FEDERAL UNIVERSITY OF SANTA CATARINA
TECHNOLOGY CENTER
AUTOMATION AND SYSTEMS DEPARTMENT
UNDERGRADUATE COURSE IN CONTROL AND AUTOMATION ENGINEERING

Torben Castro Borba

Composition and scheduling of project defense committees: mathematical programming modeling, user interface, and experiments.

Florianópolis, Brazil
2023

Torben Castro Borba

Composition and scheduling of project defense committees: mathematical programming modeling, user interface, and experiments.

Final report of the subject DAS5511 (Course Final Project) as a Concluding Dissertation of the Undergraduate Course in Control and Automation Engineering of the Federal University of Santa Catarina.
Supervisor: Prof. Eduardo Camponogara

Florianópolis, Brazil
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Borba, Torben Castro
Composition and scheduling of project defense committees : model in mathematical programming, user interface, and experiments. / Torben Castro Borba ; orientador, Eduardo Camponogara, 2023.
68 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia de Controle e Automação, Florianópolis, 2023.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Otimização Matemática. 3. Timetable. 4. Modelagem. I. Camponogara, Eduardo. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. III. Título.

Torben Castro Borba

Composition and scheduling of project defense committees: mathematical programming modeling, user interface, and experiments.

This dissertation was evaluated in the context of the subject DAS5511 (Course Final Project) and approved in its final form by the Undergraduate Course in Control and Automation Engineering

Florianópolis, July 11, 2023.

This work is dedicated to my classmates and my dear parents.

ACKNOWLEDGEMENTS

I would like to extend my gratitude to Prof. Eduardo Camponogara for his invaluable advice and support throughout the development of this work. Our meetings significantly aided in the development of both the model and this report.

“Remember that all models are wrong; the practical question is how wrong they have to be to not be useful.”
(George Box, 1976)

DECLARAÇÃO DE PUBLICIDADE

Florianópolis, 02 de Julho de 2023.

Na condição de representante do Departamento de Automação e Sistemas (DAS) na qual o presente trabalho foi realizado, declaro não haver ressalvas quanto ao aspecto de sigilo ou propriedade intelectual sobre as informações contidas neste documento, que impeçam a sua publicação por parte da Universidade Federal de Santa Catarina (UFSC) para acesso pelo público em geral, incluindo a disponibilização *online* deste documento no Repositório Institucional da Biblioteca Universitária da UFSC. Além disso, declaro ciência de que o autor, na condição de estudante da UFSC, é obrigado a depositar este documento, por se tratar de um Trabalho de Conclusão de Curso, no referido Repositório Institucional, em atendimento à Resolução Normativa n° 126/2019/Cun.

Por estar de acordo com esses termos, subscrevo-me abaixo.

ICP-Edu
Documento assinado digitalmente
Eduardo Camponogara
Data: 13/07/2023 19:50:02-0300
CPF: ***.988.180-**
Verifique as assinaturas em https://v.ufsc.br

Eduardo Camponogara
Departamento de Automação e Sistemas

ABSTRACT

This study seeks to resolve the complex problem of efficiently designing project defense committees and scheduling their events. In more details, it investigates the constraints and preferences of all stakeholders - students, advisors, supervisors, examiners, and chairs - and the influence these factors have on committee design and event scheduling. The currently manual and resource intensive task of forming committees and planning project defenses poses significant logistical issues. The method often leads to sub-optimal results, including scheduling conflicts, overload of responsibilities on committee members, and compromised quality of evaluations, among other issues. These inefficiencies highlight the necessity for an automated solution. To this end, the project adopts mathematical programming, a technique capable of providing optimal solutions to these kinds of intricate problems. The methodology used involved developing a user-friendly data collection application utilizing Google Workspace apps, including Google Forms, Google Sheets, and Google Apps Script. This system handles user data, pre-processes it, and transfers it to a computational system, which was implemented using the Julia programming language with the JuMP package for mathematical optimization. The model was applied as a case study to the first semester of 2023, managing to successfully schedule 17 out of 18 project defense events. The study highlights that the single unsuccessful case was due to the infeasibility of overlapping schedules among the involved individuals. Therefore the system proved to be a tremendous enhancement to the previous manual process, reducing the workload for event organizers, and demonstrating its suitability for future semesters. This comprehensive solution to the project defense scheduling problem not only offers immediate relief to event organizers but also presents a scalable model that can be adapted to suit future needs.

Keywords: Mathematical Optimization. Modeling. Timetable Problems.

RESUMO

Este estudo busca resolver o complexo problema de projetar comitês de defesa de projetos e programar seus eventos de forma eficiente. Especificamente, investiga as restrições e preferências de todas as partes interessadas - estudantes, orientadores, supervisores, examinadores e presidentes - e a influência desses fatores no design do comitê e na programação de eventos. A tarefa atualmente manual e intensiva em recursos de formação de comitês e planejamento de defesas de projetos apresenta significativos problemas logísticos. O método frequentemente leva a resultados subótimos, incluindo conflitos de agenda, sobrecarga de responsabilidades nos membros do comitê e comprometimento da qualidade das avaliações, entre outros problemas. Essas ineficiências destacam a necessidade de uma solução automatizada. Para isso, o projeto adota a programação matemática, uma técnica capaz de fornecer soluções ótimas para esses tipos de problemas complexos. A metodologia utilizada envolveu o desenvolvimento de um aplicativo de coleta de dados amigável ao usuário, utilizando aplicativos do Google Workspace, incluindo Google Forms, Google Sheets e Google Apps Script. Este sistema lida com dados do usuário, pré-processa-os e os transfere para um sistema computacional, que foi implementado usando a linguagem de programação Julia com o pacote JuMP para otimização matemática. O modelo foi aplicado como estudo de caso ao primeiro semestre de 2023, conseguindo programar com sucesso 17 dos 18 eventos de defesa de projeto. O estudo destaca que o único caso de insucesso foi devido à inviabilidade de sobreposição de agendas entre os indivíduos envolvidos. Portanto, o sistema provou ser uma melhoria tremenda em relação ao processo manual anterior, reduzindo a carga de trabalho dos organizadores de eventos e demonstrando sua adequação para semestres futuros. Esta solução abrangente para o problema de agendamento de defesa de projeto não apenas oferece alívio imediato aos organizadores de eventos, mas também apresenta um modelo escalável que pode ser adaptado para atender às necessidades futuras.

Palavras-chave: Otimização Matemática. Modelagem. Problemas de Timetable.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

In the modern world, many tasks are primed for automation or optimization. However, these tasks are often still conducted manually or semi-manually, incurring considerable stress and leading to substandard outcomes. One such task is the organization and scheduling of academic project defenses. This work aims to present an automated, optimized solution for this problem, considering the constraints and preferences of candidates, examiners, chairpersons or simply chairs, supervisors, and advisors.

Project defenses are formal events where students present their project to a panel of experts, serving as a platform to demonstrate their understanding of their research work. These events allow students to showcase their expertise in a specific subject formally and knowledgeably. They are often a significant component of an academic degree. The appropriate composition of a project defense committee can greatly enhance the overall event, benefiting all participants.

Every academic institution has specific requirements for scheduling and constituting project defense committees. Typically, these committees comprise at least one examiner, a chairperson, and an advisor. In this study's context, the project defense participants include the student, their advisor and supervisor, an examiner, and a chairperson.

The motivation behind addressing this problem lies in the complex and resource demanding nature of composing and scheduling project defense committees. Given the unique roles, constraints, availability, and capabilities of the participants, manual management becomes increasingly challenging as the number of events grows. This challenge substantiates the need for a computational system to automatically and optimally assign committee members and schedule events.

Problems of this nature fall into a class known as timetable problems (EVEN; ITAI; SHAMIR, 1975). These problems typically involve resources, time slots, events, constraints, and objectives. In our case, the resources are the participants, the time slots refer to the available time windows for scheduling defenses, the events are the project defense sessions, the constraints pertain to the composition of the committee, and the objective is to maximize the number of successfully composed and scheduled project defense events.

Timetable problems appear in various domains. For instance, (BABAEI; KARIMPOUR; HADIDI, 2015) analyze different approaches for the university course timetabling problem. This problem is a classical application of timetabling, where the goal is to allocate teachers, students, and courses, given limited resources such as classrooms and equipment. Other traditional applications include public transportation timetable generation (PARBO; NIELSEN; PRATO, 2016) and employee timetable scheduling (GUYON et al., 2010).

Various techniques exist to address problems of this sort, including Mathematical Programming (GUYON et al., 2010), Genetic Algorithms (BURKE; ELLIMAN; WEARE, 1994), Simulated Annealing (ABRAMSON; KRISHNAMOORTHY; DANG, et al., 1999), and other Heuristic Algorithms (CAPRARA et al., 2006). In this study, we adopt a mathematical programming formulation. This formulation ensures the synthesis of an optimal solution and an interpretable model that directly aligns with the original problem's constraints, variables, and objectives. Moreover, we have access to efficient, state-of-the-art algorithms for solving problems of this nature.

1.1 OBJECTIVES

The main objective of this work is to address the problem of composing project defense committees and scheduling their activities. In more detail, we propose a solution capable of managing the entire process, starting with the initial data collection from the users to the resolution of the optimization problem that results in the optimal timetable. This optimal timetable should align with the organizer's objectives for successful project defense events. This includes finding examiners interested in specific projects and accommodating particular requests from examiners or chairs regarding the events.

Furthermore, a secondary yet important objective is that the deployed solution should be easily maintainable and customizable, with the intent of facilitating its use in subsequent semesters.

1.2 CONTRIBUTIONS

The main contribution of this work lies in the automation of the process for composing and scheduling project defense events. This can be divided into three main components: the derivation of a mathematical programming model that faithfully represents the process; the development of a system adept at collecting, processing, and storing the necessary user data; and a computational system capable of implementing and solving the mathematical programming problem.

1.3 REPORT STRUCTURE

The document is structured into seven chapters.

Chapter 2 defines the problem of concern and its constraints in detail, illustrating them with a small-scale example.

Chapter 3 reviews the fundamental concepts necessary to comprehend this work. This includes an overview of mathematical programming, its classes, and algorithms, as well as the formalization of a timetable problem, its components, applications, and solution methods.

Chapter 4 focuses on modeling our problem using a mathematical programming formulation. This process is divided into three main steps: modeling of variables, constraints, and the objective. This chapter constitutes the most crucial part of our work.

Chapter 5 elaborates on the implementation of our solution, which is divided into two distinct segments. The first segment covers data gathering, pre-processing, storing, and analysis. The second segment focuses on solving the optimization problem and drawing conclusions from its results.

Chapter 6 showcases a case study of our work, specifically the project defense events scheduled for the first academic semester of 2023. We implement our solution and analyze the results, providing an evaluation of the events for which we could not derive a solution.

Chapter 7 concludes our work, summarizing the findings and discussing potential future directions.

2 PROBLEM FORMULATION

In the educational landscape, particularly at the higher education level, project defenses serve as a significant element in the evaluation of student's academic achievements. They are usually comprised of a committee with the primary role of assessing the student's comprehension, critical thinking, and presentation skills. The composition of project defense committees and the scheduling of these defenses, however, often poses a complex logistical challenge.

The problem lies in creating a fair and efficient system that respects the constraints and preferences of all parties involved – students (candidates), advisors and supervisors (counselors), evaluators and chairs (committee members). This challenge includes finding a compatible match between students, advisors, supervisors and committee members based on their expertise, availability and interest, also ensuring that the committee members have a manageable workload.

The current methods employed to address this problem are often manual, time-consuming, and prone to errors and inefficiencies. They may also result in sub-optimal outcomes, such as committee members being overloaded with project defenses or being assigned to evaluate projects outside their area of expertise. Similarly, scheduling conflicts can arise due to the disparate availability of the candidates, examiners, and advisers. This can lead to delays, rescheduling, and added stress for all parties involved.

This problem not only affects the logistic efficiency of academic institutions but also potentially impacts the quality of evaluations provided during project defenses, thereby affecting the overall academic output of the department. Hence, there is a critical need for a more efficient and effective solution to the design and scheduling of project defense committees. This report aims to explore this problem and develop a solution that will better accommodate the needs and constraints of all parties involved.

The problem instance that we will explore in this work concerns the composition of project defense committees in the Department of Automation and Systems Engineering at the Federal University of Santa Catarina (UFSC). The scenario we consider is comprised of a student, an advisor, a supervisor (where the advisor can also be the supervisor), an examiner, and a committee chair for each project defense event. Given that the student, advisor, and supervisor have already been determined, the problem left at hand is to allocate an evaluator, a committee chair and a suitable schedule for each project defense event, while respecting the constraints that govern the composition of the committee.

In sequence, we explore the constraints involving the process of design and composition of the defense committees.

- Each available time slot can only be allocated to one event, and similarly, each event can only be allocated to one time slot. As a result, it is ensured that at most one event is scheduled per time slot.

- To allocate an event to a time slot, it is necessary to ensure that all participants are available during that time slot. If any participant is not available, the event cannot be scheduled for that time slot.
- The examiner for an event must hold one of the following positions: UFSC professor, visiting professor, postdoctoral fellow, doctoral student or external professional.
- The committee chair of an event must be a UFSC professor from the final-project organizing committee.
- The advisor, examiner, and chair of each event must be different individuals.
- One person can evaluate more than one event, but each event can only be evaluated by one person.
- One person can chair over more than one event, but each event can only be chaired by one person.

2.1 AN EXAMPLE

In this section, we present an illustrative example that showcases the design and composition of defense committees. The purpose of this example is to demonstrate how the manual timetable process works in practice and highlight the problems that can arise during its implementation.

In this example, our objective is to schedule and organize four project defenses events. Each committee consists of one undergraduate student, one advisor, one supervisor, one examiner, and one chairperson. It is important to note that, in this scenario, we consider that only p_3 or p_4 can be assigned as the chair of a committee. The provided data for this example is presented on Tables 1, 2 and 3.

To solve the problem manually, we can generate supporting tables that illustrate the availability of personnel (see Table 4) and the potential schedules an event can take, this is the common availability of the student, advisor and supervisor (see Table 5). Utilizing these tables, we can allocate one committee at a time by examining the possible schedules for the event and then considering the availability of the individuals interested in the event. The goal is to match an examiner and chairperson who are available during the same schedule, while respecting the constraints of composition of the committee. One possible solution for the problem is shown in Table 6.

It is notable that throughout this manual process, several decisions need to be made. These decisions involve determining which available personnel should be allocated to each event and selecting the schedule for each event. The basis for these

Table 1 – People of the illustrative example.

Id	Occupation	Availability
p_1	UFSC Professor	$[s_1, s_2, s_5, s_6, s_7, s_8]$
p_2	UFSC Professor	$[s_3, s_4, s_5, s_6, s_7, s_8]$
p_3	UFSC Professor	$[s_1, s_2, s_3, s_4, s_5, s_7, s_8]$
p_4	UFSC Professor	$[s_5, s_6, s_7, s_8]$
p_5	UFSC Professor	$[s_1, s_2, s_3, s_4, s_5]$
p_6	Visiting Professor	$[s_7, s_8]$
p_7	PhD student	$[s_1, s_2, s_5, s_6]$
p_8	PhD student	$[s_2, s_3, s_4, s_5, s_6]$
p_9	Postdoctoral fellow	$[s_4, s_5]$
p_{10}	External professional	$[s_1, s_5]$
p_{11}	External professional	$[s_1, s_2, s_5, s_6]$
p_{12}	External professional	$[s_1, s_2, s_5, s_6]$
p_{13}	External professional	$[s_5, s_6, s_7, s_8]$
p_{14}	Undergraduate student	$[s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8]$
p_{15}	Undergraduate student	$[s_1, s_2, s_5, s_6]$
p_{16}	Undergraduate student	$[s_3, s_4, s_5, s_6]$
p_{17}	Undergraduate student	$[s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8]$

Source: Author.

decisions is a set of criteria that align with the objective of maximizing the successful scheduling of events. Manually executing these decisions is a time intensive iterative process, which offers no assurance of arriving at an optimal solution. To illustrate this with our concise example, should we assign the schedule s_5 to any event other than e_3, this particular event will not be able to be successfully allocated. This situation arises because the only examiner showing interest is, coincidentally, the sole available president.

In this simple example, arriving at a satisfactory solution might seem trivial. However, in a practical situation involving a substantial number of events, this is not the case. It is worth noting that in a real world scenario, a good decision may require considering numerous additional criteria, such as a fair distribution of evaluation responsibilities, among others. This provides the motivation for developing an automated solution to the problem, which is also capable of leading to an optimal solution based on the given criteria.

Table 2 – Schedule for the illustrative example.

Id	Date	Hour
s_1	15/05/2023	09:00
s_2	15/05/2023	10:00
s_3	15/05/2023	11:00
s_4	15/05/2023	12:00
s_5	16/05/2023	09:00
s_6	16/05/2023	10:00
s_7	16/05/2023	11:00
s_8	16/05/2023	12:00

Source: Author.

Table 3 – Events of the illustrative example.

Id	Author	Supervisor	Advisor	Interested
e_1	p_{14}	p_{10}	p_1	$[p_6, p_7, p_8]$
e_2	p_{15}	p_{11}	p_1	$[p_6, p_7]$
e_3	p_{16}	p_{12}	p_2	$[p_4, p_5, p_9]$
e_4	p_{17}	p_{13}	p_3	$[p_2, p_5, p_9]$

Source: Author.

Table 4 – People availability for the illustrative example.

Id	Occupation	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
p_1	UFSC Professor	🟩	🟩	🟥	🟥	🟩	🟩	🟩	🟩
p_2	UFSC Professor	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩
p_3	UFSC Professor	🟩	🟩	🟩	🟩	🟩	🟥	🟩	🟩
p_4	UFSC Professor	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩
p_5	UFSC Professor	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥
p_6	Visiting Professor	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩
p_7	PhD student	🟩	🟩	🟥	🟥	🟩	🟩	🟥	🟥
p_8	PhD student	🟥	🟩	🟩	🟩	🟩	🟩	🟥	🟥
p_9	Postdoctoral fellow	🟥	🟥	🟥	🟩	🟩	🟥	🟥	🟥
p_{10}	External professional	🟩	🟥	🟥	🟥	🟩	🟥	🟥	🟥
p_{11}	External professional	🟩	🟩	🟥	🟥	🟩	🟩	🟥	🟥
p_{12}	External professional	🟩	🟩	🟥	🟥	🟩	🟩	🟥	🟥
p_{13}	External professional	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩
p_{14}	Undergraduate student	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
p_{15}	Undergraduate student	🟩	🟩	🟥	🟥	🟩	🟩	🟥	🟥
p_{16}	Undergraduate student	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥
p_{17}	Undergraduate student	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩

Source: Author.

Table 5 – Schedule for the illustrative example.

Id	Interested	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
e_1	$[p_6, p_7, p_8]$	🟩	🟥	🟥	🟥	🟩	🟥	🟥	🟥
e_2	$[p_6, p_7]$	🟩	🟩	🟥	🟥	🟩	🟩	🟥	🟥
e_3	$[p_4, p_5, p_9]$	🟥	🟥	🟥	🟥	🟩	🟩	🟥	🟥
e_4	$[p_2, p_5, p_9]$	🟥	🟥	🟥	🟥	🟩	🟥	🟩	🟩

Source: Author.

Table 6 – Event allocations in the illustrative example.

Id	Author	Supervisor	Advisor	Evaluator	President	Schedule
e_1	p_{14}	p_{10}	p_1	p_7	p_3	s_1
e_2	p_{15}	p_{11}	p_1	p_7	p_3	s_2
e_3	p_{16}	p_{12}	p_2	p_4	p_3	s_5
e_4	p_{17}	p_{13}	p_3	p_2	p_3	s_7

Source: Author.

3 CONCEPTUAL REVIEW

3.1 MATHEMATICAL OPTIMIZATION

In this section, we delve into the realm of mathematical optimization, exploring its diverse classes and focusing on the primary algorithm pertinent to our class of greatest interest. Our review is fundamentally grounded on the work of (VANDERBEI et al., 2020).

Mathematical optimization is a field of applied mathematics concerned with calculating the best possible values for decision variables, such that an optimal behavior is induced and the constraints of the mathematical model are satisfied. This optimal outcome is achieved by either minimizing a performance criterion, such as cost, or maximizing one, such as profit. The universal language for expressing optimization problems in a declarative and consistent form is called mathematical programming.

An essential process in mathematical optimization is modeling the problem in mathematical programming. The goal of mathematical models is to represent real systems in a simplified, yet faithful manner, at least for certain situations and pre-established assumptions. Modeling is certainly not a trivial task to perform, usually taking several iterations to represent the system in the desired way. Model validation is performed through numerical analyses, simulations, and testing.

The three main components of a mathematical optimization problem are the decision variables, constraints, and the objective function. Let us introduce these concepts through a straightforward example: a factory with limited resources wishes to maximize its profit by producing the most demanded products for a particular day.

Decision variables constitute the elements that we have control over and seek to determine in the optimization problem. They represent the decisions that can be made within the given problem. In our example, these variables can be the quantities of different products to be manufactured.

Constraints are the restrictions and limitations imposed on the decision variables. They define the feasible region within which a solution must reside. In our scenario, it could be the availability of resources and the minimum quantity of each product that needs to be produced.

The objective function is the function that we are trying to optimize (either minimize or maximize) subject to the constraints. This function quantifies the goal of the optimization problem in terms of the decision variables. In this example, it could be to maximize the overall profit of the factory.

Using the mathematical programming notation, a generic optimization problem is described as:

$$\begin{aligned} \text{Maximize} \quad & f(x) \\ \text{Subject to} \quad & g(x) \geq 0 \\ & h(x) = 0 \\ & x \in \mathbb{R}^n \end{aligned}$$

where:

- $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function, the function to be minimized or maximized;
- $g : \mathbb{R}^n \to \mathbb{R}^g$ e $h : \mathbb{R}^n \to \mathbb{R}^h$ are the constraints that form the feasible solution space of the model;
- $x \in \mathbb{R}^n$ is the decision variable vector, the variables whose values define the solution of the problem.

Optimization problems can be divided into several classes, each possessing unique properties and requiring distinct solution methods. By categorizing optimization problems, we are able to better understand their mathematical structure, thus facilitating the development of specific algorithms designed to exploit these characteristics. For the scope of this work, we introduce two important classes of optimization problems: Linear Programming and Integer Programming.

3.1.1 Linear Programming

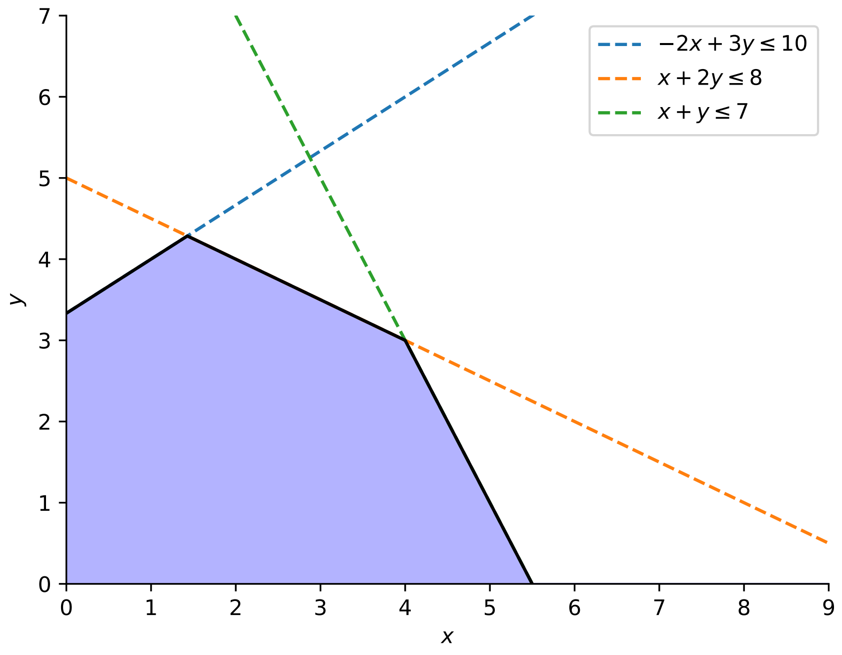
The definition of Linear Programming (LP) refers to a category of problems in which the decision variables are continuous and non-negative. Moreover, both the objective functions and constraints involved in these problems are linear functions. Using the mathematical programming notation, a generic linear programming problem is described as:

$$\begin{aligned} \text{Minimize} \quad & f(x) \\ \text{Subject to} \quad & g(x) \leq b \\ & x \geq 0 \\ & x \in \mathbb{R}^n \end{aligned}$$

where:

- $x = (x_1, x_2, ..., x_n)$ is a vector of non-negative decision variables;
- $f(x)$ and $g(x)$ are linear;

Figure 1 – Example of the feasible space of a 2-dimensional LP problem.



Source: Author.

The LP problem seeks to find the values of the decision variables that minimize/maximize the objective function, while satisfying all the linear constraints. Those constraints define the feasible region, which in the LP case, is a polytope in the n-dimensional space of decision variables. The optimal solution to the LP problem is a point within the feasible region that satisfies all the constraints and minimizes/maximizes the objective function.

In an LP problem, if a solution exists, the optimal one will always be located at one of the vertices of the feasible region, denoted by a polytope. Algorithms, such as the simplex method, are designed to efficiently traverse the vertices of the feasible region in pursuit of optimality. Figure 1 showcases a polytope representing the feasible space of a 2-dimensional LP problem.

LP problems can be effectively solved through a range of efficient algorithms, including the Simplex method and the Interior Point method. LP problems belong to the polynomial complexity class P. Although the Interior Point method is an algorithm with a worst-case polynomial complexity, the Simplex method lacks this guarantee. However, both methods remain widely used in state-of-the-art solvers, like Gurobi. The Simplex method's exponential complexity case is rare in most practical applications, which explains its continued use.

3.1.2 Integer Programming

Integer programming (IP) is a class of problems where the decision variables are required to take integer values. In other words, the solution to an IP problem must satisfy not only the constraints of the problem, but also the requirement that the decision variables take on integer values.

The general mathematical programming form of an IP problem is as follows:

$$\begin{aligned} \text{Minimize} \quad & f(x) \\ \text{Subject to} \quad & g(x) \leq b \\ & x \geq 0 \\ & x \in \mathbb{Z}^n \end{aligned}$$

where $x \in \mathbb{Z}^n$ is a vector of integer decision variables, $f(x)$ is the objective function, $g(x)$ are the constraints and b is a vector of constant values.

The integrality constraint ($x \in \mathbb{Z}^n$) restricts the solution space of the problem, since only a finite set of possible values are allowed for the decision variables. For example, if a decision variable represents the number of units of a certain product to produce, the integrality constraint would require that only integer values (e.g., 1, 2, 3, ...) are allowed, rather than fractional values (e.g., 1.5, ... , 2.25, ...). This can make optimization problems significantly more difficult to solve, since it restricts the set of feasible solutions and can result in a combinatorial explosion of possible solutions. For example, an IP problem with only 12 binary decision variables can have up to 2^{12} possible integer solutions, which can make the search for an optimal solution much more complex.

To solve IP optimization problems, specialized algorithms such as branch-and-bound, cutting planes, and branch-and-cut are used. These algorithms iteratively divide the feasible region of the problem into smaller subregions and solve LP problems over each subregion until an integer solution is found.

3.1.2.1 Branch-and-bound

Here, we present the branch-and-bound method, an efficient algorithm for solving IP problems. This algorithm is designed to find the optimal solution of an IP problem, provided it is executed in full. The primary concept behind this method is to divide the solution space into smaller subproblems (LP problems) and explore them systematically to find the optimal solution. The algorithm achieves this by exploring numerous subproblems, establishing both upper and lower bounds for each one and eliminating those that cannot improve on the current best known solution. By searching and eliminating all possible solutions, the algorithm is designed to provide an optimal solution if one exists in the problem. Therefore, if the algorithm is executed to its end and finds a solution, it is guaranteed to be optimal. If the algorithm is not executed fully, it can provide a gap between the obtained solution and the upper bound of the optimal solution.

The core concept behind the branch-and-bound algorithm is to partition the problem into simpler subproblems, which take the form of LP problems in this context. The process of transforming an IP problem into an LP problem is known as relaxation, which involves relaxing the integrality constraints on the decision variables, thereby allowing them to take continuous values. While the resulting relaxed problem may yield solutions that are not applicable to the original problem, it can provide both upper and lower bounds that can lead to a complete solution. It is worth remembering that LP problems belong to the class P and can be solved using efficient methods such as the Simplex method or the Interior Point method.

The branch-and-bound algorithm represents the solution space of the problem using a search tree. The root node of the tree represents the relaxed version of the original problem, with each subsequent node representing a subproblem of its parent. The tree grows via the branching process that involves partitioning subproblems using variable constraints, such as setting a decision variable z to either $z \leq p$ or $z > p$. Normally, the branching process uses a depth-first or best-first search strategy. In order to avoid unnecessary computation, subproblems with bounds worse than the current best known solution can be pruned, eliminating the need to explore their respective subtree. The terminal nodes, also known as leaves, represent complete solutions to the problem. Depending on the constraints, these solutions can be either feasible or infeasible. If a feasible solution is found at a leaf node, its objective function value can be compared to the current best known solution to determine whether an improved solution has been discovered. Although the ideal stopping criteria for the branch-and-bound algorithm is to explore or eliminate all possible solutions of the problem, this may not be possible in complex scenarios. Therefore, in those cases, the stopping criteria is typically based on the number of iterations, time constraints, or a specific gap between the obtained solution and the current upper bound.

We provide a simple example of the branch-and-bound algorithm using the classic 0/1 knapsack problem. This problem can be solved by integer programming, and its definition is as follows: Given a set of n items, denoted as $i \in \{1, ..., n\}$, with corresponding weights w_i and values v_i, the objective is to determine the maximum value that can be obtained by selecting a subset of items such that the total weight does not exceed the capacity W of the knapsack.

The 0/1 knapsack problem can be formulated as an integer programming problem, which is defined as follows:

$$\begin{aligned} \text{Maximize} \quad & \sum_{i=1}^{n} v_i \cdot x_i \\ \text{Subject to} \quad & \sum_{i=1}^{n} w_i \cdot x_i \leq W \\ & x_i \in \{0,1\} \quad \forall i \in \{1,...,n\} \end{aligned}$$

where x_i is the binary decision variable that represents whether item i is included in the knapsack or not. Specifically, $x_i = 1$ if item i is included, and $x_i = 0$ otherwise.

Let us consider a specific instance of the 0/1 knapsack problem:

- Knapsack capacity $W = 8$;
- Number of items $n = 4$;
  - Item 1: $w_1 = 3$, $v_1 = 12$;
  - Item 2: $w_2 = 4$, $v_1 = 16$;
  - Item 3: $w_3 = 5$, $v_1 = 45$;
  - Item 4: $w_4 = 6$, $v_1 = 55$.

We will now solve this instance of the problem step-by-step using the branch-and-bound algorithm. Table 7 shows all the solutions x of the subproblems and their corresponding objective value ζ.

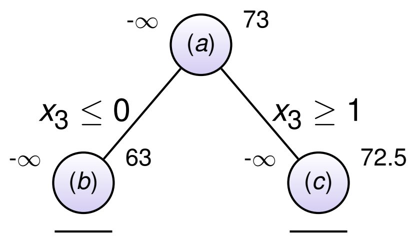
Table 7 – Branch-and-bound subproblem solutions

node	(x_1, x_2, x_3, x_4)	ζ
(a)	$(0, 0, 0.4, 1)$	73
(b)	$(0.66, 0, 0, 1)$	63
(c)	$(0, 0, 1, 0.5)$	72.5
(d)	$(0, 0.5, 0, 1)$	63
(e)	$(1, 0, 0, 0.83)$	57.83
(f)	$(1, 0, 1, 0)$	57
(g)	$x_3 \geq 1$ & $x_4 \geq 1$	Infeasible
(h)	$(0, 0, 0, 1)$	55
(i)	$(0, 1, 0, 0.66)$	52.66
(j)	$(1, 1, 0, 0)$	28
(k)	$x_3 \leq 0$ & $x_1 \geq 1$ & $x_4 \geq 1$	Infeasible

Source: Author.

We first obtain a non-valid solution by solving the relaxed version of the original problem. This solution is non-valid because the decision variable x_3 takes a non-integer value, which means that we need to start branching. To do this, we branch on the

Figure 2 – Second level of the branch-and-bound tree

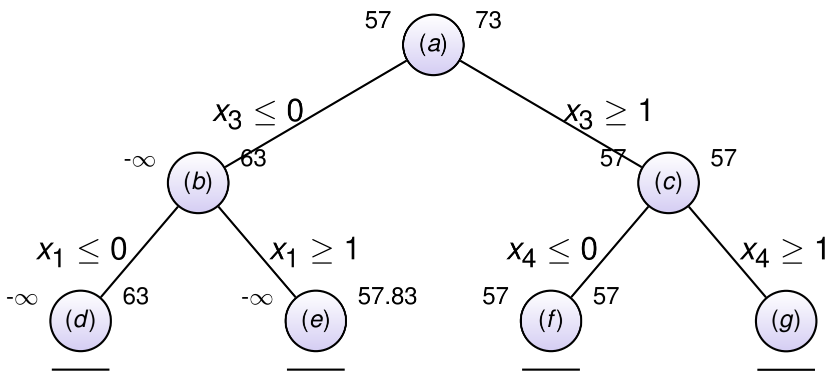


Source: Author.

variable that is not an integer, choosing the two closest integer values. By solving all the subproblems at the second level of the tree, we obtain the tree shown in Figure 2.

We then proceed to solve the third level of the tree using the same branching logic. At node (f), we find a complete solution, which allows us to update our lower bound up the tree. Since node (g) is infeasible, we can finish the search on the right branch. Figure 3 illustrates this step of the algorithm.

Figure 3 – Third level of the branch-and-bound tree



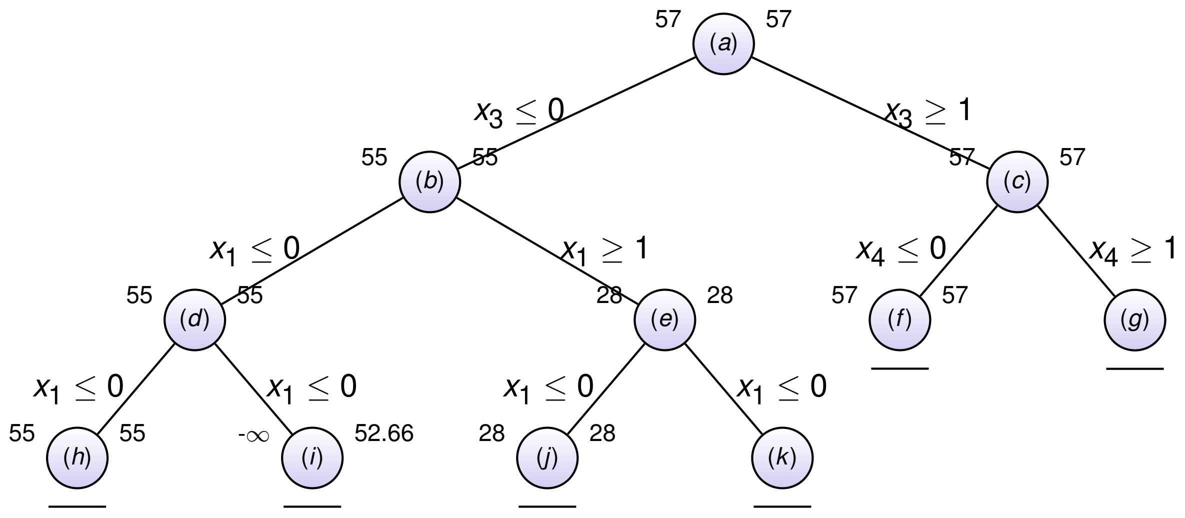
Source: Author.

At the fourth level, we find new complete solutions at nodes (h) and (j), an infeasible node at (k) and we note that (i) has an upper bound lower than its parent's current lower bound. Therefore, we don't need to branch any further, and we can guarantee that the global optimum of this problem is $x = (1, 0, 1, 0) \rightarrow 57$. The full branch-and-bound tree is depicted in Figure 4.

3.2 TIMETABLE PROBLEM

The problem of timetable refers to the difficulty of creating an efficient and effective schedule or timetable that accommodates all the necessary activities and resources within a certain time frame. The main challenge in creating a timetable is to allocate lim-

Figure 4 – Fourth level of the branch-and-bound tree



Source: Author.

ited resources such as time, space, and personnel in a way that maximizes productivity and efficiency, while respecting the problem constraints.

This problem requires careful planning and coordination to ensure that all the necessary activities are scheduled at the right time and place, with the appropriate resources and personnel. Therefore, solving this problem manually can lead to a lot of work, sometimes resulting in obtaining an ineffective or inefficient solution. Furthermore, there are often various constraints and factors to consider when creating a timetable, such as the availability of resources and personnel, the number of students or customers to be served, and the desired outcomes or objectives. These constraints can further complicate the problem and make it difficult to create an optimal timetable.

Following a similar approach as shown in (GROSS; YELLEN; ZHANG, 2013), we formally define the timetable problem, as being comprised of the following elements: T denotes a set of time slots; R denotes a set of resources; E denotes a set of events; C denotes a set of constraints; and O represents the objective of the problem. The timetable problem is to allocate time slots and resources to events in a manner that improves the objective, while concurrently satisfying the constraints. We delve in more details about each of these components.

Time slots T are discrete units of time during which events can be scheduled. Their durations could range from brief to extended periods, could be constant or variable, and a time slot could be exclusive or not to a particular event. The number of available time slots and their individual durations stand as significant parameters of the problem.

Resources R encompass all elements, excluding time slots, that can be allocated to an event. The nature of these elements vary based on the problem context. For instance, in a management context, it could refer to personnel with distinct skill sets.

Resources may be limited in several aspects such as availability, capacity, and efficiency. Understanding these parameters is crucial to achieve an optimal management of these resources.

Events E are individual units of work or tasks that need to be scheduled. Each event typically requires certain resources and must take place within a designated time slot. These events may possess a time duration that must be accommodated within the allocated time slot.

Constraints C represent the inherent rules and limitations that arise when assigning time slots and resources to events. Some common restrictions encountered in the timetabling problem include:

- Availability Constraints. Certain resources are accessible only during specific time slots. For instance, an employee might have a fixed work schedule for the week.
- Capacity Constraints: Resources may possess a limited capacity. For example, an employee may only handle a predetermined number of tasks each day.
- Precedence Constraints: Some events must be performed before others can begin. For instance, in a manufacturing process, certain parts need to be assembled prior to others.
- Grouping Constraints: Certain events may need to be bundled together within the same time slot. For instance, in a manufacturing process, there could be tasks that need to be executed concurrently.

A timetable problem can be interpreted as an assignment problem, signifying that under certain circumstances, it can be solved without a specific objective, rendering any feasible solution as optimal. In contrast, in other scenarios, an objective is essential to direct the assignment process, such as minimizing costs or maximizing the number of successfully allocated events.

Given the comprehension of these components, the timetable problem can be perceived as a multidimensional assignment issue, with tasks extending beyond merely assigning events to time slots. The challenge also lies in allocating resources and respecting various constraints. This complexity amplifies, particularly when the number of events, resources, time slots, and constraints is substantial.

Multiple methodologies, including optimization algorithms, heuristic methods, and machine learning techniques can be employed to solve this problem. The choice of method often depends on the specific requirements of the problem and the resources available for solving it. In this work, we utilize a mathematical programming formulation to tackle the timetable problem. Mathematical programming is an extremely apt way of representing such a problem. Its systematic approach allows for human comprehension, providing insights into the problem and its solution. Moreover, it is designed to identify the optimal solution within the boundaries of a given set of constraints.

This problem arises in various contexts, including schools, universities, transportation systems, and businesses. In sequence, we discuss the application in education in more details.

3.2.1 Application: Education

In an educational setting, the problem of timetable is particularly challenging because it involves scheduling a variety of courses and activities that may require different resources and personnel. We explore an example of the timetable problem applied to the scheduling and composition of university classes.

In this example, the events are the classes that need to be scheduled. Each class is an individual event that must be allocated a time slot and resources. Given enough resources multiple classes can occur simultaneously.

Resources can be anything required to conduct a class, which may encompass elements such as teachers, classrooms, and equipment.

Time slots are the periods during which the classes can take place. In a university context, a week could be segmented into Monday to Friday, and a day further divided into morning, afternoon and evening segments. Here, each time slot can correspond to a constant 50 minute duration. Breaks can be scheduled between each class and at each transition of period. In this scenario, a lecture might span more than one time slot and may extend across multiple days.

Constraints represent the conditions that must be upheld when constructing the university timetable. These might include: Availability Constraints, where each professor, faculty member, or student possesses a schedule that restricts their availability; Capacity Constraints, where each classroom has a maximum occupancy limit; Sequencing Constraints, such as a lab class which must always be after its respective theoretical class.

The objective refers to the optimization target of the timetable. In this educational context, the aim could be to maximize the number of students who secure their first choice of classes, or alternatively, to minimize the total duration of classroom idleness.

The main motivation for solving this problem is to ensure that the educational institution operates in an effective way. A well made timetable can enhance the overall learning and teaching experiences, make better use of resources, and improve the overall functioning of the school. Constructing a custom timetable based in the school's principles can provide opportunities for the institution to experiment with various teaching theories and methodologies. For instance, the school could impose unique specifications in its timetable to stimulate diverse learning experiences. One such specification could be that every week day begins with a creative class — such as music, art, or physical education — to kickstart the learning process.

However, it is notable that creating a custom timetable is a complex procedure, particularly as the problem increases in the number of variables and constraints. Consequently, specific algorithms and heuristics are required to effectively solve the problem. Despite these challenges, the benefits of achieving an effective timetable make it a problem worth tackling. Consequently, this has prompted numerous authors to conduct research in this field, aiming to facilitate the modeling and resolution of such complex problems.

In the study conducted in (DE WERRA, 1985), an overall introduction to the timetabling problem is provided. The paper explores the diverse categories of timetabling problems encountered in education, namely school timetabling, course timetabling, and exam timetabling. Furthermore, it delves into the constraints that emerge within these problem domains. The article also critically appraises early methodologies employed for addressing timetabling problems, such as graph coloring and heuristic search algorithms. Also, highlighting more contemporary approaches, including mathematical programming.

4 PROBLEM MODELING

In this chapter, we initiate the process of modeling. First, we outline the data at our disposal, partitioning it into distinct sets to enhance its comprehensibility. Subsequently, we provide a concise overview of the problem in a mathematical fashion. Finally, we proceed to model the entirety of the problem using mathematical programming notation.

4.1 PROBLEM FORMULATION AS A TIMETABLE PROBLEM

The time slots within our problem maintain a constant duration of one hour, with each slot capable of being assigned to at most one event. The total count of time slots depends upon the number of expected events. Typically, there is not a significant disparity between the number of anticipated events and the number of allocated time slots. This fact, however, complicates the process of scheduling these events due to the limited flexibility.

The resources of the event are the participants, all of whom have limitations in terms of their availability. Notably, the examiner is further constrained by their particular interest in evaluation. Certain resources, such as the undergraduate student, examiner, and committee chair, are essential for the event. Others, however, possess a greater degree of flexibility. For instance, the advisor can also assume the role of the supervisor, and neither the advisor nor the supervisor are mandated to attend the event.

The events of our problem are the project defense presentations, each of which must occur within a single time slot. As for the necessary resources, each event requires the presence of the respective undergraduate student, an examiner, and a chair. The advisor and supervisor also have the option to participate in the event, should they so desire.

The constraints of the event are the particular conditions that must be followed in the composition and scheduling of the project defense presentations. These are identical to those previously outlined in the problem definition chapter. For the convenience of the reader, we shall reiterate them:

- Each available time slot can only be allocated to one event, and similarly, each event can only be allocated to one time slot. As a result, it is ensured that at most one event is scheduled per time slot.
- To allocate an event to a time slot, it is necessary to ensure that all participants are available during that time slot. If any participant is not available, the event cannot be scheduled for that time slot.
- The examiner for an event must hold one of the following positions: UFSC professor, visiting professor, postdoctoral fellow, or doctoral student.

- The chairperson of an event must be a UFSC professor from the conclusion projects organizing committee.
- The advisor, examiner, and chairperson of each event must be different individuals.
- One person can evaluate more than one event, but each event can only be evaluated by one person.
- One person can preside more than one event, but each event can only be presided by one person.

The main objective of our timetable problem is to maximize the occurrence of events, adhering to the designated resources, constraints, and time slots. The residual events represent those which could not be accommodated within the optimal timetable and, consequently, require individual intervention for scheduling.

4.2 SETS

The problem at hand is marked by a high degree of complexity, considering the several factors that can influence the outcomes of events, depending on the choices made by the participants. In order to gain knowledge on the problem with more depth, we classify the gathered data into distinct sets, facilitating our understanding of the relationships and patterns within the data. The selection and definition of these sets can substantially aid the subsequent modeling process.

There are numerous methodologies to partition data into meaningful categories. We opted to segment the data utilizing some of the key components of a timetable problem: time slots, resources, and events.

The time slots in our problem represent the scheduled allotments for project defense presentations during the given semester. This set is referred to as the available schedule set, represented by $\mathcal{H}$. Each instance within the set corresponds to an available time slot for a project defense presentation. Each time slot contains two attributes: one for the date and the other indicating the time.

$\mathcal{H}$ **(available schedule):** Each available date to schedule for the defense $h \in \mathcal{H}$ has the following attributes:

- date(h);
- time(h).

The resources of the problem consist of the participants. This set is named as the people set, denoted by $\mathcal{P}$, where each instance within the set corresponds to a unique individual. Each person possesses inherent data such as name, occupation, and email, in addition to variables that correlate with the other key components such as availability, possibility and interest in evaluation.

$\mathcal{P}$ **(people set):** Each person $p \in \mathcal{P}$ has the following attributes:

- name(p);
- email(p);
- occupation(p) $\in$ { UFSC Professor, Visiting Professor, Postdoc, Doctoral Student, Professional, Undergraduate Student };
- availability(p) $\subset \mathcal{H}$;
- possibility(p) $\subset \mathcal{E}$;
- interest(p) $\subset \mathcal{E}$.

The events in our problem relate to the project defense presentations. This collection is identified as the event set, symbolized by $\mathcal{E}$. Each instance within the set corresponds to a distinct project defense event. Every event possesses inherent data such as the title, undergraduate student, advisor, and supervisor involved in the event. Additionally, there is data that needs to be determined through the resolution of the timetable problem, such as the chairperson, examiner, and schedule of the event.

$\mathcal{E}$ **(event set):** Each thesis defense event $e \in \mathcal{E}$ has the following attributes:

- title(e);
- undergraduate(e) $\in \mathcal{P}$;
- advisor(e) $\in \mathcal{P}$;
- supervisor(e) $\in \mathcal{P}$;
- president(e) $\in \mathcal{P}$;
- evaluator(e) $\in \mathcal{P}$;
- schedule(e) $\in \mathcal{H}$.

Lastly, we define critical subsets to facilitate the modeling process. The first is a subset of the schedule set that includes the common availability between the undergraduate student, advisor, and supervisor. We denote this subset by $\mathcal{H}_e$, where e refers to the respective event.

$$\mathcal{H}_e = \text{availability}(\text{undergraduate}(e)) \cap \text{availability}(\text{advisor}(e)) \cap \text{availability}(\text{supervisor}(e)). \tag{1}$$

We also denote subsets of the people set, referring to the subsets of possible examiners and possible chairs. The subset of possible examiners $\mathcal{C}_e^{\mathsf{E}}$ contains individuals of one of the following occupations: UFSC Professor, Visiting Professor, Postdoc, Doctoral Student, and External Professional, provided they are not the advisor or supervisor of the same event e.

$$\begin{aligned}\mathcal{C}_e^{\mathsf{E}} = & \forall p \in \mathcal{P} : \text{ occupation}(p) \in \{\text{UFSC Professor, Visiting Professor, Postdoc,} \\ & \text{Doctoral Student, and External Professional}\};\ p \notin \{\text{advisor}(e), \text{supervisor}(e)\}.\end{aligned} \tag{2}$$

The subset of possible chairs $\mathcal{C}_e^{\mathsf{P}}$ comprises individuals with the occupation of UFSC Professor, provided they do not occupy the position of advisor or supervisor at the event e.

$$\mathcal{C}_e^{\mathsf{P}} = \forall p \in \mathcal{P} : \text{ occupation}(p) \in \{\text{UFSC Professor}\};\ p \notin \{\text{advisor}(e), \text{supervisor}(e)\}. \tag{3}$$

In practice this subset is comprised only of the organizers of project defense events.

4.3 MATHEMATICAL PROGRAMMING FORMULATION

Having established the foundational components, we are now prepared to model the problem using mathematical programming notation. This approach offers numerous advantages. Firstly, it provides a formal representation of the problem, laying the groundwork for a systematic understanding of the issue at hand. Secondly, it enables the formulation of the optimization problem by defining variables, constraints, and an objective function. Finally, it facilitates the application of specialized algorithms associated with each class of mathematical programming problems. Such algorithms are capable of generating optimal or near-optimal solutions, thereby enhancing the efficiency of problem solving.

The construction of a mathematical programming model typically involves three main steps: defining the variables, specifying the constraints, and outlining the objective. It is essential to note that modeling is an iterative process, indicating that these steps may need to be repeated multiple times until a model that suitably represents the problem is obtained.

Another crucial aspect of modeling is validation. Given the precision of mathematical programming problems, these models can be interpreted and evaluated during the modeling process. This is particularly useful as it allows for the examination of the defined constraints and variables to identify potential ambiguities or discrepancies from the original problem. Furthermore, validation can also involve assessing whether the obtained solution makes sense given the problem context. However, it is important to understand that these methods do not offer assurance that the model represents the process in an desirable manner.

4.3.1 Variables

Defining the variables is one of the most important parts of formulating a mathematical programming problem. We divide the variables into three categories: the decision variables, which are the variables that we control and adjust to optimize the objective function as they represent the decisions that need to be made; the auxiliary variables, which are variables that are functions of other decision variables used for the purpose of facilitating the problem; the parameter variables, are fixed values that are given previous the solution of the problem, they are constant to the optimization problem but can change overtime.

Moreover, decision variables can be further divided into two types: Continuous variables, which can take any real value within their defined range; and Discrete variables, which can only assume specific, countable values. If a problem includes discrete variables, it falls within the realm of integer programming or one of its subclasses. This classification significantly alters the perspective employed in approaching and resolving the problem.

The variables should also be defined in a way that the potential solution space aligns with feasible solutions in the real world. In other words, variables cannot assume values that are nonsensical in a real world context, necessitating the establishment of lower and upper bounds for variables and/or constraints. These bounds help to refine the solution space and ensure its practical relevance.

Another crucial aspect to consider when defining the variables of a problem involves establishing and comprehending the range of the variables. If a variable is defined over a broader range than necessary, a part of it may become disconnected from the optimization problem. This disconnection can cause confusion and even introduce errors into the problem.

Now we define each of the decision variables used on our optimization problem.

The auxiliary variable $x_{e,h}$ indicates whether there is any event e scheduled at the time slot h.

$$x_{e,h} = \begin{cases} 1, & \text{if schedule}(e) = h \\ 0, & \text{otherwise} \end{cases} \quad , \forall e \in \mathcal{E}, \forall h \in \mathcal{H}. \tag{4}$$

The auxiliary variable z_e indicates whether the event e will take place.

$$z_e = \begin{cases} 1, & \text{if the event } e \text{ occurs} \\ 0, & \text{otherwise} \end{cases} \quad , \quad \forall e \in \mathcal{E}. \tag{5}$$

The auxiliary variable $y^{\mathrm{E}}_{p,e}$ indicates whether the person p is an examiner of the event e. This person must be in the set of potential evaluators $\mathcal{C}^{\mathrm{E}}_e$.

$$y^{\mathrm{E}}_{p,e} = \begin{cases} 1, & \text{if the person } p \text{ is the examiner of the event } e \\ 0, & \text{otherwise} \end{cases} \quad , \forall e \in \mathcal{E}, \forall p \in \mathcal{C}^{\mathrm{E}}_e. \tag{6}$$

The auxiliary variable $y^{\mathrm{P}}_{p,e}$ indicates whether the person p is the chairperson of the event e. This person must be in the set of possible presidents $\mathcal{C}^{\mathrm{E}}_{e}$.

$$y^{\mathrm{P}}_{p,e} = \begin{cases} 1, & \text{if the person } p \text{ is the president of the event } e \\ 0, & \text{otherwise} \end{cases} \quad , \forall e \in \mathcal{E},\ \forall p \in \mathcal{C}^{\mathrm{P}}_{e}. \tag{7}$$

The auxiliary variable $w^{\mathrm{E}}_{p,e,h}$ indicates whether the person p will act as an examiner for the event e at the schedule h.

$$w^{\mathrm{E}}_{p,e,h} = \begin{cases} 1, & \text{if } p \text{ is the examiner of } e \text{ on the date } h \\ 0, & \text{otherwise} \end{cases} \quad , \forall e \in \mathcal{E},\ \forall p \in \mathcal{C}^{\mathrm{E}}_{e},\ \forall h \in \mathcal{H}_{e}. \tag{8}$$

The auxiliary variable $w^{\mathrm{P}}_{p,e,h}$ indicates whether the person p presides over the event e on the date h.

$$w^{\mathrm{P}}_{p,e,h} = \begin{cases} 1, & \text{if } p \text{ presides over } e \text{ on the date } h \\ 0, & \text{otherwise} \end{cases} \quad , \forall e \in \mathcal{E},\ \forall p \in \mathcal{C}^{\mathrm{P}}_{e},\ \forall h \in \mathcal{H}_{e}. \tag{9}$$

4.4 RESTRICTIONS

Formulating the constraints of a mathematical programming problem is an essential step in the modeling process. Constraints delineate the limitations or restrictions on the decision variables, playing a vital role in defining the feasible region, or the set of solutions that can be selected. These constraints might arise from physical or resource limitations, regulatory requirements, or a variety of other considerations.

Similar to the variables and objective, it is important to note that the mathematical nature of the constraints determines the classification of the mathematical programming problem. A single problem can be modeled in several ways, resulting in potential variations of the same problem belonging to different classes of mathematical programming problems. Therefore, when faced with a constraint that increases the complexity of the problem undesirably, it is worthwhile to consider alternative modeling methods.

Interactions between constraints also require careful consideration. The feasible region is demarcated by these constraints. If a constraint does not restrict the feasible region of a problem, it is considered redundant and can be removed without altering the feasible region of the problem.

The constraints defined within this context should accurately reflect the real-world limitations and characteristics of the problem. This process might necessitate the employment of assumptions and approximations to simplify the original problem. Having previously identified the real-world constraints of the problem, we must now translate these into the notation of mathematical programming.

Each event e must have only one time slot allocated to it.

$$\sum_{h \in \mathcal{H}} x_{e,h} = z_e, \quad \forall e \in \mathcal{E}. \tag{10}$$

We note that, for an event e, the range of possible schedules falls within the subset $\mathcal{H}_e$, that is, the possible schedules for that event are those where the undergraduate student, his advisor, and supervisor are all available. Therefore, we can refine this constraint by considering this subset, $\mathcal{H}_e$, instead of the general schedule set $\mathcal{H}$.

$$\sum_{h \in \mathcal{H}_e} x_{e,h} = z_e, \quad \forall e \in \mathcal{E}. \tag{11}$$

Each time slot h can be occupied by at most one event e.

$$\sum_{e \in \mathcal{E}} x_{e,h} \leq 1, \quad \forall h \in \mathcal{H}. \tag{12}$$

Analogous to the previous constraint, considering only the schedules within the subset $\mathcal{H}_e$ would not result in any information loss. Consequently, we opt for this subset as it would generate a tighter constraint.

$$\sum_{e \in \mathcal{E} : h \in \mathcal{H}_e} x_{e,h} \leq 1, \quad \forall h \in \mathcal{H}. \tag{13}$$

For an event e to take place, the respective advisor, supervisor and undergraduate must have availability on the same dates.

$$\sum_{h \in \mathcal{H}_e} x_{e,h} = z_e, \quad \forall e \in \mathcal{E}. \tag{14}$$

Note that Eq. (11) is implied by Eq. (14), therefore only the latter is necessary in the final model, assuming that the variables $x_{e,h}$, considering an event e, will be defined only on the dates $h \in \mathcal{H}_e$.

One person p can evaluate more than one event e, but an event is evaluated by only one person.

$$\sum_{p \in \mathcal{C}_e^{\mathrm{E}}} y_{p,e}^{\mathrm{E}} = z_e, \quad \forall e \in \mathcal{E}. \tag{15}$$

where $\mathcal{C}_e^{\mathrm{E}} = \mathcal{C}^{\mathrm{E}} \setminus \{\text{advisor}(e), \text{supervisor}(e)\}$ is the set of people that can perform the role of examiner of the event e. We assume that the variable $y_{p,e}^{\mathrm{E}}$ is defined only for the events e where the person p can serve as examiner.

One person p can serve as chair for more than one event e, but each event can be chaired by only one person.

$$\sum_{p \in \mathcal{C}_e^{\mathrm{P}}} y_{p,e}^{\mathrm{P}} = z_e, \quad \forall e \in \mathcal{E}. \tag{16}$$

where $\mathcal{C}_e^{\mathsf{P}} = \mathcal{C}^{\mathsf{P}} \setminus \{\text{advisor}(e), \text{supervisor}(e)\}$ is the set of chairperson candidates of the event e, who did not advise the referred work. We also assume that the variable $y_{p,e}^{\mathsf{P}}$ is defined only for the events e where the person p can serve as chair.

The examiner and the chairperson of an event e must be different individuals.

$$y_{p,e}^{\mathsf{E}} \cdot y_{p,e}^{\mathsf{P}} = 0, \quad \forall p \in \mathcal{C}_e^{\mathsf{P}} \cap \mathcal{C}_e^{\mathsf{E}}, \forall e \in \mathcal{E}. \tag{17}$$

The above constraint will be defined only for individuals who can both serve as president and examiner of the event e. Note that the constraint above is non-linear (bilinear), but it can also be expressed through a linear constraint, as follows:

$$y_{p,e}^{\mathsf{E}} + y_{p,e}^{\mathsf{P}} \leq 1, \quad \forall p \in \mathcal{C}_e^{\mathsf{P}} \cap \mathcal{C}_e^{\mathsf{E}}, \forall e \in \mathcal{E}. \tag{18}$$

This modeling approach is preferred, as adding a bilinear constraint would introduce additional unneeded complexity to the problem. Maintaining the linear constraints ensures that our problem remains within the class of Integer Programming.

For an event to take place, one examiner, preferably interested in the event, must be assigned in a available date that is feasible for all event participants. Therefore, for each event e, the following constraints are necessary:

$$\begin{aligned}
&\text{For all } e \in \mathcal{E}, h \in \mathcal{H}_e, p \in \mathcal{C}_e^{\mathsf{E}} : e \in \text{interest}(p) \cup \text{possibility}(p) : \\
&\quad w_{p,e,h}^{\mathsf{E}} \leq y_{p,e}^{\mathsf{E}}, && \text{(19a)} \\
&\quad w_{p,e,h}^{\mathsf{E}} \leq x_{e,h}, && \text{(19b)} \\
&\quad w_{p,e,h}^{\mathsf{E}} \geq y_{p,e}^{\mathsf{E}} + x_{e,h} - 1, && \text{(19c)} \\
&\quad w_{p,e,h}^{\mathsf{E}} \leq 1 \text{ if } h \in \text{availability}(p), 0 \text{ otherwise.} && \text{(19d)}
\end{aligned}$$

Similarly, a chairperson must be allocated to an event at a date when the undergraduate student and all other committee members are available. Therefore, for all event e, the following constraints are necessary:

$$\begin{aligned}
&\text{For all } e \in \mathcal{E}, p \in \mathcal{C}_e^{\mathsf{P}}, h \in \mathcal{H}_e : \\
&\quad w_{p,e,h}^{\mathsf{P}} \leq y_{p,e}^{\mathsf{P}} && \text{(20a)} \\
&\quad w_{p,e,h}^{\mathsf{P}} \leq x_{e,h} && \text{(20b)} \\
&\quad w_{p,e,h}^{\mathsf{P}} \geq y_{p,e}^{\mathsf{P}} + x_{e,h} - 1, && \text{(20c)} \\
&\quad w_{p,e,h}^{\mathsf{P}} \leq 1 \text{ if } h \in \text{availability}(p), 0 \text{ otherwise.} && \text{(20d)}
\end{aligned}$$

Lastly, an examiner has a maximum number of evaluations that he is capable of conducting in a given semester.

$$\sum_{e \in \mathcal{E}} y_{p,e}^{\mathsf{E}} \leq \text{max\_eval}(p), \quad \forall p \in \mathcal{C}_e^{\mathsf{E}}. \tag{21}$$

4.5 OBJECTIVE

The objective function is a mathematical representation of the goal you want to achieve with your decisions. In a mathematical programming problem, you aim to optimize this function - that is, to find the values of the decision variables that either maximize or minimize the objective function, subject to the constraints.

The primary aim of our problem is to maximize the number of successfully composed and scheduled events. This aim can be mathematically expressed as follows:

$$\max \ \delta = \sum_{e \in \mathcal{E}} z_e. \tag{22}$$

We also have secondary objectives which could lead to the composition of more meaningful project defense events. These secondary objectives are addressed through a cascade optimization methodology, where the overall problem is broken down into an ordered sequence of simpler optimization problems. The solution to each problem in the sequence informs or constrains the subsequent one, allowing each problem to be solved independently, thus significantly simplifying the overall process.

It is important to note that this approach does not necessarily guarantee an absolute optimal solution for the overall problem. Sequentially constraining options might cause a potentially better solution to be overlooked, especially if two decisions are not independent. However, in our case, this method effectively addresses the problem, given the significantly greater importance of our primary objective compared to the secondary ones.

Consequently, the optimal solution to our initial problem, denoted by δ^*, provides the total number of successfully scheduled events. This number is subsequently established as a constraint in following problems, as illustrated in Eq. 23, thus enabling the exploration of optimal solutions that best align with our secondary objectives.

$$\sum_{e \in \mathcal{E}} z_e \geq \delta^*. \tag{23}$$

In our case, the secondary objectives we address are the maximization of the composition of events where the examiners have expressed interest, and the maximization of the preferred chair for the events. These could be segregated into two distinct problems, with the solution for each one fixed for the other, or they could be integrated into a single objective function. In our study, we have chosen to integrate both into one objective function.

Considering $\hat{p}$ as the preferred chair for all events, we propose the following secondary objective:

$$\max \ \delta_2 = \sum_{e \in \mathcal{E}} \left(y^{\mathrm{P}}_{\hat{p},e} + \sum_{p \in \mathcal{C}^{\mathrm{E}}_e \,:\, e \in \mathrm{interest}(p)} y^{\mathrm{E}}_{p,e} \right). \tag{24}$$

Finally, it is also possible to implement personalized constraints and objectives, depending on the specific requirements of the semester. A notable example would be to designate specific individuals as examiners for particular events.

4.6 FULL MODEL

By employing only the tighter constraints and discarding all redundant ones, we obtain the full optimization problem as shown in 25.

A noteworthy observation is that the final model obtained falls into the category of Integer Programming. More specifically, we are dealing with a binary integer programming problem where all variables are integers and are confined to the binary set $\{0, 1\}$.

Integer Programming problems are classified as NP-complete. This implies that there is no known polynomial time algorithm to solve the problem optimally, although potential solutions can be verified in polynomial time. Despite this classification, efficient algorithms like branch-and-bound exist, which can solve large-scale IP problems either optimally or near-optimally with notable efficiency.

$$f_1 : \max \delta = \sum_{e \in \mathcal{E}} z_e \tag{25a}$$

$$s.t. : \sum_{h \in \mathcal{H}_e} x_{e,h} = z_e, \quad \forall e \in \mathcal{E} \tag{25b}$$

$$\sum_{e \in \mathcal{E} : h \in \mathcal{H}_e} x_{e,h} \leq 1, \quad \forall h \in \mathcal{H} \tag{25c}$$

$$\sum_{p \in \mathcal{C}_e^{\mathsf{E}}} y_{p,e}^{\mathsf{E}} = z_e, \quad \forall e \in \mathcal{E} \tag{25d}$$

$$\sum_{p \in \mathcal{C}_e^{\mathsf{P}}} y_{p,e}^{\mathsf{P}} = z_e, \quad \forall e \in \mathcal{E} \tag{25e}$$

$$y_{p,e}^{\mathsf{E}} + y_{p,e}^{\mathsf{P}} \leq 1, \quad \forall p \in \mathcal{C}_e^{\mathsf{P}} \cap \mathcal{C}_e^{\mathsf{E}}, \forall e \in \mathcal{E} \tag{25f}$$

For all $e \in \mathcal{E}, h \in \mathcal{H}_e, p \in \mathcal{C}_e^{\mathsf{E}} : e \in \text{interest}(p) \cup \text{possibility}(p)$:

$$w_{p,e,h}^{\mathsf{E}} \leq y_{p,e}^{\mathsf{E}}, \tag{25g}$$

$$w_{p,e,h}^{\mathsf{E}} \leq x_{e,h}, \tag{25h}$$

$$w_{p,e,h}^{\mathsf{E}} \geq y_{p,e}^{\mathsf{E}} + x_{e,h} - 1 \tag{25i}$$

$$w_{p,e,h}^{\mathsf{E}} = 0 \text{ if } h \notin \text{availability}(p) \tag{25j}$$

$$w_{p,e,h}^{\mathsf{E}} \in \{0, 1\} \tag{25k}$$

For all $e \in \mathcal{E}, p \in \mathcal{C}_e^{\mathsf{P}}, h \in \mathcal{H}_e$:

$$w_{p,e,h}^{\mathsf{P}} \leq y_{p,e}^{\mathsf{P}} \tag{25l}$$

$$w_{p,e,h}^{\mathsf{P}} \leq x_{e,h} \tag{25m}$$

$$w_{p,e,h}^{\mathsf{P}} \geq y_{p,e}^{\mathsf{P}} + x_{e,h} - 1 \tag{25n}$$

$$w_{p,e,h}^{\mathsf{P}} = 0 \text{ if } h \notin \text{availability}(p) \tag{25o}$$

$$w_{p,e,h}^{\mathsf{P}} \in \{0, 1\} \tag{25p}$$

$$\sum_{e \in \mathcal{E}} y_{p,e}^{\mathsf{E}} \leq \text{max\_eval}(p), \quad \forall p \in \mathcal{C}_e^{\mathsf{E}} \tag{25q}$$

$$z_e \in \{0, 1\}, \forall e \in \mathcal{E} \tag{25r}$$

$$x_{e,h} \in \{0, 1\}, \forall e \in \mathcal{E}, \forall h \in \mathcal{H}_e \tag{25s}$$

$$y_{p,e}^{\mathsf{E}} \in \{0, 1\}, \forall e \in \mathcal{E}, \forall p \in \mathcal{C}_e^{\mathsf{E}} \tag{25t}$$

$$y_{p,e}^{\mathsf{P}} \in \{0, 1\}, \forall e \in \mathcal{E}, \forall p \in \mathcal{C}_e^{\mathsf{P}} \tag{25u}$$

The second optimization problem is presented in 26, where δ^* represents the objective determined in the first optimization problem. It is crucial to note that it is possible to introduce new terms as constraints or components of the objective function to attain solutions that better align with the requirements of the organizers.

$$f_2: \max \delta_2 = \sum_{e \in \mathcal{E}} \left(y^{\mathrm{P}}_{\hat{p},e} + \sum_{p \in \mathcal{C}^{\mathrm{E}}_e \,:\, e \in \mathrm{interest}(p)} y^{\mathrm{E}}_{p,e} \right) \tag{26a}$$

$$s.t.: \sum_{e \in \mathcal{E}} z_e \geq \delta^* \tag{26b}$$

$$(25b) - (25u) \tag{26c}$$

5 IMPLEMENTATION

In this chapter, we delve into the specifics of our computational system's implementation. The system is divided into two parts. The first pertains to data acquisition via forms, and the second involves the processing and resolution of the problem at hand.

5.1 DATA ACQUISITION

The initial step in our project encompasses the acquisition, storage, and processing of the data essential to our problem. We seek a solution that is easy to maintain, straightforward to understand, and resilient to potential issues.

The process begins with the identification of appropriate tools for user data collection, storage, and processing. The selected tools ought to conform to the established criteria previously outlined. Following this, we delve into details regarding the application of these tools in addressing the problem at hand. Lastly, we introduce a description of the process flow associated with the data acquisition phase of our solution.

5.1.1 Adopted Toolset

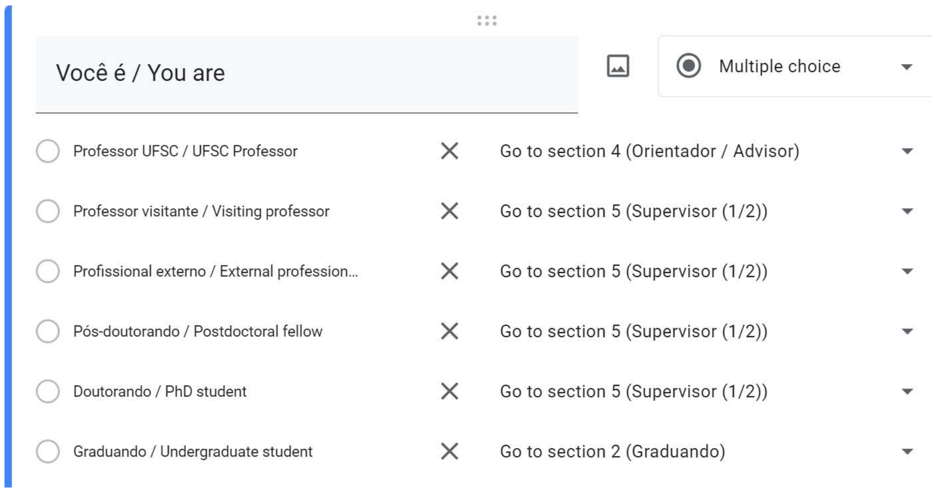
The initial toolset we considered for this project consisted of a web form application integrated with a relational database. However, acknowledging potential issues a custom solution might generate, particularly with regards to hosting and maintenance, we opted for well established tools that avoid these complications. The tools we ultimately selected belong to Google's workspace suite: Google Forms, Google Sheets, and Google Apps Script. These tools provide an integrated, user-friendly, easy-to-maintain, and robust system, aligning perfectly with our predetermined criteria.

Google Forms is an intuitive online tool adept at creating a variety of data collection applications, such as surveys, quizzes, and event registration forms. This platform facilitates users in designing, distributing, and modifying forms, besides enabling real-time response analysis and automatic data visualization through charts. Google forms are characterized by its user-friendly interface, customizable templates, and seamless integration with other Google services. The previously mentioned qualities significantly influenced in our decision to select Google Forms as the primary instrument for data collection from users engaged in project defense events.

To demonstrate the functionality of the Google Forms application, we provide two figures. Figure 5 displays a sample question from our Google Form application, indicating the available answer options and the corresponding sections to which each answer leads. Figure 6 presents a pie chart summarizing the responses to the aforementioned question, offering a convenient visual representation of the users form answers.

Google Sheets is a cloud-based spreadsheet tool. It allows for the creation,

Figure 5 – An example of a Google form question.



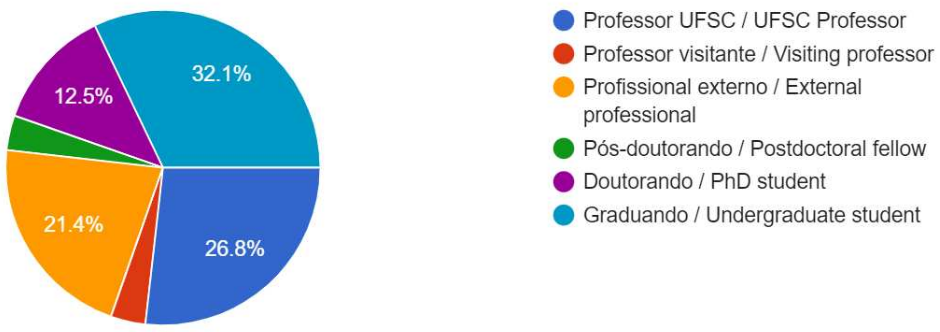
Você é / You are

Multiple choice

Professor UFSC / UFSC Professor	Go to section 4 (Orientador / Advisor)
Professor visitante / Visiting professor	Go to section 5 (Supervisor (1/2))
Profissional externo / External profession...	Go to section 5 (Supervisor (1/2))
Pós-doutorando / Postdoctoral fellow	Go to section 5 (Supervisor (1/2))
Doutorando / PhD student	Go to section 5 (Supervisor (1/2))
Graduando / Undergraduate student	Go to section 2 (Graduando)

Source: Author.

Figure 6 – An example of a Google form question answer summary.



Source: Author.

editing, and sharing of spreadsheets in real time, facilitating both individual work and collaborative efforts. While Google Sheets encompasses a broad array of features typically associated with spreadsheet applications, our particular interest lies in its robust integration capabilities with other Google services. The reason for selecting Google Sheets in our project lies in its intuitive user interface, flexibility, integration with other Google apps, and accessibility across multiple devices.

It is noteworthy to mention that in a more sophisticated, large-scale project, employing a more scalable data storage option, such as a relational database like PostgreSQL, would be more adequate. However, in our current context, we are engaged with a small scale project where data is input each semester, and our focus lies in developing a solution that can be easily maintained in subsequent semesters. Thus, adhering to the principle of Occam's Razor, which advocates for simplicity in decision making, we have opted for the straightforward and accessible Google Sheets as our primary data storage mechanism.

Google Apps Script is a cloud-based scripting language developed by Google that allows for the automation, extension, and integration of Google Workspace applications, including Google Sheets, and Google Forms, among others. Utilizing JavaScript as its foundation, it enables the creation of custom functions to automate tasks, and build web applications directly within the Google Cloud environment. Thus, we use Google Apps Script to integrate the previously discussed tools and automate the process of data acquisition.

5.1.2 Development

To begin, we develop an adaptive data acquisition form that is segmented based on user responses and occupation. In the context of project defense events, we categorize users into six distinct occupational groups: UFSC Professor, Visiting Professor, Postdoc, Doctoral Student, External Professional, and Undergraduate Student. An individual's specific occupation dictates their potential roles within the committee, which can include academic, advisor, supervisor, examiner, and chairperson. Given that an individual may assume multiple roles across different project defense events, the form navigates through each possible role, offering users the option to bypass specific roles to prevent unnecessary engagement.

The relation between an individual's occupation and their potential roles within the event are further elaborated here. Undergraduate students uniquely have a one-to-one relationship with a role within the event, i.e., they are the academics whose work will be evaluated. UFSC Professors are the only ones who can assume the role of an advisor, but they may also fulfill all other roles within the committee, provided it respects the committee's composition restrictions. External professionals can exclusively serve as supervisors. The roles of Visiting Professors, Postdocs, and Doctoral Students can extend to the supervisor or examiner roles within a project defense event.

Every role within the event needs distinct data. Academics, for instance, must supply information pertaining to their work, which includes both the title and abstract of their thesis, in addition to details about their advisor and supervisor. Conversely, advisors and supervisors are obliged to furnish specific data indicating the projects they are involved in. Evaluators, on the other hand, are expected to disclose information about the research works they have an interest in on a scale of 1 to 3. A rating of 1 signifies no interest in undertaking the evaluation, a rating of 2 suggests that the evaluator may be willing to assess the work, while a rating of 3 denotes an interest in performing the evaluation. Regardless of the role, there are some common data points that all individuals filling the forms must provide. These include their full name, email address, occupational details, and availability status.

By default, responses from the form are directly linked with a Google Sheets application. This application is sectioned into various spreadsheets, each serving a specific function. One spreadsheet is dedicated to storing the raw data obtained from the form responses, while others are designed to house organized data sets, such as the People set, Event set, and Available Schedule set that were introduced in the previous chapter.

To automate steps of the process of data acquisition we develop scripts within the Google Apps Script environment. These scripts can be manually invoked or triggered by specific events, such as the submission of a new form response. We will now delve into further detail regarding these automation scripts.

We primarily utilize Google Apps Script for two main purposes: to automate the creation and maintenance of questionnaire items, and to transform raw form data into an organized tabular format. The scripts related to the former purpose run after the initial stage of data acquisition, which occurs after undergraduate students submit their information. Conversely, the scripts associated with the latter purpose are executed after the form submission deadline has been reached.

To automate form creation and maintenance, we have developed two scripts. The first script is used to populate questions relating to the work of the undergraduate student and their respective advisor and supervisor. This script is activated upon the opening of a form, meaning that every time someone accesses the form, the script executes, ensuring the questions are always up-to-date.

The second script is tailored to generate questions that bridge the evaluators with the projects they may be interested in. For every project, a unique question is formulated, the questions are comprised of the project's title and abstract as its content. The evaluators are required to express their level of interest by rating each project on a scale from 1 to 3. These scripts are executed only once, following the completion of the initial data acquisition stage.

The scripts designed to automate the transformation of raw data into an organized tabular format follow similar format of the mathematical sets delineated in the modeling chapter. Accordingly, we propose three distinct scripts: one for the Schedule set, another for the Event set, and a final one for the Person set.

The Schedule dataset is relatively straightforward. Its data varies depending on the academic semester and is not based on the form answers. We have developed a script that receives as input a range of time slots and the specific days when events are expected to be scheduled. This script produces a table of all generated schedules in the DateTime format, which is a commonly used format in data handling for representing points in time.

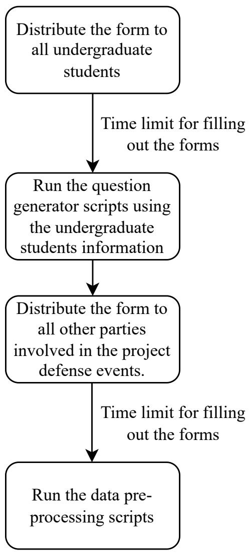
The Event dataset presents more complexity as it necessitates data from the project that the undergraduate submitted, as well as the evaluators interest data. Initially, we load the undergraduate data, then query our raw data for evaluators interested in each project. Specifically, this table houses data about the student, their advisor and supervisor, their project, and those who have shown interest in their work.

Finally, the Person dataset comprises specific data about individuals, as they themselves submitted. This encompasses their name, occupation, email, telephone, availability, interest in evaluation, and maximum number of evaluations. We regard the data submitted by the individual as the primary source of information. Consequently, when discrepancies arise between the information a student submitted about their advisor or supervisor, we default to the information that the advisor or supervisor submitted as the accurate record.

5.1.3 Process Flow

The process of data acquisition is divided into two phases: the initial phase involves collecting data from undergraduate students that are set to defend their projects within the semester, and the subsequent phase pertains to gathering data from all other participants involved in the semester's project defense events. In between these phases, automation scripts are executed to simplify the data acquisition process. A graphical representation of this process flow can be found in Figure 7.

Figure 7 – Flowchart of the data acquisition process.


Distribute the form to all undergraduate students
Time limit for filling out the forms
Run the question generator scripts using the undergraduate students information
Distribute the form to all other parties involved in the project defense events.
Time limit for filling out the forms
Run the data pre-processing scripts

Source: Author.

It is critical to clarify that despite the data acquisition process being divided into two phases, each participant is required to submit their information only once. The bifurcation of the process is needed because the sequential dependence of the data requirements. Specifically, the data gathered from the undergraduates is instrumental in formulating the questions posted to the evaluators, advisors, and supervisors.

5.2 COMPUTATIONAL SYSTEM

The computational system represents the second and most crucial component of our system and is divided into two parts. The first part is tasked with the verification and analysis of the tabular data. The second part focuses on the optimization problem, exploring potential solutions and conducting subsequent analysis of these solutions.

We will now provide details about the identification of appropriate tools for the development of our solution. We are seeking tools that are intuitive, powerful, and capable of dealing with all our requirements related to data analysis and mathematical optimization. Subsequently, we discuss how these tools are utilized in the actual development process. Finally, we delve into the overall process flow of the computational system.

5.2.1 Adopted Toolset

The necessary tools for this segment of the project include a mathematical solver adept at tackling problems related to Integer Programming, a programming language compatible with an optimization framework and data analysis packages.

Mathematical optimization solvers are designed to efficiently identify the optimal solutions to optimization problems. The characteristics of the problem dictate the analysis and the algorithms required to address it. Each solver employs adaptations of established algorithms, augmenting their problem solving efficiency. In this work, we have utilized the Gurobi solver. Being a commercial, state-of-the-art solver, Gurobi is capable of managing a broad spectrum of problem classes, including Linear Programming, Integer Programming, Quadratic Programming, and their respective variants.

For this project, Julia has been selected as the preferred programming language. Julia is a high level programming language known for its user friendly syntax and readability, all the while maintaining a high level of efficiency. As a dynamically typed language, Julia is inherently interactive, which, coupled with its efficiency, renders it particularly suitable for work in fields such as machine learning, statistics, and optimization.

Julia is also equipped with a plethora of powerful packages designed for an extensive range of applications. Among these, JuMP stands out as an open-source package specifically crafted for mathematical programming within Julia. It provides a high-level interface for composing and solving various classes of optimization problems, including Linear Programming and Integer Programming.

JuMP distinguishes itself from other optimization packages by its unique algebraic modelling style. This approach allows us to express optimization problems using conventional mathematical notation directly in the code. Once the problem is formulated, the JuMP package converts it into a format interpretable by mathematical solvers.

To further understand JuMP, we introduce its syntax for the essential components of a mathematical optimization problem: the model, the solver, the variables, the constraints, and the objective. The code snippet below showcases those essential parts. Note that the complete code of the optimization problem is showcased in Appendix A.

```
1. opt_model = Model(Gurobi.Optimizer);
2. @variable(opt_model,x[e=1:length(event_set),h=1:length(schedule_set)],set=MOI.ZeroOne());
3 .@variable(opt_model,z[e=1:length(event_set)],set=MOI.ZeroOne());
4. for e in 1:length(event_set)
5.     @constraint(opt_model, sum(x[e,He[e]]) == z[e]);
6. end
7. @objective(opt_model, Max, sum(z))
8. optimize!(opt_model)
```

The first line declares the optimization model and its respective solver, a notable feature of JuMP that allows for seamless solver changes without needing to recast the entire problem in a different notation.

The second and third lines declare the problem's variables. We link them with the optimization model, setting their range and type. Here, $x_{e,h}$ is a two-dimensional variable, and z_e is a one-dimensional one. Both are binary variables.

Lines four to six declare an equality constraint of the problem. Constraints can be equalities or inequalities that relate the decision variables to a constant or other decision variables. Here, we link $x_{e,h}$ with z_e.

The seventh line sets the problem's objective. The objective can be a maximization, minimization, or even without an objective, meaning that the algorithm will strive only to find a feasible solution. This part of the code needs the objective expression and the linked model. The eighth line is the command to optimize the problem.

5.2.2 Development

The initial segment of our computational system focuses on the loading and analysis of the organized form data. This data, having already undergone preliminary pre-processing, now necessitates loading into Julia in a manner most suited for the ensuing optimization problem, whilst maintaining data integrity. Furthermore, we conduct an analysis of the data to identify user input errors. Lastly, we review the data for coherence.

In order to import data from the Google Sheets application into Julia, we can establish a direct link via the Google Cloud API or simply download the sheet information, given that we do not need to modify it. We have chosen to use the Google Cloud API, but if any issues arise during this process, the user can easily download the sheets and use them directly.

Subsequent to loading the data, it is essential to reorganize it into a more understandable and manageable format. For this purpose, we employ Julia's DataFrame package, which stores data in a tabular format and enables various database operations.

The next step involves identifying and rectifying any inconsistencies within the data, such as misspellings, incoherent information, and other issues not addressed during Google Apps Scripts pre-processing stage. At this point, we undertake a more in-depth analysis to ensure the data appropriateness for the optimization model. In certain cases, this analysis might necessitate human intervention to discern whether a data entry represents two distinct entities or a single one. We provide further details about some of these inconsistencies.

A common issue is an undergraduate student providing a name or email differing from the advisor's or supervisor's inputs. We address this by initially comparing emails and telephone numbers. If these differ, we apply a string similarity function, such as the Damerau-Levenshtein distance, to measure the similarity between two strings based on the minimum number of operations needed to transform one into the other. Depending on this distance, we either automatically classify them as equal or prompt the user to confirm whether these two strings refer to the same person.

Another issue arises when users input data where they should not. This typically occurs with external professionals who, due to a hurried review of the form, overlook some instructions and provide unnecessary information. We rectify this by analyzing the user's occupation in relation to the information provided. For instance, an external professional expressing an interest in evaluating an event they are already supervising.

The second segment concerns the model constructed in JuMP, its generated solutions, and an analysis on the results. The model's solutions offer numerous insights. We conduct analyses on the varying solutions and, if present, on the events that failed to be scheduled, necessitating an investigation into their scheduling failure.

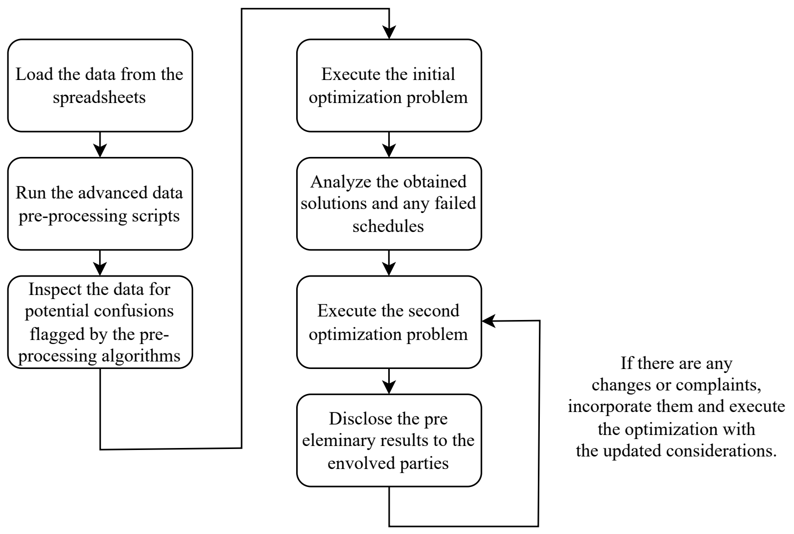
The initial part of this process involves converting the data from the DataFrames into the mathematical sets elaborated in the modeling section. This conversion facilitates the code writing for variables, restrictions, and objectives, as it essentially becomes a direct transcription of the model in JuMP notation.

With the sets in hand, we proceed to transcribe the developed mathematical optimization problem into Julia. Here, we also have the opportunity to customize the solution of the model, by employing different solvers or obtaining multiple optimal solutions, if they exist.

Subsequently, we assess the problem's solution and the unscheduled events. We analyze, the latter by examining the intersection of the availability of the undergraduate student and their respective advisor and supervisor, followed by the interested evaluators. We also perform tests to ensure the solution's feasibility by checking constraints such as availability of all parties and composition constraints like the advisor/supervisor, evaluator, and president being different individuals.

Lastly, we can introduce constraints and new objectives to the problem to achieve

Figure 8 – Process flow chart of the computational system.


Load the data from the spreadsheets
Run the advanced data pre-processing scripts
Inspect the data for potential confusions flagged by the pre-processing algorithms
Execute the initial optimization problem
Analyze the obtained solutions and any failed schedules
Execute the second optimization problem
Disclose the pre eleminary results to the envolved parties
If there are any changes or complaints, incorporate them and execute the optimization with the updated considerations.

Source: Author.

a more refined solution. Two common additions include maximizing the participation of interested evaluators in the projects and ensuring a certain individual presides over as many events as possible. It is also possible to add constraints specified by individuals, such as a request to include a particular evaluator in an event. We can accommodate these additions without sacrificing our optimal solution by fixing the number of successfully scheduled events as a constraint. If the revised problem is infeasible, it implies that the added constraints have restricted our problem beyond the set of original optimal solutions.

5.2.3 Process Flow

The process flow of this part of the problem is bifurcated into two segments: one illustrating the pre-processing of data for the optimization problem, and the other depicting the execution and analysis of the optimization problem. The optimization problem is executed in a cascade. This means that after our initial run, we commence refining certain aspects of the solution to identify solutions that fit other desirable criteria while maintaining an optimal solution. This iterative process can be repeated multiple times, enabling us to address complaints and challenges faced by specific individuals. The process flow diagram is illustrated in Figure 8.

6 CASE STUDY

The case study is centered on the allocation and orchestration of project defense events for undergraduate students at the Federal University of Santa Catarina, specifically within the Automation and Control Engineering Department, during the first semester of 2023. There were 18 students planning to present their defense that semester. Therefore, we were tasked with composing and scheduling 18 events within the period extending from the 10th to the 12th of July 2023. Each event lasts for one hour and cannot take place concurrently.

Table 8 provides a comprehensive display of all potential schedules for the project defense events in this semester. We have 24 time slots for 18 events, thereby creating a surplus of 33%.

Table 9 presents all individuals who submitted the required registration forms. We have a total of 49 individuals: 18 are undergraduate students, 14 are UFSC professors, 2 are visiting professors, 2 are postdoctoral fellows, 5 are doctoral students, and 8 are external professionals. The X mark for undergraduate students indicates that they don't apply to evaluations.

Table 10 displays the availability of these individuals within the designated time slots. Green symbolizes availability for the given time slot, while red denotes unavailability.

Table 11 provides an overview of all the events along with their respective author, supervisor, advisor, possible evaluators, and interested evaluators. The X mark in the supervisor column signifies that the individual did not submit the registration forms. As a result, the event was scheduled without taking their requirements into account. We emphasize that an advisor can assume the role of a supervisor. In this semester, we have encountered four such instances.

One last consideration to take before we solve the problem is that the set of possible chairs is given by $\mathcal{C}^{\mathrm{P}} = \{p_{18}, p_{25}, p_{38}\}$. The preferred president is p_{18}.

This problem, while hard to solve manually is solved instantaneously by a mathematical solver. In more details, our full model has 49050 variables, all of which were binary, along with 34482 constraints, and 42676 nonzeros. This term refers to the number of nonzero coefficients in the constraint matrix, or in other words, the number of meaningful relationships between constraints and variables.

Table 12 delineates all the assembled events and their corresponding schedules. It is noteworthy to mention that we were only unsuccessful in scheduling one event, namely event e_{19}. Therefore, we managed to schedule 17 out of 18 events successfully.

There exist many possible causes for a failed scheduling attempt. In this specific instance, event e_{19} was not scheduled due to the absence of common availability among the author, supervisor, and advisor. This observation can be easily visualized

Table 8 – Schedule Table.

sID	Date	Hour
s_1	10/07/2023	08:00
s_2	10/07/2023	09:00
s_3	10/07/2023	10:00
s_4	10/07/2023	11:00
s_5	10/07/2023	14:00
s_6	10/07/2023	15:00
s_7	10/07/2023	16:00
s_8	10/07/2023	17:00
s_9	11/07/2023	08:00
s_{10}	11/07/2023	09:00
s_{11}	11/07/2023	10:00
s_{12}	11/07/2023	11:00
s_{13}	11/07/2023	14:00
s_{14}	11/07/2023	15:00
s_{15}	11/07/2023	16:00
s_{16}	11/07/2023	17:00
s_{17}	12/07/2023	08:00
s_{18}	12/07/2023	09:00
s_{19}	12/07/2023	10:00
s_{20}	12/07/2023	11:00
s_{21}	12/07/2023	14:00
s_{22}	12/07/2023	15:00
s_{23}	12/07/2023	16:00
s_{24}	12/07/2023	17:00

Source: Author.

in Table 10. The resolution to this problem is to manually reach out directly to the individuals involved in the event to negotiate a suitable schedule from the remaining available slots.

In relation to our secondary objectives, we were successful in allocating the preferred president in 16 out of the 17 successfully scheduled events. The only event where the preferred president could not be assigned was event e_1. This was a result of the preferred president holding the role of advisor for that particular event. Among the interested examiners, we were able to compose events with them 14 out of 17 times.

On the whole, the results obtained were satisfactory. All objectives were fulfilled in the most optimal manner, leading to significantly less work to the organizers compared to previous semesters.

Table 9 – People Table.

PID	Occupation	Possibility	Interested	Max Eval
p_1	Undergraduate student	X	X	X
p_2	Undergraduate student	X	X	X
p_3	Undergraduate student	X	X	X
p_4	Undergraduate student	X	X	X
p_5	Undergraduate student	X	X	X
p_6	Undergraduate student	X	X	X
p_7	Undergraduate student	X	X	X
p_8	Undergraduate student	X	X	X
p_9	Undergraduate student	X	X	X
p_{10}	Undergraduate student	X	X	X
p_{11}	Undergraduate student	X	X	X
p_{12}	Undergraduate student	X	X	X
p_{13}	Undergraduate student	X	X	X
p_{14}	Undergraduate student	X	X	X
p_{15}	Undergraduate student	X	X	X
p_{16}	Undergraduate student	X	X	X
p_{17}	Undergraduate student	X	X	X
p_{18}	UFSC Professor	[]	[]	0
p_{19}	Undergraduate student	X	X	X
p_{20}	UFSC Professor	[]	[]	0
p_{21}	UFSC Professor	[]	[]	0
p_{22}	External professional	[]	[e_7]	1.0
p_{23}	UFSC Professor	[e_9, e_{15}]	[e_{12}]	1.0
p_{24}	UFSC Professor	[e_3, e_9]	[e_{15}]	1.0
p_{25}	UFSC Professor	[]	[]	0
p_{26}	UFSC Professor	[e_{15}]	[]	1.0
p_{27}	PhD student	[e_1, e_3, e_8, e_9, e_{10}, e_{12}, e_{18}]	[e_4, e_6, e_{15}]	2.0
p_{28}	Postdoctoral fellow	[e_1, e_{12}]	[e_9, e_{15}]	3.0
p_{29}	External professional	[]	[]	0
p_{30}	External professional	[e_{10}, e_{18}]	[e_4, e_9]	3.0
p_{31}	External professional	[]	[e_{11}]	1.0
p_{32}	External professional	[e_3, e_9, e_{10}, e_{11}, e_{13}, e_{15}]	[e_1, e_6, e_{12}, e_{16}]	2.0
p_{33}	PhD student	[e_4]	[e_{15}]	2.0
p_{34}	Visiting professor	[e_2, e_6, e_{12}]	[e_1]	2.0
p_{35}	UFSC Professor	[]	[]	0
p_{36}	UFSC Professor	[e_1, e_5, e_7, e_8]	[]	1.0
p_{37}	UFSC Professor	[e_1, e_5, e_8, e_{11}, e_{17}]	[e_9, e_{10}, e_{13}, e_{18}]	2.0
p_{38}	UFSC Professor	[]	[]	0
p_{39}	External professional	[]	[e_{18}]	1.0
p_{40}	Postdoctoral fellow	[e_{11}, e_{12}, e_{16}, e_{18}]	[e_1]	2.0
p_{41}	External professional	[]	[e_{10}]	1.0
p_{42}	UFSC Professor	[]	[e_{14}]	1.0
p_{43}	PhD student	[e_5]	[e_2, e_{13}, e_{16}]	3.0
p_{44}	UFSC Professor	[]	[e_4, e_8]	1.0
p_{45}	PhD student	[]	[]	0
p_{46}	UFSC Professor	[]	[]	0
p_{47}	Visiting professor	[e_6, e_{14}, e_{16}]	[e_5, e_7]	2.0
p_{48}	External professional	[]	[]	0
p_{49}	PhD student	[e_3, e_6, e_8, e_{12}, e_{18}]	[e_2, e_{14}]	3.0

Source: Author.

Table 10 – Availability Table.

pID	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	s_{16}	s_{17}	s_{18}	s_{19}	s_{20}	s_{21}	s_{22}	s_{23}	s_{24}
p_1	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
p_2	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟥
p_3	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟥
p_4	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥
p_5	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥
p_6	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥
p_7	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟥	🟥	🟩	🟥	🟥	🟥	🟩	🟥	🟥	🟥	🟩	🟥
p_8	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
p_9	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩
p_{10}	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
p_{11}	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
p_{12}	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩
p_{13}	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
p_{14}	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩
p_{15}	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥
p_{16}	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟥	🟥
p_{17}	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩
p_{18}	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
p_{19}	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥
p_{20}	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟥	🟥	🟥
p_{21}	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥
p_{22}	🟩	🟩	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
p_{23}	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟩	🟩
p_{24}	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥
p_{25}	🟥	🟩	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥
p_{26}	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥
p_{27}	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟩	🟩	🟩	🟩
p_{28}	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
p_{29}	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥
p_{30}	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
p_{31}	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
p_{32}	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
p_{33}	🟩	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
p_{34}	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥
p_{35}	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟥	🟥	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩
p_{36}	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟥	🟩	🟥	🟥	🟩	🟩	🟩	🟩
p_{37}	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥
p_{38}	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥
p_{39}	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥	🟥	🟥	🟥
p_{40}	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
p_{41}	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
p_{42}	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
p_{43}	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩
p_{44}	🟥	🟩	🟥	🟥	🟥	🟥	🟩	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥	🟥	🟩	🟩	🟩	🟥	🟥	🟩	🟥
p_{45}	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩
p_{46}	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩
p_{47}	🟩	🟩	🟩	🟩	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
p_{48}	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩
p_{49}	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩

Source: Author.

Table 11 – Event Table.

eID	Author	Supervisor	Advisor	Possibility	Interested
e_1	p_1	p_{18}	p_{18}	$[p_{27}, p_{28}, p_{36}, p_{37}]$	$[p_{32}, p_{34}, p_{40}]$
e_2	p_2	p_{27}	p_{20}	$[p_{34}]$	$[p_{43}, p_{49}]$
e_3	p_3	X	p_{38}	$[p_{24}, p_{27}, p_{32}, p_{49}]$	$[p_{49}]$
e_4	p_4	p_{49}	p_{20}	$[p_{33}]$	$[p_{27}, p_{30}, p_{44}]$
e_5	p_5	X	p_{42}	$[p_{36}, p_{37}, p_{43}]$	$[p_{47}]$
e_6	p_6	p_{21}	p_{21}	$[p_{34}, p_{47}, p_{49}]$	$[p_{49}, p_{27}, p_{32}]$
e_7	p_7	X	p_{42}	$[p_{36}]$	$[p_{22}, p_{47}]$
e_8	p_8	p_{20}	p_{20}	$[p_{27}, p_{36}, p_{37}, p_{49}]$	$[p_{44}]$
e_9	p_9	X	p_{35}	$[p_{23}, p_{24}, p_{27}, p_{32}]$	$[p_{28}, p_{30}, p_{37}]$
e_{10}	p_{10}	p_{41}	p_{42}	$[p_{49}, p_{27}, p_{30}, p_{32}]$	$[p_{37}, p_{41}]$
e_{11}	p_{11}	p_{31}	p_{36}	$[p_{32}, p_{37}, p_{40}]$	$[p_{31}]$
e_{12}	p_{12}	p_{45}	p_{46}	$[p_{27}, p_{28}, p_{34}, p_{40}, p_{49}]$	$[p_{23}, p_{49}, p_{32}]$
e_{13}	p_{13}	p_{48}	p_{26}	$[p_{32}]$	$[p_{37}, p_{43}]$
e_{14}	p_{14}	p_{32}	p_{25}	$[p_{47}]$	$[p_{42}, p_{49}]$
e_{15}	p_{15}	p_{38}	p_{38}	$[p_{23}, p_{26}, p_{49}, p_{32}]$	$[p_{24}, p_{27}, p_{28}, p_{33}]$
e_{16}	p_{16}	p_{29}	p_{20}	$[p_{40}, p_{47}]$	$[p_{32}, p_{43}]$
e_{17}	p_{17}	X	p_{25}	$[p_{37}]$	[]
e_{19}	p_{19}	p_{39}	p_{37}	$[p_{49}, p_{27}, p_{30}, p_{40}, p_{49}]$	$[p_{37}, p_{39}]$

Source: Author.

Table 12 – Scheduled and Composed Events.

eId	Author	Supervisor	Advisor	Examiner	Chairperson	Schedule
e_1	p_1	p_{18}	p_{18}	p_{34}	p_{25}	s_{15}
e_2	p_2	p_{27}	p_{20}	p_{49}	p_{18}	s_6
e_3	p_3	X	p_{38}	p_{49}	p_{18}	s_{10}
e_4	p_4	p_{49}	p_{20}	p_{30}	p_{18}	s_5
e_5	p_5	X	p_{42}	p_{47}	p_{18}	s_{21}
e_6	p_6	p_{21}	p_{21}	p_{27}	p_{18}	s_9
e_7	p_7	X	p_{42}	p_{47}	p_{18}	s_{11}
e_8	p_8	p_{20}	p_{20}	p_{44}	p_{18}	s_{14}
e_9	p_9	X	p_{35}	p_{30}	p_{18}	s_2
e_{10}	p_{10}	p_{41}	p_{42}	p_{37}	p_{18}	s_1
e_{11}	p_{11}	p_{31}	p_{36}	p_{40}	p_{18}	s_8
e_{12}	p_{12}	p_{45}	p_{46}	p_{32}	p_{18}	s_{23}
e_{13}	p_{13}	p_{48}	p_{26}	p_{43}	p_{18}	s_7
e_{14}	p_{14}	p_{32}	p_{25}	p_{42}	p_{18}	s_{12}
e_{15}	p_{15}	p_{38}	p_{38}	p_{33}	p_{18}	s_{18}
e_{16}	p_{16}	p_{29}	p_{20}	p_{43}	p_{18}	s_{13}
e_{17}	p_{17}	X	p_{25}	p_{37}	p_{18}	s_{22}
e_{19}	p_{19}	p_{39}	p_{37}	X	X	X

Source: Author.

7 CONCLUSION

In this study, we addressed a practical management challenge: the organization and scheduling of project defense events, specifically tailored to the Control and Automation major at UFSC. Our comprehensive solution optimally resolves this issue, thereby saving a significant amount of time for event organizers each semester. Due to its efficiency and ease of maintenance, this solution is ideally suited for future semesters.

Our approach employed mathematical programming to create an elegant, efficient, and interpretative model. This ensures that in future semesters, event organizers can intuitively incorporate, exclude, or modify elements in the model according to their evolving needs.

We also developed a data collection application using Google Workspace apps such as Google Forms, Google Sheets, and Google Apps Script. These user-friendly tools require no hosting and mitigate potential future instabilities. The designed applications automatically generate forms, handle user data, pre-process it, and transfer the information to our computational system in an organized manner.

Additionally, we developed a computational system to handle the optimization and analysis of our problem and its solutions. This was implemented using Julia, with the JuMP package to handle mathematical optimization. This system has excellent interactivity, flexibility, and efficiency, which made it well-suited to our project's needs.

Specifically, we applied this solution to the project defense events of the Control and Automation major during the first semester of 2023. We had 18 events to schedule and managed to successfully allocate 17 of them. The only unsuccessful case was one that was infeasible from the outset, as the individuals involved in the event did not have any overlapping schedules.

With respect to our secondary objectives, which aimed to further improve the quality of the events, we managed to allocate the preferred chair in 16 out of the 17 successfully scheduled events. The only event where the preferred chair could not be assigned was due to the preferred chair holding the role of advisor for that particular event. Among the interested examiners, we were able to include them in the events 14 out of 17 times.

Therefore, we conclude that the obtained results were exceptional. We managed to automate the entire process, significantly reducing the workload for the event organizers, and also deliver superior project defense events with interested and capable examiners.

Despite the completeness nature of the system, we acknowledge the potential for continuous improvement and maintenance. We recommend further development in the form structure to enhance clarity for all parties involved. There also exists considerable scope for generating new scripts that can further facilitate comprehension of the forms, thereby minimizing user errors. Lastly, this system needs to dynamically adapt to the preferences of the organizers, thereby necessitating the integration of features that better align with the needs of organizers and users.

REFERENCES

ABRAMSON, David; KRISHNAMOORTHY, Mohan; DANG, Henry, et al. Simulated annealing cooling schedules for the school timetabling problem. **Asia Pacific Journal of Operational Research**, v. 16, p. 1–22, 1999.

BABAEI, Hamed; KARIMPOUR, Jaber; HADIDI, Amin. A survey of approaches for university course timetabling problem. **Computers & Industrial Engineering**, v. 86, p. 43–59, 2015. Applications of Computational Intelligence and Fuzzy Logic to Manufacturing and Service Systems. ISSN 0360-8352. DOI: https://doi.org/10.1016/j.cie.2014.11.010.

BURKE, Edmund K; ELLIMAN, David; WEARE, Rupert. A genetic algorithm based university timetabling system. In: PROCEEDINGS of the 2nd East-West International Conference on Computer Technologies in Education. [S.l.: s.n.], 1994. P. 35–40.

CAPRARA, Alberto; MONACI, Michele; TOTH, Paolo; GUIDA, Pier Luigi. A Lagrangian heuristic algorithm for a real-world train timetabling problem. **Discrete Applied Mathematics**, Elsevier, v. 154, n. 5, p. 738–753, 2006.

DE WERRA, D. An introduction to timetabling. **European Journal of Operational Research**, v. 19, n. 2, p. 151–162, 1985. ISSN 0377-2217. DOI: https://doi.org/10.1016/0377-2217(85)90167-5.

EVEN, Shimon; ITAI, Alon; SHAMIR, Adi. On the complexity of time table and multi-commodity flow problems. In: IEEE. 16TH annual symposium on foundations of computer science (sfcs 1975). [S.l.: s.n.], 1975. P. 184–193.

GROSS, Jonathan L.; YELLEN, Jay; ZHANG, Ping. **Handbook of Graph Theory, Second Edition**. 2nd. [S.l.]: Chapman; Hall/CRC, 2013. ISBN 1439880182.

GUYON, O.; LEMAIRE, P.; PINSON, É.; RIVREAU, D. Cut generation for an integrated employee timetabling and production scheduling problem. **European Journal of Operational Research**, v. 201, n. 2, p. 557–567, 2010. ISSN 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2009.03.013.

PARBO, Jens; NIELSEN, Otto Anker; PRATO, Carlo Giacomo. Passenger Perspectives in Railway Timetabling: A Literature Review. **Transport Reviews**, Routledge, v. 36, n. 4, p. 500–526, 2016. DOI: 10.1080/01441647.2015.1113574.

VANDERBEI, Robert J et al. **Linear programming**. [S.l.]: Springer, 2020.

APPENDIX A – COMPLETE OPTIMIZATION CODE

The following Julia code defines all the sets as they were presented in the modeling chapter of this document. The sets are constructed based on structured tabular data that is exported from Google Sheets and subsequently transformed into a DataFrame object in Julia.

```julia
### Schedule Set
mutable struct Schedule
    date::DateTime
end

function generate_schedule_set(schedule_df::DataFrame)
    schedule_set = Vector{Schedule}();
    n = size(schedule_df)[1];
    for i in 1:1:n
        push!(schedule_set, Schedule(schedule_df.Schedule[i]));
    end

    return schedule_set
end

### Person Set
mutable struct Person
    name::AbstractString
    email::AbstractString
    occupation::AbstractString
    availability::Any
    possibility::Any
    interest::Any
    max_eval::Int
    Person(Name, Email, Occupation) = new(Name, Email, Occupation);
    Person(df::DataFrame, i::Int64) = new(df.Name[i], df.Email[i], df.Occupation[i]);
    Person(df::DataFrame, i::Int64, availability::Any, interest::Any) = new(df.Name[i], df.Email[i],
    df.Occupation[i]);
end

function generate_people_set(people_df::DataFrame, schedule_set::Vector)
    people_set = Vector{Person}();
    n = size(people_df)[1];
    for i in 1:1:n
        person = Person(people_df, i)
        person.availability = [schedule for schedule in schedule_set if schedule.date
        in people_df.Availability[i]]
        if ismissing(people_df[i, "Max Evaluations"])
            person.max_eval = 0;
        else
            person.max_eval = people_df[i, "Max Evaluations"];
        end
        push!(people_set, person);
    end
    return people_set
end

### Event Set
mutable struct Event
    title::AbstractString
    academic::Person
    advisor::Person
    supervisor::Person
    interested::Any
    president::Person # Assign
    evaluator::Person # Assign
    schedule::Schedule # Assign
    Event(title::AbstractString, academic::Person, advisor::Person, supervisor::Person) = new(title,
    academic, advisor, supervisor);
end

function generate_event_set(event_df::DataFrame, people_df::DataFrame, people_set::Vector,
schedule_set::Vector)
    event_set = Vector{Event}();
    n = size(event_df)[1];
    # Add the people objects
    c1, c2, c3 = 0, 0, 0
    academic, advisor, supervisor = Any, Any, Any
    for i in 1:1:n # for each event
        println("Event $i")
        for person in people_set
            if event_df.Academic[i] == person.name
                #println("Academic: $(person.name)")
                academic = person
                c1 += 1
            end
            if event_df.Advisor[i] == person.name
                #println("Advisor: $(person.name)")
                advisor = person
                c2 += 1
            end
            if event_df.Supervisor[i] == person.name
                #println("Supervisor: $(person.name)")
                supervisor = person
                c3 += 1
            end
        end
        if c1 != i # Check if the academic is in the event set
            print("Academic $(event_df.Academic[i]) Error!!!")
            c1 += 1;
        end
        if c2 != i # Check if the advisor is in the event set
            print("Check if the Advisor $(event_df.Advisor[i]) didn't fill the form")
            person = Person("Fake Advisor $i", "FakeAdvisor$i@email.com", "Fake")
            person.availability = [schedule for schedule in schedule_set]
            advisor = person;
            push!(people_set, advisor);

            c2 += 1;
        end
        if c3 != i # Check if the academic is in the event set
            print("Check if the Supervisor $(event_df.Supervisor[i]) didn't fill the form")
            person = Person("Fake Supervisor $i", "FakeSupervisor$i@email.com", "Fake")
            person.availability = [schedule for schedule in schedule_set]
            supervisor = person;
            push!(people_set, supervisor);
            c3 += 1;
        end
        push!(event_set, Event(event_df.Title[i], academic, advisor, supervisor));
    end
    @assert c1 == c2 == c3
```

```julia
    # Add the possiblity and interest to the people_set
    for p in 1:size(people_df)[1]
        person = people_set[p]
        person.interest = [event for event in event_set if event.title in people_df.Interested[p]]
        person.possibility = [event for event in event_set if event.title in people_df.Possible[p]]
    end
    return event_set
end;
```

The following Julia code relates to the formulation of the optimization problem. It takes as input the three sets defined earlier. The code defines all variables, sets, subsets, constraints, and objectives as outlined in the modeling chapter. It is important to note that there is no need to utilize any specific optimizer attribute to solve this problem.

```julia
function optimization_full_model(people_set, event_set, schedule_set)

    # Local Variables
    # .potential evaluator names
    C_E = [[p for p in 1:length(people_set) if people_set[p].occupation in ["PhD student",
    "Postdoctoral fellow", "Visiting professor", "UFSC Professor","External professional"] &&
    people_set[p]     [event_set[e].advisor, event_set[e].supervisor]] for e in 1:length(event_set)];
    # .potential president names
    possible_presidents_names = ["Prof1","Prof2","Prof3"]
    C_P = [[p for p in 1:length(people_set) if people_set[p].name in possible_presidents_names &&
    people_set[p]     [event_set[e].advisor, event_set[e].supervisor]] for e in 1:length(event_set) ];
    # .Availability in common of the academic, advisor and supervisor
    He = [[h for h in 1:length(schedule_set) if schedule_set[h] in
    intersect(event_set[e].academic.availability, event_set[e].advisor.availability,
    event_set[e].supervisor.availability)] for e in 1:length(event_set)];

    # Optimization Model
    opt_model = Model(Gurobi.Optimizer);

    # Optimization Variables
    @variable(opt_model, x[e=1:length(event_set),h=1:length(schedule_set)], set=MOI.ZeroOne());
    @variable(opt_model, z[e=1:length(event_set)], set=MOI.ZeroOne());
    @variable(opt_model, yP[p=1:length(people_set),e=1:length(event_set)], set=MOI.ZeroOne());
    @variable(opt_model, yE[p=1:length(people_set),e=1:length(event_set)], set=MOI.ZeroOne());
    @variable(opt_model, wP[p=1:length(people_set),e=1:length(event_set),h=1:length(schedule_set)],
    set=MOI.ZeroOne());
    @variable(opt_model, wE[p=1:length(people_set),e=1:length(event_set),h=1:length(schedule_set)],
    set=MOI.ZeroOne());

    # Constraints
    #.1
    for e in 1:length(event_set)
        temp = @constraint(opt_model, sum(x[e,He[e]]) == z[e])
        set_name(temp, "c1_e:$e")
        temp = @constraint(opt_model, sum(x[e,setdiff(1:length(schedule_set),He[e])]) == 0)
        set_name(temp, "c1!_e:$e")
    end # = #
    #.2
    for h in 1:length(schedule_set)
        Le = [e for e in 1:length(event_set) if h in He[e]]
        temp = @constraint(opt_model, sum(x[Le,h]) <= 1)
        set_name(temp, "c2_h:$h")
        temp = @constraint(opt_model, sum(x[setdiff(1:length(event_set),Le),h]) == 0)
        set_name(temp, "c2!_h:$h")
    end # = #
    #.3
    for e in 1:length(event_set)
        temp = @constraint(opt_model, sum(yE[C_E[e],e]) == z[e])
        set_name(temp, "c3_e:$e")
        temp = @constraint(opt_model, sum(yE[setdiff(1:length(people_set),C_E[e]),e]) == 0)
        set_name(temp, "c3!_e:$e")
    end # = #
    #.4
    for e in 1:length(event_set)
        temp = @constraint(opt_model, sum(yP[C_P[e],e]) == z[e])
        set_name(temp, "c4_e:$e")
        temp = @constraint(opt_model, sum(yP[setdiff(1:length(people_set),C_P[e]),e]) == 0)
        set_name(temp, "c4!_e:$e")
    end # = #
    #.5
    for e in 1:length(event_set)
        Lp = [_p for _p in intersect(C_E[e],C_P[e])]
        for p in Lp
            temp = @constraint(opt_model, yP[p,e] + yE[p,e] <= 1)
            set_name(temp, "c5_e:$(e) p:$(p)")
        end
    end # = #
    #.6
    for e in 1:length(event_set)
        Le = [_p for _p in C_E[e] if event_set[e] in union(people_set[_p].interest,
        people_set[_p].possibility)]
        for h in He[e]
            for p in 1:length(people_set)
                if p in Le
                    temp = @constraint(opt_model, wE[p,e,h] <= yE[p,e] )
                    set_name(temp, "6Ea_e:$e p:$p h:$h")
                    temp = @constraint(opt_model, wE[p,e,h] <= x[e,h] )
                    set_name(temp, "6Eb_e:$e p:$p h:$h")
                    temp = @constraint(opt_model, wE[p,e,h] >= yE[p,e] + x[e,h] - 1 )
                    set_name(temp, "6Ec_e:$e p:$p h:$h")
                    if schedule_set[h] in people_set[p].availability
                        temp = @constraint(opt_model, wE[p,e,h] <= 1 )
                    else
                        temp = @constraint(opt_model, wE[p,e,h] <= 0 )
                    end
                    set_name(temp, "6Ed_e:$e p:$p h:$h")
                else
                    temp = @constraint(opt_model, wE[p,e,h] == 0)
                    temp = @constraint(opt_model, yE[p,e] == 0)
                end
            end
        end
    end # = #
    #.7
    for e in 1:length(event_set)
        for h in He[e]
            for p in 1:length(people_set)
                if p in C_P[e]
                    temp = @constraint(opt_model, wP[p,e,h] <= yP[p,e] )
                    set_name(temp, "7Pa_e:$e p:$p h:$h")
                    temp = @constraint(opt_model, wP[p,e,h] <= x[e,h] )
                    set_name(temp, "7Pb_e:$e p:$p h:$h")
                    temp = @constraint(opt_model, wP[p,e,h] >= yP[p,e] + x[e,h] - 1 )
                    set_name(temp, "7Pc_e:$e p:$p h:$h")
                    if schedule_set[h] in people_set[p].availability
                        temp = @constraint(opt_model, wP[p,e,h] <= 1 )
                    else
                        temp = @constraint(opt_model, wP[p,e,h] <= 0 )
                    end
                    set_name(temp, "7Pd_e:$e p:$p h:$h")
                else
                    temp = @constraint(opt_model, wP[p,e,h] == 0)
                    temp = @constraint(opt_model, yP[p,e] == 0)
                end
            end
        end
    end # = #
    #.8
    for p in 1:length(people_set)
        person = people_set[p];
        temp = @constraint(opt_model, sum([yE[p,e] for e in 1:length(event_set)]) <= person.max_eval)
        set_name(temp, "8:p:$p")
    end

    # Objective
    @objective(opt_model, Max, sum(z))

    # Show the complete model
    # print(opt_model)

    # Optimize
    #.Optimization parameters
    set_optimizer_attribute(opt_model, "OutputFlag", 0)
    # set_optimizer_attribute(opt_model, "TimeLimit", 100)
    # set_optimizer_attribute(opt_model, "MIPGap", 1e-6)
    # set_optimizer_attribute(opt_model, "NumericFocus", 3)
    # set_optimizer_attribute(opt_model, "MIPFocus", 3)
    # set_optimizer_attribute(opt_model, "NonConvex", 2)

    optimize!(opt_model)
    @show termination_status(opt_model)
    # return

    # Show solution
    allocated_events = Int(sum([value(z[e]) for e in 1:length(event_set)]))
    println("Successfully allocated $(allocated_events) events out of the $(length(event_set))
    possible events")

    for e in 1:length(event_set)
        # println("TEST x[e,h]:$([value(x[e,_h]) for _h in 1:length(test_schedule_set)[1]])")
        # println("TEST yA[p,e]:$([value(yA[_p,e]) for _p in 1:length(test_people_set)[1]])")
        # println("TEST yP[p,e]:$([value(yP[_p,e]) for _p in 1:length(test_people_set)[1]])")

        # =
        #.Find the allocated schedule
        _, h = findmax([value(x[e,_h]) for _h in 1:length(schedule_set)])
        #.Find the allocated evaluator
        _, p_evaluator = findmax([value(yE[_p,e]) for _p in 1:length(people_set)])
        #.Find the allocated president
        _, p_president = findmax([value(yP[_p,e]) for _p in 1:length(people_set)])
        event = event_set[e]
        schedule = schedule_set[h]
        evaluator = people_set[p_evaluator]
        president = people_set[p_president]
        println("_"^40)
        if value(z[e]) == 1
            println("Event $e was successfully allocated to $schedule")
            println("Title:$(event.title)")
            println("Author:$(event.academic.name)")
            println("Advisor:$(event.advisor.name)")
            println("Supervisor:$(event.supervisor.name)")
            println("Evaluator:$(evaluator.name)")
            println("President:$(president.name)")
            #.Export solution
            event_set[e].schedule = schedule;
            event_set[e].evaluator = evaluator;
            event_set[e].president = president;
        else
            println("Event $e wasn't allocated")
            println("Title:$(event.title)")
            println("Author:$(event.academic.name)")
            println("Advisor:$(event.advisor.name)")
            println("Supervisor:$(event.supervisor.name)")
            #.Export solution
            #event_set[e].schedule = Schedule();
            #event_set[e].evaluator = Person();
            #event_set[e].president = Person();
        end
        println("_"^40)
    end
    # =#
end;
```