Universidad Internacional de Valencia

Virtual and Face-to-face Course Timetabling using multiobjective Genetic Algorithms based on Dynamic Gene Spaces

Titulación:
Máster Universitario en Inteligencia Artificial
Curso académico 2021-2022

Alumno/a: Vélez Falconí Martín
D.N.I: 1724556681

Director/a de TFM: Diego Hernán Peluffo-Ordóñez

Convocatoria:

Tercera

09 Mayo 2022

De:
Planeta Formación y Universidades

Do your duty today and repent tomorrow

Mark Twain

Acknowledgement

My sincerely acknowledgement to the University Metropolitan for the access and help present to achieve this work.

Contents

List of Figures

List of Tables

List of algorithms

Abstract

The course schedule problem is a frequent problem encountered in educational institutions. Since the problem belongs to the NP-Hard family, this problem remains a case study. Many papers propose various solutions for different subcategories of problems. There are some popular solutions for the course schedule problem in college. However, the spread of the SARS-CoV-2 virus drastically changes classical education. University education at the beginning of the crisis started as full distance education. Besides, the mass vaccination phases around reduce the restriction of social distance, causing some universities to return to normal or combine distance education with face-to-face education. Therefore, the classical timetable structure is sometimes different from that of the face-to-face course.

The literature reports a few case studies of virtual timetables. The cases included in the literature are hard-constrained courses as they are connected to multiple entities. In addition, the problem model is novel and most of the cases are different from each other. This manuscript explores the virtual and face-to-face course scheduling problem in a specific university case.

The case study university is assigned to multiple geographical locations around the country called Ecuador. This university in the pre-post covid program has its curriculum divided independently around its locations. However, the current conditions of the university decided to combine the courses from various locations into one distance learning course. This presents a problem as their internal differences in their schedule and time allocation depend on the availability of the course per location. In addition, the course is also related to several careers. This causes several overlaps in its calendar and organizational problems.

The manuscript proposes a solution based on a variation of the Genetic Algorithm to minimize the schedule of internal university policy violations. The proposed algorithm differs from other Genetic Algorithms. Since it dynamically constrains the possible gene values using prior information from the other genes. Since the university is starting to combine virtual and face-to-face courses, the proposed solution also includes a multiphase Genetic Algorithm to solve this task.

The internal policies are either flexible or strict depending on the type of course. The strict policy is an arbitrary case that will never occur and is represented as a hard constraint. On the

other hand, flexible policies are events that must be reduced, and represent the soft constraint. The internal policy takes precedence over the others, so the genetic algorithm includes multi-objective functions. The metrics evaluated in this article were based on the measurement requested by the university.

After data analysis and hyperparameter selection, the genetic algorithm was tested. The results show that the proposed dynamic space algorithm could be a more e□cient solution compared to a normal genetic algorithm which in the experiments could not solve the hard constraints.

Introduction 1

The problem of setting the best fit for certain resources (classroom, professors, and courses) in a finite set time is known as educational timetabling problems. These types of problems are well-known that in most cases they belong to NP complexity, (Even et al., 1975). The literature often explores three types of model problems: University Course Timetabling (CTT), Examination Timetabling (ETT), and School Course Timetabling (HTT) (Esquivel y Lucía, 2014). However, there are other types of model problems that are used to solve specific solutions, such as thesis defense timetabling, energy consumption optimization for classrooms, and others. Due to the pandemic crisis, new timetabling problem models appeared based on virtual education. This kind of problem model is fully constrained are characterized as it does not depend on the classroom. However, it is a novel topic that has a lack of literature and explores specific case scenarios.

An example is the case study, the Ecuadorian university named Metropolitan University of Ecuador (UMET) has multiple locations throughout the country. This university is a complex case since recently it is implementing two course-modality types. It includes face-to-face and virtual courses. The face-to-face course is the same problem as the timeslot assignment, but the virtual course represents a challenging task. Since the virtual course timetabling must consider the relation around the multiple locations. Therefore, the timeslot is fully constrained by the professor, career level, career calendar, and the day.

This Master's thesis explores a solution for the UMET that has both types of virtual and face-to-face course modalities. In addition, it has complex restrictions that must be met or reduced depending on the policy.

Based on an exhaustible literature review and strict analysis of the problem and constraint, the proposed solution in this work is based on a two-phase timetabling using a genetic algorithm (GA). The first phase solves a new problem model-based curriculum course virtual problem in multiple locations and the second is a classic CCT with a shared course in di□erent careers. The genetic algorithm used in this work is of type multi objective that uses a dynamic calculation of a gene space which e□ciently solves the hard constraint and reduces the number of fails for the soft constraint. Some classical techniques -being out of the scope of this study- are explained in Dorado-Sevilla et al. (2021).

This manuscript is divided into 6 chapters, the first one is introduction 1 which describes the problem and the literature review. Next Chapter 2 is the objectives. Followed by Chapter 3 that contains the methodology, and explains the dynamic gene space method in the GA. Chapter 4 shows metrics, evaluating hyper-parameter, results, and discussion. Finally, Chapter 6 contains the limitations and

future work.

1.1 Key terms and brief overview

1.1.1 Key terms

1.1.1.1 Time-Slots

A time slot is defined as a set of consecutive hours in a specific day.

1.1.1.2 Course

A course is a composition of time-slots, a professor, classroom (if it has a face-to-face component). The course has a total of class hours per component. The total number of hours must be distributed on di□erent days.

1.1.1.3 Course Schedule Calendar

Each career is based in this location. It has a schedule that shows the available hours for a course.

1.1.1.4 Tentative Students

It represents the amount of students that can take a course as the maximum.

1.1.1.5 Location

It describes the physical site where the students are available to take a course. In the university parameterization, each location is described as the combination of a city, faculty, and campus.

1.1.1.6 Class Room

A room is a physical space used for face-to-face courses in some locations, which has a maximum capacity.

1.1.1.7 Career

A career is a set of courses divided by levels, which represents the area of specialization of a group of students. The career can be available or not in some locations.

1.1.1.8 Level

The level contains the courses based on the curriculum. It represents the level of the advance of a student in their career.

1.1.2 Brief overview

The Metropolitan University of Ecuador (UMET) is allocated in multiple regions around Ecuador, see Figure 1.1. The known cities until the day of the publish date of this work were Quito, Guayaquil, and Machala. Each city has multiple campuses, Quito has 3 campuses named "Los Chillos", "6 Diciembre", and the "La Coruña". Guayaquil has the campuses "Plaza Quil", "Garzota" and "Francisco". Machala includes the campuses "Junin" and "Pajonal".

Figure 1.1: *UMET distribution around Ecuador*

The university has a mean of two thousand students per semester, see Figure 1.2. Additionally, the university o□ers common or di□erent careers in the di□erent cities 1.2. Therefore, some courses are shared between cities and careers. In normal conditions, these courses were taught to distinct groups of students according to their city. However, the COVID-19 crisis changes the university policies.

In the pandemic crisis in 2020, the Ecuadorian Council for Higher Education decreed the ordinance RPC-SE-03-No.046-2020. It temporally bans face-to-face classes, Espinoza Cordero et al. (2020). In 2022, after the vaccination phase in Ecuador, a new ordinance was decreed, it proposes a hybrid education based on virtual and face-to-face course modalities.

Due to the background mentioned before, the university raised a Structured Virtual Learning Management System (SVLMS). The system proposes to share the virtual courses around the di□erent locations in Ecuador as it is described in the paper of Espinoza Cordero et al. (2020). Subsequently, the shared course between cities conglomerates their students. As a consequence, the number of students and career-related had increased, see Figure 1.3.

It caused a conflict in the University's management of scheduling the courses which are based on career level distribution. The principal issues are the overlapping schedules for the di□erent virtual courses for students from distinct locations. It incurred directly to overlap in time-slots of the courses assigned to multiple career levels. Besides, the timetabling of the virtual course a□ects the timetabling of face-to-face classes.

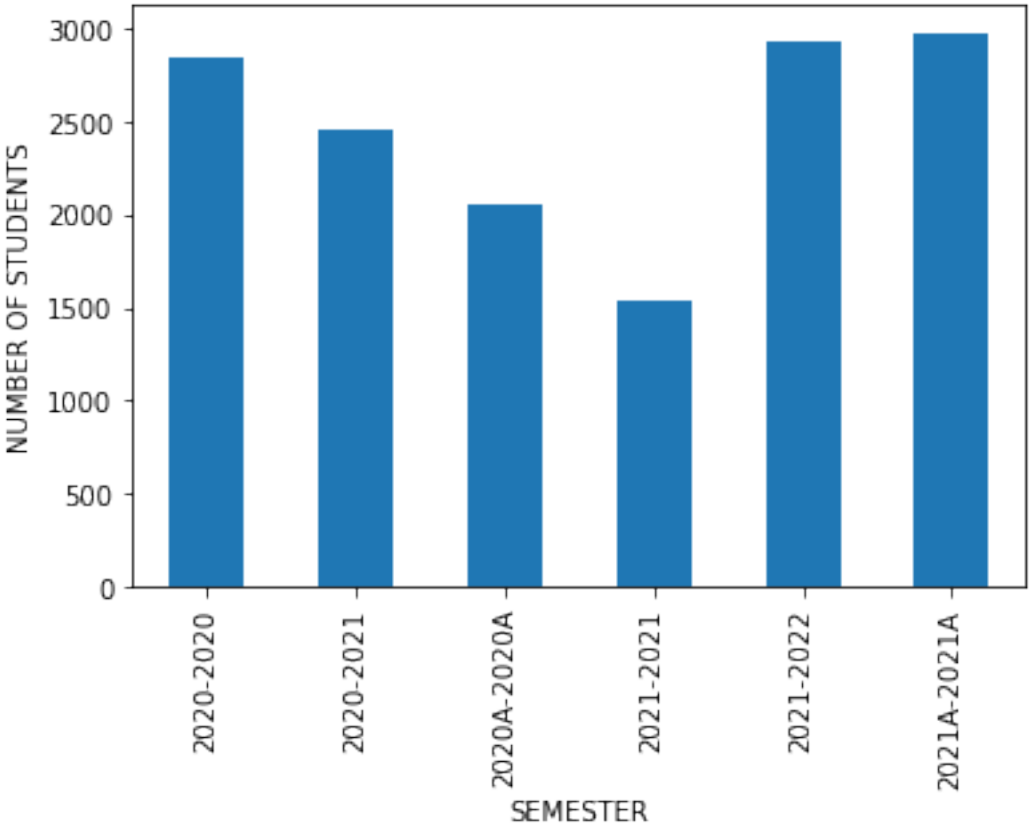

Figure 1.2: *Number of students the last 6 semester*

The University policies to schedule the courses are like a combination of well-known university timetabling problems such as Post-Enrolment Course Timetabling (PE-CTT) and Multi-phase Course Timetabling (M-CTT) explained in Subsection 1.3.2. Nevertheless, the fact that the courses are related to careers in di□erent cities and as far as possible the schedule of the career level should not have overlaps and extends these sub-categories of problems to a new kind of problem.

1.1.3 Entities of the Structured Virtual Learning Management System (ESVLMS)

The ESVLMS provides a complex structure based on the modality of the courses, see Figure 1.4. The courses are divided into two components virtual and face-to-face. In the case of virtual courses, They are shared between career and location. Each component has some specific restrictions, which are detailed in Subsection 1.2.1. For example, the course in Scientific Communication. It is run by the first semester of the chemistry career, and the third level of the information careers in the cities of Quito and Guayaquil. Even more, this course is divided into two components, the face-to-face and the virtual. The virtual component has the same professor and schedule in any university location, and it does not require a classroom. On the other hand, the face-to-face component has a di□erent professor and class according to the location.

The definition of the terms of Figure 1.4 is presented in subsection 1.1.

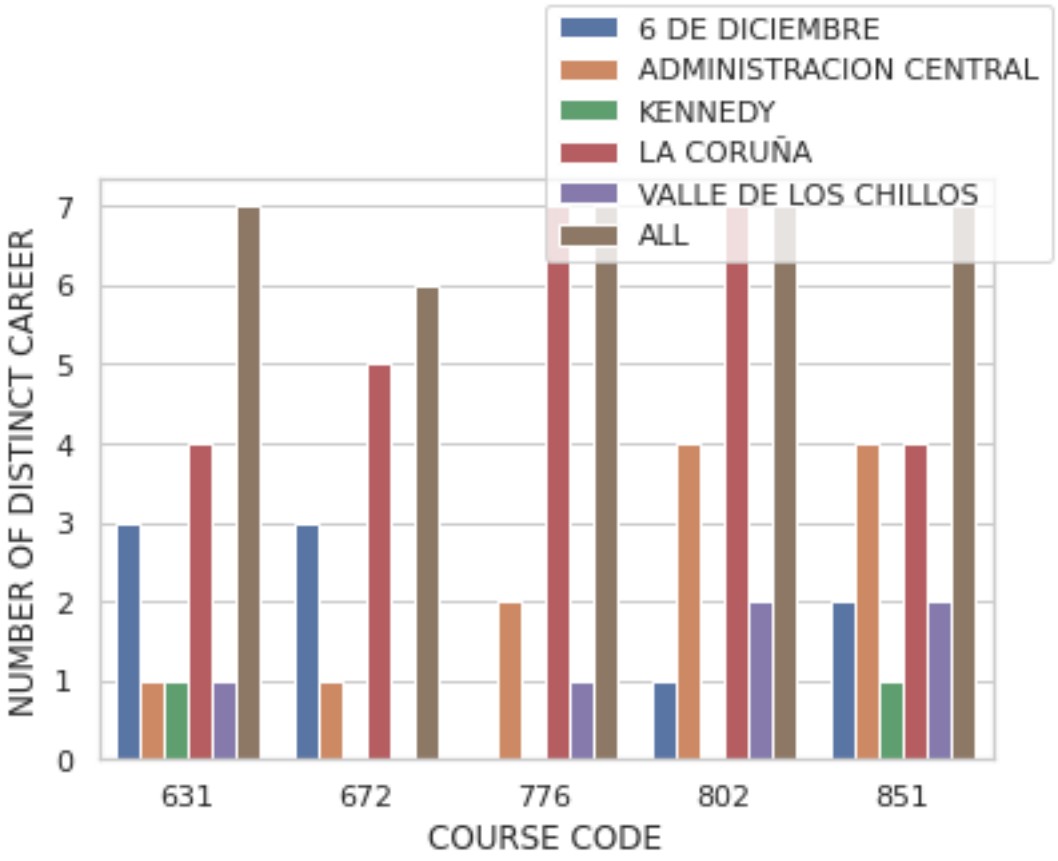

Figure 1.3: *Top 5 courses with major number of distinct related career*

1.2 Problem Description

In addition to the above, important constraints and in-context characteristics are to be considered to state the problem that this work is dealing with. To schedule must follow the accomplish the hard constraint and reduce the number of exceptions of the soft constraint, it is necessary to consider that some constraints are for each type of course component. Hard and soft restrictions are based on internal university policies.

1.2.1 Hard Constraints

1.2.1.1 Virtual modality

- **CHC1** The professor can give one course in a time-slot.
- **CHC2** The course assigned professor is immutable and it depends on human decisions.
- **CHC3** A course component has a total of hours per week, these hours must be distributed in a day per hour based on the distribution of Table 1.1.
- **CHC4** A group of tentative students cannot have two di□erent non-consecutive time slots for a course on the same day.

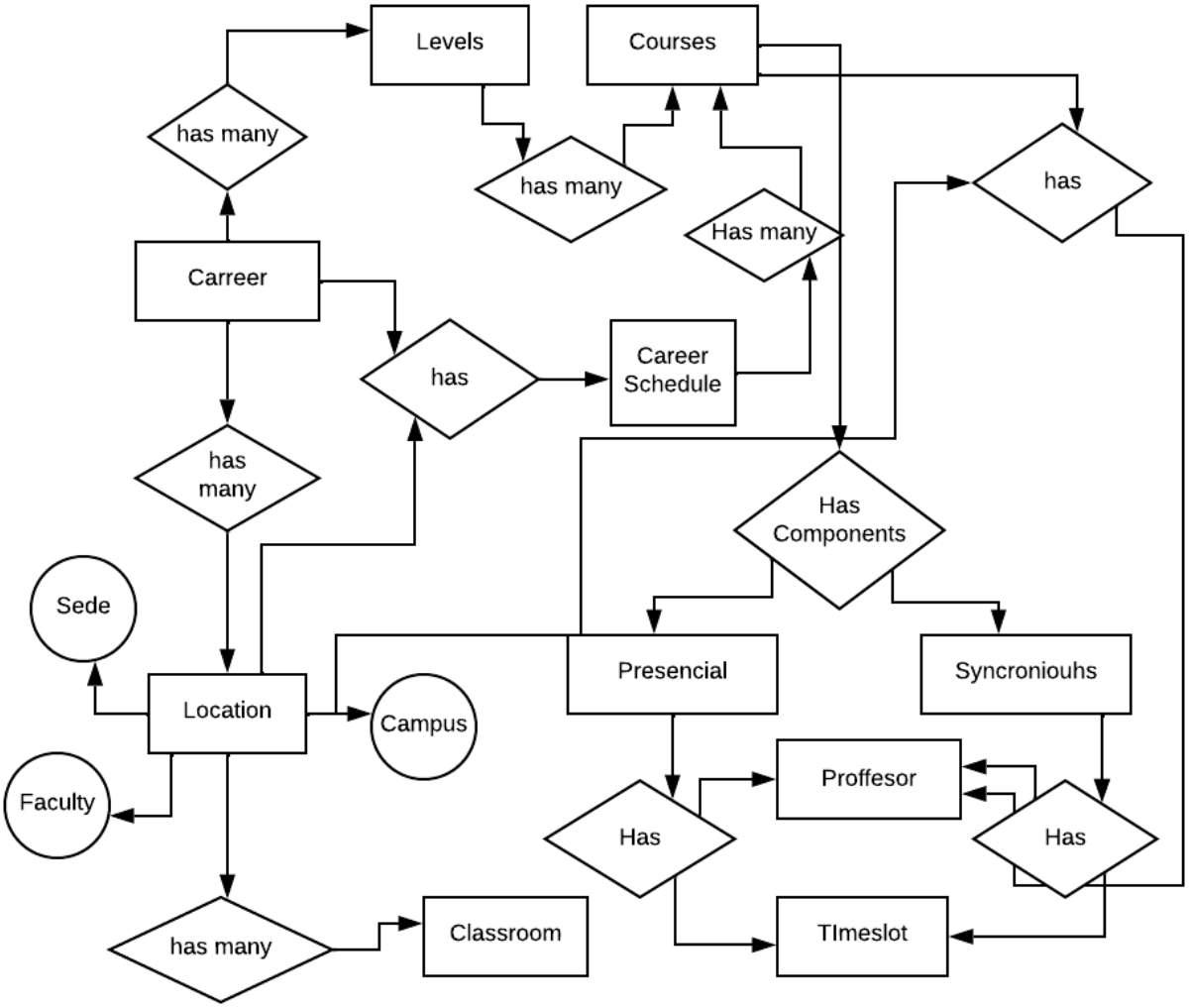

Figure 1.4: *Courses (Virtual/Face-to-face) relationship*

- **CHC4** A Day could not be used to schedule more than one time for the related course. It must include the selected days for each type of component.

1.2.1.2 Face-to-face modality

- **FHC1** The classroom cannot have two di□erent courses at the same time-slot.
- **FHC2** The classroom must have the same or major capacity that the tentative students require for the course.
- **FHC3** The course is scheduled based on their location.
- **FHC4** The professor must be in the course location.

Total Time	# Days	Hours/Day 1	Hours/Day 2	Hours/Day 3	Hours/Day 4	Hours/Day 5
2	1	2	0	0	0	0
3	1	3	0	0	0	0
4	2	2	2	0	0	0
5	2	3	2	0	0	0
6	2	3	3	0	0	0
7	3	2	2	3	0	0
8	3	2	3	3	0	0
9	3	3	3	3	0	0
12	5	3	3	3	3	0
15	5	5	5	5	5	5

Table 1.1: ***Total hours distribution per day.***

1.2.2 Soft Constraint

1.2.2.1 Common Constraint

- **CSC1** The courses have a tentative number of students per career, semester, and location. If a course is shared between many careers, then the available hours of the course are the career schedule with a large number of tentative students.
- **CSC2** If the total days is less than 3 per component then the chosen days must be non-sequential.

1.2.2.2 Virtual Constraint

- **VSC1** Minimize the number of overlaps for career level based on the priority of the number of students, considering the number of tentative students in the distinct locations.
- **VSC2** The timeslots must be assigned according to the career calendar from the course career which has the major number of students tentative students in any location.

1.2.2.3 Face-to-face

- **FSC1** Minimise the number of overlaps for career's level and location based on the priority of the number of students.
- **FSC2** The timeslots must be assigned according to the career calendar from the course career which has the major number of students, tentative students, in some locations.

This work is intended to solve the problem of timetabling under the condition of multiple schedules overlaps among courses related to multiple careers and limitless timeslot for other policies such as professors or days parameterization.

1.3 Background

This section is a review of the solution for CTT and the literature model problem. Subsection 1.3.2.6 is reserved for timetabling for the virtual course since it is a new brand topic that is related to this work.

1.3.1 CTT general problem

The CTT is well known that it belongs to the family of NP-Hard, Even et al. (1975). In a general view, the problem is defined as the relocation of resources in some time-slots. The literature explores different methods based on heuristic or non-deterministic solutions. Authors like Cruz-Rosales et al. (2022a), Huang y Wang (2022), Abduljabbar y Abdullah (2022a) develop a solution using a genetic algorithm. Even more, there are some modifications to GA, for example, Abdullah et al. (2007) mixes GA with local search. Cruz-Rosales et al. (2022b) used Mixed-Integer Programming with GA to produce a multi-alignment heuristic process. Another one was the use of GA with a great deluge algorithm in the article of Abdullah et al. (2012a).
Other solutions for timetabling are the Integer programming approaches such as the articles of Rappos et al. (2022), cuckoo search Zheng et al. (2022), binary programming Welahetti y Samarathunga (2022), and other meta-heuristic solutions.

Other solutions are explored and divided into sub-phases to solve the problem. The thesis of Hossain y Zibran (2007) decomposed the problem into subproblems. The subproblems were formulated in mathematical models. The proposed solution was ordering the mathematical model in phases, each phase solved the problem using Integer programming. Another case was the bachelor thesis of Photo (2013), which describes a multiphase timetabling using a genetic algorithm, in the case study the genetic algorithm is fully completed with the fitness function. Another solution for this problem was described by Abduljabbar y Abdullah (2022b), which is based on a GA. The manuscript describes the high adaptability of the genetic algorithm.

1.3.2 CTT Model Problems

A survey paper Ceschia et al. (2022a) expands the definition of CTT and includes three problem models: Post-Enrolment Course Timetabling (PE-CTT), Curriculum-Based Course Timetabling (CB-CTT), and other problems formulated in the 2019 International Timetabling Competition. These problems are related to the frequent problem for ITC. However, there are authors that supplementary the problem model as multi-phase course timetabling, thesis defense, and another problem model. The following subsections will expand the definition of these problem models and describe the best current best solution for these problem models.

1.3.2.1 Post-Enrolment Course Timetabling (PE-CTT)

It was formulated in the 2007 timetabling competition, the authors in the paper Lewis y Thompson (2015) describe as problem model as scheduled events into time-slots where the student enrollment

in the course is causing conflicts. The objective function is based on the penalization of students attending the different courses at the same time-slot. The benchmark analysis from Ceschia et al. (2022a), shows the best performance reached for PE-CTT was local search methods, in this case in specific where the local search algorithm uses an adaptive neighborhood size, a partial neighbor of the complete neighbor calculates the neighbor size, and it used tabu search to find the best neighbor Nagata (2018).

1.3.2.2 Curriculum-Based Course Timetabling (CB-CTT)

Mühlenthaler y Wanka (2013) defines the problem model of CB-CTT based on the conflicts of an entity named curricula. In this case scenario, the curricula are formed by a set of courses whose schedules do not overlap with the other curricula courses. The main difference between PE-CTT and CB-CTT is that PEC-CTT does not include the entity of curricula, and it is focused on student post-enrollment. According to the authors Ceschia et al. (2022a), the best solution to this problem is the evolution algorithm described in the work of Abdullah et al. (2012b).

1.3.2.3 Multiphase Course Timetabling (M-CTT)

M-CTT was introduced by Esmaeilbeigi et al. (2022). This problem shows its difference since it is based on the optimization of course level and students. The level is defined as a phase that the students are forced to take to graduate and are restricted to prerequisites. The proposed solution to this problem model was a fix-and-optimized metaheuristic using integer programming.

1.3.2.4 ITC-2019

The author details this subcategory based on the work of Müller et al. (2018) for the International Timetabling Competition 2019. In this article, he describes the classic problem with a weekly schedule. In other words, the schedule will differ for each week. According to the benchmark analysis of Ceschia et al. (2022a), the best solutions for this subcategory were the Fix-and-Optimize metaheuristic which was based on MIP with some improvements Ceschia et al. (2022b). Another solution was described in the work of Gashi y Sylejmani (2019). This solution is based on Simulated Annealing, but with the additional use of incremental penalization. This kind of penalization is calculated based on rules which describe the kind class of the penalty hard or soft.

1.3.2.5 Other educational timetabling problems

There are some problems that are not related to this work. For example, the authors Battistutta et al. (2019) describe the thesis defense timetabling model. Another formulation is high school timetabling, which was formally introduced by Post et al. (2012). An additional problem is the scheduling of college courses to reduce energy consumption described by Sun et al. (2021). In the article, the author creates using a genetic algorithm. Finally, the exam schedule, which has a problem model is defined as exam scheduling without capacity, based on the constraint of the student and exam tuple Carter et al. (1996).

1.3.2.6 Virtual Courses

The virtual university schedule is a novel problem caused by the COVID-19 constraint. Most of these papers refer to real problem data which makes the case study have di□erent constraints. The literature shows the di□erent approaches of others to solving this problem. The author Şimşek (2021), describes the di□culty of a larger number of overlaps in the time slots for multiple connections that may exist. The author describes a multi-objective mathematical model that has the bandwidth to restrict the number of connections.
The paper Barnhart et al. (2022) describes the problem with unlimited space for students due to COVID-19. The proposed solution is based on integer scheduling, which is applicable to highly constrained schedules.

Objectives

2

To develop an adaptive scheduling algorithm that can e□ciently solve the hard constraint and minimize the soft constraint of the di□erent career components. Considering the priority of reducing the number of students a□ected by overlaps in their career levels. The algorithm must be highly adaptive to new policies.

1. To solve all hard constraints according to their modality without exceptions.

2. To reduce the number of tentative students a□ected in the event that a soft restraint is not feasible to avoid.

3. To combine the partial solutions of the di□erent modalities and generate the complete schedule.

4. To produce an algorithm with a high ability to adapt to new policies or related problem models.

Methodology

3

The main goal is to accomplish all the hard constraints and reduce the number of courses that ignore soft constraints by considering the priority of the number of tentative students in the course. Moreover, the proposed solution requires a high ability to adapt to new policies. It implies that Algorithm must be scalable.

As stated above in Subsection 1.3, the problem conditions have not been explored in the scenarios of the current literature. Therefore, the proposed solution is based on a recurrent solution to solve CTT problems. The approach is the Genetic Algorithm (GA) since it gives the facility of combining the best timetabling to find the best fitness value. Additionally, it gives the freedom to solve and adapt to future policies the solution which is a requirement of this project.

As is well-known, a genetic algorithm creates one or many populations, and they are mutated or cross-over to select the best population in the fitness function. For that reason, the proposed solution is based on multi-objective GA that uses dynamic gene space. This gene space prevents the mutation from producing an invalid timetabling or ignoring a hard constraint. Since it uses dynamic programming techniques and combinatory pre-calculated sets to calculate the available values that a gene can be set.

Based on the nature of the problem, the task was divided into two parts. The first part is to solve the virtual constraint and based on that solution solve the face-to-face constraint. Both solutions implement a genetic algorithm. The complete workflow is shown in Figure 3.2.

In the following sections, it explains the principal functions and customization in the case they exist for the used GA to solve this problem. The workflow of the GA used in this work is shown in Figure 3.2.

See the full main code for the virtual courses genetic algorithm in the Appendix A.

3.1 Genes

The genes are represented in an array whose length varies according to the part of the problem they are facing. For the virtual courses, the gene is a numeric array of length 5, each value represents a time slot in a day. The values of the time slot are around 1 to 120, it represents the number of hours in a week. When a soft constraint does not fit with a time-slot inside the array, the gene is marked to be mutated.

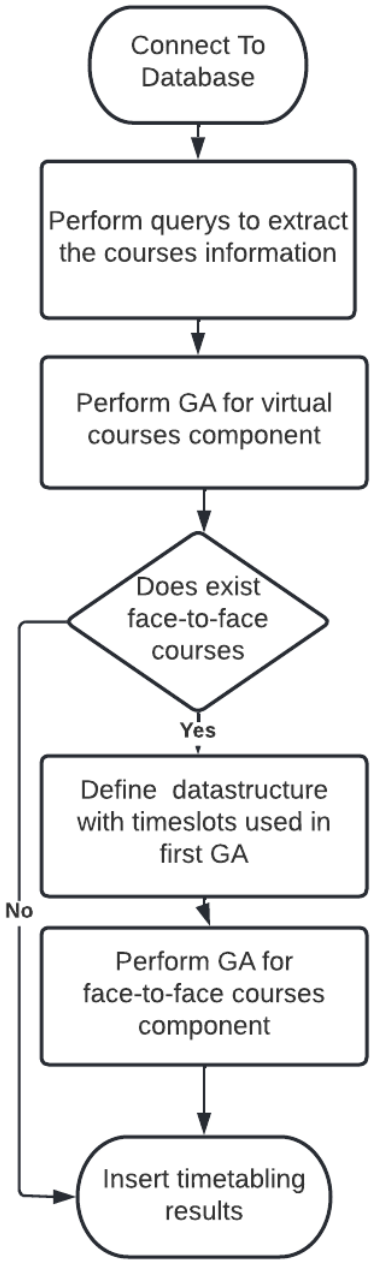

Figure 3.1: ***Complete Workflow***

In the case that a day has less than 5 days, the extra days are set with 0. For a face-to-face course, a gene is a numeric array with double length. The 5 last elements in the time-slot are filled with the code of the classroom for each of the first elements in the array.
In the case that the available space does not satisfy the course required time-slots, the gene is marked to achieve a mutation in the next round.

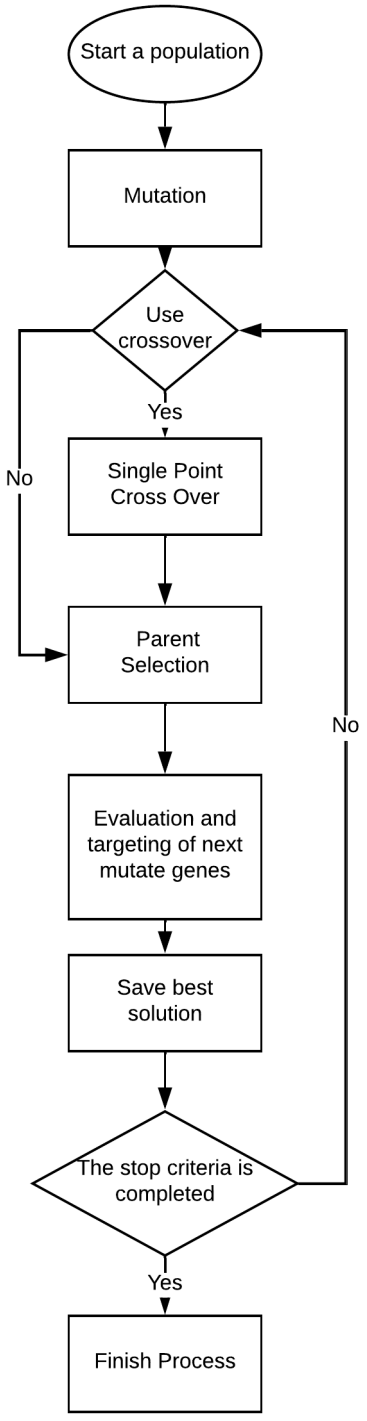

Figure 3.2: ***GA Workflow***

3.2 Chromosome

A numeric 2-D array that contains multiple genes. The rows are the genes, and their position represents a course. It is ordered based on the number of tentative students. The order of the chromo-

some gives priority in the assignation of time-slot to the course with a major population of tentative students. Figure 3.3 represents the population, where the columns of courses are the representation of the order of the courses in the array.

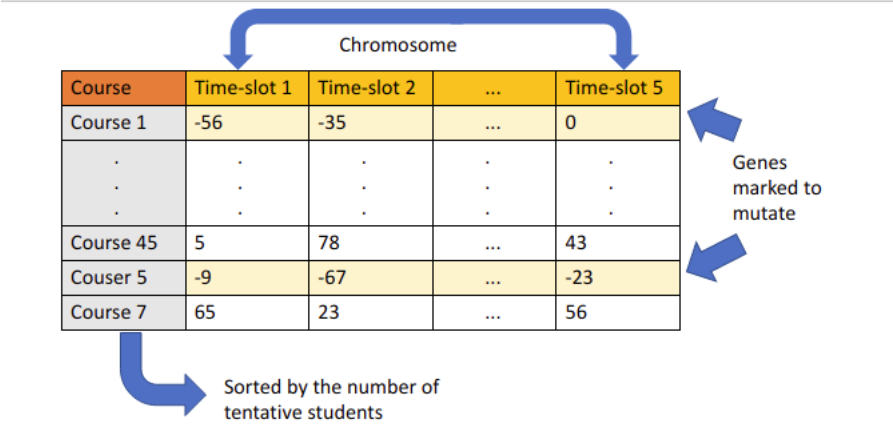

Figure 3.3: ***Chromosome Representation***

3.2.1 Gene Space

The available gene space is calculated from the information of previously selected genes, and the career schedule. From previous information, it is obtained the occupied timeslots are related to the career level and the course professor. The gene space for virtual courses is calculated by Equation 3.1.

$$TS = CS \quad {}^{1}CL \; , \; P^{0} \tag{3.1}$$

- **S** represents the set of timeslots that are not used.
- **CS** represents the set of timeslots that belong to the career calendar (It solves the soft constraint VSC2 in the case there are no available timeslots, this set is replaced with the available days).
- **CL** represents the set of timeslots that are used for all the career levels which are related to the course (It solves the soft constraint VSC1 in the case there are no available timeslots, this set is removed from Equation).
- **P** represents the set of timeslots that are used by the professor.

Equation 3.1 follows the CHC1, CSC2, CSC2 for virtual courses.

In the case of face-to-face courses, Equation 3.2 adds a new set that contains the classroom schedule hours (CC) and the a□ected virtual days (VD). CC is the result of the available classroom which had been scheduled in some time-slots. On the other hand. VD is the time-slots that belong to a day that appears on the virtual component scheduled.

$$S = CS \quad {}^{1}CL \; , \; P \; , \; CC \; , \; VD^{0} \tag{3.2}$$

Equation 3.2 solves the hard common constraint related with equation 3.1 and hard constraint FHC2, FCH3, FHC4. Additionally, it partially solves the software constraints FSC1 and FSC2. The analysis of the space if feasible to solve some soft constraints is viewed in the subsection 3.2.2.

3.2.2 Gene Space Exploration & Gene Selection

The detection of a gene space that has a valid solution is a complex problem. It is especially complicated to detect whether the gene space fits to solve some soft constraints or not. In the case that there is not a valid solution inside the gene space calculated based on some soft constraint, the gene space is replaced to expand space. The expanded space has some of the removed time-slots of the set CS or CL. The first step to analyzing if the gene space is useful or not is transforming the time slots into days, and time tuple.

To analyze if the CSC3 constraint has a solution in the gene-space, we compared the gene-space available day with a combination of available days that correspond to required days, Table 1.1. If there is a match or multiple matches with the required days of the course and the hours then perform a random selection of these matches, else penalized this population and ignore the CSC3 restriction. To solve the constraint CHC4, perform a search of the days which contain the required hours. If the number of hours of days has the required hours and days, create a combination set based on it or select one.

In the face-to-face modality, there is an additional step to be verified, which is the constraint FCH1. It searches the combination of time-slot days has available classrooms.

Figure 3.4 depicts the process of the genetic algorithm.

In the case of a day has less than 3 days, the algorithm must check the existence of an available time-slot that contains a set of time-slot that have days non-consecutive. To solve it a combination of the non-consecutive days was calculated using Algorithm 1. The gene space is divided into days and searches if the gene space has at least one of the day's combinations pre-calculated.

Algorithm 1: Generator of Non-Consecutive Days Set

Input: N /* //LIST OF DAYS */
Input: r /* //NUMBER OF DAYS */
Input: R = »¼ /* COMBINATION OF DAYS */
Function CombinationGenerator(N, r, R):
  if r = 0 **then**
    Return R
  end
  foreach ¹$pos, item$º 2 $enumerate$¹Nº **do**
    foreach F 2 r **do**
      Return $CombinationGenerator$¹N»pos ¸ 2 :¼, r 1, R ¸ »$item$¼º
    end
  end
End Function

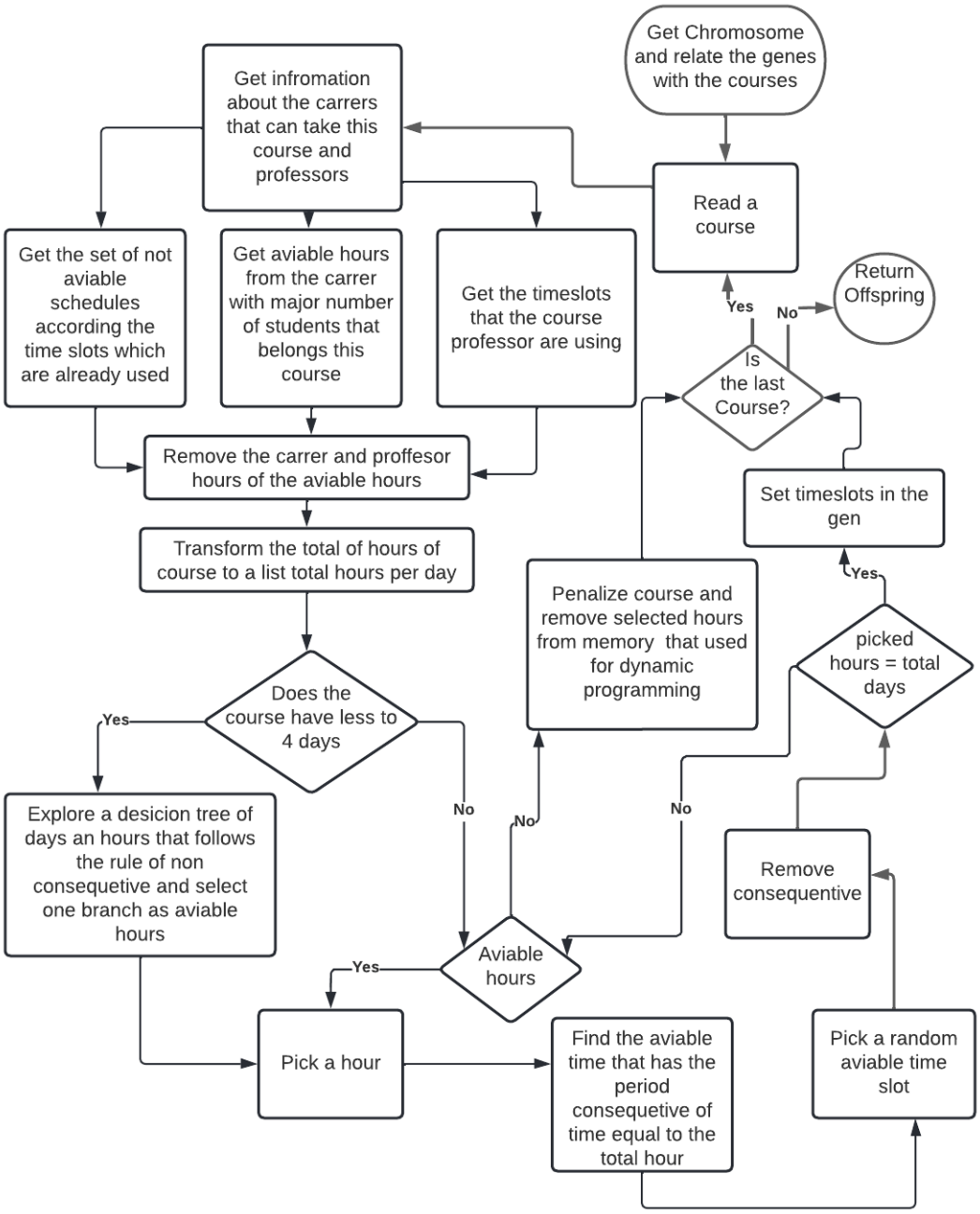

Figure 3.4: ***Population Generation.***

3.3 Parent Selection

To avoid the generation of erroneous chromosomes, the total number of parents is maintained for the next generation. In other words, no random genes are introduced. The literature reports some parent selection algorithms that are useful for scheduling problems, (Obaid et al., 2012). The algorithms explored in this manuscript are:

- Steady-State

- Tournament:
- Rank
- Stochastic Universal

3.3.1 Mutation

The mutation selects a static percentage of genes to mutate to minimize the fitness function. For simplicity, the static mutation percentage in this paper is described as the mutation index.
In addition, genes that are tagged or violate a constraint are also mutated with the mutation index. Algorithm 2 explains the proposed method in detail.

Algorithm 2: Mutation Algorithm

Input: O□spring 1-D

1. Transform the o□spring to 2-D array
2. Initialize auxiliary dictionary for gene space calculation
3. Sort the dataframe of the courses
4. Select random index of the population
5. **if** gene is target to be mutate
   (a) Start calculating the space gene.
   (b) Calculate a gene using the gene space
   (c) Update values of the gen
6. Save the gene timeslot
7. Save the values in the auxiliary dictionary
8. Return o□spring.

Output: O□spring

3.4 Cross Over

The proposed algorithm does not totally fit with the characteristics of crossover, since the create multiple partitions of the di□erent chromosomes could invalidate the dynamic space generated. Since it could produce more conflicts than expected between the genes of the new chromosome. It is represented in Figure 4.13 with a single point crossover. Even more, the Obaid et al. (2012) paper tested a genetic algorithm with di□erent genetic functions. The functions that were singled point, two points, uniform cross over, scatter. The results show that the single point was most e□cient.

Because of its background, the single point or non-cross-over represents a feasible candidate for the algorithm.

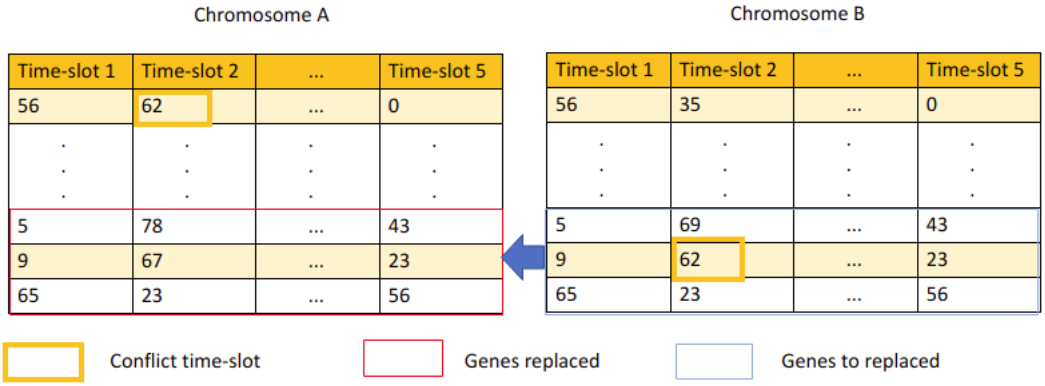

Figure 3.5: ***Crossover Problem***

3.5 Fitness Function

The fitness is calculated according to the type of resection violated. The weights represent the expectation or priority that a gene does not violate a constraint. In the case that, a gene infringed a hard constraint, the penalty is higher than the possible accumulation of another case, and its weight value is 100000. In the situation that a gene misses VSC1 or FSC1 constraint, the penalty value is equal to the total tentative number of students multiplied by 100. In the case of VSC2 or FSC2, the penalty value is the number of tentative students whose schedule does not correspond to the career of the major number of students. The fitness function is described in Equation 3.3.

$$minimize \quad \tilde{O} \; T_{SC2} \; , \; 100 \quad \tilde{O} \; T_{SC1} \; , \; 100000 \quad T_{HC} \tag{3.3}$$

T_{SC1} Correspond to the tentative student's number that the course timetabling does not satisfy the VSC1 or FSC1.
T_{SC2} Correspond to the tentative student's number that the course timetabling does not satisfy the VSC2 or FSC2.
T_H Correspond to the number of hard constraints that were ignored

An internal requirement disposed of that the gene from the virtual course will avoid penalization of the constraint CSC2. However, this internal policy does not apply to a face-to-face course.

The fitness function for face-to-face courses is like the virtual courses, but includes a term for CSC2, see Equation 3.4.

$$minimize \quad \tilde{O} \; T_{SC1} \; , \; 100 \quad \tilde{O} \; T_{SC2} \; , \quad \tilde{O} \; T_{CSC2} \; , \; 100000 \quad T_{HC} \tag{3.4}$$

T_{CSC2} It corresponds to the tentative student's number that the course timetabling does not satisfy the CSC2 constraint.

Results & Discussion

4

The chapter is divided into 5 sections. Section 4.1 contains a description of the section used in this work. Section 4.3 restricts the data set used for the experiments. Section 4.4 explains the selection of the hyperparameters. Section 4.5 compares the proposed GA with a simple GA. Finally, section 4.6 constrains the experiment and discussion.

4.1 Metrics

To compare the e□ciency of the solution, the university provides some metrics based on its objectives. The metrics are the number of tentative students that may have overlapped and failed soft constraint. In addition, some nonparametric tests are based on the assumption of similarity between the data. The method for multidimensional data was Kruskal-Wallis, (Kruskal y Wallis, 1952). For unidimensional data, the test used was the Wilcoxon singned-rank test, described in Wilcoxon (1945).

4.1.1 Number of courses passed at career level (COCL)

. Represents the total number of courses passed with other courses at the same career level. The COCL is defined by Equation 4.1.

$$COCL = \sum_{cl} \sum_{c} \sum_{cr=c,1} f_{COCL}(f_{CCL}(cl^{o}), f_{CCL}(cl), c^{o}) \; f_{COCL}(f_{TS}(c^{o}), f_{TS}(cr^{oo})) \tag{4.1}$$

$$f_{COCL}(t_x, t_y^{o}) = \begin{cases} 1, & \text{if } t_x \setminus t_y < ; \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

- CL represents the set of all the available career level.
- f_{CCL} is a function that gives all the available course for career level.
- f_{TS} is a function that receives a course and returns the set of available time-slots.

4.1.2 Number of tentative students with course over-leaps in career-level (SCOCL)

The SCOCL is described in Equation 4.3. It is cumulative of the tentative students in each course that have overlapped. The metric does not represent the actual number of students who have an overlap in their schedules, since it is possible that the tentative number of students share the same courses. However, since the tentative number of students does not represent the actual students, it is impossible to identify whether they are expected in the overlapping courses.

$$SCOCL = \sum_{cl} \sum_{c} \sum_{cr=c+1} f_{SCOCL}(f_{TS}(c), f_{TS}(cr)) \tag{4.3}$$

$$f_{SCOCL}(c_1, c_2) = \begin{cases} f_{ALU}(c_1) + f_{ALU}(c_2), & \text{if } f_{TS}(c_1) \cap f_{TS}(c_2) \neq \emptyset \\ 0, & \text{otherwise} \end{cases}$$

- CL represents the set of all the available career level.
- f_{CCL} is a function that gives all the available course for career level.
- f_{TS} is a function that receives a course and return the set of available time-slot.
- f_{ALU} is a function that receives a course and return the number of tentative students for this course.

4.1.3 Number of courses with wrong calendar (CWC)

The number of courses, their time-slots, and their gene space was calculated by ignoring time-slots that belong to the career calendar. CWC is defined in Equation 4.4.

$$CWC = \sum_{c} f_{CWC}(c) \tag{4.4}$$

$$f_{CWC}(c) = \begin{cases} 1, & \text{if } f_{TS}(c) \neq f_{CS}(c) \\ 0, & \text{otherwise} \end{cases}$$

- f_{TS} function that returns the set of time-slot for certain course.
- f_{CS} function that returns the set time-slots that belong career calendar based in the career with the maximum number students of a course.

4.1.4 Number of courses with consecutive days (CD)

The number of tentative students that belong to a course which their time-slots which used consecutive days that ignore constraint CSC2. It is defined in Equation 4.23.

$$CD = \sum_{c} f_{CD}(c) \tag{4.5}$$

$$f_{CD}(c) = \begin{cases} 1, & \text{if } \exists x \exists y \in f_{DAYS}(c), (x = y + 1 \vee y = x + 1) \\ 0, & \text{otherwise} \end{cases}$$

- f_{DAYS} a function that returns the set of days of a course

4.1.5 Number of tentative students with consecutive days (SCD)

The number of tentative students that belong to the courses and their time-slots used consecutive days that ignore constraint CSC2. It is defined in Equation 4.22.

$$SCD = \sum_{c} f_{SCD}(c) \tag{4.6}$$

$$f_{SCD}(c) = \begin{cases} f_{ALU}(c), & \text{if } \exists x \exists y \in f_{DAYS}(c), (x = y + 1 \vee y = x + 1) \\ 0, & \text{otherwise} \end{cases}$$

- f_{DAYS} a function that returns the set of days of a course
- f_{ALU} a function which receives a course and returns the number of tentative students for this course.

4.2 Technical Description

The section is divided into the Subsection 4.2.1 which has the concerning information about the device used for the experiments and Subsection 4.2.2 describes the software and libraries used in this manifest.

4.2.1 Equipment description

The used computer has the following description:

- OS: Ubuntu 20.04.2 LTS x86 64
- Kernel: 5.13.0-40-generic
- Shell: bash 5.0.17
- Desktop Environment: GNOME
- CPU: Intel i7-8565U (8) @ 4.600GHz

- GPU: Intel UHD Graphics 620
- Memory: 12250MiB / 15693MiB
- RAM: 16 GB

4.2.2 Software Description

The implementation was developed in python 3.6 using the pandas McKinney et al. (2010) to manage the data and perform queries to the university oracle database. The statistical tests were performed with scipy (Virtanen et al., 2020). The plots were drawn with seaborn (Waskom, 2021). Finally, for the creation and development of a genetic algorithm, the classes of PyGaD were extended (Gad, 2021).

4.3 Datasets Description:

The experiment was set up using two data sets from previous semesters. The first data set corresponds to the semester 2021-2021 and the other one corresponds to the last semester 2021-2022. In the case of semester 2021-2021, the courses, in their majority were virtual. So, there was no necessity to use the second phase of the methodology. The second semester includes virtual and face-to-face classes, so, it was used for the timetabling of two phases.

4.3.1 Data-Set 1: Semester 2021-2021

The semester has a considerable number of entities that take part in timetabling detailed in Table 4.1. Figure 4.1 shows the number of courses related to a career level.

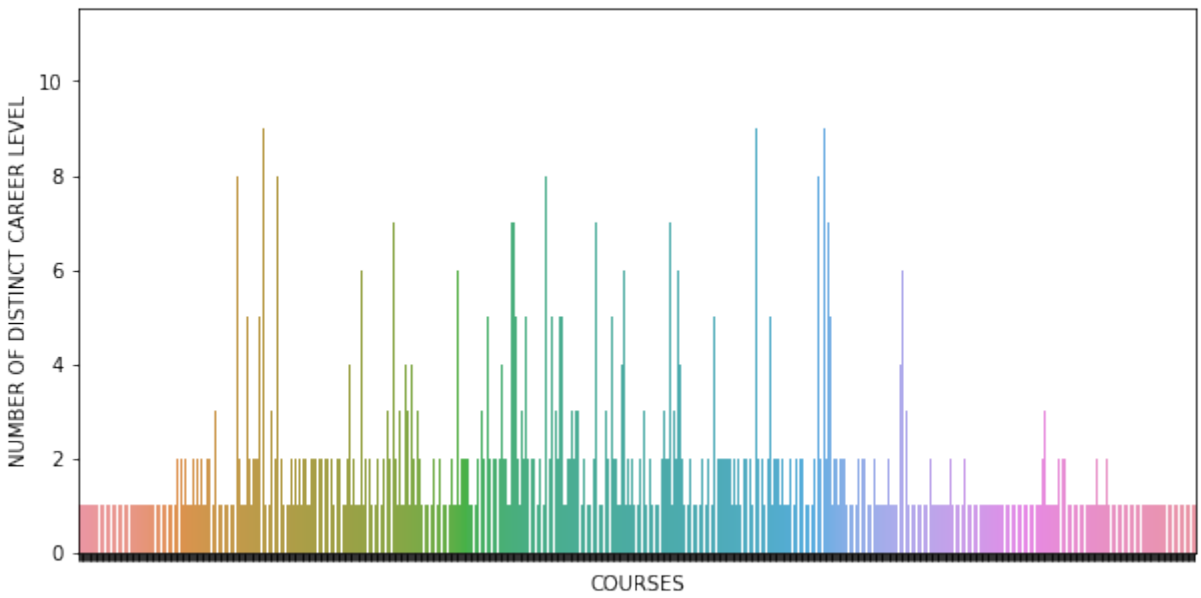

Figure 4.1: *Number of career level related with a course*

Most of the courses are related from one to four career-level. The courses at career-level majors than six are isolated cases. It represents most conflicts in the timetable. It is around 2 or 4 careers. Figure 4.2 shows the distribution of the number of related career-level with courses. The VSC1 constraint of the career level depends on the timetabling of these courses. It shows that there exists a great conflict with the courses. However, it totally depends on the number of relations with the professor.

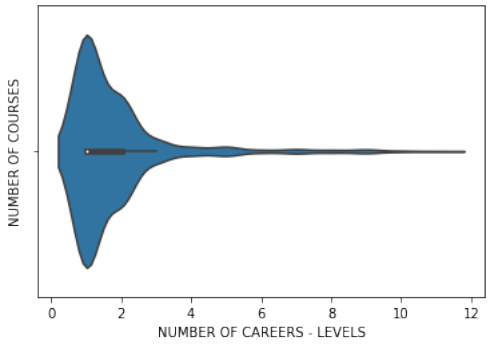

Figure 4.2: *Distribution of courses in career-level*

The CHC1 is the hardest constraint since the professor is related to the courses and the career levels. Figure 4.4 shows there is a high number of career-related courses that a professor gives. Figure 4.6 shows that the professors have a mean of six courses and 6 career levels. Most professors give around two and 8 courses this semester and they are related from 4 to 10 career-level. This data indicates that the overlaps in the timetable are extremely related to a professor.

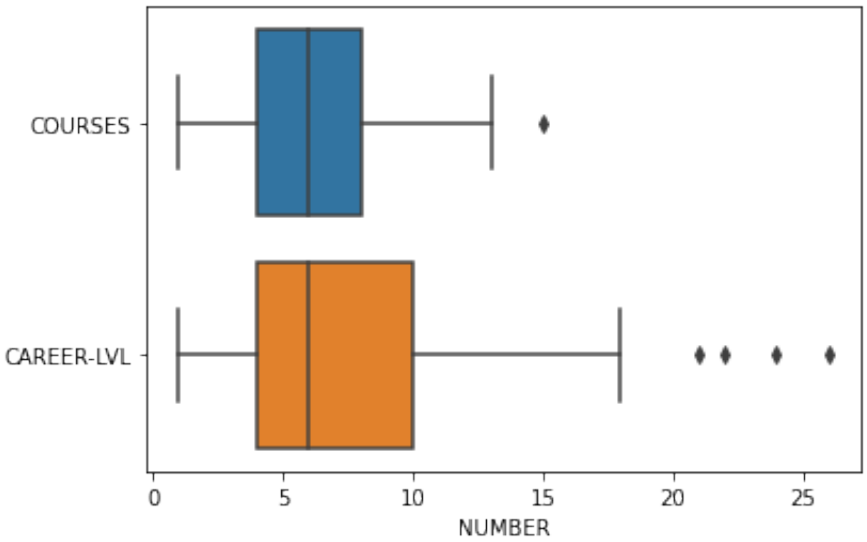

Figure 4.3: *Box-Plot career-levels and courses related with a professor*

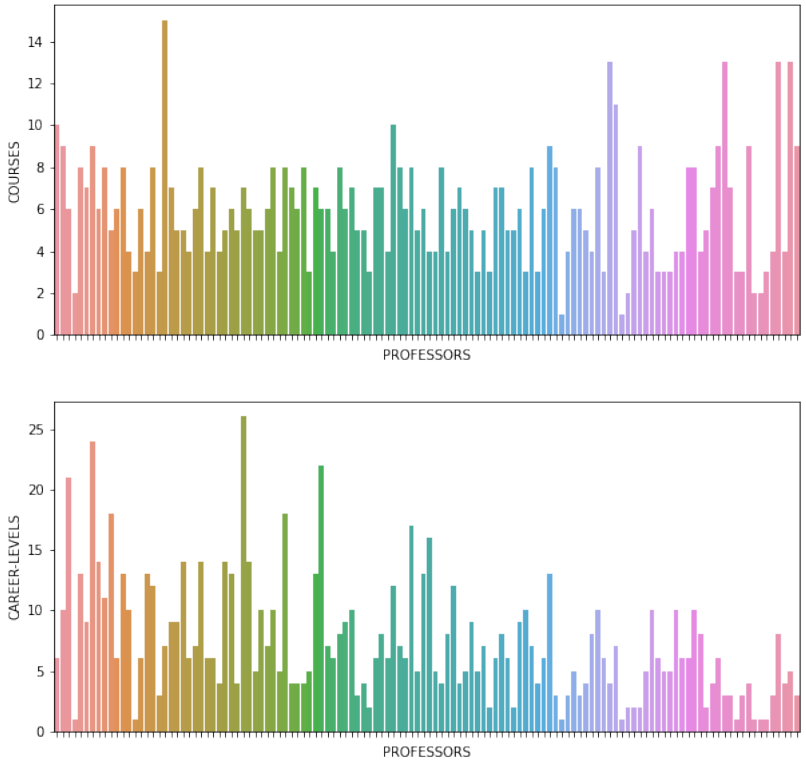

Figure 4.4: *Number of courses related with a professor*

Finally, the number of students is essential to solving the soft constraint CSC1, Figure 4.5 represents the number of students for the course. The courses with the highest number are the priority. Figure 4.6 presents that are courses extremely populated with a number superior 100 tentative students. The courses had less priority. These courses will have the highest priority in the timetable assignment.

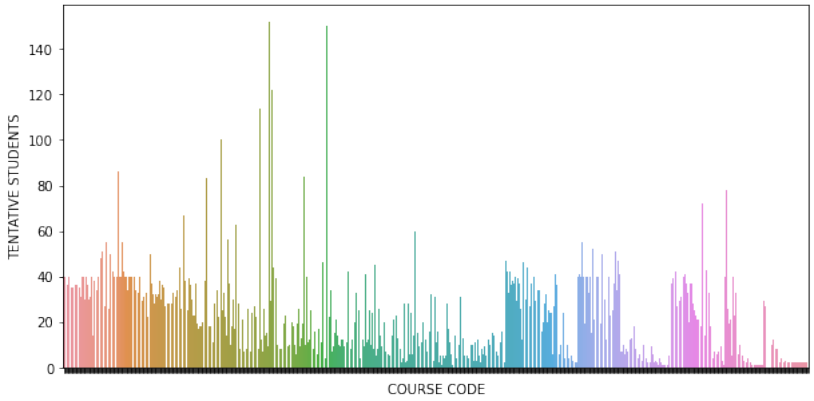

Figure 4.5: *Number of tentative students related with a course*

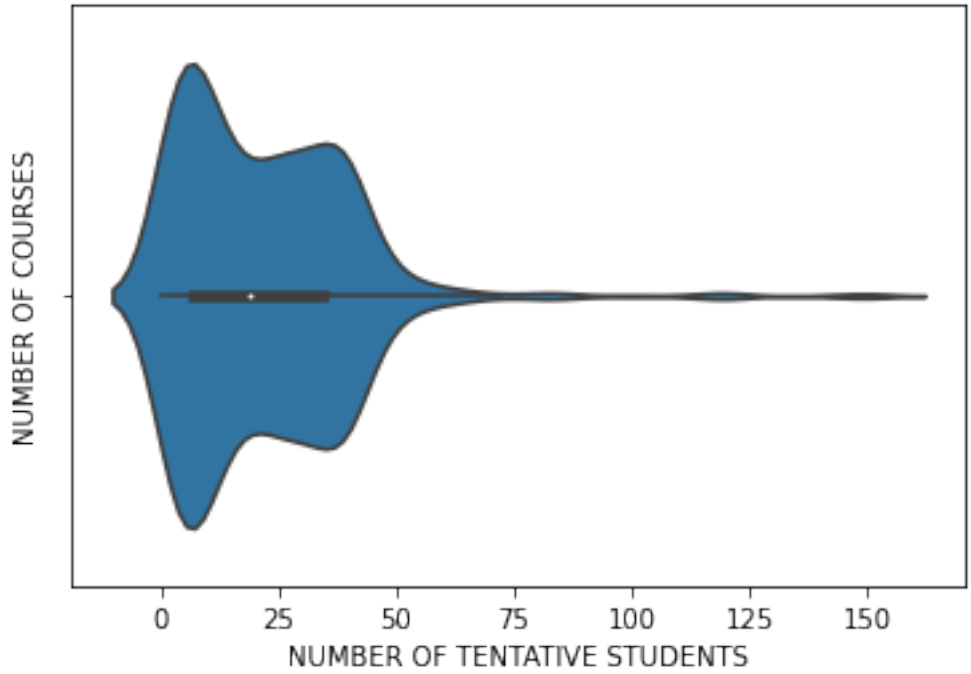

Figure 4.6: *Distribution of tentative students related with a course*

4.3.2 Dataset 2: Semester 2021-2022

Dataset 2 corresponds to the semester 2021-2022. It was the first semester in which was implemented a hybrid modality. The number of entities is described in Table 4.1.

Variable	Virtual Components	Face-to-face Components
Courses	480	152
Location	5	5
Hours	1447	321
Career	9	7
Classroom	0	836
Career Level	73	46
Courses in di□erent Career level's	1060	201
Professor	118	31

Table 4.1: ***Number of elements of the related variables***

The semester reported some courses which have some hour classes in a face-to-face modality and the rest of the hour in virtual. Other cases are courses are fully virtual or face-to-face. Figure 4.7 shows the relation between professors and classmates for the components of the courses.

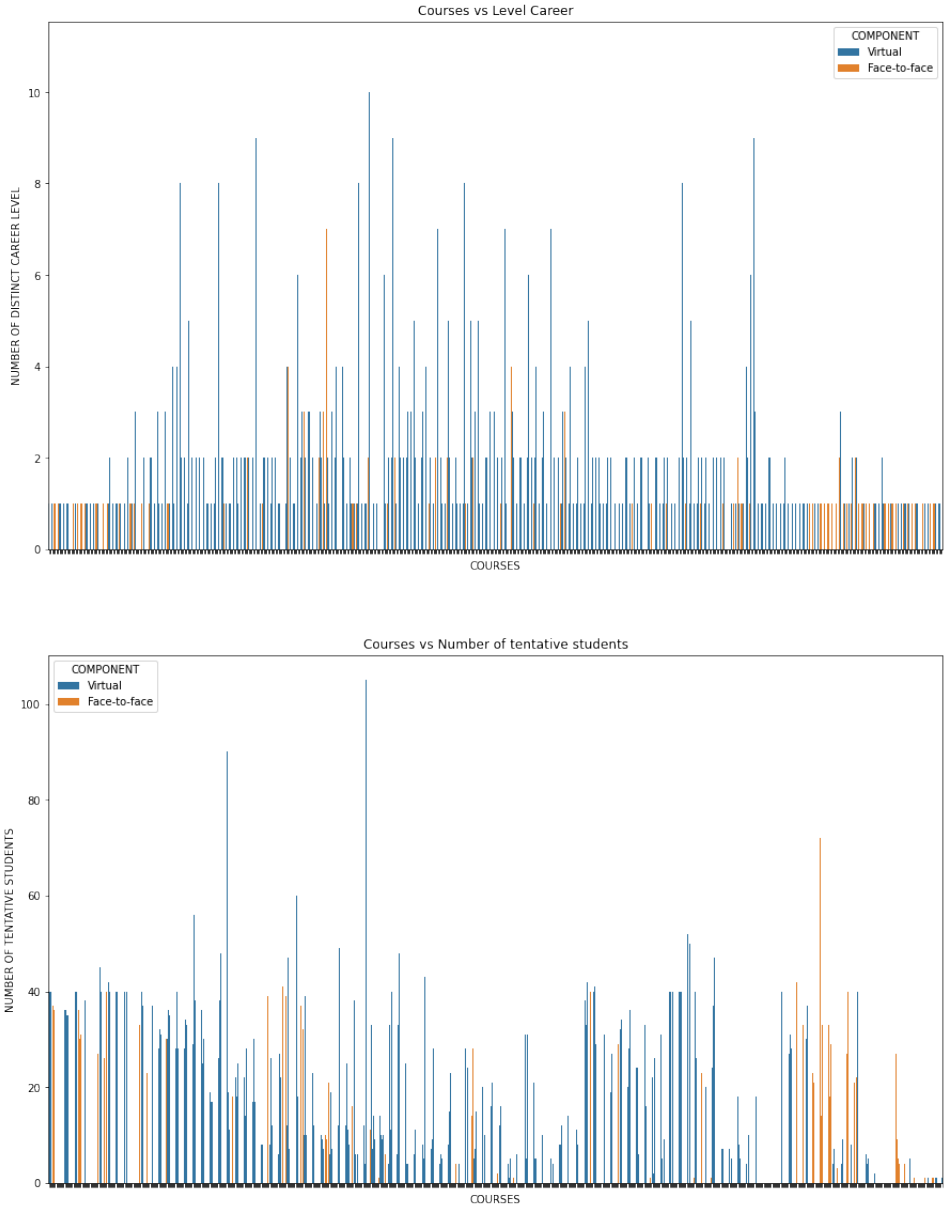

Figure 4.7: *Courses relation based in their modalities*

In a simple view, face-to-face modalities are often with a higher number of tentative students than virtual. However, the number of tentative students and career level are divided by the location where they belong to. It makes the di□erence between virtual components which accumulate the students of various locations. Figure 4.9 represents the number of courses per location including virtual.

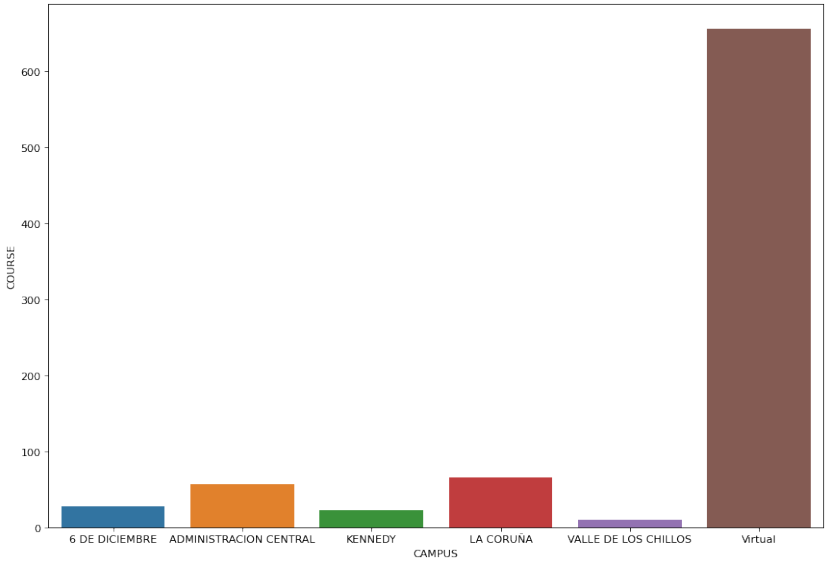

Figure 4.8: *Total Courses per location*

Figure 4.10 compares the virtual and face-to-face components. Additionally, Figure 4.9 exhibits that comparison with the number of tentative students.

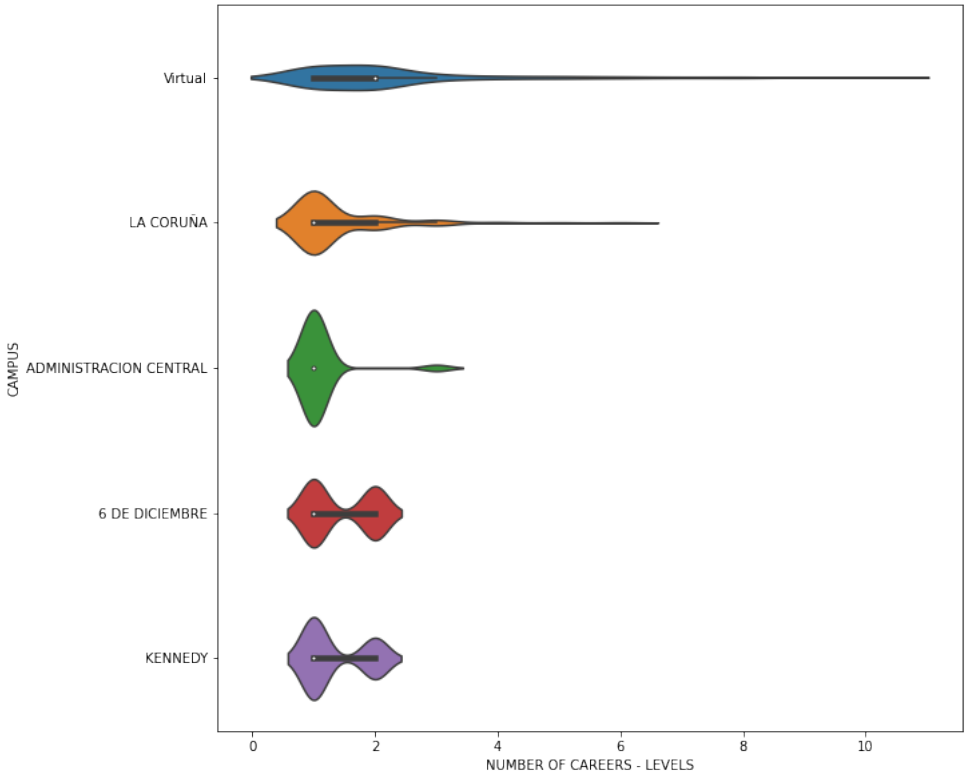

Figure 4.9: *Total of courses based in their location*

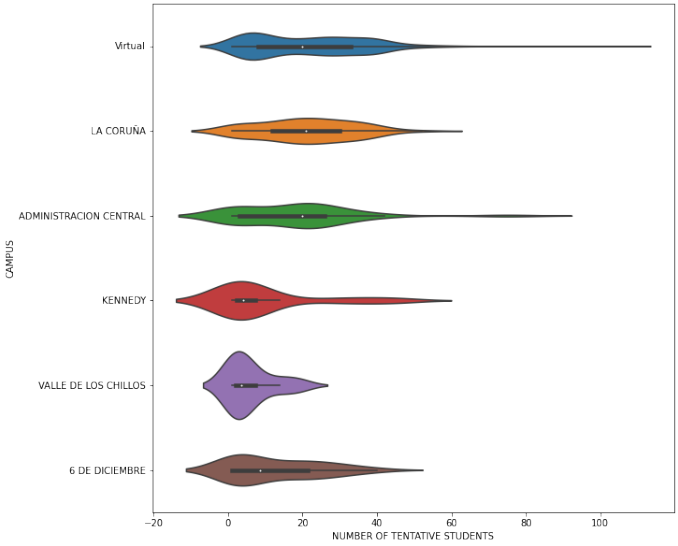

Figure 4.10: *Tentative students based on location*

The virtual courses represent the hardest task for the timetabling algorithm since there are related to a major number of career levels and it has a major number of courses. On the other hand, the course's face-to-face timetable is not fully related, however, the number of students is higher than the virtual one. It similarly happens in the relationship with the professor, see Figure 4.12. Thus, the virtual component persists as complicated timetabling. The university has around 836 di□erent classrooms are available to use in the timetable. The capacity of the classroom according to their campus is represented in the sample in Figure 4.11.

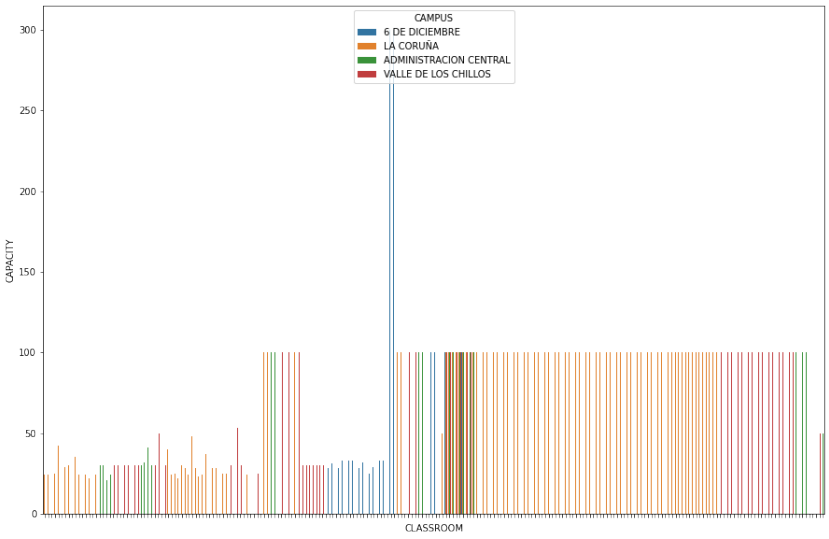

Figure 4.11: *Classroom capacity per campus*

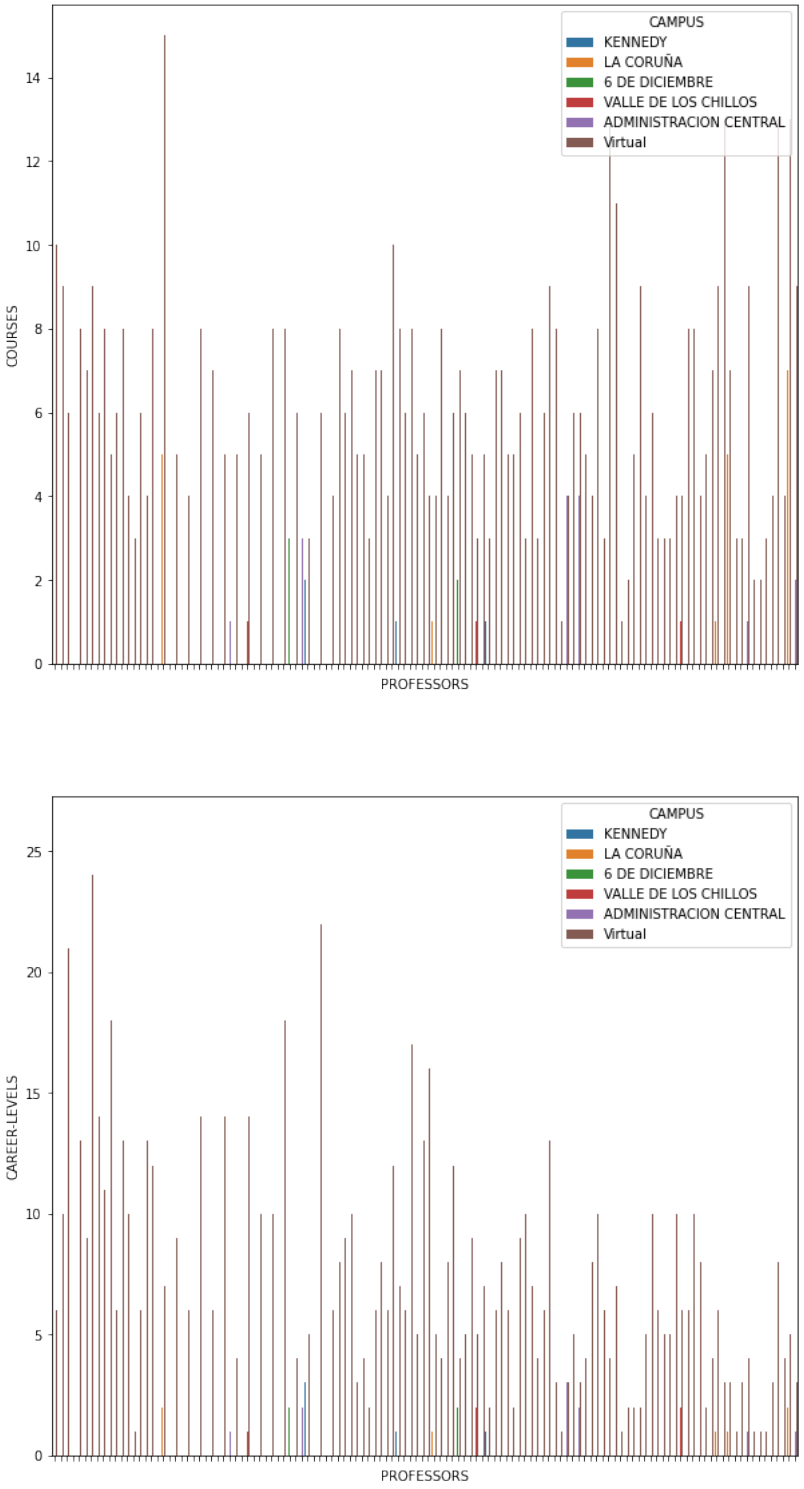

Figure 4.12: *Relation professors in di erent location*

4.4 Hyper-parameter Selection

To select the best mutation number and selection parent type was performing an experiment. The experiment used the Database described in Section 4.3.2. For each type of parent frequently used

in the literature (Steady Selection, Stochastic Universal Selection, and Tournament) and Rank) and for di□erent type of mutation index (0.05,0.1,025,0.35) were calculated the fitness function in 40 generations, see Table 4.3.

Mutation (%)	Rank	Steady-State	Stochastic Universal	Tournament
0.05	309132.4±10648.19	294303.33±8356.55	337247.0±6676.01	362358.33±15736.19
0.1	311012.4±11559.09	306104.83±10443.39	365401.0±15276.19	348448.33±10950.93
0.25	311521.2±10315.84	316516.6±7311.19	365433.0±12560.46	335078.67±20093.25
0.35	312089.8±5636.47	311637.0±12346.45	355305.67±13102.61	366714.0±4589.29

Table 4.2: ***Fitness value for di erent parent selection and mutation (%)***

The results obtained for this experiment were tested by Kruskal-Wallis. The first The first hypothesis checked was:

- H_o : The mean is the same for diverse selection types and mutation index.
- H_a:The mean is di□erent.

The statistic and the p-value were 61.065 and 1.65 10 ^{7} corresponding. The p-value is less than 0.05, so the mean is di□erent among the experiment, and the null hypothesis is rejected.
The fist hypothesis checked was:

- H_o : The mean for a parent selection type is the same in any mutation index.
- H_a: The mean are di□erent.

Value	Rank	Steady-State	Stochastic Universal	Tournament
statistic	0.474286	2.840000	8.937462	10.031770
pvalue	0.924503	0.416957	0.030134	0.018298

Table 4.3: ***Fitness value for di erent parent selection and mutation (%)***

Table 4.3 contains the result of Kruskal-Wallis test for mean generation in the parent.

Since the mutation index does not a□ect the parent selection with the major value of fitness function. The next hypothesis tests the independence of the parent selection types:

- H_o : The mean of a mutation equal to 0.05 is the same for each selection type.
- H_a: The mean is di□erent.

The statistic was 16.17 and the p-value of 0.0014, which e□ectively means that the means are different. Due to this fact, the Stochastic Universal and Tournament were rejected. The next step is to detect if there are di□erences between rank and steady selection. To perform that was used the Wilcoxon Test. The obtained result was a statistic of 101 and a p-value of 0.89, so the null hypothesis was not rejected. As a result, the rank and steady selection, with any mutation value could represent a satisfactory solution.

Table 4.4 contains the mean of time in seconds spent performing the experiments. This experiment used a cross-over single point function. Additionally, the time spent to create 5 chromosomes is (26.6 ± 0.6) s.

Mutation (%)	Rank (s)	Steady-State(s)	Stochastic Universal (s)	Tournament(s)
0.05	2438.5	2657.15	2515.22	2893.69
0.1	2156.93	2916.54	2637.75	2560.23
0.25	2285.58	3258.28	2560.81	2408.61
0.35	2983.99	3042.26	2878.75	2616.19

Table 4.4: ***Execution Time for di erent parent selection and mutation (%)***

After the results were obtained; an additional experiment was performed to obtain the ideal crossover function. The experiment compares if a crossover could improve the results. Figure 4.13 includes five experiments over fifty generations using a single-point crossover function and the genetic algorithm without a crossover function. The hyper-parameter used in this experiment was steady parent selection and mutation number equal to 0.35.

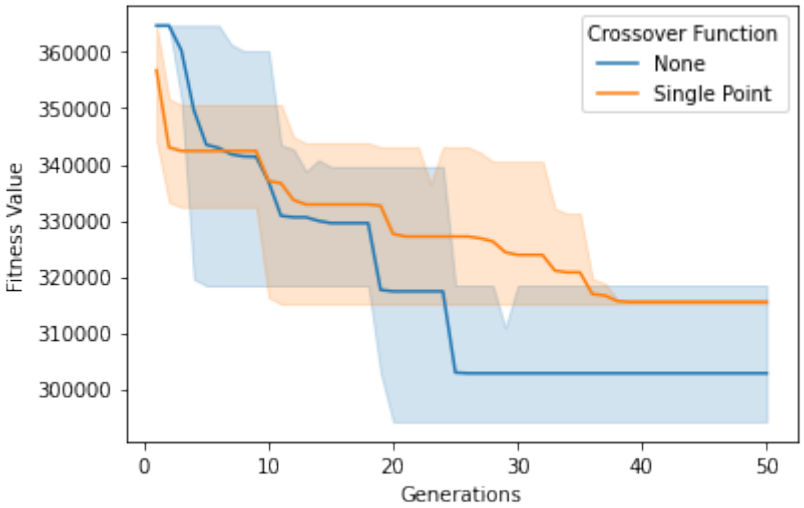

Figure 4.13: *Comparison of crossover function in 50 generations*

The results exposed in Figure 4.13 show a higher value of the fitness function is produced by the non-crossover function. To conclude this section, the experimental results show that the best

hyperparameters are a rank or steady parent selection, and non-use of a crossover function.

4.5 Experimental Comparison of Classic GA vs Dynamic Gene Space GA

The experiment was performed to compare a multi-objective normal GA with dynamic space gene GA using the same fitness function over two hundred generations. The Dynamic Space GA used a mutation index of ten%, steady parent selection, and it does not use a crossover function. The classic GA was set up with a random population. The time spent on the execution of the classic GA was around the (4120 ± 320) s and the Dynamic Gene Space (8950 ± 150) s. The proposed algorithm, described in this manifest, takes around two times the time that the classic GA spends. However, the value of the fitness function is greater than the classic, see Figure 4.14.

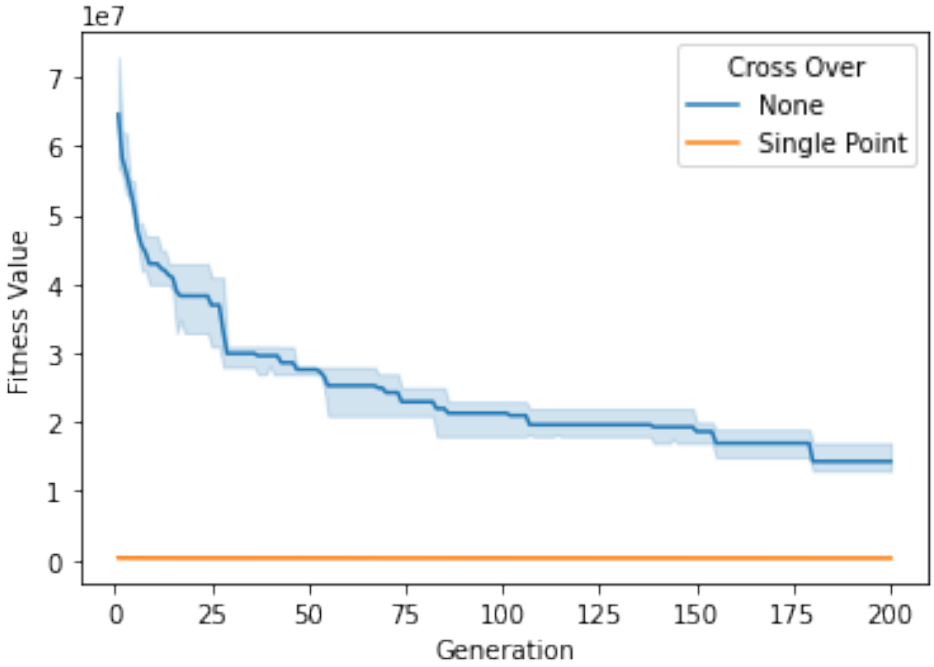

Figure 4.14: *Comparison of crossover function in 50 generations*

The next experiment uses the same start population for the Classic GA and the dynamic Gene Space GA. However, over two hundred generations the classic GA could not improve any results, see Figure 4.15.

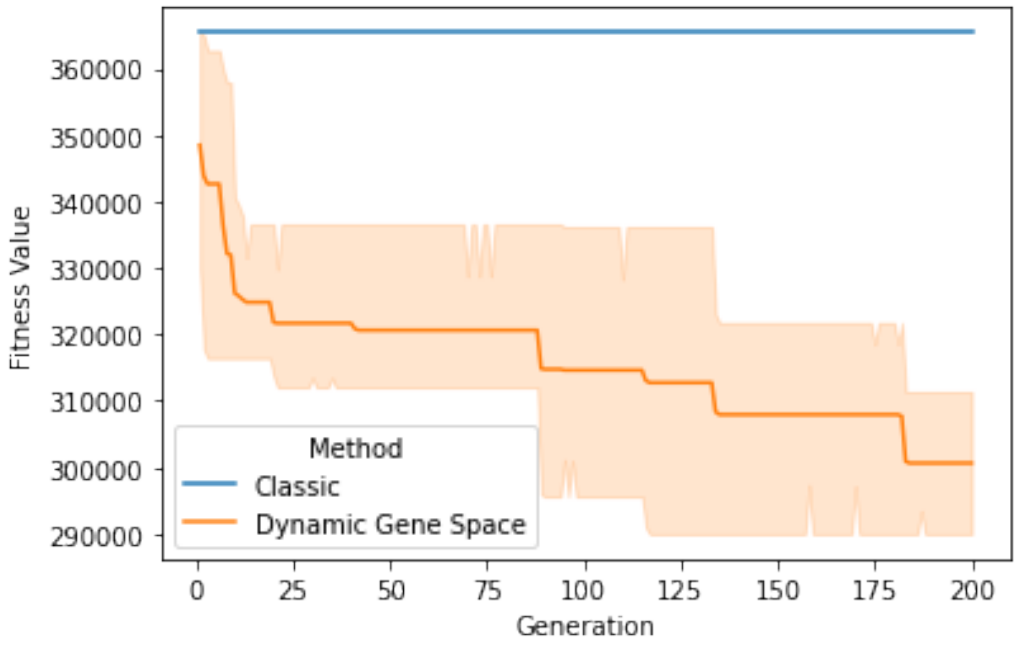

Figure 4.15: *Comparison of crossover function in 50 generations*

4.6 Experimental Result

4.6.1 Data-Set 1

The GA computes two hundred generations in five di□erent experiments. After 125 generations the GA does not show any improvement in fitness value. It started using a population with a fitness value of 11578. The minimum value reached was 59634. Figure 4.16 shows the evolution of fitness value through two hundred generations.

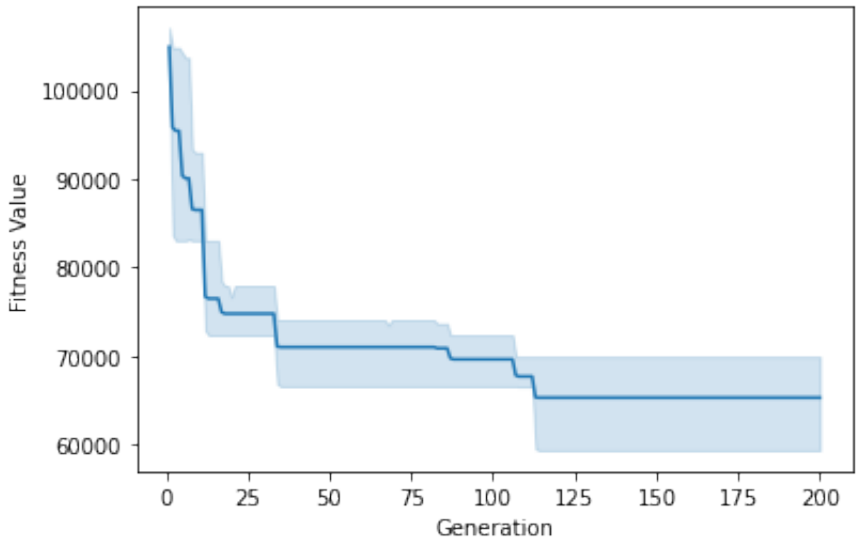

Figure 4.16: *Result of the Fitness vs generation*

The value of fitness function has a major dependency on the number of tentative students a□ect overlap in the career level (SCOLC) and less significant the tentative students which their calendar does not adjust to the career calendar (SWC). Since the GA has multiple objectives, the SCOLC has a high priority. In this sense, Figure 4.17 shows that SWC values are alternative in minimum and major through the generation to improve the SCOLC.

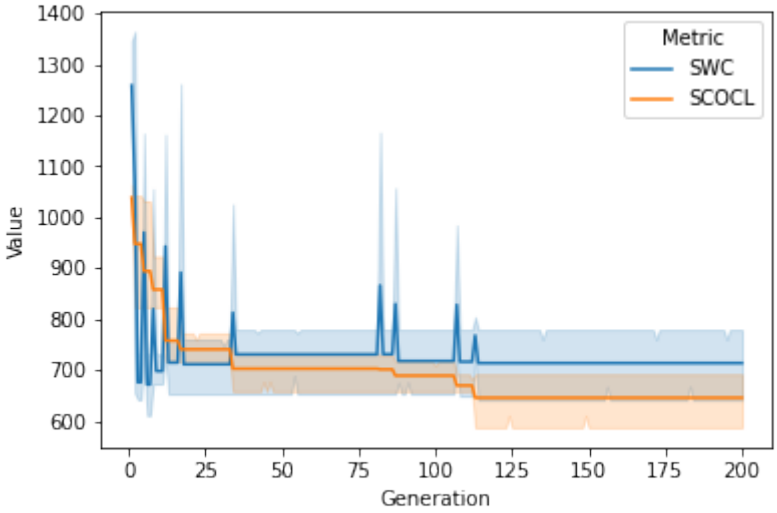

Figure 4.17: *Number of tentative students related with a course*

Another aspect is the reduction of the values of SCOLC did not imply a reduction in COLC. Figure 4.18 shows in the final generation how the COLC is increasing, compared to the SCOLC that started to decrease.

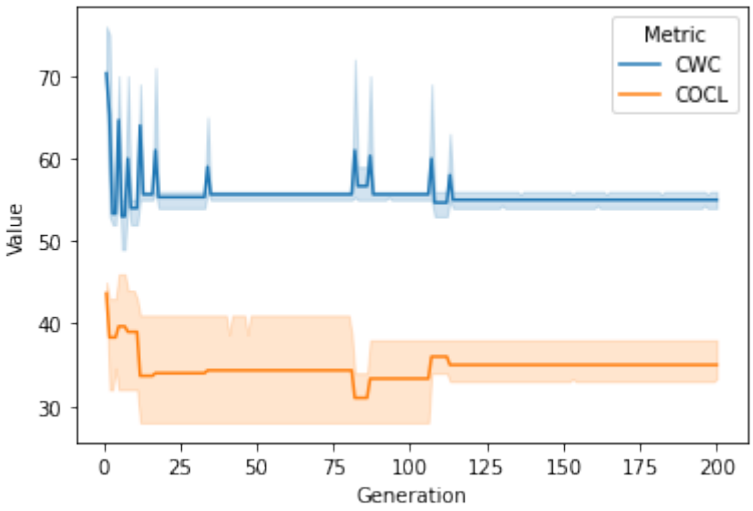

Figure 4.18: *Number of tentative students related with a course*

4.6.2 Data-Set 2

In the first part, the algorithm searches for the best solution for virtual courses. It excludes the hard constraint FC1, however, the task represents the hardest computational work due to the multiple relations that exist. The virtual GA started with a fitness value for a population of 377459 and it was minimized after eight hundred generations to 287920, see Figure 4.19. It repents an improvement of 1000 students, see Figure 4.21.

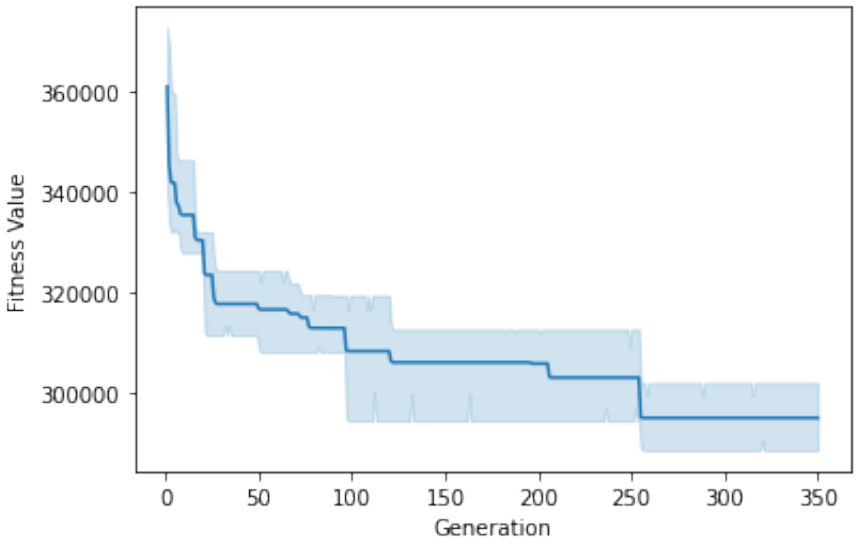

Figure 4.19: *Virtual component fitness value*

In this dataset, the values of CWC or SWC reach peak values when the COLC or SWOLC is reduced. After some generations, the CWC is reduced with a soft major value of the previous generation, see Figure 4.20 and Figure 4.21.

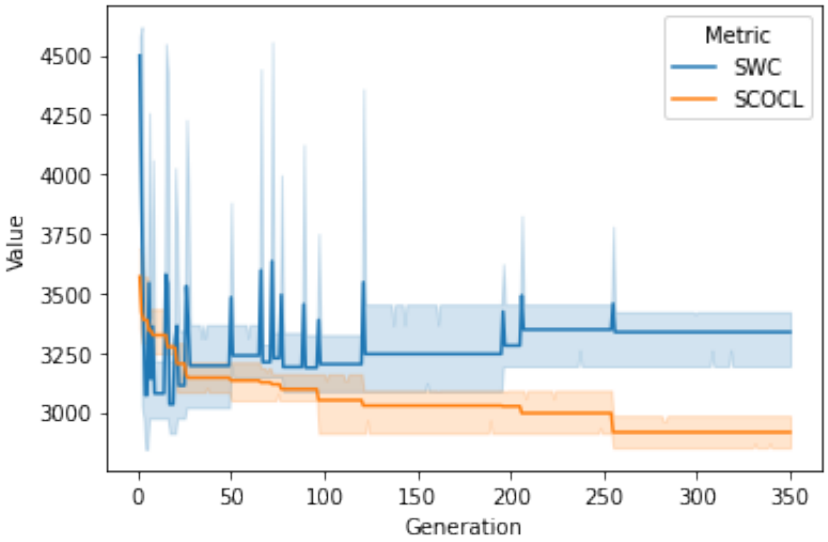

Figure 4.20: *Number of courses a ected in virtual courses*

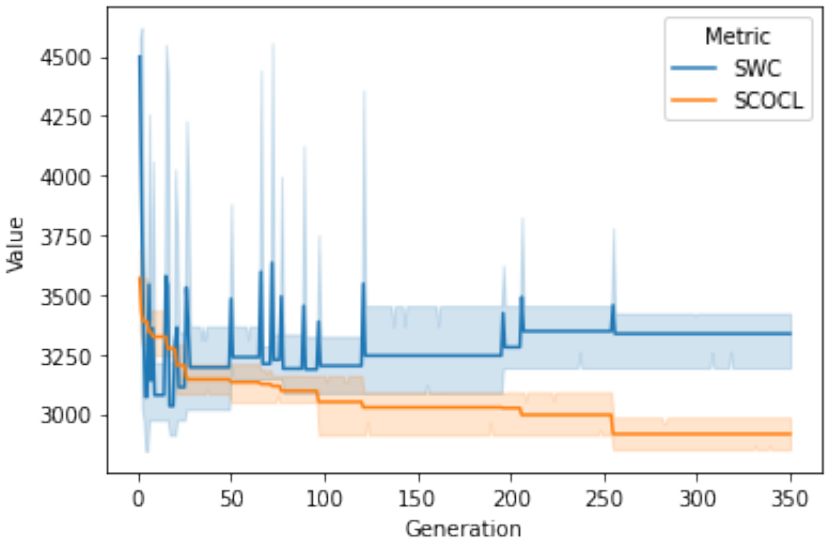

Figure 4.21: *Number of students a ected in virtual courses*

The results time spent was around 7946 seconds per execution. The results of the first genetic algorithm were utilized as non-usable or restricted time slots in the gene space calculation for the face-to-face GA. In other words, it represents the time slot used for a career-level, professor, or virtual component that restricts the timetabling for the face-to-face component.

The reduced number of face-to-face courses results in a CWC and COLC value equal to zero. However, the CD and SCD contain values di erent from zero. Figure 4.23 shows the evolution through the generation. The minimization of the number of tentative students is shown in Figure 4.22 and the related value of CD is shown in Figure 4.23.

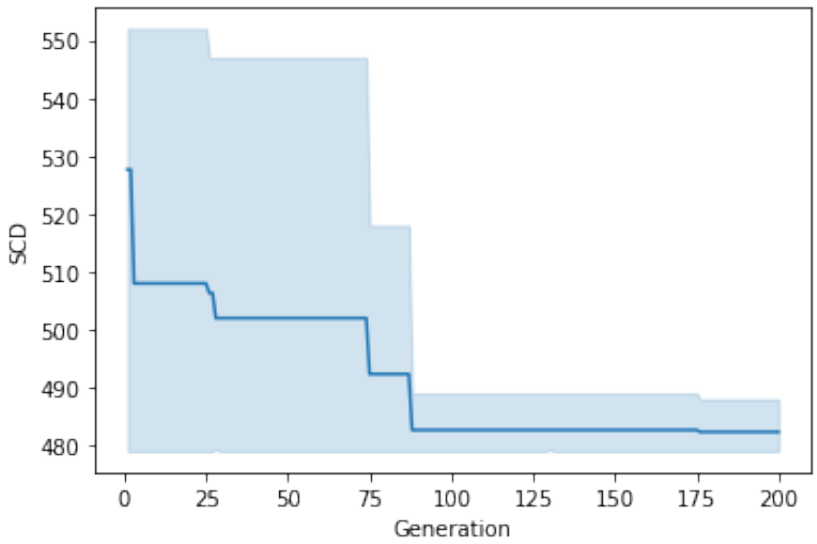

Figure 4.22: *Number of tentative students with consecutive days in face-to-face courses*

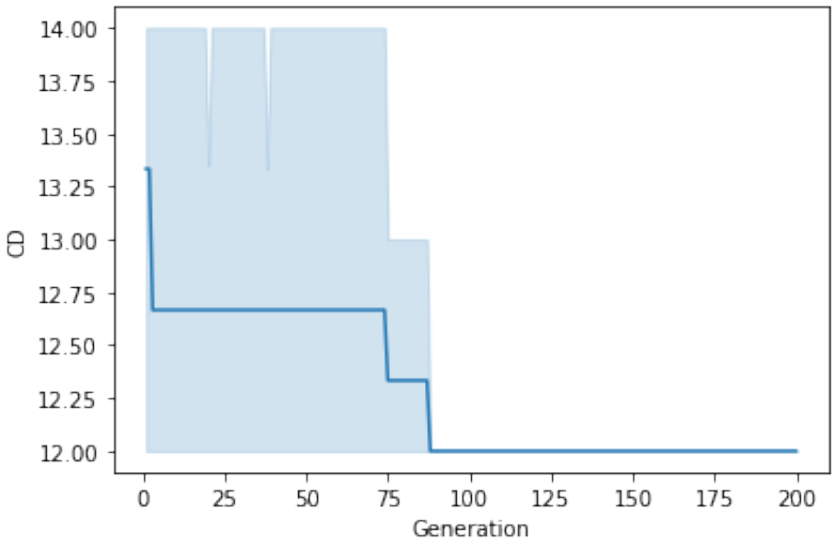

Figure 4.23: *Number of courses with consecutive days in face-to-face courses*

The time spent by execution was 3013 seconds. The results show that the genetic algorithm for face-to-face courses is useful. However, it is important to consider that the hardest task was the optimization of the virtual course component.

Conclusions

5

The CTT problem has multiple reported solutions, however, COVID-19 constructed a novel university distance learning course scheduling problem. The current literature is small and restricted by the case study. Metropolitan University o□ers a combination of distance and face-to-face courses. However, there is no literature attacking this problem. Moreover, the relationship caused by the combination of courses at multiple locations further complicates the problem. Therefore, the manuscript uses a multiphase methodology, which combines two multi-objective genetic algorithms that were modified to use a dynamic genetic space. The most relevant aspects of this work are:

1. The hard constraint was easily solved using the proposed methodology of dynamic gene space. In most of the cases, the hard constraint was solved in the first generation. However, it limits the genetic search space. For example, the pre-selection of teachers for a course represents an overwhelming reduction of elements of genetic space. Since a professor's schedule is not repeatable, and the professor may be related to multiple courses, the course is also related to multiple careers. As a result, the assigned professors are no longer available for the linked career levels.

2. The soft constraint besides the hard constraint needs multiple generations to minimize the priority metrics values. The reductions of the value as a high priority cause on many occasions an increase in other metric values. In most cases, multiple executions return di□erently. Therefore, It is extremely recommended to execute the genetic algorithm multiple times to reach the best fitness values

3. The division and combination of the problem in di□erent task results are useful for the case test scenarios. However, the gene space of the second Genetic Algorithm must consider the results obtained from the first one. Thus, the results of face-to-face courses had a reduced search gene space. Nevertheless, the results were e□cient since there was a limited number of courses in the current dataset for the face-to-face courses.

4. A classic GA is not feasible to solve the manuscript problem using the methodology proposed in this work. However, the dynamic gene space GA shows its e□ciency in solving the hard constraint and minimizing the soft constraints.

5. The face-to-face genetic algorithm was built based on the virtual genetic algorithm, which

implies that the algorithm satisfies the requirement of high adaptability to new policies and internal policy changes. The conversion of the virtual genetic algorithm to the face-to-face genetic algorithm was not a complicated task.

Limitations & Future prospects

6

6.1 Limitations

This work was limited by the computational power for the experiments since the process of each generation and the calculation of the gene space spend a high computational power. Even more, the lack of a bibliography of high related to the type of problem was a threat to finding a solution. The internal policy of the university of the static selection of professors exponentially reduces the gene space. Moreover, the unequal hourly load produces a major number of overlaps.
Another limitation is the dataset provided by the university which contains only two-semester for experimental purposes. Due to the generation of courses by semester being strict depending on the last semester, the work could not explore a robust case that contains face-to-face components. Furthermore, the artificial generation of the data could represent an ine□cient task, since the courses per semester do not have a pattern, and each semester the internal policies are altered.

6.2 Future prospects

Since the database is extremely related to multiple entities, it is a clever idea to board this problem using no relational identities to generate the gene space faster.

The exploration of di□erent methods and metrics, like taboo search and Integer programming could represent another type of solution. Novel solutions could be combined in the phase to solve potential parts of the problem.

The next step in this work will be an exploration of the data set that includes a major number of face-to-face hours.

Glosary

CB-CTT Curriculum-Based Course Timetabling.

CHC1 Common Hard Constraint 1.

CHC2 Common Hard Constraint 2.

CHC3 Common Hard Constraint 3.

CHC4 Common Hard Constraint 4.

CSC1 Common Soft Constraint 1.

CSC2 Common Soft Constraint 2.

CTT University Course Timetabling.

ETT Examination Timetabling.

FHC1 Face-to-face Hard Constraint 1.

FHC2 Face-to-face Hard Constraint 2.

FHC3 Face-to-face Hard Constraint 3.

FHC4 Face-to-face Hard Constraint 4.

FSC1 Face-to-face Soft Constraint 1.

FSC2 Face-to-face Soft Constraint 2.

GA Generict Algorithm.

HTT School Course Timetabling.

ITC International Timetabling Competition.

M-CTT Multi-phase Course Timetabling.

MPI Mixed-Integer Programming.

PE-CTT Post-Enrolment Course Timetabling.

SVLMS Structured Virtual Learning Management System.

UMET Metropolitan University of Ecuador.

VSC1 Virtual Soft Constraint 1.

VSC2 Virtual Soft Constraint 2.

Main code script

A

```
from re import A
from sql import *
import random
import copy
import numpy as np
from pygad import GA
from collections import Counter

pel=56
mutation_num_genes=20
SYNC=get_materia_s_pel(pel) #indeces materias

HORARIO_SYNC=get_horario_s(pel)
DATOS_SYNC=get_datos_s_pel(pel)
INFO_SYNC=get_info_s_pel(pel)
DOCENTE_SYNC=get_doc_s_pel(pel)
MAT_AULAS_HOURS_CONTRAINT=[]
MAT_DOC_CONTRAINT=[]

SHAPE=(SYNC['OFG_NUMERO'].size,5)

CONSTRAINT=[]
DURATION=[]

INVERVALOS_DIAS={}
HORARIO_SYNC.HORAFINAL = HORARIO_SYNC.HORAFINAL -1
HORARIO_SYNC.HORAINICIAL = HORARIO_SYNC.HORAINICIAL -1
HORARIO_SYNC.DIADESDE = HORARIO_SYNC.DIADESDE -1
HORARIO_SYNC.DIAHASTA = HORARIO_SYNC.DIAHASTA -1
acc=0

MAX_HORAS=24

def non_consecutive_combinator(rnge, r, prev=[]):
    if r == 0:
        yield prev

    else:
        for i, item in enumerate(rnge):
            for next_comb in non_consecutive_combinator(rnge[i+2:], r-1, prev+[item]):
                yield next_comb


for dia,intervalo in enumerate(range(0,7)):
    INVERVALOS_DIAS[dia]=list(range(acc,acc+15))
```

```
    acc=acc+MAX_HORAS

def clean_hours(int_list,length):
    return [int_list[i] for i in range(0, len(int_list))
            if sum(np.diff(int_list[i:i+length]))==length-1]
def check_option(x,w):
    c=0
    if (x[0]==0 and x[-1]==6):
        return False
    for i in x:
        if w[i]==0.:
            return False
        if w[i]==0.5:
            c+=1
    if c>1:
        return False
    else:
        return True
def seleccion_horario(dias,horas,domain):
    list_days=map(split_day_week,domain)
    b_hour,b_day,c=-1,0,0
    days_count=np.zeros((7))
    days={0:[],1:[],2:[],3:[],4:[],5:[],6:[]}
    for day,hour in list_days:
        hour=join_day_week(day,hour)
        days[day].append(hour)
        if b_hour+1==hour and b_day==day:
            c+=1
        else:
            c=1
        b_hour,b_day=hour,day
        if c>days_count[day]:
            days_count[day]=c
    max_hours=list(filter(lambda x:x>=max(horas),days_count))
    result,selected_day=False,None
    if len(max_hours)>=horas.count(max(horas)) and \
        len(list(filter(lambda x:x>=min(horas),days_count)))-len(max_hours)>=1 \
        or len(max_hours)>=dias:
        result=True

    if result:
        if dias<4:
            max_hours=(days_count>=max(horas)).astype(int)
            prob_hours=max_hours+(max_hours-(days_count>=min(horas)
                                             ).astype(int)==-1).astype(int)*0.5
            desicion=non_consecutive_combinator([0,1,2,3,4,5,6], dias)
            days_list=list(filter(lambda x: check_option(x,prob_hours),desicion))

            if days_list:
                selected_day=random.choice(days_list)
                selected_day=list(filter(lambda x:prob_hours[x]==1,selected_day))+\
                    list(filter(lambda x:prob_hours[x]==0.5,selected_day))
        if not selected_day:
            options=np.where(days_count>=max(horas))[0]
            selected_day=random.sample(list(options),dias-1)
            options_min=list(set(np.where(days_count>=min(horas))[0])
                             -set(selected_day))
            selected_day=selected_day+[random.choice(options_min)]
```

```
            horario=[]
            for i,day in enumerate(selected_day):
                horario.append(random.choice(clean_hours(days[day],horas[i])))
            if horario:
                return horario

    return [-666]

def get_horas_dias(t_horas):
    if t_horas<=3:
        dias=1
        horas=[t_horas]
    elif t_horas==4:
        dias=2
        horas=[2,2]
    elif t_horas==5:
        dias=2
        horas=[3,2]
    elif t_horas==6:
        dias=2
        horas=[3,3]
    elif t_horas==7:
        dias=3
        horas=[3,2,2]
    elif t_horas==8:
        dias=3
        horas=[3,3,2]
    elif t_horas==9:
        dias=3
        horas=[3,3,3]
    elif t_horas==11:
        dias=4
        horas=[3,3,3,2]
    elif t_horas==12:
        dias=4
        horas=[3,3,3,3]
    elif t_horas==13:
        dias=5
        horas=[3,3,3,3,1]
    elif t_horas==14:
        dias=5
        horas=[3,3,3,3,2]
    elif t_horas==15:
        dias=5
        horas=[3,3,3,3,3]
    return dias,horas
def obtener_intervalos(ofg_numero,schedule_lvl,schedule_carr,schedule_doc,limit=0):
    info=INFO_SYNC.loc[INFO_SYNC['OFG_NUMERO']==ofg_numero][
        ['CAR_CODIGO','MAA_NIVEL']].drop_duplicates()[limit:]
    domain=[]
    domain_2=[]
    for row, elem in info.iterrows():
        domain=domain+schedule_lvl.get(str(elem.CAR_CODIGO)+'-'+str(elem.MAA_NIVEL),[])
    docente=get_docente(ofg_numero)
    if docente:
        domain=domain+schedule_doc.get(docente,[])
    return domain,domain_2
def set_list(dic,codigo,lista):
```

```
    if codigo in dic:
        dic[codigo]+=lista
    else:
        dic[codigo]=lista

def save_best_solution_sync(data):
    data[data<-999]=data[data<-999]/1000
    data[data<0]=-1*data[data<0]
    datos=[]
    for index,row in SYNC.iterrows():

            dias,horas=get_horas_dias(row.DID_HORAS_SINCRONICAS)
            for num_dia,i in enumerate(filter(lambda x: x>0,horas)):
                info=DATOS_SYNC.loc[DATOS_SYNC['OFG_NUMERO']==row.OFG_NUMERO]
                dia,hora=split_day_week(data[index][num_dia])
                for row2,values in info.iterrows():
                    for time in range(hora+1,hora+i+1):
                        datos.append({'OFG_NUMERO':int(values.OFG_NUMERO),
                        'CAR_CODIGO':int(values.CAR_CODIGO),
                        'MAA_NIVEL':int(values.MAA_NIVEL)
                        ,'HDD_DIA':int(dia+1),
                        'HDD_HORA_INICIAL':int(time),
                        'HDD_HORA_FINAL':int(time+1),
                        'FAC_CODIGO':int(values.FAC_CODIGO),
                        'PRA_NUMERO':int(values.PRA_NUMERO),
                        'CAM_CODIGO':int(values.CAM_CODIGO),
                        'SED_CODIGO':int(values.SED_CODIGO),
                        'MAT_CODIGO':int(values.MAT_CODIGO[0]),'PEL_CODIGO':pel,
                        'DOCENTE_AUTOR':int(values.DOCENTE_AUTOR)
                        if not np.isnan(values.DOCENTE_AUTOR) else 0,
                        'DID_CODIGO':int(values.DID_CODIGO)})
    return pd.DataFrame(datos)
def set_horarios(ofg_numero,horas,schedule_lvl,schedule_carr,schedule_ofg,schedule_doc):
    info=INFO_SYNC.loc[INFO_SYNC['OFG_NUMERO']==ofg_numero][['CAR_CODIGO','MAA_NIVEL']].drop_duplicates()
    docente=get_docente(ofg_numero)
    if docente:
        set_list(schedule_doc,docente,horas.copy())
    set_list(schedule_ofg,str(ofg_numero),horas.copy())
    for row, elem in info.iterrows():
        set_list(schedule_lvl,str(elem.CAR_CODIGO)+'-'+str(elem.MAA_NIVEL),horas.copy())

        #set_list(schedule_carr,str(elem.CAR_CODIGO),horas)


def get_probability(todos,values_to_select_from):
    freq=[todos.count(i) for i in  values_to_select_from]
    freq=max(freq)-np.array(freq)
    if sum(freq)!=0:
        freq=freq/sum(freq)
    return freq

def get_docente(ofg_numero):
    docente=DOCENTE_SYNC[DOCENTE_SYNC['OFG_NUMERO']==ofg_numero]
    if not docente.empty:
        docente=docente.iloc[0]['DOCENTE_AUTOR']
        if not np.isnan(docente):
            return str(int(docente))
```

```
    return False
def remove_domains(ofg_numero,schedule_lvl,schedule_carr,
                   schedule_doc,dias,horas,intv2,domain):
    c=0
    docente=get_docente(ofg_numero)
    aux_domain=domain.copy()
    horario=None
    if docente:
        tmp_domain=schedule_doc.get(docente,[])
        horas_disponibles2=intv2 - set(tmp_domain)
        horario=seleccion_horario(dias,horas,horas_disponibles2)
    else:
        tmp_domain=[]
    if -666 not in horario:
        return True,False,horario
    if not tmp_domain:
        while(tmp_domain!=domain):
            c=c + 1
            domain,_=obtener_intervalos(ofg_numero,schedule_lvl,schedule_carr,
                                        schedule_doc,c)
            if domain != aux_domain:
                horas_disponibles2=intv2 - set(domain)
                horario=seleccion_horario(dias,horas,horas_disponibles2)
                aux_domain=domain.copy()
            if -666 not in horario:
                return True,False,horario
    else:
        return False, True, horario
def select_day(ofg_numero,t_horas,schedule_ofg,schedule_lvl,schedule_carr,schedule_doc):
    dias,horas=get_horas_dias(t_horas)
    domain,_=obtener_intervalos(ofg_numero,schedule_lvl,schedule_carr,schedule_doc)
    intv_completo=set(np.array([INVERVALOS_DIAS[i]
                                for  i in INVERVALOS_DIAS]).flatten())
    intv_param=set(intervalo_dias(ofg_numero))
    horas_disponibles_todo=intv_completo - set(domain) #horas disponibles sin tomar encuenta la parametrizacion
    horas_disponibles_param=intv_param - set(domain) #horas disponibles sin tomar en cuenta la parametrizacion
    penalizar_no_param=False
    penalizar_cruze_nivel=False
    horario=seleccion_horario(dias,horas,horas_disponibles_param)
    if -666 in horario:
        horario=seleccion_horario(dias,horas,horas_disponibles_todo) #toma encuenta la parametrizacion y salto de dias
        penalizar_no_param=True
    #Admite cruzes de horarios
    if -666 in horario:
        penalizar_cruze_nivel,penalizar_no_param,horario=remove_domains(
            ofg_numero,schedule_lvl,schedule_carr,schedule_doc,dias,
            horas,intv_completo,domain)

    if -666 not in horario:
        for x,hora_selecionada in enumerate(np.absolute(horario)):
            horas_horario=range(hora_selecionada,hora_selecionada+horas[x])
            set_horarios(ofg_numero,list(horas_horario),schedule_lvl,schedule_carr,schedule_ofg, schedule_doc)
        if penalizar_no_param:
            horario=list(-1*np.array(horario)) #se detectica que no hizo uso de la parametrizacion
        if penalizar_cruze_nivel:
            horario=list(np.array(horario)) #se detectica que no hizo uso de la parametrizacion

    return horario
```

```
def intervalo_dias(ofg_numero):
    """
    Busca obtiene el conjunto de intervalos paramatrizados
    """
    intervalo=[]
    horarios_busqueda=HORARIO_SYNC[HORARIO_SYNC['OFG_NUMERO']==ofg_numero]
    for  row,horario_busqueda in horarios_busqueda.iterrows():
        inicio=horario_busqueda['DIADESDE']
        fin=horario_busqueda['DIAHASTA']
        hora_ini=horario_busqueda['HORAINICIAL']
        hora_fin=horario_busqueda['HORAFINAL']
        for dia in range(inicio,fin+1):
            intervalo+=list(range(hora_ini+(dia*MAX_HORAS),(dia*MAX_HORAS)+hora_fin))
    return intervalo

def join_day_week(day,hour):
    return day*MAX_HORAS+hour
def split_day_week(value):
    day=value//MAX_HORAS
    time=value%MAX_HORAS
    return int(day),int(time)

def fitness_func(solution, solution_idx):
    actual_populations=solution.reshape(*SHAPE)
    return validate_data(actual_populations)

def check_docente(did_horas_sincronicas,ofg_numero,schedule_doc,schedule_lvl,pobla):
    _,horas=get_horas_dias(did_horas_sincronicas)
    docente=get_docente(ofg_numero)
    horario_docente=schedule_doc.get(docente,[])
    info=INFO_SYNC.loc[INFO_SYNC['OFG_NUMERO']==ofg_numero][['CAR_CODIGO','MAA_NIVEL']].drop_duplicates()
    lista_cruzes_lvl=[]
    for row, elem in info.iterrows():
        lista_cruzes_lvl=lista_cruzes_lvl+\
            schedule_lvl.get(str(elem.CAR_CODIGO)+'-'+str(elem.MAA_NIVEL),[])
    for i, hora in enumerate(horas):
        horario=list(range(int(pobla[i]),int(pobla[i])+hora))
        for j in horario:
            if j in horario_docente:
                return True
            if j in lista_cruzes_lvl:
                return True
    return False
def set_data(did_horas_sincronicas,ofg_numero,schedule_lvl,
             schedule_carr,schedule_ofg, schedule_doc,pobla):
    _,horas=get_horas_dias(did_horas_sincronicas)
    for i, hora in enumerate(horas):
        horario=list(range(int(pobla[i]),int(pobla[i])+hora))
        set_horarios(ofg_numero,horario,schedule_lvl,schedule_carr,
                     schedule_ofg, schedule_doc)

def get_cruzes_docente(did_horas_sincronicas,schedule_doc,pobla,docente):
    _,horas=get_horas_dias(did_horas_sincronicas)
    for i, hora in enumerate(horas):
        horario=list(range(int(pobla[i]),int(pobla[i])+hora))
        for j in horas:
            if j in schedule_doc[docente]:
```

```
                return -10000
def validate_data(actual_populations):
    mal_horario=np.where((np.sum(actual_populations,axis=1)<-1)&(np.sum(actual_populations,axis=1)>-1000))[0]
    c_data= save_best_solution_sync(actual_populations.copy())
    prof_mal=-1000000 *len(c_data[['HDD_HORA_INICIAL','HDD_DIA','DOCENTE_AUTOR',
                                'OFG_NUMERO']].drop_duplicates().groupby(
                                ['DOCENTE_AUTOR','HDD_HORA_INICIAL','HDD_DIA'])
                                .filter(lambda x: len(x) > 1)['OFG_NUMERO']
                                .drop_duplicates().values)
    ofgs=c_data[['OFG_NUMERO','CAR_CODIGO','MAA_NIVEL',
                 'HDD_HORA_INICIAL','HDD_DIA']].drop_duplicates().groupby(
    ['CAR_CODIGO','MAA_NIVEL','HDD_DIA','HDD_HORA_INICIAL']
    ).filter(lambda x: len(x) > 1)['OFG_NUMERO'].drop_duplicates()
    ofgs_day_bad=c_data[c_data['HDD_DIA']>7]['OFG_NUMERO'].values
    index=list(SYNC.index[SYNC.OFG_NUMERO.isin(ofgs)].values)
    actual_populations[index]=-1*actual_populations[index]
    index=list(SYNC.index[SYNC.OFG_NUMERO.isin(ofgs_day_bad)].values)
    if index:
        actual_populations[index][0]=-666
    return -len(np.where(np.any(actual_populations==-666,axis=1))[0]
                )*1000000 -SYNC.loc[mal_horario]['CUPO_SUBGRUPO'].sum()\
        -100*SYNC[SYNC.OFG_NUMERO.isin(set(ofgs))].CUPO_SUBGRUPO.sum()+prof_mal-len(
            index)*100000



def start_random_population():
    population=np.zeros(SHAPE)
    create_schedule(population,flag=True)
    return population

def create_schedule(actual_populations,flag=False):
    schedule_ofg={}
    schedule_carr={}
    schedule_lvl={}
    schedule_doc={}

    mutation_indices = random.sample(list(range(0, SHAPE[0])), mutation_num_genes)
    for index,row in SYNC.iterrows():
        chk_docente=check_docente(row.DID_HORAS_SINCRONICAS,row.OFG_NUMERO,
                    schedule_doc,schedule_lvl,np.absolute(actual_populations[index]))
        if flag or index in mutation_indices or  np.any(actual_populations[index] <0) \
            or chk_docente:
            result_days=select_day(row.OFG_NUMERO,row.DID_HORAS_SINCRONICAS,
                        schedule_ofg,schedule_lvl,schedule_carr,schedule_doc)
            for i,val in enumerate(result_days):
                actual_populations[index][i]=val
        else:
            set_data(row.DID_HORAS_SINCRONICAS,row.OFG_NUMERO,schedule_lvl,
                     schedule_carr,schedule_ofg, schedule_doc,
                     np.absolute(actual_populations[index]))
    return actual_populations

class GACustomize(GA):
    def random_mutation(self, offspring):
        for offspring_idx in range(offspring.shape[0]):
            create_schedule(offspring[offspring_idx].reshape(SHAPE))
        return offspring
```

```
inputs=[start_random_population().flatten() for i in range(5)]

GANN_customize = GACustomize(num_generations=500,
                             sol_per_pop=5,
                             num_parents_mating=2,
                             fitness_func=fitness_func,
                             num_genes=len(inputs),
                             mutation_type="random",
                             initial_population=inputs,
                             keep_parents=0,
                             save_best_solutions=True,
                             mutation_num_genes=15
                             )
GANN_customize.run()
```

Bibliography

Abduljabbar, I. A. y Abdullah, S. M. (2022a). An evolutionary algorithm for solving academic courses timetable scheduling problem. *Baghdad Science Journal*, 19(2):0399.

Abduljabbar, I. A. y Abdullah, S. M. (2022b). An evolutionary algorithm for solving academic courses timetable scheduling problem. *Baghdad Science Journal*, 19(2):0399–0399.

Abdullah, S., Burke, E. K., y McCollum, B. (2007). A hybrid evolutionary approach to the university course timetabling problem. In *2007 IEEE Congress on Evolutionary Computation*, pages 1764–1768.

Abdullah, S., Turabieh, H., Mccollum, B., y McMullan, P. (2012a). A hybrid metaheuristic approach to the university course timetabling problem. *J. Heuristics*, 18:1–23.

Abdullah, S., Turabieh, H., McCollum, B., y McMullan, P. (2012b). A hybrid metaheuristic approach to the university course timetabling problem. *Journal of Heuristics*, 18:1–23.

Barnhart, C., Bertsimas, D., Delarue, A., y Yan, J. (2022). Course scheduling under sudden scarcity: Applications to pandemic planning. *Manuf. Serv. Oper. Manag.*, 24:727–745.

Battistutta, M., Ceschia, S., Cesco, F. D., Gaspero, L. D., y Schaerf, A. (2019). Modelling and solving the thesis defense timetabling problem. *Journal of the Operational Research Society*, 70(7):1039–1050.

Carter, M. W., Laporte, G., y Lee, S. Y. (1996). Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society*, 47(3):373–383.

Ceschia, S., Di Gaspero, L., y Schaerf, A. (2022a). Educational timetabling: Problems, benchmarks, and state-of-the-art results. *arXiv preprint arXiv:2201.07525*.

Ceschia, S., Di Gaspero, L., y Schaerf, A. (2022b). Educational timetabling: Problems, benchmarks, and state-of-the-art results. *arXiv preprint arXiv:2201.07525*.

Cruz-Rosales, M. H., Cruz-Chávez, M. A., Alonso-Pecina, F., Peralta-Abarca, J. d. C., Ávila-Melgar, E. Y., Martínez-Bahena, B., y Enríquez-Urbano, J. (2022a). Metaheuristic with cooperative processes for the university course timetabling problem. *Applied Sciences*, 12(2):542.

Cruz-Rosales, M. H., Cruz-Chávez, M. A., Alonso-Pecina, F., Peralta-Abarca, J. d. C., Ávila Melgar, E. Y., Martínez-Bahena, B., y Enríquez-Urbano, J. (2022b). Metaheuristic with cooperative processes for the university course timetabling problem. *Applied Sciences*, 12(2).

Dorado-Sevilla, D. F., Pelu□o-Ordóñez, D. H., Lorente-Leyva, L. L., Herrera-Granda, E. P., y Herrera-Granda, I. D. (2021). An interactive framework to compare multicriteria optimization algorithms: Preliminary results on nsga-ii and mopso. In Bindhu, V., Tavares, J. M. R. S., Boulogeorgos, A.-A. A., y Vuppalapati, C., editors, *International Conference on Communication, Computing and Electronics Systems*, pages 61–76, Singapore. Springer Singapore.

Esmaeilbeigi, R., Mak-Hau, V., Yearwood, J., y Nguyen, V. (2022). The multiphase course timetabling problem. *European Journal of Operational Research*, 300(3):1098–1119.

Espinoza Cordero, C. X., Socorro Castro, A. R., Soler McCook, J. M., Hernández Toazo, H., y Guerra Maldonado, C. P. (2020). Sistema estructurado de gestión del aprendizaje virtual de la universidad metropolitana del ecuador. *Revista Universidad y Sociedad*, 12(5):404–414.

Esquivel, L. y Lucía, L. (2014). Modelo matemático para la programación de un horario escolar con multilocalización de docentes. *Maestría en Ingeniería Industrial. Universidad del Valle, Cali, Colombia*.

Even, S., Itai, A., y Shamir, A. (1975). On the complexity of time table and multi-commodity flow problems. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pages 184–193.

Gad, A. F. (2021). Pygad: An intuitive genetic algorithm python library. *arXiv preprint arXiv:2106.06158*.

Gashi, E. y Sylejmani, K. (2019). Simulated annealing with penalization for university course timetabling.

Hossain, S. y Zibran, M. (2007). A multi-phase approach to the university course timetabling problem. pages 73–76.

Huang, Q. y Wang, Y. (2022). Application of genetic algorithm in university teaching management system. In *Innovative Computing*, pages 613–620. Springer.

Kruskal, W. H. y Wallis, W. A. (1952). Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, 47(260):583–621.

Lewis, R. y Thompson, J. (2015). Analysing the e□ects of solution space connectivity with an e□ective metaheuristic for the course timetabling problem. *European Journal of Operational Research*, 240(3):637–648.

McKinney, W. et al. (2010). Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX.

Mühlenthaler, M. y Wanka, R. (2013). Fairness in academic course timetabling. *CoRR*, abs/1303.2860.

Müller, T., Rudová, H., y Müllerová, Z. (2018). University course timetabling and international timetabling competition 2019. In Proceedings of the International Timetabling Competition 2019.

Nagata, Y. (2018). Random partial neighborhood search for the post-enrollment course timetabling problem. *Comput. Oper. Res.*, 90(C):84–96.

Obaid, O. I., Ahmad, M., Mostafa, S. A., y Mohammed, M. A. (2012). Comparing performance of genetic algorithm with varying crossover in solving examination timetabling problem. *J. Emerg. Trends Comput. Inf. Sci*, 3(10):1427–1434.

Photo, M. (2013). University course timetable using multiphase genetic algorithm : a case study of bindura university. Bindura University of Science Education.

Post, G., Ahmadi, S., Daskalaki, S., Kingston, J., Kyngas, J., Nurmi, C., y Ranson, D. (2012). An xml format for benchmarks in high school timetabling. *Annals of operations research*, 194(1):385–397. eemcs-eprint-19627.

Rappos, E., Thiémard, E., Robert, S., y Hêche, J.-F. (2022). A mixed-integer programming approach for solving university course timetabling problems. *Journal of Scheduling*.

Şimşek, A. B. (2021). A course timetabling formulation under circumstances of online education. *Journal of Turkish Operations Management*, 5:781 – 791.

Sun, Y., Luo, X., y Liu, X. (2021). Optimization of a university timetable considering building energy e□ciency: An approach based on the building controls virtual test bed platform using a genetic algorithm. *Journal of Building Engineering*, 35:102095.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., y SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

Waskom, M. L. (2021). seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021.

Welahetti, L. y Samarathunga, D. (2022). A binary linear programming model for a case study in university timetabling problem.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. biometrics bulletin, 1 (6), 80-83.

Zheng, H., Peng, Y., Guo, J., y Chen, Y.-C. (2022). Course scheduling algorithm based on improved binary cuckoo search. *The Journal of Supercomputing*, pages 1–26.