FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

A Metaheuristic Approach to Improving University Timetables Using Genetic Algorithms and Simulated Annealing

Miguel Azevedo Lopes

Mestrado em Engenharia Informática e Computação

Supervisor: Prof. Daniel Castro Silva

August 1, 2024

A Metaheuristic Approach to Improving University Timetables Using Genetic Algorithms and Simulated Annealing

Miguel Azevedo Lopes

Mestrado em Engenharia Informática e Computação

August 1, 2024

Abstract

Every semester at the Faculty of Engineering of the University of Porto, timetables for classes, professors, and rooms are generated using specialized software. However, these solutions often fall short of meeting non-essential constraints, such as consolidating professor courses into fewer days and eliminating free intervals between class lessons. Consequently, manual optimizations are conducted each semester at the Department of Informatics Engineering - a complex and time consuming task. This dissertation proposes an automated solution to optimize timetables, aiming to eliminate the need for manual interventions.

The problem addressed falls under the broad category of university timetabling problems, commonly referred to in the literature as the University Course Timetabling Problem (UCTP), and, more specifically, under the Curriculum-Based Course Timetabling variant (CB-CTT). It differs slightly from typical timetabling problems because the starting point is an already existing, feasible solution.

An analysis of state-of-the-art methodologies highlighted metaheuristic approaches as particularly effective for similar optimization challenges. To tackle this NP-hard problem, two main metaheuristic approaches were employed: genetic algorithm and simulated annealing. Additionally, a greedy local search was developed to serve as a baseline comparison for the two main approaches.

The proposed approaches were evaluated by optimizing timetables from two different semesters at the Faculty of Engineering of the University of Porto. The evaluation considered adherence to stipulated constraints and computational metrics including time and memory consumption.

Results showed the genetic algorithm to be the best performing algorithm, producing the highest quality timetables. Simulated annealing also improved upon the starting solution, but less reliably and with worse overall results.

Keywords: university course timetabling problem, timetabling optimization, metaheuristics, curriculum based course timetabling

Resumo

Todos os semestres, na Faculdade de Engenharia da Universidade do Porto, são gerados horários de aulas, professores e salas utilizando de software especializado. No entanto, na maior parte das vezes, estas soluções não conseguem satisfazer restrições não essenciais, tais como a consolidação de aulas de professores em menos dias e a eliminação de intervalos livres entre aulas. Consequentemente, todos os semestres são efectuadas optimizações manuais no Departamento de Engenharia Informática - uma tarefa complexa e morosa. Esta dissertação propõe uma solução automatizada para a otimização de horários, com o objetivo de eliminar a necessidade de intervenções manuais.

O problema abordado enquadra-se na categoria alargada de problemas de geração de horários universitários, tipicamente designada na literatura como Problema de Geração de Horários Universitário e, mais especificamente, na variante de Geração de Horários Baseada em Currículo. Difere ligeiramente dos problemas típicos de geração de horários porque o ponto de partida é uma solução já existente e viável.

Uma análise das metodologias mais avançadas destacou as abordagens metaheurísticas como particularmente eficazes para desafios de otimização semelhantes. Para resolver este problema NP-hard, foram utilizadas duas principais abordagens metaheurísticas: um algoritmo genético e algoritmo de arrefecimento simulado. Adicionalmente, foi desenvolvida uma pesquisa local gulosa para servir de base de comparação para as duas abordagens principais.

As abordagens propostas foram avaliadas através da otimização de dois horários de semestres distintos da Faculdade de Engenharia da Universidade do Porto. A avaliação considerou a adesão às restrições estipuladas e métricas computacionais, incluindo tempo e consumo de memória.

Os resultados mostraram que o algoritmo genético foi o algoritmo com melhor desempenho, produzindo os horários de maior qualidade. O algoritmo de simulated annealing também melhorou a solução inicial, mas de forma menos fiável e com piores resultados globais.

Keywords: problema de geração de horários universitários, otimização de horários, metaheurísticas, geração de horários baseada em currículo

Agradecimentos

Gostaria de primeiro, agradecer à minha mãe. Não só por ser a minha primeira amiga, mas também por ser a melhor que tenho.

Ao meu pai, por me ensinar o valor do trabalho. Sabendo do quão gostaste da cadeira de Investigação Operacional, acho que gostarias do tema deste trabalho.

Aos meus amigos, por darem sentido à minha vida. O principal dissabor de acabar o meu percurso académico e iniciar a minha vida profissional, é saber que não passarei tanto tempo convosco.

Ao Professor Daniel Silva, não só por toda a ajuda ao longo deste trabalho, mas por ter ido sempre além daquilo que era a sua obrigação. Ao longo do meu percurso académico tive vários professores, a maior parte dos quais me deixaram com um sentimento negativo em relação à academia. O último semestre fez me esquecer as más experiências. Acho que o melhor elogio que lhe posso fazer é que me deixou com vontade de um dia ser (bom) professor. Obrigado!

Miguel Azevedo Lopes

"Act out being alive, like a play. And after a while, a long while, it will be true"

John Steinbeck, East of Eden

Contents

List of Figures

List of Tables

+

Chapter 1

Introduction

Timetabling is a complex challenge encountered across various sectors and is crucial in coordinating schedules and optimizing resources. In academia, the task becomes intricate, requiring careful attention to the specific needs of educational institutions. The challenge often involves coordinating students, teachers, classrooms, and timeslots within specific constraints, which, in many instances, are contingent on the policies of individual educational institutions. University timetabling adds further complexity due to diverse degree programs with unique and evolving curricula. This necessitates a nuanced and adaptable approach.

The complexity of this process can be exemplified by a standard bachelor's degree, which has a three-year duration, thereby necessitating the creation of three distinct course offerings during each semester, corresponding to the specific curriculum of that degree. Within each of these offerings, multiple classes are formed, each comprising a fixed number of students. The number of classes is proportional to the number of students signed up for these offerings, so different courses will have different numbers of classes. As a result, a unique and tailored timetable is formulated for each class, encompassing the combination of lessons that the class' students must attend. A course is typically composed of lectures (theoretical lessons) or practical lessons, and, in most cases, both types.

In the previous example, the focus is placed on generating timetables for classes, however it's important to acknowledge that timetables are just a grouping of events, in this case lessons, selected from a certain perspective: class timetables group all the events a class of students must attend, however, these lessons can also be grouped to form professor timetables, which contain all the lessons a given professor must teach, or even timetables for the rooms these events (lessons) are taking place in.

The complexity inherent in the timetable generation process becomes pronounced, particularly within the context of the Faculty of Engineering at the University of Porto, where the student body currently comprises nearly 8000 individuals, and the faculty roster exceeds 600 professors (FEUP, 2023). This translates to around 1800+ lessons per semester and almost 450+ classes belonging to 60+ degrees. The intricacy of this process is further accentuated by the continual growth in the number of students and professors observed each year. Notably, the physical infrastructure of the

faculty, specifically the available rooms (around 150), has not undergone expansion, which further complicates the process.

1.1 Context and Motivation

The Faculty of Engineering at the University of Porto currently handles the task of timetable generation by using specialized software for this task, provided by Bullet Solutions (2023). However, the generation of the timetables remains a time-intensive process, often extending over several weeks. Invariably, the resultant solution tends to adhere primarily to mandatory constraints, leaving an optimal or even a near-optimal solution unrealized. The exigency of time constraints precludes the exhaustive exploration necessary for the software to find the most optimal timetable configuration within the stipulated time frame.

To address the lack of quality in the generated timetables, the Department of Informatics Engineering assumes the task of manually fine-tuning the timetables of the degrees under its purview. This manual optimization consists of enhancing the timetables, by changing them to make them adhere to supplementary, non-essential constraints. These constraints optimize the timetables from the perspective of the several stakeholders that are affected by them. For example:

- Professors' timetables are optimized to consolidate lessons into as few days as practicable within their timetable, so they can have more free time for other activities (whether it be research, administrative tasks, or even working a separate corporate job, which is the case of some invited lecturers);
- Classes' timetables are optimized to avoid isolated lessons in the timetable, as well as consecutive theoretical lessons (as opposed to an intertwined mix of practical and theoretical lessons); they are also optimized to be similar (in terms of timeslot occupancy) when they belong to the same curricular year of a degree, and to be compatible with other lessons of different curricular years of the same degree, so that students who fail courses can retake them in the following year.

This hands-on approach is necessitated by the imperative to not only meet the baseline requirements, a task which the existing timetabling generation software already fulfills, but also to ensure a more efficacious utilization of available resources and enhance the overall quality of the generated timetables from the perspective of both professors and students (classes). The motivation behind this dissertation lies in recognizing the challenges embedded in the existing method of optimization, which is very time-consuming and doesn't always produce the best solutions.

1.2 Main Objectives, Research Questions and Methodology

The main goal of this dissertation is to replace the labor-intensive manual optimization currently conducted by the Department of Informatics Engineering's staff.

The initial phase of this work entailed a research process to identify algorithms capable of addressing the unique challenges inherent in timetable optimization. Following this, the selected algorithms were developed, and their respective performances were compared.

Throughout the development of this dissertation, three key questions were posed, which helped guide the research process. Those questions are highlighted bellow:

- **R.Q.1.:** What algorithms have demonstrated most efficacious in solving similar timetabling problems?
- **R.Q.2.:** How do those algorithms adapt to a problem of optimization of an existing timetable?
- **R.Q.3.:** How do the selected algorithms compare to a naive approach in terms of performance?

1.2.1 Algorithm Selection

An exploration of state-of-the-art algorithms was undertaken to identify the most suitable algorithm for optimizing university timetables. This involved an extensive review of various research papers that provided insights and analyses on the topic. Examining these research papers aimed to glean information regarding the comparative performance, and effectiveness of different algorithms in addressing the specific challenges posed by this dissertation. The process of algorithm selection followed these key steps:

1. **Identification of Algorithmic Approaches:** The initial step involved identifying the most effective approaches/types of algorithms in the literature;
2. **Performance Assessment on Benchmark Datasets:** Subsequently, the selection was refined by considering the algorithms' performance on benchmark datasets. This approach provided an objective basis for comparison, as it allowed for evaluating algorithms when applied to the same problem with identical datasets.
3. **Relevance in the Context of the Specific Problem:** From the shortlisted algorithms, the final selection was made based on how closely each algorithm's problem definition aligned with the particular requirements of the problem that motivated this dissertation.

This process resulted in the selection of a genetic algorithm, for its proven versatility, on both real-world and benchmark problem applications, and simulated annealing, for its positive performance in optimizing existing timetables when utilized in hybrid approaches. A more in-depth analysis of these types of algorithms is conducted in sections 2.4 and 2.5. Also, additional insights into the reasoning behind the selection are presented, alongside a review of existing approaches in the literature, on chapter 3.

1.2.2 Performance Evaluation

The assessment of each implemented algorithm focused on its efficacy in generating a timetable that complies with the specified problem constraints. The adherence to these constraints was evaluated through the fitness value of the solution, provided by the fitness function, which measures a timetable's conformity to the imposed constraints (as described in section 5.3).

In recognition of the pivotal role of time, as emphasized in the introduction of the problem, an additional evaluative dimension will center on the computational efficiency of each algorithm. This criterion specifically measures the algorithm's capacity to improve the existing solution's fitness value within several time frames, considering both the speed and memory consumption of the process. This analysis will allow the staff at the Department of Informatics to choose the best algorithm to optimize the schedules based on the time available to conduct the process, as well as whether the resources available to run the algorithm are limited in memory and processing threads (a regular computer), or for when a high-performance computer is available, which in some algorithms might allow for a broader exploration of the search space, and consequently lead to better solutions.

In essence, the assessment encompasses the algorithm's ability to meet imposed constraints and its computational efficiency.

1.3 Achievements and Contributions

The key contributions made by this dissertation can be outlined as follows:

- Review of the existing literature on the University Course Timetabling Problem, with a special focus on Curriculum-Based Course Timetabling Problem;
- Development of three approaches that optimize a pre-existing timetable to satisfy as many constraints as possible: a genetic algorithm, a simulated annealing algorithm and a greedy local search algorithm;
- Develop and evaluate the performance of three solutions for a version of the University Course Timetabling Problem, which considers constraints that haven't previously been addressed in the literature.

An additional achievement will be a decrease in the workload of the staff at the Department of Informatics Engineering responsible for optimizing the timetables;

1.4 Document Outline

Building upon the introductory section, which elucidated the context, motivation, primary objectives, research questions, methodology, and achievements and contributions, the subsequent sections of this paper are now delineated.

In Chapter 2, essential background information is provided, covering terminology related to timetabling, an exploration of the University Course Timetabling Problem, an elucidation of the timetabling process at the Faculty of Engineering, an analysis of the types of constraints involved, and an introduction to the testing dataset employed for the research, as well as background on the algorithms employed to solve this problem, namely genetic algorithms and simulated annealing.

Chapter 3 delves into the existing body of work in the timetabling optimization field, exploring various types of approaches to the University Course Timetabling Problem, with a special emphasis on metaheuristics approaches.

Chapter 4 concentrates on the problem under investigation and outlines the developed solutions. It formalizes the optimization challenge and delineates the problem specific constraints.

Chapters 5,6 and 7 present the three approaches developed to tackle the problem: using a genetic algorithm, simulated annealing and greedy local search, respectively.

Finally, chapter 8 includes the results comparing the three algorithms, as well as the main findings, and in chapter 9 the conclusions are presented.

Chapter 2

Background

In this chapter, essential terminology and foundational concepts pertinent to this research are discussed, beginning with an overview of the University Course Timetabling Problem (UCTP) and its significance in optimizing academic schedules. The specific timetabling strategies employed by the Department of Informatics Engineering (DEI) of the Faculty of Engineering at the University of Porto (FEUP) are explored, highlighting the challenges and solutions unique to this problem. Additionally, a comprehensive background on the two primary algorithmic approaches utilized in this study, genetic algorithms and simulated annealing, is provided, detailing their mechanisms. This groundwork will equip readers with the necessary context to understand and appreciate the methodologies and findings presented in subsequent sections.

2.1 Terminology

Within the framework of this dissertation, certain terms will be consistently employed. To clarify, the definitions of these terms in the context of the problem under consideration are presented below:

- **Degree** refers to an academic program of study that students enroll in, typically spanning multiple years, and culminating in the awarding of an academic qualification. Each degree comprises a collection of curriculums, one for each semester of study, and these curriculums are composed of various courses;
- **Course** denotes a specific subject or unit of study within a degree program, typically taught over a semester. Examples include Calculus, Algorithms and Data Structures, Thermodynamics, etc;
- **Class** denotes a collective assembly of students who have lessons together. Each class possesses a timetable comprising multiple assigned courses;
- **Lessons** encompass either lectures (theoretical lessons) or practical lessons (in the Faculty of Engineering of the University of Porto, a further distinction is made within practical lessons, classifying them as either practical, theoretical-practical or laboratory lessons,

however, for the purposes of optimization, this distinction is disregarded). Consequently, the term "class" consistently pertains to a group of students and not to an individual lesson;

- **Professor** is the term used to refer to the faculty staff member instructing a lesson;
- **Room** is the physical space where lessons take place. Rooms can have different capacities and be of different types: computer room, regular room, auditorium, or laboratory. Some types of rooms are typically used for certain types of lessons: computer, laboratory and regular rooms are typically destined for practical lessons, as opposed to auditoriums, which have a bigger capacity and therefore can host the lectures (theoretical lessons), which are normally attended by more than one class at the same time;
- **Shift** is a group of practical lessons assigned to classes that attend the same theoretical lessons.

2.2 The University Course Timetabling Problem

The University Course Timetabling Problem involves the strategic scheduling of courses, students, and instructors within a limited set of resources. The objective is to generate a timetable that adheres to the constraints set by an educational institution or to maximize compliance with these constraints (Abdipoor et al., 2023).

The problem of scheduling timetables for universities isn't new, with papers published addressing computational solutions to this problem dating at least as far back as 1966 (Almond, 1966).

Among all variations of this problem, there is a universal constraint that dictates that specific pairs of events cannot be scheduled during the same timeslot, for example, because a student can't attend two lessons at once, nor can a professor teach more than one lesson at a time. This constraint is the basis for a comparison made between this problem and the NP-hard problem of graph coloring (Lewis and Thompson, 2015).

2.2.1 International Timetabling Competition (ITC) and Benchmark Datasets

Acknowledging the longstanding complexities inherent in the domain of timetabling, the International Timetabling Competition aims to draw participants from diverse research fields, fostering the development of innovative solutions. Its primary objectives include promoting multidisciplinary approaches to advance research and bridging the gap between theoretical advancements and practical applications in operational research related to timetabling problems (McCollum et al., 2010). The competition has seen five editions, with the most recent one occurring in 2021, with a focus on sports timetabling. Four of the five editions, have been focused on academic

timetabling, with three of them being specifically dedicated to university related timetabling, except for the third edition in 2011, which focused on high school timetabling[1].

The most recent edition focused on university timetabling occurred in 2019, and highlighted solutions to problem instances from real universities, as opposed to the 2007 edition which utilized datasets fabricated by a research group.

The benchmark datasets employed in the 2007 competition continue to be utilized even after 15 years to assess the efficacy of contemporary approaches (Akkan et al., 2022). These benchmark datasets assume a pivotal role in ascertaining the relative performance of various methodologies employed to address the challenges posed by educational timetabling.

2.2.2 Curriculum-Based and Post-Enrollment Course Timetabling

Within the University Course Timetabling Problem, the literature distinguishes between two main problems: Post-Enrollment Course Timetabling and Curriculum-Based Course Timetabling.

In Post-Enrollment Course Timetabling, conflicts primarily emerge from student enrollment data, emphasizing the need to accommodate individual student schedules and preferences. These conflicts may arise from overlapping course schedules, high demand for specific courses, or individualized constraints based on student preferences (Lewis et al., 2007).

On the other hand, Curriculum-Based Course Timetabling predominantly deals with conflicts arising from the published curriculum. Here, the focus is on scheduling courses based on predefined curriculum structures, irrespective of individual student preferences. Conflicts in Curriculum-Based Course Timetabling may include ensuring that all required courses are scheduled without conflicts, meeting specific class size requirements, and adhering to institutional rules governing course scheduling (Di Gaspero et al., 2007).

Essentially, Post-Enrollment Course Timetabling primarily emphasizes accommodating individual student preferences, whereas Curriculum-Based Course Timetabling focuses on adhering to the overarching curriculum framework. The issue under examination in this dissertation effectively pertains to a Curriculum-Based Course Timetabling problem.

2.2.3 Types of Constraints

A more straightforward definition of the University Course Timetabling Problem, in contrast to the previous one, involves the creation of timetables that adhere to a specific set of constraints. Existing literature on this problem categorizes constraints into two primary types (Chen et al., 2021):

- **Hard constraints:** these constraints, when met, render the generated timetable a feasible solution. An illustrative example of a hard constraint is that a student cannot attend two lectures simultaneously.

[1] More information available online, at the PATAT website at https://www.patatconference.org/communityService.html

- **Soft constraints:** these constraints contribute to defining the quality of a feasible solution. For instance, a soft constraint may stipulate that a class should have its lessons distributed throughout the week.

2.3 Timetabling at the Faculty of Engineering of the University of Porto

2.3.1 Current Approach

This dissertation addresses the optimization of pre-existing timetables, as opposed to the University Course Timetabling Problem, which focuses on creating timetables from scratch. Each semester, the Department of Informatics Engineering at the Faculty of Engineering of the University of Porto manually refines timetables generated by the current software. This software creates timetables for the entire faculty, with resources (such as rooms and sometimes professors) shared across different degrees and departments. The process begins by collecting preference information from professors responsible for each course and configuring the software to adjust various parameters, such as optimization preferences and priorities. This process takes several weeks and involves approximately 20 people from different timetabling committees as well as administrative staff within the faculty.

After the software generates a feasible solution for the whole faculty, the refinement process begins. Each department can then make alterations to their timetables, subject to the available resources (with room availability being the main restriction), and on a first-come, first-served basis. This prioritization makes time a critical factor. The first department to request timetable alterations will have all their requests fulfilled, while subsequent departments must adjust to the new availability conditions imposed by earlier changes. To ensure their optimization efforts are not compromised, the Department of Informatics aims to be the first to request changes. The need for a rapid solution for this manual and labor-intensive task of timetable optimization makes this problem an ideal candidate for automation.

The optimization process conducted by the Department of Informatics involves maximizing the adherence of timetables of the degrees under their purview (BSc and MSc in Informatics and Computing Engineering, MSc in Artificial Intelligence, PhD in Informatics Engineering) to a multitude of soft constraints. The current manual process involves manually retrieving the timetable information generated by the software, to create a spreadsheet to help with the optimization process. Then, for each modification made during the optimization process, the individual overseeing this task ensures that the alteration does not contravene any of the previously satisfied hard constraints. For instance, when relocating a lesson from one timeslot to another, a series of verifications is required, including:

- Confirming the availability of the professor originally assigned to the lesson at the newly selected timeslot. In the event that the professor is not available during the chosen timeslot,

substitution with another available professor is necessary, warranting an additional check for the new professor's availability.

- Ensuring that the room initially allocated for the lesson is unoccupied in the new timeslot, or that another room is available with the same type and at least the same capacity

If the proposed change remains unfeasible after these considerations, the process necessitates exploring alternative modifications.

This described process, which has been simplified in this overview, is notably time-consuming, given its iteration across numerous class timetables and faculty members' schedules.

2.3.2 Testing Dataset and Previous Automation Efforts

The optimization process within the Department of Informatics Engineering prompted initiatives to simplify it. Recently, a team of students developed software to serve as a frontend interface, streamlining manual adjustments. This software aims to enhance the efficiency of individuals involved in these manual modifications.

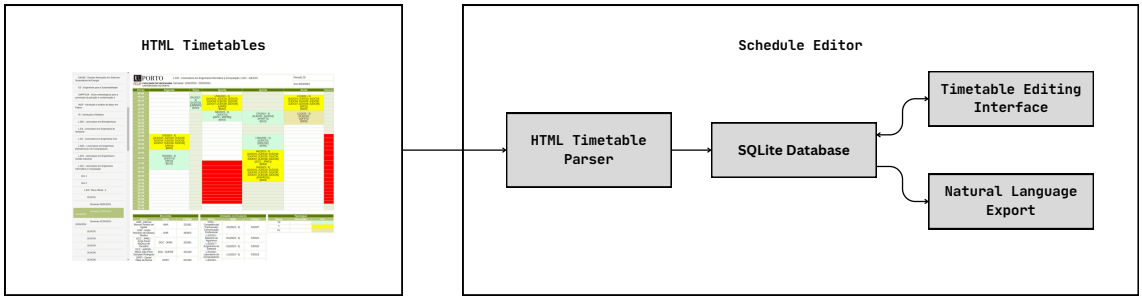

Figure 2.1: Previous Automation Efforts

The software created by the students operates as described in Figure 2.1. The scheduling software used by the faculty (Bullet Solutions, 2023) produces HTML-formatted timetables, which are parsed by the student's solution to collect data which is then stored in an SQLite database. A browser-based interface provides editing features to perform alterations (previously done in a spreadsheet). The software also verifies the validity of each alteration. The editing of timetables to perform manual optimizations is the process that this work aims to automate. Additionally, the software includes an export module that converts the performed changes into natural language, allowing them to be communicated to the faculty's administrative services for implementation.

This software has been refined by another group of students over the past semester, with efforts focused on addressing several key issues. These included resolving errors that occurred during the parsing phase, reducing the database size by eliminating redundancies, and fixing bugs in the editing interface. Additionally, they simplified the export process to remove redundant changes and streamline the output.

2.4 Genetic Algorithms

Popularized by Holland (1975), Genetic Algorithms fall into the category of evolutionary algorithms and are inspired by natural selection and the reproduction of the fittest individuals. These algorithms emulate the process of natural evolution, where populations of candidate solutions evolve towards better solutions over successive generations. The central concepts of genetic algorithms are derived from biological mechanisms, such as selection, crossover (recombination), and mutation, which contribute to the algorithm's ability to explore and exploit the solution space effectively.

A typical genetic algorithm functions as described in algorithm 1 - it starts with initializing a population 2.4.2, composed of a fixed number of individuals (also referred to as chromosomes - 2.4.1), each representing a potential solution to the problem at hand.

The algorithm runs for a fixed number of iterations, known as generations. In each iteration, the population is evaluated in respect to its fitness value, which quantifies how well an individual solves the problem (2.4.3). Based on their fitness values, a subset of the population is selected to form a mating pool. This selection process (2.4.4) often employs techniques such as roulette wheel selection, tournament selection, or rank-based selection, which probabilistically favor individuals with higher fitness values.

The selected parents undergo crossover operations (2.4.5), where pairs of chromosomes exchange segments of their genetic material to produce offspring for the next generation.

To maintain genetic diversity and avoid premature convergence to local optima, mutation operations (2.4.6) are applied, randomly altering the genetic information of individuals. The algorithm typically terminates when a maximum number of generations is reached, or when an individual with a fitness value meeting a predefined threshold is found, indicating an acceptable solution has been discovered (Alhijawi and Awajan, 2023).

Algorithm 1 Genetic Algorithm - GA ($population\_size, max\_generations, fitness\_function$)

```
// Initialize variables
generation ← 0
// Generate an initial population
population ← initializePopulation(population_size)
// Main loop
while generation < max_generations do
  // Evaluate the fitness of each individual
  evaluateFitness(population, fitness_function)
  // Select parents for next generation
  parents ← selection(population)
  // Create new population of offsprings using crossover
  population ← crossover(parents)
  // Introduce mutations
  mutate(population)
  // Increment generation counter
  generation ← generation + 1
end while
// Return best individual found
return bestIndividual(population)
```

2.4.1 Individual/Chromosome Representation

Drawing from genetic algorithms, a candidate solution is often termed a chromosome or individual. The representation of a chromosome depends primarily on the nature of the problem. Traditionally, as proposed by Holland (1975), encoding involves using an array of bits, known as binary representation. However, non-binary representations, are also viable alternatives (Whitley, 1994). The use of arrays with different types and structures follows the same core principles, as long as they are of a fixed size. This fixed size feature of genetic representations is advantageous because it simplifies the alignment of components, particularly facilitating crossover operations where genetic material from two parents is exchanged to create new offspring.

On a more granular level of the chromosome, there is the gene. A chromosome is composed by a set of genes. Within the context of the solution, a gene represents one or more configuration values of the solution (Eiben and Smith, 2015).

2.4.2 Population Initialization

A genetic algorithm uses its population to explore various areas of the search space, so, initializing the several individuals that compose its population properly has a strong impact on the performance of the algorithm. In their review of genetic algorithm theory, Alhijawi and Awajan (2023) highlight two main characteristics that impact performance:

- Population Diversity: low diversity leads a genetic algorithm to behave like a local search algorithm with the increased overhead of maintaining multiple similar solutions. A diverse

population allows the algorithm to explore different areas of the search space which can increase the odds of finding a global optimum (Sudholt, 2020);

- Population Size: the bigger the population, the more areas of the search space can be explored. A small population can lead to premature convergence after multiple crossover operations. On the other hand, a large population might result in redundancy at the expense of computational resources (Kazimipour et al., 2014).

Initialization criteria for generating initial populations can be categorized based on randomness, compositionality, and generality. The randomness criterion distinguishes between stochastic techniques, which produce varied populations using different seeds, and deterministic techniques, which create the same population regardless of the seed. The compositionality criterion separates non-compositional techniques, using a single-step process, from compositional techniques, which involve multiple steps. Lastly, the generality criterion differentiates between generic techniques, which are domain-independent, and specific application techniques, tailored for particular problems (Alhijawi and Awajan, 2023).

2.4.3 Fitness Function

Every genetic algorithm relies on a fitness function to evaluate how well each individual in the population meets its objectives. This function acts as a specialized type of objective function that condenses the assessment of a solution's alignment with its goals into a single measure/value.

Performance is evaluated using this objective function, which quantifies specific criteria relevant to the problem domain. Based on the outcomes of the objective function (or functions, in the case of multi-objective optimization), the fitness function determines the likelihood of each individual reproducing in subsequent generations (Whitley, 1994). This ensures that individuals demonstrating higher fitness levels, as assessed by the defined criteria, are prioritized for further evolutionary processes within the algorithm.

As highlighted by Eiben and Smith (2015), the goal in optimization challenges can often contradict the term "fitness function". While "fitness" implies a focus on increasing the quality of solutions, the objective is not always to maximize fitness; in many cases, such as the one presented in this work, it involves minimization.

2.4.4 Selection Operator

A pool of parents is formed for mating by the selection operator. This operator directs the genetic algorithm toward the best answer by favoring individuals with higher fitness over those with lower fitness. The selection operator can significantly impact population diversity. Several techniques for selection are widely used. Some of the most recognized include:

- Roulette-wheel Selection (Holland, 1975): this fitness proportionate method assigns selection probability based on individual fitness. Higher fitness individuals have a greater chance of selection, akin to segments on a roulette wheel;

- Tournament Selection (Brindle, 1980; Suh and Van Gucht, 1987): randomly chosen subsets of the population compete, and the fittest individual from each subset is selected. This method allows control over selection pressure through tournament size;
- Elitist Selection (De Jong, 1975): the best individuals are directly carried over to the next generation, ensuring that the highest-quality solutions are retained. This method can be combined with other selection techniques to enhance population quality.

Recognizing the role that the selection operator plays in maintaining (or reducing) population diversity, several selection operators have been proposed, namely gender selection.

A standard genetic algorithm (GA) typically does not differentiate between genders, allowing any two individuals to mate. However, natural reproduction involves gender-specific pairing, which has inspired the incorporation of gender in genetic algorithms to potentially improve performance by promoting diversity.

Gender selection (Tahera et al., 2008) refers to the process of assigning a gender to each individual (solution) of the population. This is done based on a gender probability factor that ranges from 0 to 1, controlling the number of males and females in the population.

More concretely, and to promote population diversity, AbouElhamayed et al. (2016) proposed a gender selection operator that involves dividing the population into two halves based on fitness, with higher fitness individuals termed as "females" and lower fitness individuals as "males". Candidates from each half are selected using tournament selection, where individuals compete based on their fitness scores within their respective groups. The fittest individual from each half is then chosen as a parent for offspring through crossover, aiming to maintain genetic diversity by mating high and low fitness individuals. This method helps to prevent premature convergence, enhance population diversity, and improve the algorithm's ability to explore and exploit solutions in complex optimization problems.

2.4.5 Crossover Operator

The crossover operator plays a pivotal role in genetic algorithms, facilitating the exchange of genetic material between parent individuals to create offspring with potentially improved fitness. This operator mimics biological recombination, where segments of genetic information from two parent solutions are combined to produce new solutions. Types of crossover operators include:

- Single-Point Crossover (Holland, 1975): a random point is selected along the length of the chromosomes of two parent individuals. Offspring are created by exchanging the genetic material beyond this point between the parents;
- Two-Point Crossover: involves selecting two random points along the length of the chromosomes of two parent individuals. Genetic material between these two points is exchanged to create offspring;

- K-point Crossover (De Jong, 1975): the one-point and two-point crossover techniques are specific instances within the broader K-point crossover framework. Therefore, the principles underlying one-point and two-point crossovers are generalized to K-point crossover, where cross points are chosen randomly;
- Uniform Crossover (Spears and De Jong, 1991): assigns equal probability to each gene (individual component of a chromosome) being inherited from either parent. Each gene position is independently selected for inheritance from either parent, resulting in offspring that may have more varied genetic makeup.

2.4.6 Mutation Operator

Crossover operators reduce population diversity by producing offspring with genes (individual component of a chromosome) identical to those of their parents. In contrast, the mutation operator sustains diversity by randomly altering genes in offspring. The key concept is to introduce random changes in offspring genes. In binary representations, a mutation might be as simple as flipping a bit of the bit string used to represent the solution.

The mutation operator is governed by a mutation probability, P_m, which is typically kept low to prevent the genetic algorithm from resembling a random search.

2.4.7 Replacement Techniques

Once the genetic algorithm generates new offspring, it proceeds to form the next generation population. In simple terms, this involves selecting the newly produced offspring along with parents that did not undergo crossover to populate the next generation. Various methods can be employed for this process:

- Delete-all (Generational) Replacement Technique (Holland, 1975): This method replaces all individuals from the current generation (parents) with the new offspring to form the next generation;
- Steady State Replacement Technique (Rechenberg, 1973; Holland, 1975): During each iteration, this technique replaces a small subset (n) of the population with n newly generated individuals. The selection of the n individuals to be replaced varies from approach to approach;
- Elitism Replacement Technique (Fogel, 1998; Tripathy, 1982): Here, the fittest individuals from both the parent population and the new offspring are selected to populate the next generation.

2.5 Simulated Annealing

Simulated annealing is an optimization algorithm, introduced by Kirkpatrick et al. (1983), inspired by the annealing process in metallurgy, where a material is heated and then slowly cooled to

decrease defects and improve structural integrity. The algorithm is designed to find a good approximation to the global optimum of a given function in a large search space. It operates by iteratively exploring neighboring solutions and accepting or rejecting them based on a temperature parameter that gradually decreases over time. At high temperatures, the algorithm is more likely to accept worse solutions to escape local optima, while at lower temperatures, it becomes more selective, converging towards a potentially optimal solution. This balance between exploration and exploitation makes simulated annealing a versatile tool for solving complex optimization problems.

The effectiveness of simulated annealing hinges on its cooling schedule, which dictates how the temperature decreases over iterations. Commonly used schedules include linear, exponential, and logarithmic cooling. The choice of the cooling schedule and other parameters, such as the initial temperature and the neighborhood structure, can significantly impact the algorithm's performance (Amine, 2019).

The basic functioning of Simulated Annealing is described in algorithm 2.

Algorithm 2 Simulated Annealing - SA (T_{init}, $max\_iterations$)

Initialization: Start with a random solution s
$T \leftarrow T_{init}$
$iteration \leftarrow 0$
while $iteration < max\_iterations$ **do**
  Perturbation: Slightly modify the current solution s to explore a nearby solution s'
  Evaluation: Evaluate the quality of the new solution s' compared to the current solution s
  if s' is better than s **then**
    Accept s' as the new current solution
  else
    Accept s' with a probability $P(e, e_{new}, T)$ {T is the temperature, e and e_{new} are the energy (evaluation) values of the current, s, and new solution, s', respectively}
  end if
  Decrease the temperature T according to a cooling schedule
  $iteration \leftarrow iteration + 1$
end while

return s

2.5.1 Neighboring Solutions

As mentioned earlier, the algorithm examines neighboring states of the problem to find improved solutions. These new states are generated by making adjustments to a given state. The specific method used to modify states and generate neighboring states is referred to as a move, with different move types producing a distinct set of neighboring states. These moves typically involve minimal changes to the current state, aiming to iteratively enhance the solution by improving its individual components over time (Amine, 2019).

Moves, and neighboring states are directly related to the representation of a solution, since, a move refers to a change in the configuration of solution. In their work, Amine (2019) consider neighboring solutions, solutions whose configurations "differ in few components only".

2.5.2 Acceptance Probability

The probability of moving from the current state s to a potential new state s_{new} is governed by an acceptance probability function $P(e, e_{new}, T)$. This function is influenced by the energies $e = E(s)$ and $e_{new} = E(s_{new})$ of the respective states, as well as the temperature parameter T, which changes over time. States with lower energy levels are more desirable than those with higher energy levels (i.e. have a better fitness value / evaluation than those with higher energy). To avoid the method from getting stuck in a local minimum and to ensure it can find the global minimum, the probability function P must remain positive even if e_{new} exceeds e.

As T approaches zero, $P(e, e_{new}, T)$ should approach zero if e_{new} is greater than e and should remain positive if e_{new} is less than or equal to e. When T is very low, the system increasingly prefers moves that reduce energy and avoids those that increase it. At $T = 0$, the process effectively turns into a greedy algorithm, only accepting transitions that lower the energy (Amine, 2019).

2.5.3 Cooling Schedule

The cooling schedule dictates how the temperature parameter T decreases over time. This schedule influences the algorithm's ability to explore the solution space and avoid local minima. Typically, the cooling schedule involves a gradual reduction of T according to a predefined scheme, such as exponential decay (Lundy and Mees, 1986), linear decay (Strenski and Kirkpatrick, 1991), geometric decay (Kirkpatrick et al., 1983), or logarithmic decay (Geman and Geman, 1984). An effective cooling schedule starts with a high initial temperature to explore the solution space broadly and accept worse solutions to escape local minima. As the temperature decreases, the algorithm becomes more selective, favoring moves that lead to lower energy states. The rate of temperature decrease must be carefully controlled: cooling too quickly may result in premature convergence to a suboptimal solution, while cooling too slowly may increase computational cost. Designing an optimal cooling schedule is essential for balancing exploration and exploitation to achieve a high-quality solution efficiently (Amine, 2019).

Chapter 3

Related Work

The search for literature addressing the particular issue of enhancing existing university timetables yielded no relevant results. Consequently, the research focus for identifying the most efficient approach to address the problem in this dissertation primarily centered on the existing body of literature related to the University Course Timetabling Problem (UCTP).

Given the extensive nature of the literature on this topic, emphasis was placed on reviews and surveys of approaches to the University Course Timetabling Problem, as opposed to papers describing individual approaches.

Ilyas and Iqbal (2015) reviewed distinct approaches for tackling the UCTP, including constructive heuristics, simulated annealing, tabu search, genetic algorithms, particle swarm optimization, artificial bee colony, and hybrid algorithms combining these methods. The paper provides an overview of the operational mechanisms behind each technique and summarizes the key results and performance characteristics presented in the reviewed literature. Hybridization of local search methods with population-based metaheuristics is highlighted as a popular strategy to leverage the strengths of different approaches - local search for faster exploration and population-based for avoiding local optima traps. The authors analyze the occurrences of different hard and soft constraints across studies, noting the high variability that makes finding generic solutions challenging. Concluding, they emphasize the importance of hybridization to tackle the NP-complete nature of UCTP, while extensive experimental evaluation on benchmarks is crucial to advance research in this complex domain.

In their review, Teoh et al. (2015) examined distinct approaches to academic scheduling problems, including tabu search, genetic algorithms, simulated annealing, particle swarm optimization, and others. The paper provides insights into the operational mechanisms of each method and summarizes the results presented in the reviewed literature. Concluding, the authors draw insights into the advantages and disadvantages of these approaches.

Bettinelli et al. (2015) analyzed algorithms designed specifically for the curriculum-based variant of the University Course Timetabling Problem, specifically algorithms developed to address problems introduced in the International Timetabling Competition of 2007. The authors initially

scrutinized mathematical models and exact algorithms, subsequently shifting their focus to heuristic and metaheuristic algorithms. A thorough description of these algorithms is provided, with a notable observation by the authors that metaheuristics generally exhibit superior performance compared to those based on mathematical models.

In their review, Pandey and Sharma (2016) examined distinct approaches proposed for the NP-hard university timetabling problem, including linear programming, constraint satisfaction methods, graph coloring techniques, and metaheuristic algorithms like genetic algorithms, ant colony optimization, particle swarm optimization, and artificial bee colony optimization. The paper provides an overview of the operational mechanisms behind each methodology and summarizes the key results and performance characteristics presented in the reviewed literature. Metaheuristic nature-inspired algorithms, especially those combining population-based approaches with local search heuristics, are highlighted for their ability to effectively explore large search spaces and find good approximate solutions. Concluding, the authors draw insights into the advantages of leveraging parallelism in these bio-inspired algorithms and emphasize the importance of extensive experimental evaluation on standardized benchmarks alongside theoretical analysis to advance research in this challenging domain.

Oude Vrielink et al. (2017) systematically reviewed the practices in timetabling within higher education institutions, focusing on the differences between academic research and commercial software used for scheduling. The study highlights the increasing complexity and constraints faced by higher education institutions, driven by limited resources and rising demands for flexibility and availability. It identifies that recent advances in timetabling algorithms, particularly metaheuristics like genetic algorithms, simulated annealing, and tabu search, show significant promise in improving solution quality. These techniques, along with hybrid approaches, excel due to their adaptability and efficiency in handling diverse and complex timetabling constraints. However, a gap remains between theoretical models and practical applications, necessitating more comprehensive and generalizable solutions for effective implementation in real-world scenarios.

Alghamdi et al. (2020) assessed multiple optimization algorithms for the University Course Timetabling Problem, although the exact number was not explicitly stated in the paper. The authors provide an overview of the advantages and disadvantages of 16 different approaches to the problem, delving into the operational mechanisms of genetic algorithms and particle swarm optimization. However, the paper lacks clear conclusions regarding the relative performance of the analyzed algorithms, making it challenging to determine the most effective methods among those examined by the authors.

The systematic mapping study by Bashab et al. (2020) provides a comprehensive review of metaheuristic algorithms applied to University Timetabling Problems in 131 studies from 2009 to early 2020, highlighting trends, current methods, and research gaps. The study reveals the dominance of hybrid algorithms in recent literature, with a significant focus on solution proposals. Emerging metaheuristic techniques such as grey wolf optimizer and cat swarm optimization show potential for future research. The study underscores the necessity for adaptive and multi-objective approaches and advocates for further systematic reviews to support ongoing advancements in this

field.

In their survey, Chen et al. (2021) aimed to discern perspectives, trends, challenges, and opportunities in the University Course Timetabling Problem, concurrently examining the advantages and disadvantages of existing methodologies. The survey encompassed 35 papers, revealing that the majority belonged to the category of metaheuristic approaches, with 18 approaches falling under this classification. The second-largest category comprised operational research (OR) approaches.

The survey by Ceschia et al. (2023) provides an extensive review of educational timetabling problems, emphasizing standard formulations and benchmark instances. Six key formulations, including course based university timetabling, are analyzed for their features, relevance, and usability. The paper highlights the importance of benchmark datasets for validating new approaches and ensuring reproducibility. It concludes that while solution quality varies across different techniques, metaheuristics and hybrid approaches often excel, particularly genetic algorithms, simulated annealing, and tabu search, due to their flexibility and robustness in handling complex constraints.

Abdipoor et al. (2023) identified several limitations in previous surveys of the University Course Timetabling Problem, particularly focusing on the area of metaheuristics. These limitations included a narrow focus on a single problem variant, inadequate reporting of solution quality, lack of analysis of hybrid methods, and shortcomings in critical analysis. The survey covered papers from 2015 to 2022 related to metaheuristic approaches to the University Course Timetabling Problem, reviewing a total of 45 papers. Notably, 28 of these papers were classified as hybrid approaches, underscoring the popularity of this sub-category within metaheuristics. This survey stands out for clearly comparing approaches, highlighting their performance on benchmark datasets.

3.1 Types of Approaches to the University Course Timetabling Problem and Main Findings

Surveys of methodologies addressing the University Course Timetabling Problem categorize these approaches differently, with some studies identifying up to six primary categories Chen et al. (2021); Alghamdi et al. (2020). Notably, Abdipoor et al. (2023) consolidate these various categorizations into five main groups: operational research based approaches, metaheuristics, hyperheuristics, multi-objective techniques, and hybrids.

Upon scrutinizing surveys and reviews of methodologies for addressing the University Course Timetabling Problem, it became evident that metaheuristic approaches (including hybrid approaches based on metaheuristics) stood out as the predominant choice among researchers in this field. This prominence is discernible in the substantial disparity in the number of approaches documented in the literature for each category. In a comprehensive review of metaheuristic techniques (Abdipoor et al., 2023), a search for approaches to the University Course Timetabling Problem from 2015 onwards revealed that among 108 approaches to the problem, 77 were categorized as metaheuristics or hybrids involving metaheuristics, while only 31 were encompassed within the other defined

categories. Another review underscored the significance of metaheuristics, emphasizing their efficacy in providing "accurate and optimal solutions to the timetable scheduling issue" (Alghamdi et al., 2020).

In addition to the limited representation of recent approaches falling into alternative categories such as operational research-based approaches, hyperheuristics, and multi-objective techniques, there were specific drawbacks emphasized in the literature for these other categories, particularly in addressing the specific problem considered in this dissertation:

- Operational research based approaches were characterized by one survey as “lacking in producing good quality solutions compared to other approaches” (Chen et al., 2021);
- Hyperheuristics are a relatively less explored field, with the term "hyperheuristics" initially introduced in 2000 to denote heuristics for selecting heuristics within the context of combinatorial optimization. More recently, this category has expanded to encompass not only the process of heuristic selection but also heuristic generation (Burke et al., 2013). Significantly, in the context of hyperheuristics, a review concentrated on educational timetabling characterized the body of research specifically dedicated to addressing the University Course Timetabling Problem as being limited in number (Pillay, 2016). Moreover, as pointed out by Oude Vrielink et al. (2017), the goal of hyperheuristics is to find a generally applicable methodology, that applies to a range of problem instances, as opposed to a methodology to solve a specific problem instance;
- Multi-objective or multi-criteria approaches are defined as strategies to address problems "where optimal decisions need to be taken in the presence of trade-offs between two or more objectives that may be in conflict" (Chang, 2015). Notably, one of the surveys analyzed underscored that when using this type of approach, "a large computational effort is often required in finding the Pareto front, even more so when it is desirable to have the solutions evenly spread along the Pareto fronts" (Chen et al., 2021). Several algorithms classified into other categories (metaheuristics, hyperheuristics, hybrids) can also be classified as multi-objective approaches;
- Hybrids, include all algorithms that combine multiple of the aforementioned categories. Within this category, matheuristics (Maniezzo et al., 2021) stands out for its positive performance in solving University Course Timetabling Problems. In the 2019 International Timetabling Competition, the first two winning results were based on matheuristics.

In Table 3.1, a summary of the advantages and disadvantages of different categories of approaches to the University Course Timetabling Problem are presented. It should be noted that some of these categories overlap, making it harder to establish a relative comparison. Notably, all problems which consider conflicting soft constraints in their optimization can be considered multi-objective approaches, as they are in fact optimizing for multiple objectives, and trying to find a solution close to the Pareto front, which contains feasible solutions that violate as little soft

constraints as possible. However, some works consider the problem of multi-objective specifically (Akkan et al., 2022; Gülcü and Akkan, 2020), so a category is defined for those approaches.

The rise in popularity of matheuristics as well as other types of hybrids, merit further exploration given their distinct performance in the ITC of 2019. However, the problem presented at the 2019 edition of the competition more resembles a post-enrollment version of the University Course Timetabling Problem, as opposed to the curriculum-based version, which is the type of problem faced at the Faculty of Engineering of the University of Porto. Moreover, a comparative analysis of the results of the competition was only published in April 2024 (Müller et al., 2024), at which point, the development of this work was well underway. Despite their positive performance on benchmark datasets, the review of literature on the topic is lacking, namely on the comparative strengths of these new mathematical-based approaches when compared to their counterparts.

A decision was made to delve more extensively into the metaheuristic literature to find a solution to the problem presented in this dissertation. This decision was driven not only by the challenges highlighted in other methodological categories but primarily by the substantial volume of research dedicated to metaheuristics and the positive performance, which is highlighted in every review paper analyzed, as well as the positive results of this type of methods in both the 2007 and 2019 International Timetabling Competition (McCollum et al., 2010; Müller et al., 2024). This abundance of literature may be interpreted as an indication of the scientific community's confidence in the promise of this category, and more importantly, the increased volume of literature facilitates the selection process, providing a panoply of analyses and comparisons of various approaches.

3.2 Single and Multi-Stage Approaches

Approaches to the University Course Timetabling Problem can also be categorized into two types based on the number of stages involved in addressing soft and hard constraints.

Specifically, one-stage approaches aim to generate a solution that simultaneously satisfies both soft and hard constraints in a single stage. On the other hand, two-stage or multi-stage approaches seek to generate a solution that adheres to hard constraints initially and then, in a separate stage (or stages), addresses the satisfaction of soft constraints (Abdipoor et al., 2023; Teoh et al., 2015).

Multi-stage approaches are particularly relevant to this dissertation, as the primary objective is to optimize existing feasible timetables to align with soft constraints. In other words, the focus is on performing only the second stage in a multi-stage approach. Within this category, simulated annealing is particularly popular as a second stage algorithm (Lewis and Thompson, 2015; Song et al., 2018; Goh et al., 2020; Gülcü and Akkan, 2020).

3.3 Metaheuristic Approaches

Metaheuristics are algorithms that combine "basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring a search space" Blum and Roli (2003).

Table 3.1: Summary of Strengths and Weaknesses of Categories of Approaches to the University Course Timetabling Problem

	Advantages / Strengths	**Disadvantages / Weaknesses**
Operational Research	Mathematical-based approaches have gained traction in recent approaches to the University Course Timetabling Problem (however, mostly hybrids, classified as matheuristics).	Purely operational research-based approaches have presented lacking results when compared to other types of approaches.
Hyperheuristics	Offer good results when a applied to several different problems and problem instances.	Offer versatility at the expense of performance when compared to approaches developed specifically to tackle a single problem instance. Implementation is typically more complex than other types of approaches.
Multi-objective/Multi-criteria	Focus on balancing multiple, often conflicting goals.	Require a high computational effort to find solutions that evenly spread along the Pareto fronts of the several objectives;
Hybrids	Recent approaches, namely in the context of timetabling competitions have proved successful. A shift in the literature favors hybrid approaches, namely in the field of hybrid metaheuristics. In the 2019 ITC matheuristics were amongst the best classified solutions.	The combination of several methodologies tends to increase the overall complexity of these methods, thereby raising the development burden.
Metaheuristics	Depending on the approach, allow for either a high explorative (evolutionary algorithms) or high exploitative (local search approaches) capacities. Some approaches, namely genetic algorithms, have become popular when tackling real-world problems for their versatility.	Some approaches can get stuck in local optimum values. Most algorithms are subject to parameter configuration, for which there isn't a single best configuration - this configuration can impact the results severely. Some approaches offer good results to several problem instances, at the expense of timeliness.

Despite the considerable body of research on metaheuristic methods and their applications in addressing the University Course Timetabling Problem, no singular solution emerges as dominant. This lack of a standout solution is primarily attributed to substantial performance variations, evident not only among different instances of the same problem but also when dealing with variations of the problem, such as those involving a slightly different constraint set.

In the realm of metaheuristic approaches, three primary types of algorithms can be discerned: single solution-based, commonly referred to as local search algorithms; population-based approaches; and hybrids, which combine the aforementioned two (Abdipoor et al., 2023).

3.3.1 Single Solution-Based Metaheuristics

Single solution approaches focus on modifying and improving a single candidate solution. This category of metaheuristics encompasses approaches such as simulated annealing and tabu search. Single solution-based methods are characterized by a heightened exploitative ability, enabling them to uncover high-quality solutions within a relatively shorter time frame (Abdipoor et al., 2023; Teoh et al., 2015).

In terms of particular methods, simulated annealing, introduced by Kirkpatrick et al. (1983), has been prominently featured in several review papers for its capacity to generate high-quality solutions. However, it is noteworthy that simulated annealing is highly sensitive to parameter settings, a characteristic that requires careful consideration and tuning for optimal performance (Abdipoor et al., 2023; Teoh et al., 2015).

3.3.2 Population-Based Metaheuristics

Population-based metaheuristics are algorithms that use a set of potential solutions, called a population, to explore the search space.

This category can be further subdivided into two primary subcategories: evolutionary algorithms and swarm intelligence. Evolutionary algorithms encompass widely utilized methods such as genetic algorithms, which are prominently featured in the literature. Swarm intelligence algorithms, on the other hand, include approaches such as particle swarm optimization or ant colony optimization (Abdipoor et al., 2023).

These methods exhibit superior exploratory capabilities and are more commonly featured in the literature, namely when tackling real-world problems (Abdipoor et al., 2023; Teoh et al., 2015).

Genetic algorithms have emerged as the most prevalent technique among evolutionary algorithms, primarily due to their exceptional flexibility when applied to various problem instances. However, it is noteworthy that they have not consistently achieved success in identifying feasible solutions under stringent time constraints (Abdipoor et al., 2023).

Genetic algorithms' popularity can be attributed to its versatility in solving various problem instances. Notably, Yousef et al. (2016) presented a hybrid genetic algorithm that integrates local search as a mutation operator. Additionally, the authors leverage GPU parallelization to accelerate computation. The integration of local search with genetic algorithms is not novel, with multiple

authors having explored this approach and demonstrating promising results, not only in the context of university timetabling (Abdullah et al., 2009; Kohshori and Abadeh, 2012) but also in other timetabling problems (Goel and Deb, 2002; Rahoual and Saad, 2006).

Multiple reviews and surveys analyzed highlighted the emergence of new population-based metaheuristics, like the bat algorithm and grey wolf optimizer, with Abdipoor et al. (2023) suggesting further research into the effectiveness of these newly proposed algorithms when applied to the University Course Timetabling Problem. However, multiple authors (Camacho-Villalón et al., 2018; Piotrowski et al., 2014; Weyland, 2010) have written about the growing amount of metaphors within the realm of metaheuristic search algorithms, which conceal the fact that a lot of the newly proposed algorithms are just versions of already existing ones (sometimes identical ones Weyland (2015)), with a new metaphor to mask the similarities shared between them. Moreover, in their paper, Camacho Villalón et al. (2020) dismantle both the bat and grey wolf optimizer algorithms as a "reiteration of ideas introduced first for particle swarm optimization".

3.3.3 Hybrid Approaches

According to the results in benchmark datasets, hybridization of approaches emerges as the most effective strategy in the literature. Integrating different methods has been shown to enhance performance by mitigating the weaknesses of each and leveraging their respective strengths. Hybrids can be broadly categorized into two main types: Integrative, where a technique is an embedded component within another, and Collaborative, where information exchange occurs between the algorithms contributing to the solution—whether sequentially, intertwined, or in parallel—but without being intrinsic parts of each other (Abdipoor et al., 2023).

An example of an integrative approach, with good results in benchmark datasets is the approach presented by Yousef et al. (2016), which uses local search as a mutation operator in a genetic algorithm.

Within the realm of hybrid approaches, collaborative approaches are of particular relevance to this work, as they typically start by constructing a feasible solution, and follow that with a second algorithm that improves upon that solution.

Several collaborative approaches have also produced great results in benchmark datasets. In their work, Song et al. (2018) start by constructing a feasible solution using a greedy heuristic, and then use a simulated annealing-based iterated local search to enhance that solution. Initially, simulated annealing is applied. The resulting solution then undergoes an improvement perturbation. If this perturbation yields a better result than the current solution, it is accepted.

Similarly, Song et al. (2021) used a greedy algorithm to find a feasible solution and then introduced a new method for combining multiple neighborhoods, where only one neighborhood is selected at each iteration to balance between a large search space and high computational efficiency. Finally, it includes a competition-based restart strategy, where two SA-based multi-neighborhood local search procedures are implemented during the search process. The elite solution from the two procedures is then selected as the initial solution for the next round of search.

3.4 Summary of Findings

The analysis of existing research on the University Course Timetabling Problem (UCTP) reveals several trends and conclusions regarding the most effective methodologies.

Metaheuristics, particularly hybrid metaheuristics, have emerged as the predominant solution strategy for UCTP over the past few years. Their flexibility, robustness, and ability to handle complex constraints effectively make them a preferred choice among researchers. Combining local search methods with population-based metaheuristics is a popular strategy, as it allows for fast exploration of solutions while avoiding the traps of local optima. This hybridization is particularly effective, as evidenced by the positive performance of metaheuristics in competitions and benchmark tests.

Single solution-based metaheuristics, such as simulated annealing and tabu search, are valued for their exploitative capabilities, which enable them to improve solutions within a relatively short time frame. Simulated annealing, despite its effectiveness, is highly sensitive to parameter settings, requiring careful tuning for optimal performance. Population-based metaheuristics, like genetic algorithms, are the most prevalent among evolutionary algorithms due to their versatility across various problem instances. These algorithms are often combined with local search methods to enhance their performance. Emerging algorithms, such as the bat algorithm and grey wolf optimizer, show potential, but are considered by some researchers as a mere rebrand of existing methods.

While metaheuristics dominate the field, other approaches are also noteworthy. Operational research-based methods, when used in their pure form, generally underperform compared to metaheuristics. However, their hybrid forms, such as matheuristics, have achieved notable success in recent competitions. Hyperheuristics aim for general applicability across various problem instances, offering good results for multiple problems, though they may lack the performance of solutions tailored to specific problems. Multi-objective or multi-criteria approaches address problems with conflicting objectives but require significant computational effort to find solutions that are evenly spread along the Pareto fronts.

Among the benchmarked approaches, metaheuristics stand out for their exceptional performance on the curriculum-based version of the UCTP, which closely aligns with the scheduling specifics of the Faculty of Engineering at the University of Porto. Within the realm of metaheuristics, collaborative approaches are particularly promising for addressing the problem at hand. These approaches typically involve a multi-stage process: initially finding a feasible solution and subsequently focusing on satisfying the soft constraints. This second stage aligns with the optimization problem being addressed in this work - rather than generating new timetables, the task is to optimize existing ones to meet as many soft constraints as possible - thus, the algorithms employed in the second stage of the collaborative approaches are especially promising for solving the problem at hand.

Chapter 4

Problem and Proposed Solution

4.1 Problem Description

As previously articulated, the principal objective of this dissertation is to optimize pre-existing timetables that already comply with hard constraints. This optimization involves enhancing the adherence to soft constraints while ensuring the resultant timetables maintain their feasible status. In other words, the newly generated timetables must align with as many soft constraints as possible as well as the totality of hard constraints. The problem formalization is presented below.

Given:

- A set of rooms $R = \{r_1, r_2, r_3, ..., r_{|R|}\}$, and for each room:
  - A set of fixed properties:
    - Type;
    - Capacity;
    - Number of computers;
    - Blocked timeslots (times at which the room is being used for other purposes other than lessons, like exams).
- A set of classes $C = \{c_1, c_2, c_3, ..., c_{|C|}\}$, and for each class:
  - A set of fixed properties:
    - Degree;
    - Curricular year;
- A set of professors $P = \{p_1, p_2, p_3, ..., p_{|P|}\}$, and for each professor:
  - A set of fixed properties:
    - For each course Cs, and for each type of lesson Tp (practical, theoretical), the lesson time that professor should teach, $L_{time}[Cs][Tp] \in \mathbb{R}$. New solutions created

from crossover often result in some professors teaching more lessons than they did in the initial feasible solution. To ensure that the teaching time for each professor in each course they teach remains consistent, this information is necessary.

  - Blocked timeslots (some professors, like invited professors, aren't available at certain times during the week).

- A set of lessons, $L = \{l_1, l_2, l_3, ..., l_{|L|}\}$, and for each lesson:
  - A set of fixed properties:
    - Begin week ($start_{week}$) and end week (end_{week}), number (most lessons have the same values, representing the whole semester, however, some lessons function as modules and therefore are only taught for some weeks of the semester);
    - The course the lesson belongs to (e.g. Linear Algebra, Physics, ...);
    - The class(es) assigned to the lesson, $\{c_{n_1}, c_{n_2}, ...\} \in C$;
    - The type and capacity of the room(s) it needs to be taught in (e.g. room with at least 20 computers, an auditorium with 200 seats, ...).
  - A set of properties to be (re)assigned:
    - Rooms (typically just one, but in some cases, more than one), $\{r_{n_1}, r_{n_2}, ...\} \in R$;
    - Professors (typically just one, but in some cases, more than one), $\{p_{n_1}, p_{n_2}, ...\} \in P$;
    - Timeslots (expressed in a range defined by the first and last timeslot of the class), $\{start, end\} \in [0, 180[$ - thirty timeslots of 30 minutes, starting at 8:00, for each day ranging from Monday to Saturday.

Find: for each lesson in L, a new assignment of rooms in R, professors in P, and time slots in [0,180[. The assignments should ensure no hard constraints are violated and minimize the number of soft constraints that are violated.

4.2 Problem Specific Constraints

In the curriculum-based version of the university course timetabling problem, typical hard constraints include:

- A professor cannot teach two lessons simultaneously
- A class cannot have overlapping lessons in its schedule
- No two lessons can be assigned to the same room at the same time

Soft constraints, meanwhile, offer more variety and often cater to problem-specific considerations.

The scheduling problem at the Faculty of Engineering is particularly constrained, with its real-world complexities becoming evident in various aspects. One notable challenge is that some lessons occur for less than a semester, lasting only a few weeks. This adds an extra layer of complexity when considering time-related occupancy constraints, as these shorter lessons must be accounted for in the overall scheduling process. Most of the curriculum-based versions of the university timetabling problems discussed in the literature do not incorporate this additional temporal dimension.

Another aspect that is often disregarded in the literature is considering that classes and professors must have a lunch break in their schedule.

The following list provides a breakdown of the constraints influencing the generation or optimization of timetables within the Department of Informatics at FEUP.

Hard Constraints:

H1 Professors can't have conflicting lessons (two lessons happening at the same time in their timetable):

$$\begin{aligned}&\forall p \in P,\ \forall L_u, L_v \in Lessons(p),\ (L_u \neq L_v) \implies \\ &(end_{L_u} \leq start_{L_v}) \vee (end_{L_v} \leq start_{L_u}) \vee \\ &(start_{week_{L_u}} > end_{week_{L_v}}) \vee (start_{week_{L_v}} > end_{week_{L_u}})\end{aligned}$$

H2 Rooms can't have conflicting lessons (two lessons happening at the same time in their timetable):

$$\begin{aligned}&\forall r \in R,\ \forall L_u, L_v \in Lessons(r),\ (L_u \neq L_v) \implies \\ &(end_{L_u} \leq start_{L_v}) \vee (end_{L_v} \leq start_{L_u}) \vee \\ &(start_{week_{L_u}} > end_{week_{L_v}}) \vee (start_{week_{L_v}} > end_{week_{L_u}})\end{aligned}$$

H3 Classes can't have conflicting lessons (two lessons happening at the same time in their timetable):

$$\begin{aligned}&\forall c \in C,\ \forall L_u, L_v \in Lessons(c),\ (L_u \neq L_v) \implies \\ &(end_{L_u} \leq start_{L_v}) \vee (end_{L_v} \leq start_{L_u}) \vee \\ &(start_{week_{L_u}} > end_{week_{L_v}}) \vee (start_{week_{L_v}} > end_{week_{L_u}})\end{aligned}$$

H4 Rooms assigned to lessons must be of the required type and size:

$$\begin{aligned}&\forall L_u \in L,\ \#Rooms(L_u) = \#Requirements(L_u),\ \forall i \in \{1,2,\ldots,\#Rooms(L_u)\}, \\ &(t_i, s_i) \in Requirements(L_u), r_i \in Rooms(L_u) \implies type(r_i) = t_i \wedge size(r_i) \geq s_i\end{aligned}$$

H5 Blocked times in schedules must be respected:

$$\begin{aligned}&\forall p \in P,\ \forall L \in Lessons(p),\ \forall b \in B_p,\ (end(L) \leq start(b)) \vee (start(L) \geq end(b)) \\ &\forall r \in R,\ \forall L \in Lessons(r),\ \forall b \in B_r,\ (end(L) \leq start(b)) \vee (start(L) \geq end(b)) \\ &\forall c \in C,\ \forall L \in Lessons(c),\ \forall b \in B_c,\ (end(L) \leq start(b)) \vee (start(L) \geq end(b))\end{aligned}$$

H6 Professors and classes must have a free lunch period (a two consecutive timeslots between 11:30 and 14:30):

$$\forall p \in P,\ \exists t \in T_{11:30 \text{ to } 14:30-1},\ \neg(lessons(p,t) \vee lessons(p,t+1))$$
$$\forall c \in C,\ \exists t \in T_{11:30 \text{ to } 14:30-1},\ \neg(lessons(c,t) \vee lessons(c,t+1))$$

H7 Lessons of certain specified courses, $cs \in Sc$, must be taught at a specified periods, $Sp(cs)$:

$$\forall cs \in Sc,\ \forall L \in Lessons(cs),\ timeslots(L) \in Sp(cs)$$

H8 For each course,cs, and type of lesson, tp, ensure that the total time of lessons assigned to a professor, p, $Time(p,cs,tp)$, match the required time they should teach, of that course and lesson type, as specified in $L_{time_p}[cs][tp]$:

$$\forall p \in P,\ \forall cs \in Courses(p),\ \forall tp \in Types(cs), (Time(p,cs,tp) = L_{time_p}[cs][tp])$$

Soft Constraints:

S1 Distribute course's practical lessons belonging to the same shift throughout the week:

$$\forall cs \in Courses,\ \forall s \in Shifts(cs),\ \forall d \in TeachingDays,$$
$$\left| \#PracticalLessons(d,s) - \frac{\#PracticalLessons(s)}{\#TeachingDays} \right| < 1$$

S2 Avoid unoccupied timeslots in between classes' lessons in the morning or afternoon (lunch breaks are excluded - up to 3 hours between 11:30 and 14:30):

S2.1 Avoid unoccupied timeslots in between lessons in the morning:

$$\forall c \in C,\ \forall L_u, L_v \in Lessons(c),$$
$$L_u \neq L_v \wedge weekDay(L_u) = weekDay(L_v) \wedge end_{L_u} \leq 11:30 \wedge start_{L_v} \leq 11:30 \implies$$
$$end_{L_u} = start_{L_v} \vee \exists L_w \in Lessons(c),\ (end_{L_u} \leq start_{L_w} < start_{L_v})$$

S2.2 Avoid unoccupied timeslots in between lessons in the afternoon:

$$\forall c \in C,\ \forall L_u, L_v \in Lessons(c),$$
$$L_u \neq L_v \wedge weekDay(L_u) = weekDay(L_v) \wedge end_{L_u} \geq 14:30 \wedge start_{L_v} \geq 14:30 \implies$$
$$end_{L_u} = start_{L_v} \vee \exists L_w \in Lessons(c),\ (end_{L_u} \leq start_{L_w} < start_{L_v})$$

S2.3 Allow for unoccupied timeslots up to three hours, for lunch breaks, between 11:30 and

14:30:

$$\forall c \in C,\ \forall L_u, L_v \in Lessons(c),$$
$$L_u \neq L_v \wedge weekDay(L_u) = weekDay(L_v) \wedge end_{L_u} \leq 11:30 \wedge start_{L_v} \geq 14:30 \implies$$
$$start_{L_v} - end_{L_u} \leq 3 \text{ hours} \vee \exists L_w \in Lessons(c),\ (end_{L_u} \leq start_{L_w} < start_{L_v})$$

S3 Avoid having a period (a period refers to: morning, afternoon and night, for each day of the week) in classes' timetables with only lectures (as opposed to a mix of lectures and practical lessons):

$$\forall c \in C,\ \forall d \in Days,\ \forall p \in Periods,\ (\#Lectures(c,d,p) > 0) \implies (\#Practicals(c,d,p) > 0)$$

S4 Harmonize timetables between classes that have the same curriculum (classes with the same curriculum are classes from the same degree and curricular year), so that they are as similar as possible in terms of timeslot (t) occupancy, tolerating differences up to a certain threshold value

$$\forall c_i, c_j \in C \text{ where Curriculum}(c_i) = \text{Curriculum}(c_j),\ \forall d \in \text{Days},$$
$$\sum_t |\text{Occupied}(c_i,d,t) - \text{Occupied}(c_j,d,t)| \leq \text{Threshold},$$

with:

$$\text{Occupied}(c,d,t) = \begin{cases} 1, & \text{if occupied,} \\ 0, & \text{otherwise} \end{cases}$$

S5 Minimize the number of days with lessons in Professors' schedules, considering that a teacher cannot teach more than 10 hours per day (10 hours being equivalent to 20 timeslots of 30 minutes each):

Minimize:

$$|\#DaysUsed(p) - \left\lceil \frac{\sum_{L \in Lessons(p)} \text{duration}(L)}{20} \right\rceil|,\ \forall p \in P$$

subject to:

$$duration(L) = end_L - start_L + 1$$
$$\#DaysUsed(p) = \sum_{d \in Days} \text{HasLessonsScheduled}(p,d)$$

S6 Room continuity (minimizing professor movement, ensuring consecutive lessons are held in the same rooms):

$$\forall p \in P,\ \forall L_u, L_v \in L_p,\ end_{L_u} = start_{L_v} \implies Rooms(L_u) = Rooms(L_v)$$

S7 Lessons of certain specified degrees, $Dg \in Sd$, and curricular year, $Cy \in CurricularYears(Dg)$

must be taught at a specified periods, $Sp(Dg, Cy)$:

$$\forall Dg \in Sd, \forall Cy \in CurricularYears(Dg),\ \forall L \in Lessons(Dg, Cy),\ timeslots(L) \in Sp(Dg, Cy)$$

4.3 Proposed Solution

Although most of the review papers analyzed don't highlight any specific algorithm that stands out from others, both genetic algorithms and simulated annealing are notable for their recurring appearance in timetabling competitions and their good results in tackling benchmark datasets. The review papers emphasize the intricate nature of each problem, which leads to different algorithms performing variably across different problem instances. However, some conclusions about the algorithms' exploitation and exploration abilities can be inferred.

Firstly, simulated annealing presents good results in shorter time periods due to its exploitative capacity. On the other hand, genetic algorithms can produce better results over longer periods because of their greater exploratory capacity. Additionally, the surveys and reviews indicate that hybrid algorithms tend to perform better overall. A deeper look into some of these collaborative approaches reveals that simulated annealing is often used in the second stage. This is particularly relevant in the context of this problem since, similarly to a collaborative method where the first stage focuses on finding a feasible solution (without hard constraint violations), and the second stage optimizes this solution (minimizing the number of violated soft constraints), in this approach, the starting point is an already feasible solution and the aim is to minimize the number of soft constraints while maintaining feasibility.

Considering this, a choice was made to initially develop two algorithms: a genetic algorithm, due to the high number of constraints which could require a more exploratory approach, and simulated annealing, which has proven successful in integrative approaches where the starting point is an already feasible solution, as is the case in this situation.

Additionally, a third algorithm was developed, which conducts a greedy local search, by repurposing the mutation/neighborhood operators already defined for the two previous approaches. This algorithm was developed to act as a baseline for comparison with the two other approaches.

4.3.1 Integration with Existing Automation Efforts

As previously discussed in 2.3.2, some automation efforts have already been made to facilitate the optimization of timetables by the Department of Informatics. Figure 4.1 illustrates how the automation of optimization efforts will be integrated with the existing software developed to support these efforts.

The previously developed software will parse the timetables generated for the entire faculty by the Bullet Solutions (2023) software and store this data in a database.

Following this process, the staff of the Department of Informatics manually adjusts the teaching staff assignments in the generated timetables (initially, many lessons are assigned to virtual

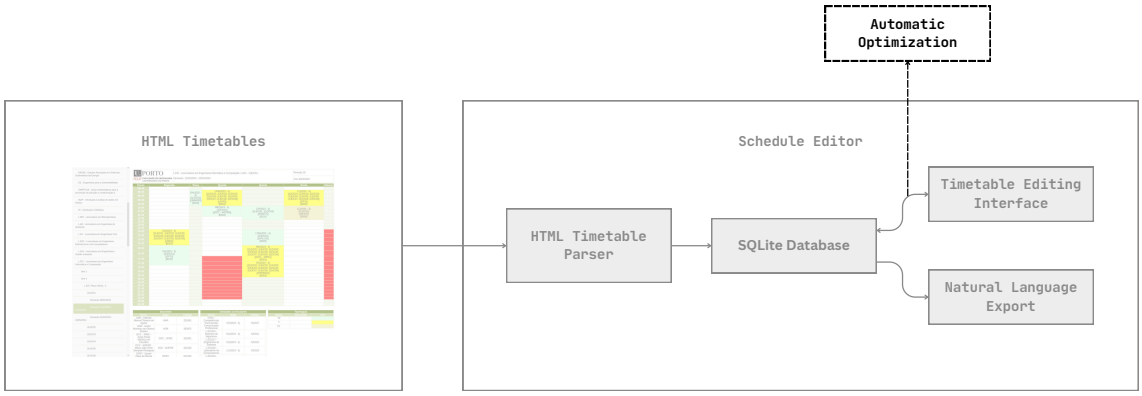

Figure 4.1: Integration with Existing Automation Efforts

professors to allow scheduling even when the actual professor is not yet determined) using the interface in the software that allows for manual editing.

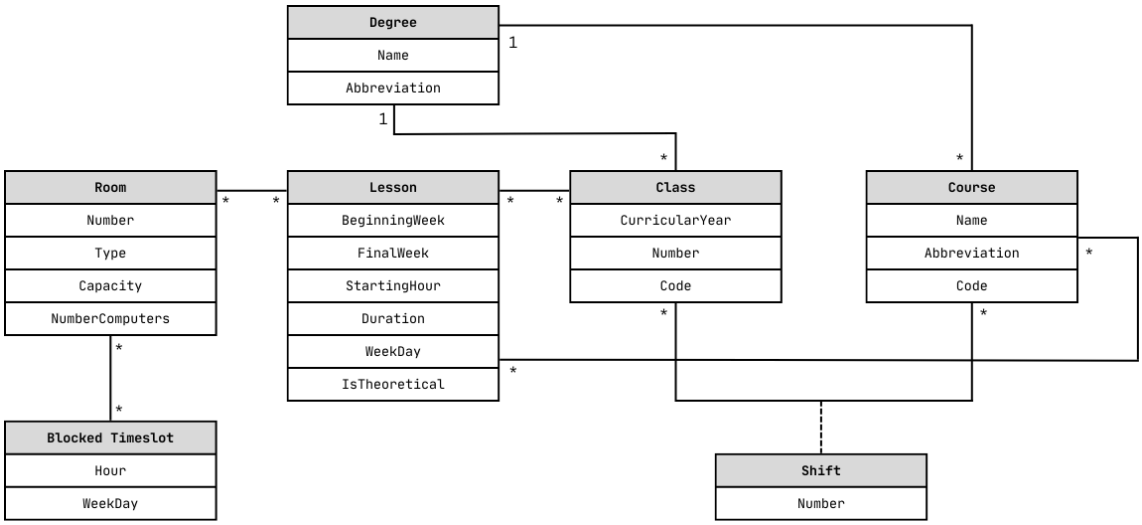

Figure 4.2: Database Schema

The optimization work developed in this dissertation will then utilize the database to optimize the timetables (limiting the optimization to timetables of degrees under the purview of the Department of Informatics) and output an optimized solution in the same format. This process involves converting the information in the database, which has its schema described in Figure 4.2, into a solution representation more adequate to the optimization process (later described in section 5.1).

Subsequently, the existing software will employ its natural language export feature to communicate the changes to the administrative services of the faculty, facilitating the execution of these changes.

4.3.2 Adaptability to Other Problem Instances

The developed solution is highly adaptable to the timetabling problems at other institutions. This adaptability is largely due to the modular design of the fitness function and the flexibility in defining constraint enforcement. To tailor the solution to the specific needs and constraints of different institutions, it is simply a matter of adding or removing constraint enforcement within the fitness function. This allows for easy customization to address unique requirements and priorities of various academic environments.

Furthermore, the solution's effectiveness can be enhanced by adapting the mutation and neighborhood operators used in the genetic algorithm and simulated annealing. By incorporating problem-specific operators, the search process becomes more efficient, improving the chances of finding optimal solutions that align with the institution's goals. This adaptability ensures that the developed algorithms can be fine-tuned to the particular characteristics of each problem instance, making them versatile tools for a wide range of university course timetabling problems.

4.3.3 Technologies Used

For the first approach, a genetic algorithm, a choice was made to develop the algorithm and corresponding operators from scratch, rather than using existing libraries. This decision was based on the simplicity of implementation. The most labor-intensive task is not in the genetic algorithm's implementation, but in evaluating each individual's fitness. This evaluation is problem-specific and would be necessary regardless of whether a library was used. Moreover, after investigating several popular genetic framework options (Keijzer et al., 2002; Gagné and Parizeau, 2002; Mohammadi et al., 2017; Gad, 2023), none included the gender selection operator as described in AbouElhamayed et al. (2016). To expedite execution, OpenMP (OpenMP Architecture Review Board, 2021) was employed to parallelize both the mutation and fitness calculation.

For the second approach, which is based on Simulated Annealing and uses multiple neighborhoods of solutions, as well as for the third approach, a greedy local search which also uses multiple neighborhoods, OpenMP was used to parallelize the search for a feasible solution in each neighborhood.

All algorithms were implemented in C++, mainly for it's direct compatibility with the OpenMP API (OpenMP Architecture Review Board, 2021).

Chapter 5

Genetic Algorithm

The algorithm developed employs the fundamental principles of genetic algorithms to tackle the task of optimizing the existing schedule. The following sections offer a detailed explanation of each step in the process. A notable feature of this algorithm is its utilization of CPU parallelization to enhance computational efficiency. Additionally, it incorporates local search as mutation operators and dynamically adjusts probabilities for the selection of mutation operators, in an attempt to enhance the algorithm's effectiveness in exploring and exploiting the search space.

5.1 Chromosome/Individual Representation

For the problem being tackled, a choice was made to represent a solution as a set of genes - events, or more precisely, lessons. In turn, each lesson contains the following information:

- Information that can be changed in the optimization process:
  - Room(s), the room(s) allocated for that lesson;
  - Professor(s), the professor(s) allocated for that lesson;
  - Timeslots (the first and last, marking the beginning and end of the lesson).
- Static information that is used to identify constraint violations and compute the fitness value:
  - Course(s), the course(s) the lesson belongs to;
  - Classe(s), the classes(s) the lesson belongs to (which in turn provides us with information regarding the degree, curricular year and shift);
  - Lecture or Practical, a flag that marks whether or not the lesson is theoretical or practical;
  - Begin and End Weeks;
  - Required Room Type, indicates which type of room the lesson should be taught in (regular room, auditorium, computer room);

– Required Room Capacity, indicates how many people the room should be able to holds;

– Required Room Capacity, in case the room is a computer room, it indicates how many computers it should have.

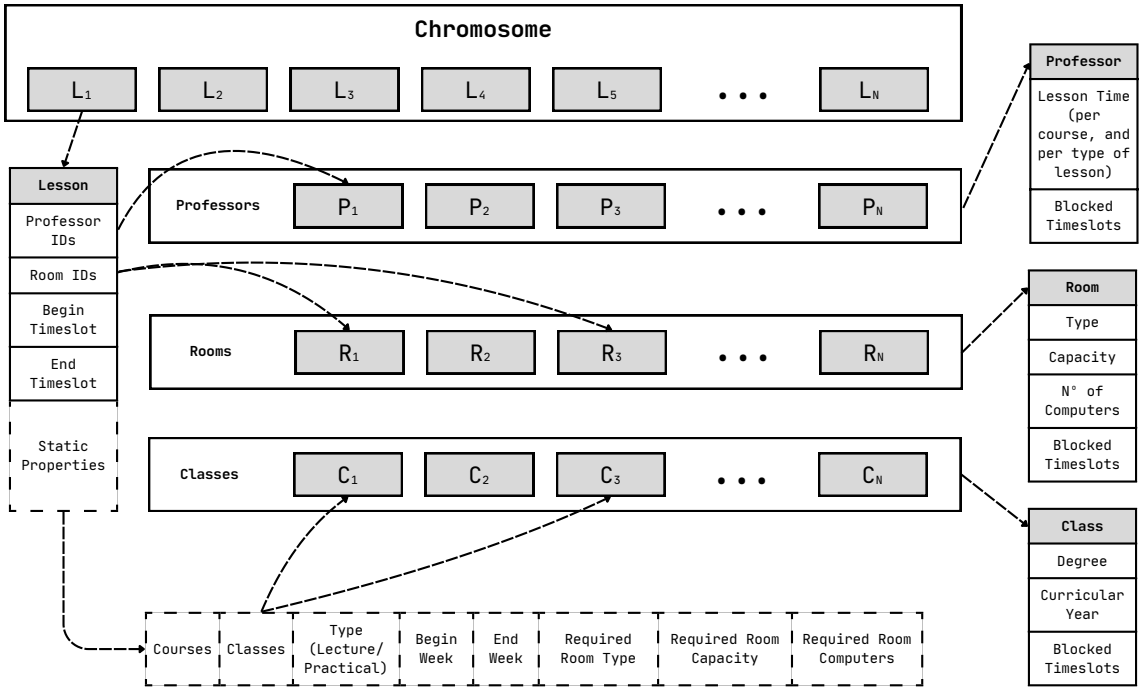

Figure 5.1: Individual Representation

With the information contained within the set of lessons (the chromosome - illustrated in Figure 5.1), it's possible to map out timetables for the rooms, professors, and classes, which in turn have static information of their own, as previously described in section 4.1. The chromosome includes two main types of information: mutable information, which changes during the generation of new solutions, and static information, which is used to verify the quality of new solutions and check for constraint violations, and remains the same throughout the optimization process.

The mutable information includes the assigned rooms' IDs (which can be mapped out to Room objects that hold additional details), the assigned professors' IDs (which can be mapped out to Professor objects, that also hold more information), and timeslots assigned to that lesson. In contrast, the lesson object also contains various pieces of static information necessary for evaluating the solution and identifying constraint violations. For example, to ensure that the room assigned to a lesson meets the required type, capacity, and number of computers, there must be a match between the details in the assigned room's object and the static information in the lesson object regarding room requirements.

5.2 Population and Initialization

The population is composed of a fixed number of individuals. There are several initialization techniques, with different categorizations based on their characteristics, as presented in section 2.4.2. In the context of the academic scheduling, genetic algorithm approaches often opt for a random initialization technique to create the initial population (Moreira, 2008; Deng et al., 2011; Abdelhalim and El Khayat, 2016; Febrita and Mahmudy, 2017).

In the context of this project, the starting point is a pre-existing solution that satisfies all hard constraints, so the initialization process differed slightly from traditional approaches.

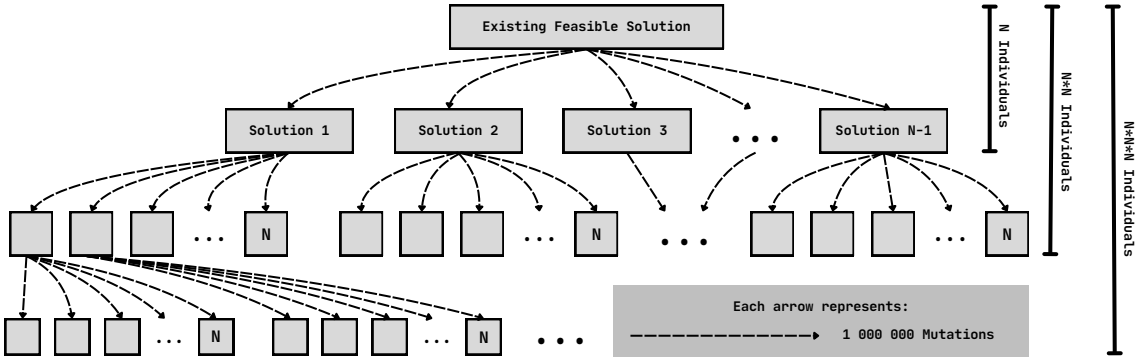

Figure 5.2: Initialization Technique

Instead of developing an algorithm to randomly create new solutions, the mutation operators, presented in section 5.6, were used to alter the existing solution in order to create new ones (allowing for constraints to be violated during the mutation process, and solutions with worst fitness to be created). The first individual of the initial population was the already existing solution. As Figure 5.2 illustrates, the remaining individuals were generated by applying 1 million mutations to the already existing solution. The mutation process was done iteratively: the first N-1 individuals generated were derived from the pre-existing individual, then for each one of the N-1 new individuals (derived from the first), N new individuals were generated by applying mutations (resulting in (N-1)*N new individuals, for a total of 1+(N-1)+(N-1)*N = N*N individuals), and for each one of those , another N were generated. As an example, for N=10, in the first iteration 9 new individuals are created, in the second iteration 9*10=90 new individuals are created, and in the last iteration 90*10=900 individuals are created, totaling 1+9+90+900=1000 individuals.

The iterative initialization process is hoped to enhance diversity in the population by introducing multiple layers of variation. Rather than generating all individuals directly from the pre-existing solution, the method involves sequential mutation steps where each round of newly created individuals undergoes further mutations. This cascading effect is expected to allow for a broader exploration of the solution space, as each iteration propagates unique changes, potentially reducing the likelihood of converging prematurely to a local optimum. By iteratively mutating individuals generated in previous rounds, the process aims to capture a wider array of genetic

variations, fostering a more diverse population compared to a straightforward approach where all individuals are derived solely from the initial solution.

Population diversity is extremely important in genetic algorithms, since as pointed out by Alhijawi and Awajan (2023), "a population with a low diversity level leads to a GA like a local search algorithm". In populations with low diversity the effectiveness of crossover, which relies on diverse parents to generate novel offspring, is diminished. Additionally, mutations, while introducing some variation, have a reduced impact on a population with similar genetic makeup, limiting their ability to explore entirely new areas of the search space. Consequently, the GA exhibits local search-like behavior, iteratively refining solutions within a restricted region instead of exploring the broader landscape for potentially superior global optima.

5.3 Solution Evaluation/Fitness Function

The fitness function is responsible for evaluating the fitness value of an individual. In this context, since the goal is to reduce violations of constraints, the optimization goal is to minimize the fitness value. Unlike other genetic operations, which are generally problem-independent, the fitness function is problem-dependent. In the context of this problem the fitness function, denoted as f, for a given timetabling solution T, can be defined as:

$$f(T) = \lambda_H \cdot N_H + \sum_{i=0}^{|Types\ of\ S|} \lambda_{S_i} \cdot N_{S_i} \tag{5.1}$$

where N_H refers to the number of hard constraints violated, N_{S_i} to the number of soft constraints of type i violated, and λ_H and λ_{S_i} are weights denoting the relative importance of the hard constraints, H, and soft constraints, S. Notably, different types of soft constraints S_i hold a varying degree of importance, reflected in the individual weights assigned to them. In contrast, all hard constraints share the same penalty, as a violation of any single hard constraint renders the solution infeasible and unacceptable.

$$\lambda_H > \sum_{i=0}^{|Types\ of\ S|} \lambda_{S_i} \cdot N_{S_i} \tag{5.2}$$

The stringent nature of hard constraints necessitates zero tolerance for violations. Consequently, the scenario where a single hard constraint is violated, even while satisfying multiple soft constraints, is considered more severe than the situation where all soft constraints are violated while no hard constraints are breached. Therefore, as illustrated in equation 5.2, the cumulative penalty incurred from all soft constraint violations must be less than the penalty associated with any single hard constraint violation, ensuring a prioritization of hard constraint adherence. Determining the actual penalty value for a hard constraint is not an easy task, mainly because determining the maximum number of violated soft constraints that a given solution can have is also not simple. As was previously described in subsection 4.2, there are several soft constraints. The maximum number of possible soft constraints violations will depend on several factors, namely

the quantity of lessons, which varies depending on the dataset being used. Therefore, one way to assign a value to the hard constraint penalty, is to evaluate the initial solution, and determine how many soft constraints are being violated, and use that value as a reference to calculate the value of the hard constraint penalty.

In light of the optimization challenge commencing with an initially feasible solution, it is imperative that the fitness value of the final solution T' always remains below the threshold set by λ_H. To put it succinctly, equation 5.3 highlights a fundamental criterion for the final solution - strict adherence to hard constraints, as any violation of a hard constraint would render the obtained solution inferior to the initial one.

$$f(T') < \lambda_H \tag{5.3}$$

5.4 Parent Selection

To determine parents for crossover, a gender selection process was implemented, drawing inspiration from AbouElhamayed et al. (2016). This process involves sorting the population based on fitness value and then segregating individuals by gender as follows:

- Females are individuals with the best fitness value (i.e., the lowest value)
- Males are individuals with the worst fitness values (i.e., the highest values)

From each gender pool, a parent is chosen using tournament selection.

According to AbouElhamayed et al. (2016), this selection process "can delay the population convergence and increase the diversity of the population", which as was previously highlighted, can lead to better results, and reduce the probability of getting stuck on a local optima.

5.5 Crossover, Replacement and Elitism

As pointed out in the study of crossover operators conducted by Singh and Gupta (2022), and previously evidenced (albeit on a more general level) by the No Free Lunch theorem (Wolpert and Macready, 1997), it's not possible to affirm one crossover operator as better than another, especially when averaged across a multitude of problem applications. As such, a focus was placed on operators utilized by the literature on the curriculum-based course timetabling problem.

In their work, which obtained positive results on benchmark datasets, Yousef et al. (2016) used single-point crossover. Similarly, in their genetic algorithm approach, Abdullah et al. (2009) also used this method with very positive results on benchmark datasets. Unlike the previous two, Kohshori and Abadeh (2012) utilized uniform crossover, however their work isn't applied to benchmark datasets, making it hard to effectively compare to the previously presented approaches. In their genetic algorithm approach, also tested on benchmark datasets, AbouElhamayed et al.

(2016) tested several crossover operators (but failed to specify which in their work), and concluded that single and double-point crossover produced the best results.

Based on the prevalence of this operator in the literature, specifically in approaches with good benchmark results, a single-point crossover method was employed. This process involves generating two new individuals by taking the Lessons array of each parent, splitting both arrays at the same randomly selected index. Lessons to the right of that point are then swapped between the two parent chromosomes. As a result, two offspring are produced, each inheriting genetic information from both parents.

Crossover was executed at a predetermined probability rate, which also dictated whether the parents would be replaced by their offspring in the new generation. If crossover occurred, both parents were substituted with their children; otherwise, the parents would proceed to the next generation without change.

In order to preserve the best solutions in subsequent generations regardless of their participation in the crossover process, an elitism mechanism was implemented. When active, this operator ensures that the two best solutions in the current generation get passed on to the next generation directly.

5.6 Mutation

For each individual in a generation, a mutation was applied with a fixed probability. A mutation involves altering a gene of the chromosome, or in this context, modifying one lesson within the solution.

If a chromosome is chosen for mutation, a specific mutation operator is then selected. Each mutation operator determines the type of mutation that will be applied to the gene. In this case, eight distinct mutation operators were defined, tailored to the specificities of the problem and informed by empirical observations of which types of changes typically yield improved results during manual optimization. The changes introduced by each operator on a lesson are detailed below:

- M1 Room Change: This operator identifies an equivalent room (having the same requirements in terms of type, capacity, and number of computers), which is free at the time of the lesson being mutated, and assigns the lesson to that room;
- M2 Timeslot Change: Modifies the times lots of the lesson to a different timeslots;
- M3 Swap Professors: This operator locates another lesson of the same course and type (lecture or practical) and exchanges professors between the two lessons;
- M4 Swap Class Lessons: Finds another lesson assigned to the same class and exchanges timeslots between the two lessons;
- M5 Swap Rooms: This operator identifies another lesson occurring at the same time with identical room requirements and swaps the rooms between the two lessons;
- M6 Double Swap Professors: Similar to "Swap Professors," but involves two consecutive swaps. Specifically, it takes three lessons of the same course and type - L1, L2, and L3 - and swaps

the professor assigned to L1 with the one assigned to L2, followed by swapping the professor assigned to L2 with the one assigned to L3. This results in the professor originally assigned to L1 moving to L3, the professor of L2 moving to L1, and the professor of L3 moving to L2;

M7 Double Swap Class Lessons: Similar to "Swap Class Lessons", but performs two consecutive swaps. Specifically, it takes three lessons of the same class - L1, L2, and L3 - and swaps the timeslots assigned to L1 with the ones assigned to L2, followed by swapping the timeslots assigned to L2 with the ones assigned to L3. This results in the timeslots originally assigned to L1 moving to L3, the timeslots of L2 moving to L1, and the timeslots of L3 moving to L2;

M8 Double Swap Rooms: Similar to "Swap Rooms", but performs two consecutive swaps. Specifically, it takes three lessons occurring at the same time with identical room requirements - L1, L2, and L3 - and swaps the rooms assigned to L1 with the ones assigned to L2, followed by swapping the rooms assigned to L2 with the ones assigned to L3. This results in the rooms originally assigned to L1 moving to L3, the rooms of L2 moving to L1, and the rooms of L3 moving to L2.

5.7 Mutation Operator Selection

Recognizing that the different mutation operators introduced in section 5.6 might have a different impact on the overall improvement of the solution, two techniques for selecting which operator is employed were developed. The choice of technique is controlled in the initial configuration of the genetic algorithm by boolean variable.

The first technique consists of selecting the operator at random. In this case, considering there are 8 operators, when a mutation occurs, there is a 12.5% chance of any operator being chosen.

The second technique records the positive impact that each operator has had in reducing each of the violated constraints, and then uses that information to dynamically calculate a probability of that operator being chosen, based on the recorded impact information, and on the specific solution it is being applied on, by taking in to account how many violations of each constraint that solution has.

$$T_m = \sum_{i=0}^{Current\ Generation} Number\ of\ Mutations_{m,i} \tag{5.4}$$

In the initial generations, the choice of mutation operator is randomly selected from a set of operators M. However, once each operator has been applied at least 1000 times, (the value of T_m, $\forall m \in M$ expressed in 5.4, is bigger than 1000), a more informed selection process is implemented. This involves utilizing previously stored data on the impact each operator has on reducing individual constraint violations.

The value of 1000 was chosen to ensure that each operator has been applied enough times to provide a statistically reliable dataset for evaluating its effectiveness. The hope is that this

threshold value balances the need for adequate data collection, allowing the algorithm to transition to a more adaptive and informed mutation selection process in a timely manner.

$$I_{m,k} = \sum_{i=0}^{Current\ Generation} max(0, VB_{m,k,i} - VA_{m,k,i}) \tag{5.5}$$

After each mutation, the change in the number of hard and soft constraint violations is recorded to inform future operator choices. The impact of a mutation operator $m \in M$ on a given constraint $k \in K$, is defined by $I_{m,k}$ in equation 5.5, where $VB_{m,k,i}$ represents the number of violations of constraint of type k before applying the mutation operator m in generation i. Similarly, $VA_{m,k,i}$ represents the number of violations of constraint of k after applying the mutation operator m in generation i.

$$P_m = \sum_{k \in K} V_k * \frac{I_{m,k}}{T_m} \tag{5.6}$$

$$\forall m \in M,\ P_{Normalized_m} = \frac{P_m}{\sum_{m \in M} P_m} \tag{5.7}$$

The value of the impact is then used to compute P_m of an operator m (5.6), where V_k represents the violations of constraint k in the chromosome being mutated. The value of P_m is then normalized (5.7) to obtain $P_{Normalized_m}$, which is the probability used to define which mutation operator is selected.

To exemplify, if the Room Change operator consistently reduces violations of Hard Constraint 5, and if the current solution being mutated exhibits a higher prevalence of violations for that same hard constraint, the probability of selecting the Room Change operator is increased. The goal with this adaptive approach is to optimize the mutation process by prioritizing operators that have historically proven effective in addressing specific constraint violations present in the current solution.

Additionally, a parameter was included to determine whether the mutation allow for a resulting solution with a worse fitness value. When set to only permit better solutions (i.e., not allowing worse fitness values), the mutation operators function as a local search algorithm, seeking to improve the solution rather than making random changes to the individual's chromosome.

5.8 Parameters

The genetic algorithm developed includes several genetic operators, which can be configured to be applied or not, or, in some cases, to be applied at varying rates. The list of parameters included in the developed algorithm is as follows:

- **Population Initialization Value, N**: a value which determines the size of the population ($Population\ Size = N^3$). It is used to perform initialization as described in 5.2;
- **Maximum Number of Generations**: the number of iterations the algorithm will run for;

- **Crossover Rate**: the probability of crossover occurring between pairs of individuals;
- **Mutation Rate**: the probability of mutation occurring in an individual;
- **Elitism**: determines if elitism is applied or not;
- **Allow Mutations to Generate Solution with Worse Fitness**: this parameter determines whether mutations that result in an individual with a worse fitness value are allowed. If not allowed, multiple mutations options are tested until the mutated solution results in a solution with a better or equal fitness value (in some cases, if all mutation possibilities with the selected mutation operator are exhausted, meaning no better solutions using that operator were found, then, no mutation is performed);
- **Varying Operator Selection Probability**: determines whether the varying probability described in section 5.7 is applied or not. If not, each operator has an equal probability of being selected.

Most of these parameters significantly impact the performance of the algorithm. For this reason, several tests were conducted to determine the best parameter values to use when comparing this algorithm with other algorithms. Although the literature reveals that there isn't an out-of-the-box set of parameters that work best universally (Czarn et al., 2004), adjustments to certain parameters can produce better results. In particular, Maskaoui et al. (2017) highlight population size, crossover, and mutation as being amongst the most impactful parameters. Additionally, in their work, Alhijawi and Awajan (2023) remark that a higher mutation rate preserves diversity, which might lead to a better outcome at the expense of timeliness. The goal of these tests is to find a parameter set that will produce the best results without sacrificing too much computational time. The results of the parameter testing, as well as the results for the overall algorithm are presented in chapter 8.

Chapter 6

Simulated Annealing

The algorithm presented in this chapter uses simulated annealing to search for new solutions within the search space. The search is conducted using several neighborhood operators. At each iteration, all neighborhoods are searched in parallel, and the best solution is selected as the candidate solution. This candidate solution is then accepted based on the probability of acceptance at that iteration, which varies depending on the fitness of the solution and the current value of the temperature.

6.1 Solution Representation

The structure adopted to represent a timetabling solution to the problem was the same utilized to represent a chromosome/individual in the genetic algorithm approach (previously described in section 5.1). To summarize, each solution is defined by an array of lessons, each lesson containing an allocation of a start and end timeslot, and one or more professors and rooms.

6.2 Initialization

In traditional simulated annealing approaches it is customary to start from a randomly initialized solution (Kirkpatrick et al., 1983). On the other hand, in hybrid algorithms that use simulated annealing, particularly in collaborative approaches, there are two main stages: the first stage employs an algorithm to find a feasible solution (such as graph coloring heuristics or greedy heuristics), and the second stage uses simulated annealing to improve upon this feasible solution.

In this particular case, the starting point is already a feasible solution, eliminating the need for a first stage focused on finding one.

6.3 Perturbation and Multi-neighborhood Search

The perturbation phase as the name indicates, refers to introducing a perturbation to the current solution to find a new candidate solution. In this case, in order to find a new candidate solution,

multiple neighborhoods are explored. Neighborhoods contain candidate solutions obtained by introducing a perturbation to the previously accepted solution. All candidate solutions are feasible, meaning they might violate soft constraints, but never hard constraints.

The perturbations are introduced by mutating the current solution. For this purpose, the mutation operators defined for the genetic algorithm (in section 5.6) were repurposed, with each operator representing a neighborhood. A neighboring solution is generated by selecting a random lesson and modifying either its timeslot, assigned room, or professor. In the previously presented genetic algorithm approach, the acceptance of mutations varied based on the selected configuration: in one of the configurations, only mutations that improved fitness values were permitted; in the other, mutations with any fitness value were allowed, including those that resulted in unfeasible hard constraints. By contrast, this algorithm considers solutions with worse fitness values, provided they do not violate hard constraints. The fitness value can deteriorate due to soft constraint violations, but hard constraints must remain satisfied.

The process of finding neighboring solutions in each one of the neighborhoods as well as evaluating them is conducted in parallel.

6.4 Solution Evaluation/Fitness Function

In order to find and evaluate neighboring solutions, it is necessary to evaluate the solution after the perturbation is performed. For this, the fitness function utilized for the genetic algorithm, described in section 5.3, is repurposed.

6.5 Acceptance and Cooling Schedule

Once a candidate solution is selected from multiple neighborhoods, a decision process commences. If the candidate solution has a fitness value, $f(S_{Candidate})$, below the current solution's fitness value, $f(S_{Current})$, (meaning the solution is better), it is accepted. However, if the value is higher (the solution is worse than the current one), it might still be accepted. The decision of whether or not to accept a worse solution is given by a probability, described in equation 6.1, calculated using a temperature parameter, and the variation in fitness value between the current solution and the worse candidate.

$$P_{acceptance} = \exp\left(-\frac{\delta E}{T}\right) = \exp\left(\frac{f(S_{Current}) - f(S_{Candidate})}{T}\right) \tag{6.1}$$

With every iteration of the algorithm, cooling occurs, meaning, the temperature value diminishes, and therefore, so does the probability of a worse candidate solution being accepted. The cooling rate defines how fast the cooling occurs; similarly to temperature, it is also a parameter that needs to be set at the beginning of the algorithm.

The cooling schedule employed follows a geometric decay, as shown in equation 6.2. This schedule adjusts the temperature's value in each iteration, k, at a rate determined by the cooling rate, Cr.

$$T_k = T_0 \cdot Cr^k \tag{6.2}$$

Chapter 7

Greedy Local Search

This algorithm was developed to take advantage of the existing local search/mutation operators defined in section 5.6, and also to serve as baseline comparison to the previously presented algorithms.

Algorithm 3 Greedy Local Search Algorithm $(InitialSolution, MaxIterations)$

Initialize S' with $InitialSolution$
Initialize list of candidate changes $Changes$ to empty
$CurrentIteration \leftarrow 0$
while $CurrentIteration < MaxIterations$ **do**
  for each lesson L in S' **do**
    for each mutation operator M in Section 5.6 **do**
      $S'_{new} \leftarrow$ Apply M to L in S'
      if Fitness(S'_{new}) < Fitness(S') **then**
        Record change $(L, M, \text{Fitness}(S'_{new}) - \text{Fitness}(S'))$ in $Changes$
      end if
    end for
  end for
  Sort $Changes$ in descending order based on impact on fitness value (Δ)
  for each change (L, M, Δ) in $Changes$ **do**
    $S'_{new} \leftarrow$ Apply M to L in S'
    if $Fitness(S') > Fitness(S'_{new})$ **then**
      $S' \leftarrow S'_{new}$
    end if
  end for
  $CurrentIteration \leftarrow CurrentIteration + 1$
end while
return S'

The operators presented in section 5.6 apply changes to a solution on a lesson level. In the approach described in algorithm 3, each of these operators is applied iteratively to search for neighboring solutions that yield better fitness values. When a better solution is found, the necessary modifications are recorded along with their impact on the fitness value. These modifications

are then applied sequentially, starting from the most impactful to the least. If applying a modification at any point results in a worse fitness value, it is skipped. This can occur because while modifications were individually tested for feasibility, their combination may lead to conflicts that violate constraints, thereby impacting fitness negatively.

Chapter 8

Results and Discussion

In this chapter, the results of experiments conducted on a computational setup featuring an Intel Xeon Gold 6258R CPU @ 2.70GHz with 112 threads and 128GB of RAM are presented and discussed. To evaluate the performance of optimization algorithms, including genetic algorithms and simulated annealing, various configurations tailored to each approach were tested. Given the stochastic nature of genetic algorithms and simulated annealing, each configuration underwent five repeated tests to capture variability and ensure robust evaluation. In contrast, due to its deterministic nature, the greedy local search algorithm underwent a single test. The following analysis examines and interprets the outcomes obtained across these experiments.

8.1 Test Datasets

As has been previously stated, the goal of this work is to provide a solution that optimizes an existing feasible solution. For the purposes of testing the algorithms developed, tests were conducted on two datasets: one referring to a semester that had already been manually optimized by the Department of Informatics Engineering, and another referring to a different semester in which optimization hadn't been performed yet.

The test datasets include data for every lesson across all degrees in the Faculty of Engineering. Since this work focuses on the needs and constraints of the Department of Informatics, a filtering process was necessary to exclude lessons from other departments. Excluding these lessons also meant limiting the resources allocated to them. To achieve this, the resources allocated to non-target lessons were converted into blocked timeslots in each resource's timetable.

For instance, a professor teaching lessons to both degrees under optimization, and those not included, will have the non-target lessons marked as blocked timeslots in their timetable to prevent conflicts during the optimization process. The same procedure applies to rooms, where lessons from other degrees are marked as blocked timeslots in the room's timetable. Additionally, some lessons in the datasets are shared by two degrees, with one falling under the Department of Informatics. These shared lessons were also excluded from the optimization. After this processing,

the datasets to be optimized contained a total of 311 lessons (for the optimized dataset) and 336 lessons (in the non-optimized dataset).

The initial solution used for testing, an already optimized dataset, has a fitness value of 101.601, calculated using penalties described in Table 8.5. Tables 8.1 and 8.2 provide the number of violations for each of the hard and soft constraints, respectively. This dataset will be referred to from now on as Dataset 1.

It is noteworthy that despite references to starting with a feasible solution, the initial solution does have a violation of Hard Constraint 2. This violation stems from an agreement between two departments to share a room, during the manual optimization process which led to the schedule that is being used in this testing process. The decision was made to keep the testing dataset unchanged, including this violation. The rationale is that a well-functioning optimization algorithm should be capable of resolving such hard constraint violations on its own.

The second solution, a non-optimized dataset, has a fitness value of 1.7835, calculated using penalties described in Table 8.5. Tables 8.3 and 8.4 provide the number of violations for each of the hard and soft constraints, respectively. This dataset will be referred to from now on as Dataset 2.

Table 8.1: Dataset 1 - Number of Hard Constraint Violations

H1	**H2**	**H3**	**H4**	**H5**	**H6**	**H7**	**H8**
0	1	0	0	0	0	0	0

Table 8.2: Dataset 1 - Number of Soft Constraint Violations

S1	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**
120	11	18	434	95	0	72

Table 8.3: Dataset 2 - Number of Hard Constraint Violations

H1	**H2**	**H3**	**H4**	**H5**	**H6**	**H7**	**H8**
0	0	0	0	0	0	0	0

8.2 Solution Evaluation/Fitness Function - Constraint Violation Penalty Values

In section 5.3, the fitness function was discussed, highlighting the necessity for specific penalty values for hard and soft constraints. The penalty values need to be carefully considered to ensure that the cumulative value of penalties for soft constraint violations do not surpass the penalty for a single hard constraint violation, as shown in equation 5.2.

Table 8.5 presents the penalty values for hard and soft constraints. A single value is attributed for all hard constraints, since just one violation makes the timetable unfeasible. Soft constraints

Table 8.4: Dataset 2 - Number of Soft Constraint Violations

S1	S2	S3	S4	S5	S6	S7
163	135	39	278	251	7	46

have each its own individual penalty, according to the relative importance attributed to it by the timetabling responsible person at the Department of Informatics.

The sum of all soft constraint penalties in both datasets (1.601 for Dataset 1 and 1.7835 for Dataset 2), is significantly lower than the penalty value assigned to a single hard constraint, which is 100. This ensures that the optimization process prioritizes resolving hard constraint violations over soft constraint violations, maintaining the feasibility of the timetable.

8.3 Genetic Algorithm

As previously described in section 5.8, there are several parameters to configure, that can have a large impact on genetic algorithms. For this reason, several tests were conducted to determine the configuration that produced the best results. The best resulting configuration is the one that will be later compared to the remaining algorithms.

All tests, except those varying the population size, utilized the exact same population of 216 individuals throughout. This method ensured consistency by creating the population once and reusing it across all experiments. The initial population size selection was based on an important factor during the testing process: time. Using a higher population size for the same number of generations would translate to longer execution times. When multiplied by five runs for each configuration, this would result in a testing time-frame that would make testing as many configurations inviable.

Almost all the tests to determine the best configuration were ran on Dataset 1, with exception of the tests which vary population size, which were tested on both Dataset 1 and Dataset 2.

8.3.1 Mutation Rate

Firstly, mutation rates were tested. To evaluate this parameter, the remaining parameters were fixed as follows:

- **Population Size (Population Initialization Value, N)**: 216 (6);
- **Maximum Number of Generations**: 1250;
- **Crossover Rate**: 1.0;
- **Elitism**: Not applied;
- **Allow Mutations to Generate Solution with Worse Fitness**: Don't allow;
- **Varying Operator Selection Probability**: Not applied;

Table 8.5: Penalty Values for Hard and Soft Constraints

Constraint	Penalty Value
H1 - H8	100
S1	0.002
S2	0.002
S3	0.005
S4	0.001
S5	0.001
S6	0.0005
S7	0.01

The crossover rate was selected to be 1.0 because anything below that would introduce variability in the application of crossover during each run. By ensuring crossover is applied consistently in every one of the five runs, it is easier to analyze the impact of varying the mutation rate. The remaining parameters - elitism, allowing for mutation to generate solutions with worse fitness, and varying operator selection probability - are binary configurations that will be tested separately and were not applied in this configuration.

By fixing these parameters, the impact of varying mutation rates on the algorithm's performance could be isolated and assessed more accurately. Six values of mutation rates were tested, with each test comprising five runs to exclude outlier results derived from the random factors in the algorithm. Results are presented in Table 8.6.

Table 8.6: Mutation Rate (Dataset 1)

Mutation Rate	Average Fitness Value (of the 5 runs)	Standard Deviation Fitness Value (of the 5 runs)	Best Fitness Value (out of the 5 runs)	Average Execution Time (min)
0	861.604	1699.42	101.601	7
0.1	1202.2913	1919.47	1.57	90
0.3	3522.6992	7168.79	1.576	98
0.5	11503.5505	25719.2047	1.56	97
0.7	201.8516	346.63169	1.573	105
0.9	1.5737	0.002864	1.569	114

The results for varying mutation rates were inconsistent across all configurations, except for the configuration with a 0.9 mutation rate. Most mutation values produced solutions that reduced the starting fitness value of 101.601, but in some runs, they also resulted in worse values, as reflected in the average fitness value of each configuration. This inconsistency could be attributed to the lack of elitist pressure; without it, crossover operations can generate suboptimal solutions.

8.3.2 Elitism

Following the results of the previous subsection 8.3.1, the next test was conducted under the same conditions, but this time with elitism active. Results are presented in Table 8.7.

Table 8.7: Mutation Rate with Elitism (Dataset 1)

Mutation Rate	**Average Fitness Value (of the 5 runs)**	**Standard Deviation Fitness Value (of the 5 runs)**	**Best Fitness Value (out of the 5 runs)**	**Average Execution Time (min)**
0.1	1.5883	0.00327	1.5845	91
0.5	1.578	0.00569	1.5695	103
0.9	1.5673	0.003899	1.562	113

With elitism applied, the values become more consistent, with a clear trend becoming visible: higher mutation results in both better results, with lower fitness values, and also higher run times.

8.3.3 Varying Operator Selection Probability

The next experiment conducted, was to test the varying mutation selection probabilities described in section 5.7. To test this configuration, the remaining parameters were fixed as follows:

- **Population Size**: 216;
- **Maximum Number of Generations**: 1250;
- **Crossover Rate**: 1.0;
- **Mutation Rate**: 0.9;
- **Elitism**: Applied;
- **Allow Mutations to Generate Solution with Worse Fitness**: Don't allow;
- **Varying Operator Selection Probability**: Applied;

The mutation rate used was the one that presented the best results in the previous tests. The results of the tests using the varying probability for mutation operator selection are presented in Table 8.8:

The results in Table 8.8 indicate a negative outcome, contrary to expectations. Both the average fitness value across the five runs and the best fitness value from all the runs underperform compared to the scenario where each operator is selected with equal probability.

Table 8.8: Varying Mutation Operator Selection Probability (Dataset 1)

Varying Mutation Operator Selection Probability	**Average Fitness Value (of the 5 runs)**	**Standard Deviation Fitness Value (of the 5 runs)**	**Best Fitness Value (out of the 5 runs)**	**Average Execution Time (min)**
Applied	1.5834	0.004749	1.5785	97.6
Not Applied	1.5673	0.003899	1.562	113

8.3.4 Allow Mutations to Generate Solution with Worse Fitness

The next parameter variation tested was allowing mutations to create solutions that result in a worst fitness value. To test this configuration, the remaining parameters were fixed as follows:

- **Population Size**: 216;
- **Maximum Number of Generations**: 1250;
- **Crossover Rate**: 1.0;
- **Mutation Rate**: 0.9;
- **Elitism**: Applied;
- **Allow for Mutation to Generate Solution with Worse Fitness**: Allow;
- **Varying Operator Selection Probability**: Not applied;

The outcomes, presented in Table 8.9 were the worst obtained so far. In genetic algorithms, mutation acts as a preserver of diversity. When mutation operators are permitted to create solutions that violate more constraints, resulting in worse fitness values, and when the mutation rate is set as high as 0.9, excessive diversity can be introduced. This can prevent convergence towards a better solution, even with elitist pressure, which might explain the underwhelming results obtained with this configuration.

Table 8.9: Allow Mutations to Generate Solution with Worse Fitness (Dataset 1)

Mutation Rate	**Average Fitness Value (of the 5 runs)**	**Standard Deviation Fitness Value (of the 5 runs)**	**Best Fitness Value (out of the 5 runs)**	**Average Execution Time (min)**
0.9	21.5998	44.7215	1.597	14

The reduced execution time can be attributed to the fact that, with this type of mutation, the first mutation applied is immediately accepted. In contrast, the previous approach acted as a local search, requiring a solution with better or equal fitness, which requires testing multiple mutation options until one is found, which significantly increases the time spent on each mutation.

8.3.5 Crossover Rate

The next parameter tested was crossover rate. To evaluate this parameter, the remaining parameters were fixed as follows:

- **Population Size**: 216;
- **Maximum Number of Generations**: 1250;
- **Mutation Rate**: 0.9;
- **Elitism**: Applied;
- **Allow for Mutation to Generate Solution with Worse Fitness**: Don't allow;
- **Varying Operator Selection Probability**: Not applied;

In addition to the previously tested crossover rate of 1.0, with results initially shown in Table 8.7 and now repeated in Table 8.10 for easier comparison, crossover rates of 0.8 and 0.6 were also evaluated.

Table 8.10: Crossover Rate (Dataset 1)

Crossover Rate	Average Fitness Value (of the 5 runs)	Standard Deviation Fitness Value (of the 5 runs)	Best Fitness Value (out of the 5 runs)	Average Execution Time (min)
1.0	1.5673	0.003899	1.562	113
0.8	1.5668	0.01003	1.554	113
0.6	1.569	0.00697	1.5625	157

As evidenced in Table 8.10, a crossover rate of 0.8 yielded the best results. This was true both in terms of the average fitness value and the best individual result among all the tests conducted for various crossover rates.

8.3.6 Population Size

The final configuration tested varied the population size. Previously, a population size of 216 was used. To compare this with a larger population size, a population of 512 was tested. For this test, the remaining parameters were fixed as follows:

- **Maximum Number of Generations**: 1250;
- **Crossover Rate**: 0.8;
- **Mutation Rate**: 0.9;
- **Elitism**: Applied;

- **Allow for Mutation to Generate Solution with Worse Fitness**: Don't allow;
- **Varying Operator Selection Probability**: Not applied.

The results of the tests, as well as a comparison with the best results obtained with the smaller population size of 216 are presented in Table 8.11. This increase in population resulted in a marginal improvement in the average fitness value, however, it came with an increase in the average execution time, proportional to the increase in population size: from 113 minutes to 272.

Table 8.11: Population Size (Dataset 1)

Population Size	**Average Fitness Value (of the 5 runs)**	**Standard Deviation Fitness Value (of the 5 runs)**	**Best Fitness Value (out of the 5 runs)**	**Average Execution Time (min)**
216	1.5668	0.01003	1.554	113
512	1.5626	0.00712	1.5545	272

Despite the longer running time, the convergence graphs for the best runs among all five tests for each population size, depicted in Figures 8.1 and 8.2, indicate that the population of size 216 reached a plateau in fitness value after generation 800. In contrast, the population size of 512 shows potential for continuing along a path towards further convergence to potentially better solutions, suggesting that additional iterations could lead to improved outcomes.

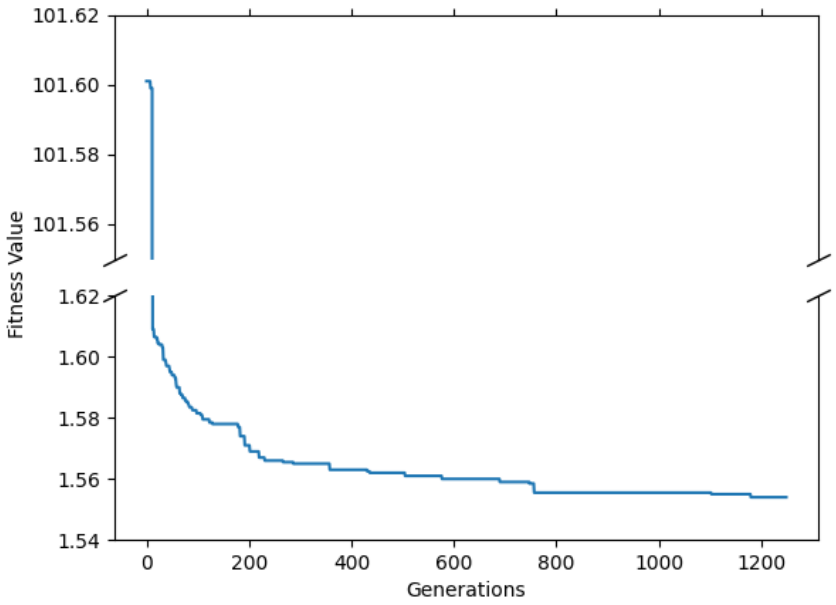

Figure 8.1: Convergence Graph - Genetic Algorithm (216 Population Size - Dataset 1)

For Dataset 2 (non-optimized dataset), the best population was also tested with the same parameters. The results are presented in Table 8.12. The results are in-line with the findings for

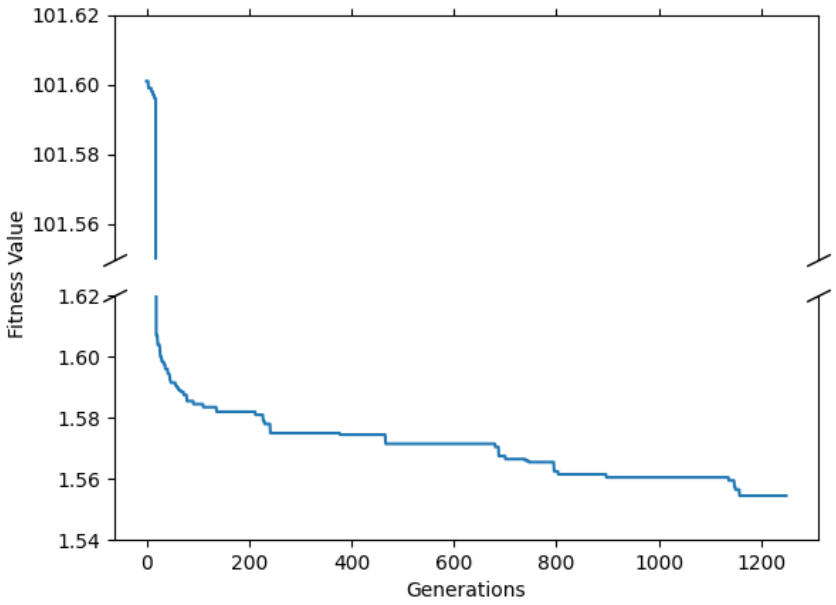

Figure 8.2: Convergence Graph - Genetic Algorithm (512 Population Size - Dataset 1)

Dataset 1 - a bigger population improves overall results and increases execution time. Moreover the convergence graphs also indicate a potential for further improvement as the population increases in size, as evidenced by the convergence graphs presented in Figures 8.3, 8.4 and 8.5.

Table 8.12: Population Size (Dataset 2)

Population Size	**Average Fitness Value (of the 5 runs)**	**Standard Deviation Fitness Value (of the 5 runs)**	**Best Fitness Value (out of the 5 runs)**	**Average Execution Time (min)**
216	0.8031	0.02797	0.767	166
512	0.7845	0.02520	0.751	221
1000	0.7599	0.03489	0.7235	339

Initially, before running tests, there was a concern regarding the genetic algorithm's memory usage due to its slightly more complex chromosome structure, as opposed to a binary encoded one. The fear was that this complexity might lead to excessive RAM consumption, rendering the algorithm impractical to run on standard computers at the Faculty of Engineering, which typically have around 8GB of RAM. However, an analysis of the process's RAM usage revealed that for the population sizes tested, memory usage was not a limiting factor. The results of this analysis, presented in Table 8.13, show that the bottleneck in increasing population size is not necessarily memory, but the proportional increase in execution time, evidenced by the results in Table 8.11, might be.

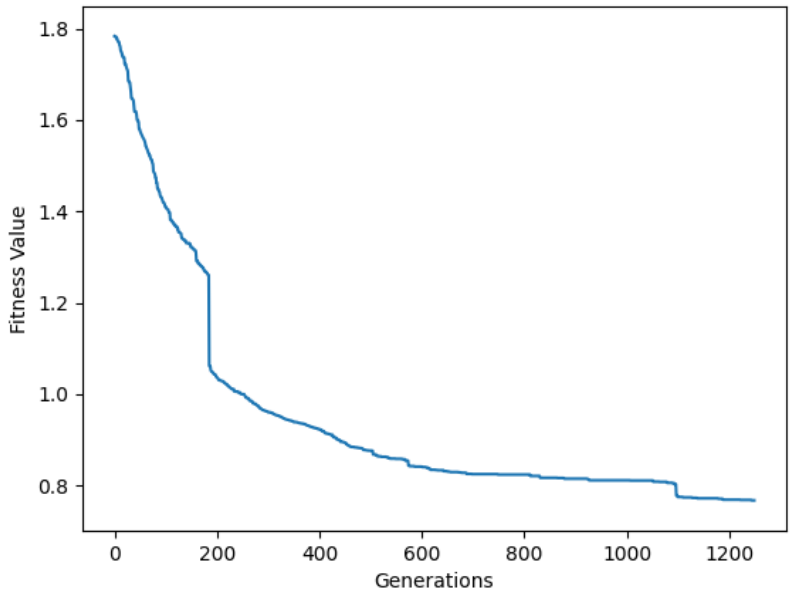

Figure 8.3: Convergence Graph - Genetic Algorithm (216 Population Size - Dataset 2)

8.3.7 Discussion

Initially, the impact of mutation rates on algorithm performance was explored while keeping other parameters fixed. The results, detailed in Table 8.6, showed varying outcomes across different mutation rates. Notably, a mutation rate of 0.9 consistently produced the lowest fitness values, indicating better performance in terms of solution quality. However, this configuration also resulted in longer execution times.

Building upon the insights from the mutation rate tests, the effect of elitism on performance was examined (Table 8.7). With elitism applied, the algorithm exhibited more consistent results across runs, particularly evident with higher mutation rates. This stability in results underlines the importance of elitism in preserving superior solutions across generations, ultimately contributing to better overall performance.

The subsequent investigation focused on varying the probability of selecting mutation operators (section 5.7). Surprisingly, the results (Table 8.8) demonstrated that using equal probabilities consistently outperformed varying probabilities in terms of both average and best fitness values. Unfortunately, due to the limited timeframe for testing, further exploration of these negative results was not feasible. It is worth noting that despite running for the same number of generations,

Table 8.13: RAM Memory Usage (Dataset 1)

Population Size	216	512
RAM Memory Utilized (MB)	298.125	700.625

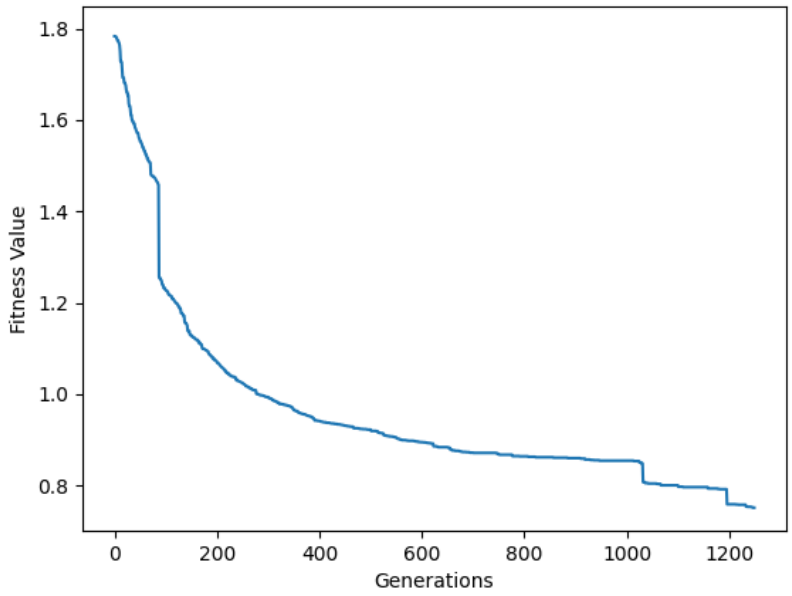

Figure 8.4: Convergence Graph - Genetic Algorithm (512 Population Size - Dataset 2)

the configuration with varying probabilities required less time for execution. Comparing the performance of both operators over an equal duration would provide valuable insights. Moreover, the operator employing varying probabilities initially uses equal probabilities for operator selection, gathering data on each operator's impact. Only after each operator has been applied at least 1000 times does it begin varying probabilities. Increasing the number of initial applications for each operator could potentially enhance the starting conditions and improve overall results.

The impact of allowing mutations to generate solutions with worse fitness value constraints and create solutions with worse fitness values was also tested (Table 8.9). Allowing such violations did not lead to improved outcomes. Instead, it resulted in inferior fitness values. This phenomenon underscores the delicate balance required in mutation operations to avoid excessive diversification that hampers convergence towards optimal solutions. Furthermore, these results reinforce findings in the literature that hybrid algorithms (in this particular case, an algorithm combining genetic algorithms with local search for mutations) perform better than non-hybrid approaches (a genetic algorithm with random mutations which allow for solutions with a worse fitness value).

Variation in crossover rate was another parameter explored to assess its influence on algorithm performance (Table 8.10). A crossover rate of 0.8 emerged as the most effective, consistently achieving the lowest average fitness values across multiple runs and yielding the best overall fitness value among all configurations. Conversely, lowering the crossover rate to 0.6 resulted in inferior outcomes. This improvement associated with a lower crossover rate (when compared to the previous best configuration of 1.0 crossover rate) may be linked to a slower loss of diversity. While crossover is a pivotal operator for creating new chromosomes from parent chromosomes, as previously noted, it can also diminish diversity over time.

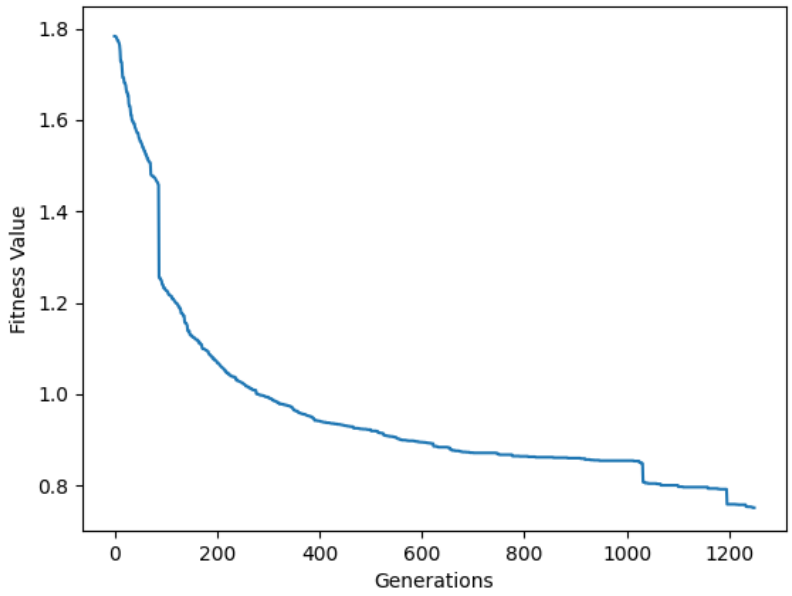

Figure 8.5: Convergence Graph - Genetic Algorithm (1000 Population Size - Dataset 2)

Finally, the impact of population size on performance was examined by comparing results between populations of 216, 512 and 1000 individuals (tables 8.11 and 8.12). Increasing the population size resulted in a slight improvement in average fitness values but significantly extended execution times. This trade-off underscores the practical considerations involved in balancing computational resources with performance gains in genetic algorithm applications. Nevertheless, if time constraints are not a concern, convergence graphs (Figures 8.1, 8.2, 8.3, 8.4 and 8.5) suggest that running the algorithm with larger populations, for a larger number of generations, may lead towards solutions with even better fitness values.

For comparison with other algorithms, the configuration with a population size of 216 was chosen, as it provides a balanced compromise between timeliness and effectiveness in reducing the fitness value.

8.4 Simulated Annealing

Determining optimal parameter values for a simulated annealing algorithm is a complex endeavor, specifically regarding the initial temperature and cooling rate, which dictate the acceptance probability of solutions with worse fitness values. To set these parameters, the first step was to establish the duration of the algorithm, which was chosen to run for 113 minutes to compare with the best configuration of the previously tested genetic algorithm (for Dataset 1).

Initial tests revealed that this duration would allow the algorithm to complete at least 400 iterations. Specific targets were then set: the algorithm should reach an 80% acceptance probability for solutions with worse fitness values by the time it completes 37.5% of its iterations (150 iterations),

decreasing to just 10% acceptance by 50% of the iterations (200 iterations). This configuration is expected to enable the algorithm to explore the search space extensively in the first 150 iterations, followed by an increasingly greedy search for the remainder of the run.

The choice of these initial parameters was based on balancing exploration and exploitation phases. Setting a high initial acceptance rate should help the algorithm escape local optima in the early stages, allowing for a broader search. As the algorithm progresses, gradually reducing the acceptance rate should increase the focus on refining the best solutions found, promoting convergence.

To compute the parameters that achieve these targets, several considerations were taken into account. It was decided that the change in energy (δE) should correspond to -0.0005, indicative of violating the least penalizing soft constraint. This choice ensures that violations of more critical soft constraints will result in larger energy changes, leading to lower acceptance probabilities.

Based on these considerations, the initial temperature T_0 and cooling rate Cr were calculated using the formulas detailed in section 6.5:

$$P_{\text{acceptance}_k} = \exp\left(-\frac{\delta E}{T_k}\right), \quad \text{then,} \quad T_k = -\frac{\delta E}{\ln(P_{\text{acceptance}_k})}$$

Replacing with values:

$$T_{150} = -0.0005/\ln(80\%) = 0.00224071$$

$$T_{200} = -0.0005/\ln(10\%) = 0.00021715$$

According to the cooling schedule formula:

$$T_k = T_0 \cdot \text{Cr}^k, \quad \text{then} \quad T_0 = \frac{T_k}{\text{Cr}^k}$$

Thus,

$$T_{150} = T_0 \cdot \text{Cr}^{150}$$

$$T_{200} = T_0 \cdot \text{Cr}^{200}$$

Solving for T_0 and Cr:

$$\text{Cr} \approx 0.9544, \quad T_0 \approx 2.462$$

In the end, the algorithm ran for an average of 653 iterations, meaning that since the acceptance probability reaches a value of 10% by iteration 200, it predominantly behaved like a greedy algorithm for approximately two-thirds of its iterations (from 200 to 653). Similar to the genetic algorithm, the simulated annealing configuration tests were repeated five times for each parameter set. The lowest fitness value achieved for the initial configuration across all runs was 1.587. The full results are presented in Table 8.14.

The graph presented in Figure 8.6, shows the convergence of the run that produced the best fitness value. The graph highlights two main factors: a tendency to converge to better fitness

Table 8.14: Simulated Annealing $(T_0, Cr) = (2.462, 0.9544)$ (Dataset 1)

Initial Temperature, T_0	2.462
Cooling Rate, Cr	0.9544
Avg. Iterations (of the 5 runs)	653
Avg. Fitness Value (of the 5 runs)	41.5881
Std. Deviation Fitness Value (of the 5 runs)	54.77
Best Fitness Value (out of the 5 runs)	1.587

values as the algorithm becomes greedier (i.e., acceptance rates are lower), and a stagnation of the fitness value after iteration 400. This stagnation can be attributed to the algorithm reaching a local optimum and being unable to find better solutions afterward. Consequently, it was decided to test parameters that maintain the acceptance of worse solutions throughout the algorithm's execution, but at a lower rate.

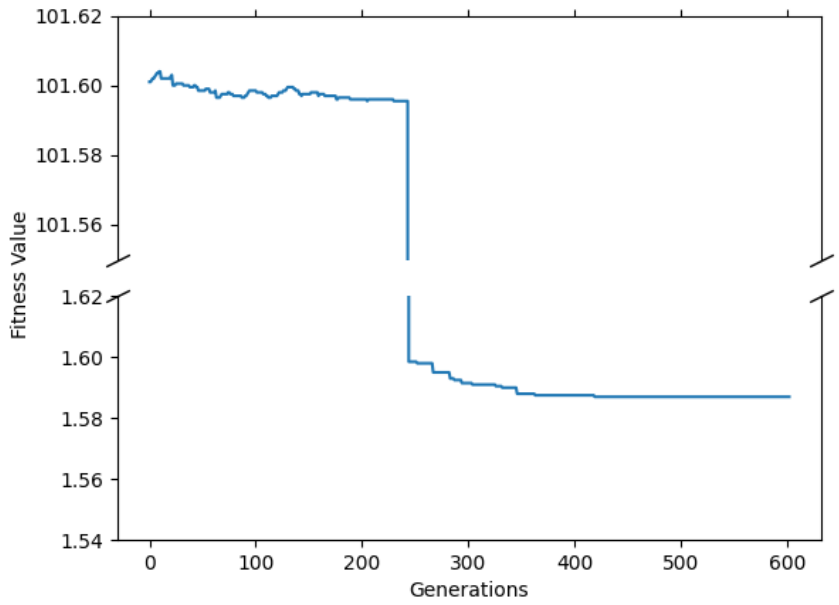

Figure 8.6: Convergence Graph - Simulated Annealing $(T_0, Cr) = (2.462, 0.9544)$ (Dataset 1)

The values used for the next round of testing ensured an acceptance rate of 80% by iteration 300, and 20% by iteration 550. The method used to calculate the parameters were the same as presented earlier in this section, and the values obtained were, $T_0 = 0.0239924$ and $Cr = 0.992128$. The results obtained are presented in Table 8.15, and the convergence graph for the best result out of the five tests is presented in Figure 8.7.

For the second dataset (Dataset 2), the stop criteria was 166 minutes, which corresponds to the average execution time of the best configuration of the genetic algorithm with a population of 216 for the same dataset.

Table 8.15: Simulated Annealing $(T_0, Cr) = (0.0239924, 0.992128)$ (Dataset 1)

Initial Temperature, T_0	0.0239924
Cooling Rate, Cr	0.992128
Avg. Iterations (of the 5 runs)	632
Avg. Fitness Value (of the 5 runs)	21.588
Std. Deviation Fitness Value (of the 5 runs)	44.72
Best Fitness Value (out of the 5 runs)	1.5795

The configuration used to test the simulated annealing algorithm on this dataset was a compromise between the previously tested configurations: 60% acceptance rate by 50% of the iterations (in this case around iteration 2200, since for the determined execution time the algorithm runs for about 4400 iterations) and 10% acceptance rate by 90% of the iterations (around iteration 3960). The results obtained are presented in Table 8.16 and in the convergence graph in Figure 8.8.

Table 8.16: Simulated Annealing $(T_0, Cr) = (0.0064262, 0.999145)$ (Dataset 2)

Initial Temperature, T_0	0.0064262
Cooling Rate, Cr	0.999145
Avg. Iterations (of the 5 runs)	4347
Avg. Fitness Value (of the 5 runs)	0.9326
Std. Deviation Fitness Value (of the 5 runs)	0.11107
Best Fitness Value (out of the 5 runs)	0.863

8.4.1 Discussion

The simulated annealing approach successfully optimized the timetable values. Testing of different parameter configurations revealed that this type of algorithm is, as highlighted by the literature, highly sensitive to parameter configuration. The best configuration out of the two tested was the one that maintained an acceptance of worse solutions throughout the algorithm's execution, but at a lower rate during the final iterations. This underscores the importance of maintaining the temperature at levels that permit at least some acceptance of worst solutions, enabling the algorithm to escape local optima. Prematurely reducing acceptance values causes the algorithm to behave like a greedy algorithm, thus failing to leverage its strengths.

For Dataset 1, using the first configuration, the algorithm failed to resolve the existing hard constraint violation in two out of five tests. In contrast, with the second configuration, it successfully resolved the hard constraint in four out of five tests. These results are consistent with the observation that this algorithm is frequently utilized in the literature as a secondary algorithm in collaborative methods, where the optimization target is an already feasible solution. This indicates that the algorithm generally excels at optimizing for soft constraint violations.

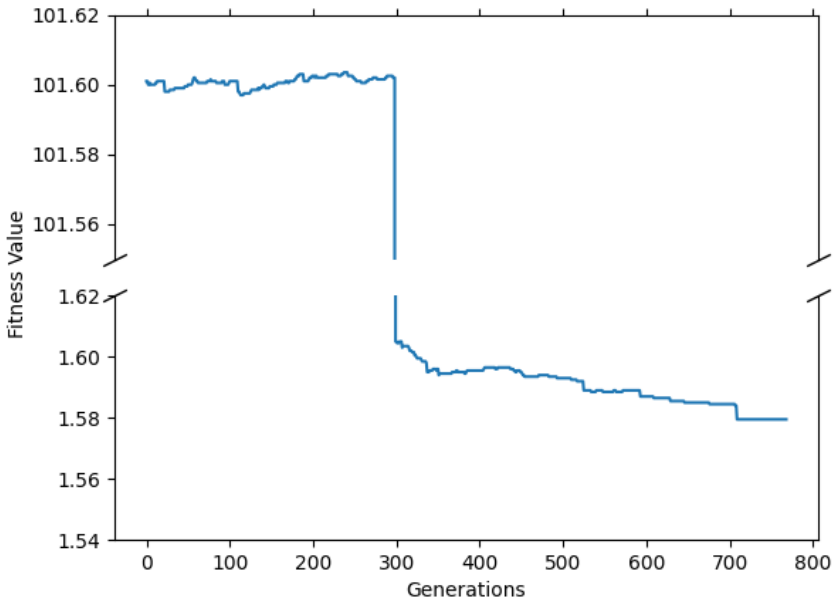

Figure 8.7: Convergence Graph - Simulated Annealing $(T_0, Cr) = (0.0239924, 0.992128)$ (Dataset 1)

Regarding the results obtained with Dataset 2, there was an average reduction in the fitness value of 47.7%. However, the high standard deviation indicates that this approach is somewhat unreliable, particularly when time constraints are a critical factor.

8.5 Greedy Local Search

The greedy local algorithm developed operates deterministically without requiring any parameter configuration. Due to its deterministic nature, starting from the same initial solution always yields the same results since randomness is not a factor in its execution.

During testing, for Dataset 1, the algorithm completed two iterations before converging to a local optimum where further improvement was not achievable. The best fitness value achieved in these conditions was 1.582, after a total runtime of 15 minutes. Table 8.17 presents the progression of fitness values across each iteration.

Table 8.17: Greedy Local Search: Best Fitness per Iteration (Dataset 1)

Iteration	**Fitness Value**
0	101.601
1	1.5875
2	1.582

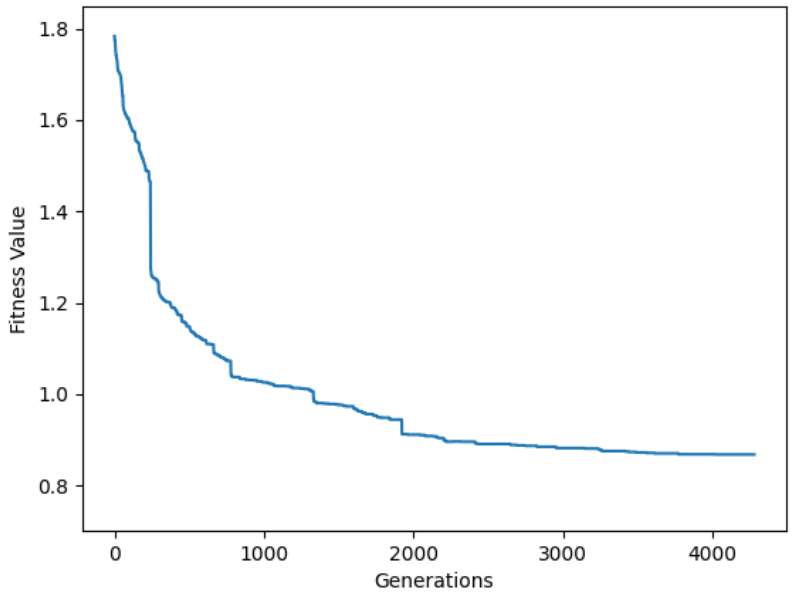

Figure 8.8: Convergence Graph - Simulated Annealing $(T_0, Cr) = (0.0064262, 0.999145)$ (Dataset 2)

As for Dataset 2, the algorithm proved even more successful, almost halving the fitness value within 14 minutes, at which point it reached a local optima from which it couldn't escape and that dictated the end of its execution. The results for this dataset are presented in Table 8.18

Table 8.18: Greedy Local Search: Best Fitness per Iteration (Dataset 2)

Iteration	Fitness Value
0	1.7835
1	1.1585
2	0.8845
3	0.854
4	0.843
5	0.84

8.6 Algorithm Comparison

Comparing all the algorithms' performance on the optimized dataset, Dataset 1, the genetic algorithm performed the best overall, as evidenced by the results in Table 8.19, which highlights the best test results for each algorithm. Genetic algorithm obtained both the best overall result of all the tests conducted, as well as the best average value on several runs. Although in the review of the literature conducted, simulated annealing was more popular in specifically tackling

the second stage of collaborative approaches, where the goal is to improve on feasible solutions, in this particular case of optimizing a near-feasible (1 hard-constraint away from being feasible) solution, it didn't provide results as positive as the genetic algorithm. Moreover, when compared to the greedy local search, it performed poorly, taking around 5 times more time to produce a result that was on average worst, and just slightly better on the best runs.

Table 8.19: Comparison of Results for Each Algorithm

Algorithm	**Average Fitness Value (Dataset 1)**	**Best Fitness Value (Dataset 1)**	**Average Fitness Value (Dataset 2)**	**Best Fitness Value (Dataset 2)**
Genetic Algorithm	1.5668	1.554	0.8031	0.767
Simulated Annealing	21.5883	1.5795	0.9326	0.863
Greedy Local Search	-	1.582	-	0.84

As for the non-optimized dataset, Dataset 2, the genetic algorithm was also the best performing algorithm, followed by the greedy local search, which outperformed the simulated annealing approach both in the best fitness value obtained, as well as on average.

The lackluster performance of the simulated annealing algorithm can be attributed to two primary factors. First, as a local search algorithm, simulated annealing struggles to escape local optima once it reaches or approaches them. Although the algorithm has an inherent ability to escape local optima by temporarily accepting worse solutions, this ability is diminished in this context. This is because the neighborhoods being searched only contain feasible solutions. To transition to a different neighborhood that might provide a better solution, the algorithm would need to accept a solution that temporarily violates hard constraints, which is not permitted by the defined neighborhood operators. This restriction likely hampers simulated annealing's ability to effectively explore the solution space and identify a significantly better timetable. Second, this algorithm benefits the least from parallelization. Both the genetic algorithm and the greedy local search algorithm fully utilize all 112 available CPU threads at their peak of usage. In contrast, simulated annealing only utilizes up to 8 threads at its peak, with one thread assigned to each neighborhood being searched simultaneously. Due to its low potential for parallelization, simulated annealing underperforms when compared to algorithms that can be heavily parallelized.

Genetic algorithms are also known for their superior exploratory ability, which might explain the better performance in scenarios where the initial solution is close to a local optimum. Unlike simulated annealing, which focuses on a single solution, the genetic algorithm is population-based. This population-based approach allows the genetic algorithm to explore multiple search directions simultaneously, significantly increasing its ability to escape local optima. By maintaining a diverse pool of potential solutions, the genetic algorithm can effectively explore a broader portion of the solution space, making it more adept at finding improved solutions even when starting from a near-optimal state.

The greedy algorithm proved particularly interesting for its speed. Its rapid execution makes it an attractive candidate for application in conjunction with either the genetic algorithm or simulated

annealing, potentially enhancing the results obtained by these methods. By quickly providing a good initial solution or refining a solution found by other algorithms, the greedy algorithm could serve as a complementary tool to further improve the overall optimization process, or even for scenarios in which a particularly fast optimization approach is required.

Chapter 9

Conclusions

This dissertation explored the application of metaheuristic approaches, specifically genetic algorithms and simulated annealing, to improve university timetabling at the Department of Informatics of the Faculty of Engineering of the University of Porto. The primary goal was to reduce the reliance on manual adjustments by automating the optimization of already feasible timetables, to better adhere to soft constraints.

The research was grounded in a review of related work, which highlighted the effectiveness of metaheuristic approaches in solving the University Course Timetabling Problem. Previous studies indicated that genetic algorithms, due to their evolutionary operations, and simulated annealing, with its probabilistic acceptance of solutions, were particularly promising for this NP-hard problem, offering both exploratory and exploitative abilities, respectively. These insights informed the development of the algorithms presented here.

The genetic algorithm was expected to provide robust solutions by effectively exploring the solution space and converging on high-quality schedules. This expectation was met, as the genetic algorithm outperformed the remaining algorithms tested.

Simulated annealing was anticipated to be especially useful for refining already good timetables by allowing uphill moves and thus escaping local optima. It's positive performance when applied in collaborative approaches was particularly interesting, since, in those approaches, simulated annealing was applied to already feasible timetables, to improve soft constraint adherence, much like in the case of the optimization challenge at the Department of Informatics. The developed algorithm improved on the starting solution, but came short of reaching the results obtained by both the genetic algorithm, and the greedy local search. The primary factor potentially explaining these subpar results is the disparity in parallelization among the algorithms. While all the algorithms were parallelized, the simulated annealing approach benefited the least from this parallelization.

A greedy local search method was tested to serve as baseline against the two others. While it was less effective than the genetic algorithm, it still provided significant improvements over the initial schedules in a very short time window, and outperformed most test runs of the simulated annealing approach.

Despite successfully fulfilling the task of optimizing timetables, this work has some limitations. Firstly, the algorithms were all tested on a server with a high thread count which benefited primarily the algorithm with a higher potential for parallelization (genetic algorithm and greedy local search).

Additionally, as a measure of the quality of the generated solutions, it would be interesting to compare the results of the optimization performed by these algorithms against the manual changes done by the staff of the Department of Informatics. Moreover, the performance of the algorithms developed is influenced by the specific parameter settings, and finding the optimal settings can be time-consuming, which meant not being able to test parameters exhaustively. There is also the inherent limitation of metaheuristic approaches, which cannot guarantee finding the global optimum solution due to their heuristic nature.

Future work directions include testing the algorithms on a machine with a diminished thread count (e.g. a laptop) to understand how the algorithms compare in an environment with less parallelization potential. Additionally, conducting further tests with different parameters might prove useful for further optimizing the performance of the simulated annealing algorithm.

This work focused on metaheuristic approaches; however, the rise in popularity and the positive performance demonstrated by matheuristic approaches and other mathematically formulated methods in the 2019 International Timetabling Competition merit further exploration of these types of algorithms.

In conclusion, this dissertation successfully demonstrated that metaheuristic approaches provide a robust and efficient means of optimizing university timetables. Among the methods tested, the genetic algorithm approach outperformed the others, underscoring and justifying its widespread popularity within the research community, particularly when applied to real-world scenarios. The empirical evidence gathered from this study suggests that genetic algorithms are well-suited for addressing the complexities and constraints inherent in university timetabling. Furthermore, the findings of this research pave the way for further advancements in automating the timetabling efforts at the Department of Informatics of the Faculty of Engineering of the University of Porto.

Bibliography

Abdelhalim, E. A. and El Khayat, G. A. (2016). A utilization-based genetic algorithm for solving the university timetabling problem (uga). *Alexandria Engineering Journal*, 55(2):1395–1409. DOI: 10.1016/j.aej.2016.02.017.

Abdipoor, S., Yaakob, R., Goh, S. L., and Abdullah, S. (2023). Meta-heuristic approaches for the university course timetabling problem. *Intelligent Systems with Applications*, 19:200253. DOI: 10.1016/j.iswa.2023.200253.

Abdullah, S., Turabieh, H., McCollum, B., and Burke, E. K. (2009). An investigation of a genetic algorithm and sequential local search approach for curriculum-based course timetabling problems. In *Proceedings of the Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2009), Dublin, Ireland*, pages 727–731.

AbouElhamayed, A. F., Mahmoud, A. S., Shaaban, T. T., Salama, C., and Yousef, A. H. (2016). An enhanced genetic algorithm-based timetabling system with incremental changes. In *2016 11th International Conference on Computer Engineering & Systems (ICCES)*, pages 122–127. DOI: 10.1109/ICCES.2016.7821985.

Akkan, C., Gülcü, A., and Kuş, Z. (2022). Bi-criteria simulated annealing for the curriculum-based course timetabling problem with robustness approximation. *Journal of Scheduling*, 25(4):477–501. DOI: 10.1007/s10951-022-00722-0.

Alghamdi, H., Alsubait, T., Alhakami, H., and Baz, A. (2020). A review of optimization algorithms for university timetable scheduling. *Engineering, Technology & Applied Science Research*, 10(6):6410–6417. DOI: 10.48084/etasr.3832.

Alhijawi, B. and Awajan, A. (2023). Genetic algorithms: theory, genetic operators, solutions, and applications. *Evolutionary Intelligence*, 17(3):1245–1256. DOI: 10.1007/s12065-023-00822-6.

Almond, M. (1966). An algorithm for constructing university timetables. *The Computer Journal*, 8(4):331–340. DOI: 10.1093/comjnl/8.4.331.

Amine, K. (2019). Multiobjective simulated annealing: Principles and algorithm variants. *Advances in Operations Research*, 2019(1):8134674.

Bashab, A., Ibrahim, A. O., AbedElgabar, E. E., Ismail, M. A., Elsafi, A., Ahmed, A., and Abraham, A. (2020). A systematic mapping study on solving university timetabling problems using meta-heuristic algorithms. *Neural Computing and Applications*, 32(23):17397–17432. DOI: 10.1007/s00521-020-05110-3.

Bettinelli, A., Cacchiani, V., Roberti, R., and Toth, P. (2015). An overview of curriculum-based course timetabling. *Transactions in Operations Research*, 23(2):313–349. DOI: 10.1007/s11750-015-0366-z.

Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization. *ACM Computing Surveys*, 35(3):268–308. DOI: 10.1145/937503.937505.

Brindle, A. (1980). *Genetic algorithms for function optimization*. PhD thesis, University of Alberta. DOI: 10.7939/R3FB4WS2W.

Bullet Solutions (2023). Bullet Solutions | Leading the Scheduling and Timetabling Evolution. https://bulletsolutions.com/. [Accessed 01-02-2024].

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Qu, R. (2013). Hyper-heuristics: a survey of the state of the art. *The Journal of the Operational Research Society*, 64(12):1695–1724. DOI: 10.1057/jors.2013.71.

Camacho-Villalón, C. L., Dorigo, M., and Stützle, T. (2018). Why the intelligent water drops cannot be considered as a novel algorithm. In *Swarm Intelligence*, pages 302–314, Cham. Springer International Publishing. DOI: 10.1007/978-3-030-00533-7_24.

Camacho Villalón, C. L., Stützle, T., and Dorigo, M. (2020). Grey wolf, firefly and bat algorithms: Three widespread algorithms that do not contain any novelty. In *Swarm Intelligence*, pages 121–133, Cham. Springer International Publishing.

Ceschia, S., Di Gaspero, L., and Schaerf, A. (2023). Educational timetabling: Problems, benchmarks, and state-of-the-art results. *European Journal of Operational Research*, 308(1):1–18. DOI: 10.1016/j.ejor.2022.07.011.

Chang, K.-H. (2015). Multiobjective optimization and advanced topics. In *e-Design*, pages 1105–1173. Elsevier. DOI: 10.1016/B978-0-12-382038-9.00019-3.

Chen, M. C., Sze, S. N., Goh, S. L., Sabar, N. R., and Kendall, G. (2021). A survey of university course timetabling problem: perspectives, trends and opportunities. *IEEE access : practical innovations, open solutions*, 9:106515–106529. DOI: 10.1109/ACCESS.2021.3100613.

Czarn, A., MacNish, C., Vijayan, K., Turlach, B., and Gupta, R. (2004). Statistical exploratory analysis of genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 8:405–421. DOI: 10.1109/TEVC.2004.831262.

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, USA. AAI7609381.

Deng, X., Zhang, Y., Kang, B., Wu, J., Sun, X., and Deng, Y. (2011). An application of genetic algorithm for university course timetabling problem. In *2011 Chinese Control and Decision Conference (CCDC)*, pages 2119–2122. IEEE.

Di Gaspero, L., McCollum, B., and Schaerf, A. (2007). The Second International Timetabling Competition (ITC-2007): Curriculum-based Course Timetabling (Track 3). In *Proceedings of the 1st International Workshop on Scheduling a Scheduling Competition (SSC 2007)*, Providence (RI), USA.

Eiben, A. and Smith, J. (2015). *Introduction to Evolutionary Computing*. Springer Berlin Heidelberg. DOI: 10.1007/978-3-662-44874-8.

Febrita, R. E. and Mahmudy, W. F. (2017). Modified genetic algorithm for high school timetable scheduling with fuzzy time window. In *2017 International Conference on Sustainable Information Engineering and Technology (SIET)*, pages 88–92. IEEE.

FEUP (2023). FEUP em Números 2022. https://sigarra.up.pt/feup/pt/p/feup%20em%20n%C3%BAmeros%202022. [Accessed 01-02-2024].

Fogel, D. B. (1998). *Artificial Intelligence through Simulated Evolution*, pages 227–296. IEEE. DOI: 10.1109/9780470544600.ch7.

Gad, A. F. (2023). Pygad: An intuitive genetic algorithm python library. *Multimedia Tools and Applications*, pages 1–14.

Gagné, C. and Parizeau, M. (2002). Open beagle: A new versatile c++ framework for evolutionary computations. *GECCO Late Breaking Papers*, pages 161–168.

Geman, S. and Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741. DOI: 10.1109/TPAMI.1984.4767596.

Goel, T. and Deb, K. (2002). Hybrid methods for multi-objective evolutionary algorithms. In *Proceedings of the Fourth Asia-Pacific Conference on Simulated Evolution and Learning (SEAL, 02).(Singapore)*, pages 188–192.

Goh, S. L., Kendall, G., Sabar, N. R., and Abdullah, S. (2020). An effective hybrid local search approach for the post enrolment course timetabling problem. *Opsearch*, 57:1131–1163.

Gülcü, A. and Akkan, C. (2020). Robust university course timetabling problem subject to single and multiple disruptions. *European Journal of Operational Research*, 283(2):630–646.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press.

Ilyas, R. and Iqbal, Z. (2015). Study of hybrid approaches used for university course timetable problem (uctp). In *2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA)*, pages 696–701. DOI: 10.1109/ICIEA.2015.7334198.

Kazimipour, B., Li, X., and Qin, A. K. (2014). A review of population initialization techniques for evolutionary algorithms. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 2585–2592. DOI: 10.1109/CEC.2014.6900618.

Keijzer, M., Merelo, J. J., Romero, G., and Schoenauer, M. (2002). *Evolving Objects: A General Purpose Evolutionary Computation Library*, page 231–242. Springer Berlin Heidelberg. DOI: 10.1007/3-540-46033-0_19.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680. DOI: 10.1126/science.220.4598.671.

Kohshori, M. S. and Abadeh, M. S. (2012). Hybrid genetic algorithms for university course timetabling. *International Journal of Computer Science Issues (IJCSI)*, 9(2):446.

Lewis, R., Paechter, B., and Mccollum, B. (2007). Post Enrolment based Course Timetabling: A Description of the Problem Model used for Track Two of the Second International Timetabling Competition. *Cardiff University, Cardiff Business School, Accounting and Finance Section, Cardiff Accounting and Finance Working Papers*.

Lewis, R. and Thompson, J. (2015). Analysing the effects of solution space connectivity with an effective metaheuristic for the course timetabling problem. *European Journal of Operational Research*, 240(3):637–648. DOI: 10.1016/j.ejor.2014.07.041.

Lundy, M. and Mees, A. (1986). Convergence of an annealing algorithm. *Mathematical Programming*, 34(1):111–124. DOI: 10.1007/bf01582166.

Maniezzo, V., Boschetti, M. A., and Stützle, T. (2021). *Matheuristics: Algorithms and Implementations*. Springer International Publishing. DOI: 10.1007/978-3-030-70277-9.

Maskaoui, Z. E., Jalal, S., and Bousshine, L. (2017). Genetic algorithm parameters effect on the optimal structural design search. *IOSR Journal of Mechanical and Civil Engineering*, 14:124–130. DOI: 10.9790/1684-140305124130.

McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A. J., Gaspero, L. D., Qu, R., and Burke, E. K. (2010). Setting the research agenda in automated timetabling: the second international timetabling competition. *INFORMS journal on computing*, 22(1):120–130. DOI: 10.1287/ijoc.1090.0320.

Mohammadi, A., Asadi, H., Mohamed, S., Nelson, K., and Nahavandi, S. (2017). OpenGA, a C++ Genetic Algorithm Library. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. DOI: 10.1109/smc.2017.8122921.

Moreira, J. J. (2008). A system for automatic construction of exam timetable using genetic algorithms. *Tékhne-Revista de Estudos Politécnicos*, 6(9):319–336.

Müller, T., Rudová, H., and Müllerová, Z. (2024). Real-world university course timetabling at the international timetabling competition 2019. *Journal of Scheduling*. DOI: 10.1007/s10951-023-00801-w.

OpenMP Architecture Review Board (2021). *OpenMP Application Programming Interface*. OpenMP Architecture Review Board, 5.2 edition.

Oude Vrielink, R. A., Jansen, E. A., Hans, E. W., and van Hillegersberg, J. (2017). Practices in timetabling in higher education institutions: a systematic review. *Annals of Operations Research*, 275(1):145–160. DOI: 10.1007/s10479-017-2688-8.

Pandey, J. and Sharma, A. (2016). Survey on University timetabling problem. In *2016 3rd International conference on computing for sustainable global development (INDIACom)*, pages 160–164. IEEE.

Pillay, N. (2016). A review of hyper-heuristics for educational timetabling. *Annals of operations research*, 239(1):3–38. DOI: 10.1007/s10479-014-1688-1.

Piotrowski, A. P., Napiorkowski, J. J., and Rowinski, P. M. (2014). How novel is the "novel" black hole optimization approach? *Information Sciences*, 267:191–200. DOI: 10.1016/j.ins.2014.01.026.

Rahoual, M. and Saad, R. (2006). Solving timetabling problems by hybridizing genetic algorithms and taboo search. In *6th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006)*, pages 467–472.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Problemata (Stuttgart). Frommann-Holzboog.

Singh, G. and Gupta, N. (2022). *A Study of Crossover Operators in Genetic Algorithms*, pages 17–32. Springer Singapore, Singapore. DOI: 10.1007/978-981-16-3128-3_2.

Song, T., Chen, M., Xu, Y., Wang, D., Song, X., and Tang, X. (2021). Competition-guided multi-neighborhood local search algorithm for the university course timetabling problem. *Applied Soft Computing*, 110:107624. DOI: 10.1016/j.asoc.2021.107624.

Song, T., Liu, S., Tang, X., Peng, X., and Chen, M. (2018). An iterated local search algorithm for the University Course Timetabling Problem. *Applied Soft Computing*, 68:597–608. DOI: 10.1016/j.asoc.2018.04.034.

Spears, W. M. and De Jong, K. A. (1991). On the virtues of parameterized uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236.

Strenski, P. N. and Kirkpatrick, S. (1991). Analysis of finite length annealing schedules. *Algorithmica*, 6:346–366.

Sudholt, D. (2020). *The Benefits of Population Diversity in Evolutionary Algorithms: A Survey of Rigorous Runtime Analyses*, pages 359–404. Springer International Publishing, Cham. DOI: 10.1007/978-3-030-29414-4_8.

Suh, J. Y. and Van Gucht, D. (1987). *Distributed genetic algorithms*. Computer Science Department, Indiana University.

Tahera, K., Ibrahim, R. N., and Lochert, P. B. (2008). *A Comparative Study of Gender Assignment in a Standard Genetic Algorithm*, page 167–174. Springer US. DOI: 10.1007/978-0-387-74935-8_12.

Teoh, C. K., Wibowo, A., and Ngadiman, M. S. (2015). Review of state of the art for metaheuristic techniques in Academic Scheduling Problems. *Artificial Intelligence Review*, 44(1):1–21. DOI: 10.1007/s10462-013-9399-6.

Tripathy, A. (1982). Numerical optimization of computer models. *Journal of the Operational Research Society*, 33(12):1166–1166. DOI: 10.1057/jors.1982.238.

Weyland, D. (2010). A Rigorous An alysis of the Harmony Search Algorithm: How the Research Community can be Misled by a "Novel" Methodology. *International Journal of Applied Metaheuristic Computing*, 1(2):50–60. DOI: 10.4018/jamc.2010040104.

Weyland, D. (2015). A critical analysis of the harmony search algorithm—How not to solve sudoku. *Operations Research Perspectives*, 2:97–105. DOI: 10.1016/j.orp.2015.04.001.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and computing*, 4:65–85.

Wolpert, D. and Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82. DOI: 10.1109/4235.585893.

Yousef, A. H., Salama, C., Jad, M. Y., El-Gafy, T., Matar, M., and Habashi, S. S. (2016). A GPU based genetic algorithm solution for the timetabling problem. In *2016 11th International Conference on Computer Engineering & Systems (ICCES)*, pages 103–109. IEEE. DOI: 10.1109/ICCES.2016.7821982.