibility to change those made before the semester. This is due to the fact that both students and teachers already use the initial assignments as a reference.

This project provides the supervisor of this process with a tool to help them calculating the assignments, reducing their workload. Not only does it generate assignments for all the groups of the semester, but can use previous assignments, total or partial, to calculate a subset of assignments (for example, the assignments for the new groups created in the middle of the semester). On top of that, the prototype developed in this thesis makes finding a set of free classrooms to hold events easy and fast, using the assignments generated previously by the system itself.

2.1 Situation overview

At the beginning of each semester, the School opens a process in which the person in charge takes the list of groups for the semester, their schedules and the list of classrooms, and performs a manual compilation of all the assignments.

There are a number of other similar procedures, like the creation of the exam timetable or the assignments of enrolled students to subject groups. However, some are not manual, but automated by a system, like the previously mentioned procedure of assigning students to groups. Seeing the potential of such tools, I was given the task of automating the assignment of classrooms to subject groups by similar means.

The procedure of assigning the classrooms is done after configuring the student groups for the semester and knowing their schedules. Even though it is a manual process, the supervisor does not start making the assignments from scratch. First, they have the knowledge of previous years, and then they have a list of preferences or premade assignments. For example, certain laboratories can only be assigned to specific groups, like the ones from the Electronic Technology of Computers subject. The system described in this document preserves these sources of information and builds on top of them.

2.2 Purpose

This project aims to help the personnel of the School manage their classrooms. It will address two main functionalities, the automation of the process of assigning classrooms to all the groups of a given semester (starting from scratch or using a previous partial or total assignment), and a tool that searches for gaps in a previous set of assignments for single or multi-day events in one or more classes.

The implementation of this system is intended to assist in the work of the supervisor for this process, and provide an efficient and flexible tool that expands the possibilities of such work. To do so, the program executes two algorithms, a genetic algorithm guided by a greedy algorithm. For a more detailed view on these algorithms the reader might refer to 3. Once the assignments have been calculated, the system will allow the users to find classrooms to hold specific events in the middle of the semester.

Along with the system, the system manuals are submitted. These have the purpose of teaching how to install, use, maintain and extend the system. Apart from the manuals, another tool to generate the necessary files for the program is handed.

2.3 Scope

The project needs to formally define the problem of assigning classrooms of the School to all the groups of the semester, conduct a study on the problem and propose a solution.

A development of a software prototype that solves the problem is planned, designed, implemented and tested. This prototype will solve the two main functionalities indicated in 2.2 and will consist of a command line application that takes input data in plain text files and outputs the solution to plain text files. The program is configured by different configuration files depending on the functionality being executed. An experimental study on the results of the software system is carried out, finding the most fitting default values for the configuration files. The project also contains the system manuals of the application, which consist of the installation, usage, user and programmer manuals.

Finally, an additional tool for automating the creation of the input files of the software is given. It uses a format agreed with the client and will use the same technical specifications of the main prototype, like the programming language and the development environment.

2.4 Project goals

We can identify from the scope the following objectives. They need to be met in order to close the project successfully:

1. Formally define the problem of assigning classrooms to the groups of the School.

2. Study the problem and the means to solve it.

3. Define the proposed solution.

4. Build a prototype that solves the problem using the algorithms described in the proposed solution.

   (a) It will receive plain text input files with the required data.

   (b) It will output the solution to plain text files.

Introduction

(c) It will be able to make the assignments starting from scratch or from a total or partial set of assignments.

(d) It will be able to search a set of free classrooms for a specific event in one or more days.

5. Make a set of experiments to find the best default values for the configuration files.

6. Write a set of manuals to cover the essentials of the system.

7. Create another software tool that will automate the creation of the input plain text files for the main system.

8. Validate solution with the users.

3   Theoretical aspects

A digital magazine Bootaku works with three freelancers. Dante, Virgil and Beatrice. Together they write a section about book reviews. Gathering data from previous sections, Bootaku wants to define and solve a problem of efficiently assign all reviews to the three critics so that the section gets the highest profit. For the assignments, Bootaku wants every book review to have one (and only one) associated freelancer. If a freelancer ends up with no reviews, the assignments are still valid if and only if the previous condition is met.

3.1 Assignment problem

The problem described before is an example of an assignment problem. It can be generalised with the following elements:

A set of n freelancers f

A set of m book reviews r

An assignment matrix of $n \times m$ assignments a_{fr} such that $a_{fr} = 0$ when freelancer f is not assigned to book review r and $a_{fr} = 1$ when freelancer f is assigned to book review r.

A profit matrix of $n \times m$ profits p_{fr} which indicate the profit obtained when assigning freelancer f to book review r and that $p_{fr} > 0$.

A valid solution is defined as a matrix of assignments where all the book reviews have a freelancer assigned to them and no book review has more than one associated freelancer.

The profit for all the assignments will then be:

$$\sum_{f=1}^{n} \sum_{r=1}^{m} a_{fr} p_{fr} \tag{3.1}$$

The optimal solution consists on having a set of assignments such that the sum of all the profits for the current assignments is maximised.

For example, imagine that for the next month's section, we have the following data. The information is represented by means of two sets: F for the freelancers and R for the reviews.

$$F = \{Dante, Virgil, Beatrice\} \tag{3.2}$$

$$R = \{DivinaCommedia, ElQuijote, VoyageAuBoutDeLaNuit, TodoModo\} \tag{3.3}$$

Then, our assignments and profits will be represented by the A and P matrices.

$$A = \begin{array}{c} Dante \\ Virgil \\ Beatrice \end{array} \begin{matrix} DC & EQ & VN & TM \end{matrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix} \tag{3.4}$$

$$P = \begin{array}{c} Dante \\ Virgil \\ Beatrice \end{array} \begin{matrix} DC & EQ & VN & TM \end{matrix} \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{pmatrix} \tag{3.5}$$

Where each row represents a freelancer and each column represents a book review. So freelancer 1 is Dante, 2 is Virgil and 3 is Beatrice. The same goes for the book reviews. Book review 1 is *Divina Commedia*, 2 is *El Quijote*, 3 is *Voyage Au Bout De La Nuit* and 4 is *Todo Modo*.

Now, we are going to study valid and non-valid solutions. As we explained before, a solution is valid if every book review has a freelancer assigned to it, and no more than one.

We will analyse four sets of values for the A matrix:

$$A1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \tag{3.6}$$

$$A2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \tag{3.7}$$

$$A3 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.8}$$

$$A4 = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \tag{3.9}$$

From these matrices, we can deduce that $A1$ and $A2$ are valid solutions, because they have one freelancer for each book review. We are not concerned with a freelancer having no book reviews assigned. However, a book review without an associated freelancer represents a non-valid solution. That is precisely the case for $A3$, the book review for *El Quijote* has not an assigned freelancer. In the case of $A4$, the fact that *El Quijote* has two freelancers assigned makes it a non-valid solution.

Now, we will give values to the P matrix in order to discuss possible optimal solutions. We will compare them with the assignment matrices $A1$ and $A2$

$$P1 = \begin{bmatrix} 9 & 1 & 5 & 4 \\ 2 & 8 & 14 & 2 \\ 7 & 11 & 10 & 6 \end{bmatrix} \tag{3.10}$$

$$P2 = \begin{bmatrix} 9 & 1 & 5 & 4 \\ 13 & 8 & 14 & 2 \\ 7 & 11 & 10 & 6 \end{bmatrix} \tag{3.11}$$

$P1$ and $A1$:

Theoretical Aspects

$$Profit = \sum_{f=1}^{n} \sum_{r=1}^{m} a_{fr} p_{fr} = 9 + 11 + 14 + 6 = 40 \qquad (3.12)$$

$P1$ and $A2$:

$$Profit = \sum_{f=1}^{n} \sum_{r=1}^{m} a_{fr} p_{fr} = 2 + 11 + 14 + 6 = 33 \qquad (3.13)$$

We can observe that for $P1$, the assignments defined in $A1$ are better than those in $A2$, because they result in a better profit. Another important remark about $A1$ is that it is the optimal solution to the problem, because it assigns the book reviews to the freelancers with the best profit value for their assigned books. Now $P2$ will be evaluated.

$P2$ and $A1$:

$$Profit = \sum_{f=1}^{n} \sum_{r=1}^{m} a_{fr} p_{fr} = 9 + 11 + 14 + 6 = 40 \qquad (3.14)$$

$P2$ and $A2$:

$$Profit = \sum_{f=1}^{n} \sum_{r=1}^{m} a_{fr} p_{fr} = 13 + 11 + 14 + 6 = 44 \qquad (3.15)$$

In the case of the profits values of $P2$, the situation is reversed. $A2$ is now the optimal solution and therefore better than $A1$.

The important thing to notice here is that the values for the P matrix right now may appear as having no meaning whatsoever. But we need to think of P as the results obtained from a profit funtion. Then, we can interpret $P1$ as values of profit in a context where Virgil has just expressed an opinion on social media about the *Divina Commedia* and caused a massive controversy. We can then say that $P1$ is a function which gives more importance to public relations and so the profit p_{21} is very low, whereas $P2$ gives more importance to views and so the profit p_{21} is higher. Of course, in a real problem you know what the function is calculating, but this shows how we can add meaning to a set of symbols in order to understand the data more

efficiently.

With this, we have discussed an assignment problem, looked at its main components and analysed its non-valid, valid and optimal solutions. Some final remarks about the relation between a general assignment problem and the Bootaku problem follow. The actors that perform the jobs, in this case the freelancers that *write* the reviews, are called the *agents*. The *tasks* to be performed are, in the Bootaku problem, the book reviews. Nevertheless, the agents in an assignment problem do not need to be persons (or even things that carry out actions), they can be machines, warehouses, or classrooms. The same can be said for the tasks.

3.2 Heuristics and greedy algorithms

One way of solving the Bootaku problem described earlier can be found in *search algorithms*.

3.3 Metaheuristics and genetic algorithms

4 Problem definition

The School of Computing Engineering of Oviedo must find a classroom for each group of a given semester. In most cases of this particular problem, just like in the Bootaku situation described in 3, there are more *tasks* (groups/book reviews) than *agents* (classrooms/freelancers). And, in the same way, a valid solution implies that all groups have one (and only one) classroom assigned.

The data for the classrooms and groups can be represented by two sets C and G.

$$C = \{c_1, c_2, ..., c_n\} \tag{4.1}$$

$$G = \{g_1, g_2, ..., g_m\} \tag{4.2}$$

Where C is the set of n elements representing all the classrooms of the School, and G a set of m elements representing the groups for a given semester.

A classroom can be a laboratory or a theory classroom. Groups, on the other hand, can be taught in English or Spanish, and have three types. Laboratory, theory and seminar groups. In this problem, because we are only interested in the classrooms that can be assigned to groups, we only consider two types. Laboratory and theory. That is, we consider that the types of classrooms are *identical* to the types of groups.

$$T = \{t_1, t_2, ..., t_p\} \tag{4.3}$$

$$L = \{l_1, l_2, ..., l_q\} \tag{4.4}$$

Therefore, each classroom c and group g have a type t. In the case of groups, they also are taught in language l.

Each group has a set of academic weeks and of group schedules. A group can attend classes weekly, every two weeks or on a non-trivial pattern, and may be taught on one or several days.

$$W_i = \{w_{i1}, w_{i2}, ..., w_{ir}\} \tag{4.5}$$

$$H_i = \{h_{i1}, h_{i2}, ..., h_{is}\} \tag{4.6}$$

Therefore, every group i has a set of weeks W_i and a set of schedules H_i. A schedule consists of a triplet in the form $(DayOfTheWeek, start(hh : mm), finish(hh : mm)$.

Every group belongs to a subject.

$$S = \{s_1, s_2, ..., s_t\} \tag{4.7}$$

A subject s is related to a subset of G groups.

With these, we have defined all the data which we need in order to solve the problem. Now we will talk about the problem constraints that we have to fulfill.

For the constraints, we call *hard constraints* those which are imperative for the solution to be valid, and *soft constraints* the ones that reflect on the overall quality of the solution but are not mandatory.

Before introducing the constraints, two new concepts are presented. Restrictions and preferences.

$$R_i = \{r_{i1}, r_{i2}, ..., r_{iu}\} \tag{4.8}$$

$$P_i = \{p_{i1}, p_{i2}, ..., p_{iv}\} \tag{4.9}$$

Restrictions and preferences can be positive or negative. A group i must be assigned to a classroom in the set of its positive restrictions, and cannot be assigned to a classroom in the set of its negative restrictions. It is prefered that the group i is assigned to a classroom in the set of its positive preferences, and preferably not in

the set of its negative preferences. With that in mind, the constraints are listed next.

Hard constraints:

Laboratory groups can only be assigned to laboratories.

Theory and seminar groups can only be assigned to theory classrooms.

A group cannot be assigned to a classroom whose capacity is less than the number of students in the group.

A group with a set of positive restrictions must be assigned to one of those classrooms.

A group with a set of negative restrictions cannot be assigned to one of those classrooms.

A group cannot be assigned to a classroom if that classroom was already assigned to another group and both groups collide (they overlap in week and schedule).

Soft constraints:

Laboratory groups of the same subject must all attend the same laboratory classroom, and if not possible, at least minimise the number of laboratories assigned to them.

Theory groups of the same name and course work in the same way, but being assigned to theory classrooms [1].

English and Spanish groups should go to different classrooms.

Every hour a number of laboratories must be empty. To cover for emergencies.

A group with a set of positive preferences should be assigned to one of those classrooms.

A group with a set of negative preferences should not be assigned to one of those classrooms.

---

[1]All the groups in the School have the format *subject.type.name*. For example the group *Com.T.1* refers to *theory* group *1* of the *Computability* subject. So all theory groups named 1 would be assigned to the same theory classroom, if possible.

Problem definition

5 Proposed solution

5.1 Search space

5.1.1 Assignments

An assignment is a tuple which associates a group with a classroom.

$$(G_i, C_j) \tag{5.1}$$

Because a group can only have *one* classroom assigned, an assignment can be identified by the *code* [1] of the group. For example, the assignment for group SI.T.1 can be identified by the code SI.T.1 as well.

Assigning just *one* classroom to each group means that the total number of assignments is calculated by the following expression.

$$TotalNumberOfAssignments = |G| \tag{5.2}$$

This implies that there are as many assignments as the number of groups for the semester.

5.1.2 Solutions

A *solution* for this problem is represented by a set of all the assignments must be performed for the semester. As presented in the previous section, the total number of assignments equals the total number of groups in that semester. So we have the next statement.

$$Solution = \{(G_1, C_x), (G_2, C_y), ..., (G_m, C_z))\} \tag{5.3}$$

Where m is the total number of groups and x, y and z are the index for the class-

---

[1]The name convention previously mentioned: *subject.type.name* (e.g. Com.T.1).

rooms assigned to the groups. Note that the classrooms are not sequential (e.g x could represent C_{12} and y represent C_3).

An *empty solution* is represented by a set of all the assignments where each assignment is *incomplete*. We mean that an assignment is incomplete when the group has no classroom assigned.

$$IncompleteAssignment = (G_i, -) \tag{5.4}$$

So, for the empty solution, we have a set with the following format.

$$EmptySolution = \{(G_1, -), (G_2, -), ..., (G_m, -))\} \tag{5.5}$$

Finally, a *partial solution* is one in which not every assignment was performed, and a *complete solution* is defined by a set in which all the assignments have been performed and each group has a classroom associated with it.

5.1.3 States

A *state* represents a phase in the problem. There can be three states. The *initial state*, which stands for the empty solution of non allocated assignments. The *final state*, which represents a complete solution with all the assignments performed. And the *intermediate states* portraying partial solutions.

A key concept to understand our solution is the following. Although by default we start the execution of the algorithms with an initial state, it is also possible to start the execution with an intermediate state. This is because we can receive as input a partial or total solution of assignments and work from there. Now we will discuss how we can jump from one state to the next, which is normally called *state expansion*.

To expand a state, one of the non performed assignments in the solution is executed. This means that every time a classroom is assigned to a group the state is being expanded. To perform an assignment, the number of possible candidates is the same as the number of classrooms. So we have the following.

$$TotalNumberOfCandidates = |C| \qquad (5.6)$$

Nevertheless, as there exist constraints that indicate wether or not the solution is valid, there are filters which reduce the number of available classrooms for a group. This allows for optimized and easy to retrieve calculations in the execution of the greedy algorithm (this will be explained later in 5.3). The important thing to note at this moment is that, because of these filters, not all states can be expanded.

5.1.4 Instances

The complexity of the calculations and completion time depend on many factors. Some of those factors follow. First, the number of groups for the semester, which directly translates to the number of assignments to be made. Second, the number of classrooms. If there are more classrooms, it is easier to avoid collisions. Obviously, the number of groups is much more volatile between semesters than the number of classes, which is likely to change very occasionally, if at all. Lastly, the case of starting the prototype with an intermediate state. This means that the set of assignments represents a partial solution given as input. The number of calculations decreases in direct proportion to the assignments already made, therefore the completion time would be lower.

Because of the constraints of the problem, there are some groups where class allocations are more straightforward. For example, a group with only one positive restriction is either going to have that classroom assigned to it or, if it collides with other group, end up unallocated. This is why all the groups that just have one available classroom are assigned first. Also, the groups which have less students have more available classrooms than those with a large number of members.

5.2 Collisions

A collision is an overlap of the timetable of two different groups. For a collision to occur, the groups must clash at least once in the same week, day and time. Collisions are an essential part of this problem, as we cannot assign a classroom to a group if another group was previously associated with that classroom and both groups collide.

5.2.1 Lazy Collision Matrix

Due to the large number of assignments that have to be made throughout the execution cycle of the genetic algorithm, the chosen data structures were properly analysed. This is where the *Lazy Collision Matrix* comes in.

Imagine that we have the following group set.

$$G = \{G_1, G_2, G_3\} \tag{5.7}$$

Then, our initial Lazy Collision Matrix would be represented by the expression below.

$$LCM = \begin{array}{c} \\ G_1 \\ G_2 \\ G_3 \end{array} \overset{\begin{array}{ccc} G_1 & G_2 & G_3 \end{array}}{\begin{pmatrix} & -1 & -1 \\ -1 & & -1 \\ -1 & -1 & \end{pmatrix}} \tag{5.8}$$

First of all, the diagonal is empty because we never compare one group against itself. Then, we can observe that the rest of values are defaulted to -1. Why? Because there are *not yet evaluated*. That is the reason behind the name of the matrix. It is *lazy* because the collisions are only calculated when needed.

Continuing with this example, imagine that we assign classroom C_x to group G_1. Then, G_2 also tries to have C_x assigned to it, so we check if both groups collide. We find out that they do, so we update the matrix.

$$LCM = \begin{array}{c} \\ G_1 \\ G_2 \\ G_3 \end{array} \overset{\begin{array}{ccc} G_1 & G_2 & G_3 \end{array}}{\begin{pmatrix} & 1 & -1 \\ 1 & & -1 \\ -1 & -1 & \end{pmatrix}} \tag{5.9}$$

Therefore, the values are updated with a 1, which indicates that both groups *collide*. This results in a different classroom C_y being assigned to G_2. Now, G_3 has that classroom also available, so we check if it clashes with G_2. We learn that they do not

Proposed solution

collide. We update the matrix again.

$$LCM = \begin{array}{c} \\ G_1 \\ G_2 \\ G_3 \end{array} \begin{array}{c} \begin{array}{ccc} G_1 & G_2 & G_3 \end{array} \\ \begin{pmatrix} & 1 & -1 \\ 1 & & 0 \\ -1 & 0 & \end{pmatrix} \end{array} \qquad (5.10)$$

The value for non-collision is 0, as observed. Because G_3 does not clash with G_2, they are both allocated in the same classroom.

We can now generalise the LCM as in the next expression.

$$LCM = \begin{array}{c} \\ G_1 \\ G_2 \\ G_3 \end{array} \begin{array}{c} \begin{array}{ccc} G_1 & G_2 & G_3 \end{array} \\ \begin{pmatrix} & g_{12} & g_{13} \\ g_{21} & & g_{23} \\ g_{31} & g_{32} & \end{pmatrix} \end{array} \qquad (5.11)$$

Where a value g_{ij} can be

$$\begin{cases} 1, & \text{if } G_i \text{ collides with } G_j \\ 0, & \text{if } G_i \text{ does not collide with } G_j \\ -1, & \text{if the collision has not yet been evaluated} \\ \epsilon, & \text{otherwise} \end{cases}$$

The main advantage of this design is that we do not have to calculate all collisions. For example, a collision between a laboratory group and a theory group would be pointless to calculate because they would never be allocated in the same classroom. Therefore we alleviate the number of calculations.

As there is only one Lazy Collision Matrix, all calculations performed by the greedy algorithm across all populations in all generations are stored in just one place. This means that all collisions are being calculated only when necessary and only once. Think of the previous example. In a future iteration the greedy algorithm wants to check if groups G_1 and G_2 collide. It access the corresponding location in the LCM,

Proposed solution

and because it contains a 1, the greedy concludes that they indeed collide. This is done with a $O(1)$ complexity, as the matrix is coded as a dictionary of dictionaries. If instead the greedy wanted to check if groups G_1 and G_3 collided, because the LCM has a -1 in that position, the greedy would have to perform the collision check and then update the matrix.

5.3 Classroom filters

A classroom filter is a function. It receives as input either the C set or a subset $I \subset C$, and outputs a new subset $F \subset C$ with the available classrooms for a given group. It filters out the classrooms that do not comply with the hard constraints for that particular group (except collisions, which are calculated later with the LCM, see 5.2.1). All groups have the *same* classroom filters.

For example, let us consider a group G_i with type T_j and x students. A type filter for G_i would remove from the result set all the classrooms with a type different from T_j. A capacity filter would then take the set resulting from the previous type filter and use it as input. Then it would eliminate from the set all classrooms with a capacity lower than x and return the new F set.

This reduces the number of classrooms that the greedy has to evaluate and therefore decreases the complexity of the calculations. Furthermore, the filters are deterministic, that is, the execution of the filters of a group will always give the same results. This means that, if needed, the filters are only performed once per group per execution.

This may lead us to this question. *If all classroom filters are executed only once per group every time you run the prototype, why bother using a lazy approach for storing them? Would it not be better to perform and store them in a dictionary at the start of the execution?* The answer is no. It is true that if we start from an empty solution the lazy approach does not present a big advantage. Nonetheless, in the case of partial solutions, it reduces the number of calculations. For example, if we want to assign a classroom to a new group created in the middle of the semester, the only thing we care about is if the group collides with any other group. The filters for the rest of the groups are, in that situation, irrelevant. This is due to the fact that they have already been allocated to their corresponding classrooms.

5.3.1 Lazy Filter Dictionary

The filters work in a similar way to the Lazy Collision Matrix. Again, the election of the data structures is crucial to optimise the execution times. That is why the classroom filters are coded using a dictionary of sets. Once more, we will explain this with an example.

We have a M set of n classroom filters, three groups G_1, G_2 and G_3, and two classrooms C_1 and C_2.

$$M = \{M_1, M_2, ..., M_n\} \tag{5.12}$$

$$G = \{G_1, G_2, G_3\} \tag{5.13}$$

$$C = \{C_1, C_2\} \tag{5.14}$$

Then, we define the dictionary as a function $Dict : Keys \rightarrow Values \cup \{\epsilon\}$ where ϵ is the $null$ character. That is, $\epsilon \notin Values$.

The keys are represented by the groups, so $Keys \equiv G$. The values depict the different sets of available classrooms for each group. Because the dictionary is as lazy as the LCM, the calculations are performed as needed, so the initial set of values are by default ϵ. Then we can say that $Values = \{\epsilon, \epsilon, \epsilon\}$.

Accordingly, we have the following cases.

$$Dict(x) = \begin{cases} \epsilon, & \text{if} x = G_1 \\ \epsilon, & \text{if} x = G_2 \\ \epsilon, & \text{if} x = G_3 \\ \epsilon, & \text{otherwise} \end{cases}$$

We are now in the first execution of the greedy algorithm. Because we start from an empty state and not from an intermediate step, the greedy will try to assign a classroom for all groups. As a result of the values in the dictionary being $null$, the greedy knows that it must execute the filters, updating the dictionary. This is the LFD

Proposed solution

after the first execution for this case.

$$Dict(x) = \begin{cases} F_1, & \text{if } x = G_1 \\ F_2, & \text{if } x = G_2 \\ F_3, & \text{if } x = G_3 \\ \epsilon, & \text{otherwise} \end{cases}$$

With each $F_i \subset C$ being the filtered classrooms for each group. Example of values for the F sets follow.

$$F_1 = \{C_1\} \tag{5.15}$$
$$F_2 = \{C_1, C_2\} \tag{5.16}$$
$$F_3 = \{\} \tag{5.17}$$

We can observe that the for G_1, classroom C_2 was filtered out. In the case of G_2, both classrooms passed al the constraints. And for G_3, there are no classes available. This is very important, because it implies that with these filters, G_3 will *always* end up without a classroom. Therefore, a complete solution cannot be found for this case unless the filters are changed.

Lastly, a final remark. We can say that from now on, until the prototype terminates, the greedy will not have to perform the filters for any of the groups again, as the results are already stored in the dictionary.

Proposed solution   25

5.4 Greedy algorithm

5.4.1 Preprocessing

5.4.2 Heuristic

5.4.3 Repairs

5.5 Genetic algorithm

5.5.1 Fitness function

5.5.2 Operators

5.5.2.1 Selection

5.5.2.2 Crossover

5.5.2.3 Mutation

5.5.2.4 Tournament

5.5.3 Parameters

6 Project planning and budget overview

6.1 Planning

6.2 Budget summary

7 Analysis

7.1 System definition

7.2 System requirements

TODO: THESE ARE THE INITIAL REQUIREMENTS, THEY WILL CHANGE

The requirements listed here are a basic overview of the fundamental functionality covered by the project. For the complete list of in-depth requirements the reader might refer to NOPE.

7.2.1 Interface

- The program must implement a CLI.

  - The CLI must show basic or complete information to the user depending on the given option flag.
  - The CLI must show the encountered errors to the user before terminating the execution.
  - The CLI must have help, license and version options.

7.2.2 Input

- The progam receives as input the classrooms, groups, group schedule and the academic weeks of each group.

- The program might optionally receive as input a subset of assignments already performed.

- The program might optionally receive as input a previous complete list of assignments but without some of the classrooms/laboratories used in it.

- The program might optionally receive as input a previous complete list of assignments but with more or less groups.

- The program might optionally recieve as input a list of classroom preferences for the groups of a particular subject, given their type (theory or laboratory) and language (english or spanish).

7.2.3 Configuration

- Program configuration must allow the user to control the parameters of the genetic algorithm.

- Program configuration must allow the user to change the version of the program.

- Program configuration must allow the user to specify the folder paths for the log and output files.

- Program configuration can change in the middle of the course.

7.2.4 Algorithm

- The program must use a genetic algorithm guided by a greedy algorithm.

- Language group requirements:

  - English groups should go to different classrooms/laboratories from the spanish groups.

- Classroom requirements:

  - Some initial classroom assignments can be specified before the execution of the program and they must remain the same.

  - The program must be able to find a gap in the current list of assignments to include a (mono/multi)-(classroom/laboratory).

  - The number of groups of the same number and course assigned to the same theory classroom must be maximised.

  - The number of groups of the same subject assigned to the same laboratory must be maximised.

  - In each time slot there must be a minimum number of free laboratories.

  - Some big laboratories must be empty for emergency reasons.

  - The program must penalise assignments where the number of students is far below the number of computers.

  - The laboratories must have some free space defined by the user.

Analysis 29

– In small laboratories (of 16 computers) there must be at least two free computers.

– The program must be able to handle a split in two of a laboratory group with only one professor (for emergencies).

7.3 Subsystem mapping

7.4 Preliminary class diagram

7.5 Analysis of use cases

7.6 Analysis of user interfaces

7.7 Test plan specification

8 System design

8.1 System architecture

8.2 Class design

8.3 Interaction and state diagrams

8.4 Activity diagram

8.5 Interface design

8.6 Technical specification of the test plan

9 System implementation

9.1 Standards and references

9.1.1 Standards

9.1.2 Licenses

The software of this project is licensed under the GNU General Public License v2.0.

9.1.3 Other references

Java Code Conventions. Set of guidelines and conventions for programmers to consider when using the Java programming language.

9.2 Programming languages

There were two programming languages considered for the implementation of the system, **C** and **Java**.

Considering that:

- The author and only developer of the system has worked with Java throughout his university studies, but only used C in one subject and in some of his personal projects.

- Java is probably less efficient than C when executing the genetic and greedy algorithms.

- Java code is more easy to run in other systems than C code.

- The program is going to be executed only a few times a year.

For this reasons, even if C would be faster in execution, because the program will not be running every day, and taking into account the other two advantages, Java was the language of choice for implementing the system.

9.3 Tools and programs used in development

9.4 System development

10   Test development

10.1 Unit tests

10.2 Integration and system tests

10.3 Usability and accessibility tests

10.4 Performance tests

11 Experimental results

12 System manuals

12.1 Installation manual

12.2 Execution manual

12.3 User manual

12.4 Programmer manual

13 Conclusions and future work

13.1 Final conclusions

13.2 Future work

14   Budget

14.1 Internal budget

14.2 Client budget

15 Annexes

15.1 Definitions and abbreviations

Listed below is a glossary of definitions and abbreviations used in the document whose meaning may not be obvious.

Glossary of definitions:

- **Genetic algorithm:** metaheuristic search and optimization algorithm.

- **Greedy algorithm:** algorithm that builds the solution in successive steps, always trying to take the optimal solution for each step

- **Heuristic:** function that gives value to each path in a search algorithm, based on current information.

- **Java:** general-purpose, high-level, object-oriented programming language.

- **Metaheuristic:** high-level heuristic that guides the search in a combinatorial optimization problem.

Glossary of abbreviations:

- **CSV:** Comma-Separated Values. Refers to a text file format.

- **CLI:** Command Line Interface.

- **GNU:** GNU is not Unix (recursive acronym). Refers to the free software project announced by Richard Stallman.

- **TXT:** Text. Refers to the text file format.

- **UNE:** in spanish, *Una Norma Española*. Refers to the Spanish Association for Standardisation.

15.2 Submission contents

16 Source code

Bibliography

[dlC18] Gonzalo de la Cruz. Metaheuristics for the assignment of students to class groups. End-of-degree thesis, School of Computing Engineering of Oviedo, 2018.

[Red] Jose Manuel Redondo. *Documentos-modelo para Trabajos de Fin de Grado/Master de la Escuela de Informática de Oviedo*. Escuela Universitaria de Ingeniería Técnica en Informática de Oviedo, 1.4 edition.