Improved Monte Carlo Tree Search for University Course Timetabling

Daniela Tomás
Master in Network and Information Systems Engineering
Department of Computer Science
Faculty of Sciences of the University of Porto
2025

Improved Monte Carlo Tree Search for University Course Timetabling

Daniela Tomás

Dissertation carried out as part of the Master in Network and Information Systems Engineering

Department of Computer Science

2025

Supervisor

Prof. Dr. João Pedro Pedroso, Professor Associado, Faculty of Sciences of the University of Porto

Acknowledgements

I would like to express my gratitude to my supervisor, Prof. João Pedro Pedroso, for all the availability, guidance, and support throughout this year. The advise and feedback always helped me stay on track.

To my parents, thank you for always believing in me and being my biggest supporters. Your encouragement gave me the strength to overcome the most challenging moments.

To my brother, thank you for the laughs, distractions, and for always being there when I needed it.

To my boyfriend, I am grateful for the endless support, motivation, and love. Thanks for listening, reminding me to take breaks, and always being willing to help.

Finally, to my university friends, thank you for sharing this journey with me. You made the experience lighter, more enjoyable, and truly unforgettable.

Resumo

A geração de horários para escolas e universidades é um problema de otimização combinatória NP-difícil, caracterizado por um espaço de pesquisa vasto e por restrições. Na Faculdade de Ciências da Universidade do Porto (FCUP), a elaboração dos horários a cada semestre para milhares de estudantes, dezenas de currículos e centenas de docentes é, atualmente, um processo manual, demorado e sujeito a erros.

Esta dissertação aborda a variante *Curriculum-Based Course Timetabling* (CB-CTT) do *University Course Timetabling Problem*, apresentando uma abordagem híbrida inovadora que combina *Monte Carlo Tree Search* (MCTS) com *Hill Climbing* (HC) e *diving*. O MCTS conduz a pesquisa global, simulando e avaliando múltiplas atribuições de evento, período e sala, priorizando os eventos mais restritos e expandindo os ramos mais promissores. Sempre que o MCTS encontra, durante a fase de simulação, um horário viável melhorado, o HC assume o controlo e aplica seis movimentos de vizinhança para refinar iterativamente a solução, enquanto preserva a sua viabilidade. A estratégia de diving foi introduzida para melhorar a efetividade do MCTS. Essa estratégia aprofunda soluções parciais promissoras, permitindo que o algoritmo se foque em regiões de maior qualidade no espaço de pesquisa.

Testes com as instâncias do terceiro *track* da ITC-2007 demonstram que a abordagem híbrida encontra consistentemente horários viáveis. No entanto, mesmo com limites de tempo alargados, a abordagem ainda não supera os melhores resultados publicados no que diz respeito à minimização de *soft constraints*.

Apesar de várias melhorias e adaptações ao algoritmo MCTS padrão, continuam a existir desafios. Ainda assim, a abordagem híbrida proposta revela um grande potencial e constitui uma base sólida para investigação futura e aplicação prática em contextos institucionais de criação de horários, como é o caso da FCUP.

Palavras-chave: *University Course Timetabling Problem*, *Curriculum-based Course Timetabling*, *Monte Carlo Tree Search*, *Hill Climbing, Diving*, ITC-2007

Abstract

Timetable generation for schools and universities is an NP-hard combinatorial optimization problem with a vast search space and hard-to-meet constraints. At the Faculty of Sciences of the University of Porto (FCUP), preparing semester schedules for thousands of students, dozens of curricula, and hundreds of lecturers is currently done manually by an activity that is time-consuming and prone to errors.

This dissertation addresses the Curriculum-Based Course Timetabling (CB-CTT) variant of the University Course Timetabling Problem by presenting a novel hybrid approach that combines Monte Carlo Tree Search (MCTS) with Hill Climbing (HC) and a diving strategy. MCTS drives the global search, simulating and evaluating multiple event, period, and room assignments, prioritizing the most constrained events and expanding the most promising branches. Whenever MCTS obtains an improved feasible timetable during simulation, HC takes over and applies six neighborhood moves to iteratively improve the solution while preserving the feasibility. A diving strategy was introduced to further enhance the MCTS effectiveness. This strategy deepens promising partial solutions, allowing the algorithm to focus on higher-quality regions of the search space.

Tests on the ITC-2007 track 3 benchmark instances show that the hybrid approach consistently finds feasible timetables. However, even with extended time limits, the approach does not yet outperform the best published results in terms of minimizing soft constraint violations.

Despite some improvements and changes to the standard MCTS algorithm, challenges remain. Nevertheless, the proposed hybrid approach demonstrates strong potential and a solid foundation for further research and practical adoption in institutional timetabling contexts like FCUP.

Keywords: University Course Timetabling Problem, Curriculum-based Course Timetabling, Monte Carlo Tree Search, Hill Climbing, Diving, ITC-2007

Table of Contents

List of Figures

List of Abbreviations

CB-CTT Curriculum-based Course Timetabling. 1, 4

COP Combinatorial Optimization Problem. 3, 13

FCUP Faculty of Sciences of the University of Porto. 1, 4, 15

HC Hill Climbing. 2, 3, 6, 7, 11, 12, 14, 22, 24, 25

ITC International Timetabling Competition. 5, 11

ITC-2007 Second International Timetabling Competition. 5, 12, 15, 24, 27, 28

MCTS Monte Carlo Tree Search. iv, 1– 3, 5, 13, 14, 16, 24, 27, 28

PATAT International Conference on the Practice and Theory of Automated Timetabling. 5

PE-CTT Post-Enrollment Course Timetabling. 4, 5, 13

SA Simulated Annealing. 6, 11, 12

TS Tabu Search. 6, 11, 13

UCT Upper Confidence Bounds for Trees. 6, 19

UCTTP University Course Timetabling Problem. 1– 4, 10, 11, 13

1. Introduction

University Course Timetabling Problem (UCTTP) is a complex combinatorial optimization problem that consists of allocating events, rooms, lecturers, and students to weekly schedules while meeting certain constraints. A particular focus of this research is on Curriculum-based Course Timetabling (CB-CTT), a variant of UCTTP where the scheduling process is centered around courses and their associated curricula.

To illustrate the complexity of the problem, consider two courses within the same curriculum: *Introduction to Programming* and *Calculus I*. Each course comes with distinct scheduling demands: the former might require three lectures per week (two theoretical and one lab), while *Calculus I* may demand two lectures (a theoretical and a practical). Each lecture requires specific resources and teaching environments: practical programming lectures typically require a computer lab, while large auditoriums are better suited for theoretical lectures to accommodate more students. Since these lectures cannot overlap, variations in lecture frequency and format add an additional layer of complexity to the scheduling process. Moreover, other specific requirements may arise depending on the institution's policies and the particular needs of courses or lecturers.

The challenge intensifies when considering a complete bachelor's or master's degree, which spans several years and involves numerous courses, each with multiple lectures and unique requirements. Given the scale and complexity of the problem, obtaining an optimal solution in usable time is typically infeasible. Nevertheless, heuristic algorithms have proved capable of producing approximate and high-quality solutions effectively.

1.1. Motivation

The process of building timetables, for instance, at Faculty of Sciences of the University of Porto (FCUP) each semester is currently time-consuming, not automated, and the final results are not always the most satisfactory. In general, existing tools focus primarily on visualizing timetables or on basic conflict detection. This situation not only results in suboptimal schedules but also increases the workload for administrative staff and impacts the satisfaction of both lecturers and students.

In the academic year 2023/2024, FCUP enrolled more than 5000 students, offered 117 different curricula, and coordinated 646 collaborators [1]. This scale underscores the need

for an effective timetabling system that can manage such scale while minimizing errors and conflicts.

1.2. Objectives

The primary objective of this dissertation is to contribute to the field of university course timetabling by proposing, implementing, and evaluating a novel method that combines Monte Carlo Tree Search (MCTS), Hill Climbing (HC), and a diving strategy.

More specifically, the objectives are to:

- Demonstrate that MCTS can be adapted to the CB-CTT problem effectively;
- Integrate MCTS with HC to refine feasible solutions;
- Introduce a diving strategy to guide search focus towards promising areas of the search space;
- Evaluate the method on the 21 benchmark instances from the Second International Timetabling Competition (ITC-2007) competition and analyze the feasibility and quality of the resulting timetables;
- Analyze the influence of key algorithmic parameters and design choices.

This dissertation does not aim to develop a complete timetabling system or integrate with visualization tools. Its contributions lie primarily in algorithmic innovation and performance evaluation.

1.3. Methodology and Contributions

To address the problem challenges, this dissertation proposes a novel hybrid approach combining two heuristic algorithms: MCTS and HC, along with a diving strategy. The MCTS heuristic search algorithm was chosen, as it has been applied to various optimization problems and has proven to be particularly effective in games. HC was also chosen to be used in conjunction with MCTS for local optimization, refining feasible solutions discovered during the global search. The diving strategy further strengthens the search process by deepening the exploration of promising partial solutions, thereby improving the chances of reaching higher-quality timetables.

Although MCTS has shown positive results in relevant areas, its application to UCTTP remains largely unexplored. Therefore, the proposed system presents an opportunity to

advance the state of the art and help in further studies by leveraging the global search capabilities of MCTS to explore diverse scheduling possibilities with HC to refine solutions locally. This hybrid approach aims to deliver both feasible and better-quality timetables while addressing the practical challenges faced by FCUP in managing large-scale scheduling.

1.4. Dissertation layout

After the introductory chapter stating the motivation, objectives, methodology, and contributions, the remainder of the dissertation is organized as follows:

- **Chapter 2 - Background:** Introduces the fundamental concepts needed to contextualize and understand the UCTTP, along with an overview of the algorithms used.
- **Chapter 3 - Related Work:** Reviews existing research on UCTTP, discussing various approaches, including operational research, metaheuristics, hyperheuristics, multi-objective, and hybrid methods.
- **Chapter 4 - Development:** Describes the design and implementation of the proposed hybrid approach.
- **Chapter 5 - Tests:** Details the experimental setup and evaluation metrics used to assess the system's performance.
- **Chapter 6 - Results:** Presents and analyzes the results obtained from the experiments.
- **Chapter 7 - Conclusion:** Summarizes the findings, discusses the contributions, and outlines potential future enhancements.

2. Background

This chapter will cover the concepts needed to contextualize and understand the University Course Timetabling Problem (UCTTP) and the algorithms used to obtain the final result, namely the Monte Carlo Tree Search (MCTS) and Hill Climbing (HC) algorithms.

2.1. Combinatorial Optimization Problems

Finding a solution for maximizing or minimizing a value is common in several real world problems. These problems often require searching for an optimal solution from a finite set of possibilities. They are characterized by their discrete nature and the challenge of finding optimal solutions in large search spaces.

Combinatorial Optimization Problems (COPs) arise in various fields, such as artificial intelligence, machine learning, auction theory, applied mathematics, and more. Some well-known COPs include the Knapsack Problem, the Traveling Salesman Problem, and Graph Coloring.

To solve these problems, various algorithmic techniques are used, including:

- **Exact algorithms:** Guarantee optimal solutions by exhaustively exploring the solution space. However, they can be computationally expensive, especially for large and complex problems, because their runtime and/or memory requirements often grow exponentially.
- **Approximation algorithms:** Produce provably near-optimal solutions, providing a quantifiable measure of how far the solution might diverge from the best possible one, making them valuable for problems where obtaining exact solutions is too demanding but some guarantees are desirable.
- **Heuristic algorithms:** Focus on practicality and efficiency by finding good solutions within a reasonable time frame. While they do not ensure optimality, they are often effective for solving large-scale problems. This type of algorithms will be the primary focus of this dissertation.

2.2. The University Course Timetabling Problem

UCTTP is an NP-hard COP that involves allocating events, rooms, lecturers, and students to weekly schedules while meeting certain predefined constraints. It falls under

the broader category of Educational Timetabling, which also includes other challenging problems such as Examination Timetabling.

Due to the size and complexity of the problem, obtaining an optimal solution in usable time is typically infeasible. Therefore, it is usually necessary to employ heuristic optimization techniques.

2.2.1 Curriculum-Based and Post-Enrollment Course Timetabling

UCTTP can be categorized into two main types:

- **Curriculum-Based Course Timetabling (CB-CTT):** Courses are grouped into predefined curricula, ensuring that no student or lecturer is allocated to overlapping courses within their curriculum.
- **Post-Enrollment Course Timetabling (PE-CTT):** Events are scheduled after students enroll in their courses, taking into account their preferred event combinations while minimizing conflicts.

These two problems differ significantly in terms of timing, flexibility, and constraints. PE-CTT is performed after the student has enrolled in their courses, while Curriculum-based Course Timetabling (CB-CTT) is performed first. PE-CTT adjusts to the student choices, providing greater flexibility, while CB-CTT follows a more rigid and predefined structure. Furthermore, PE-CTT often involves more complex constraints due to diverse student preferences, whereas CB-CTT deals with more predictable patterns. As a result, PE-CTT is appropriate for institutions with a flexible course enrollment, while CB-CTT is ideal for institutions with fixed curricula.

2.2.2 Constraints

In the context of UCTTP, constraints are often divided into two different types (Figure 2.1):

- **Hard constraints:** Ensure the feasibility of the timetable and must be strictly satisfied. Typically, a hard constraint includes avoiding overlapping events for the same student or lecturer.
- **Soft constraints:** Represent preferences to improve the quality of the solution without being mandatory. An example of a soft constraint is minimizing gaps in students' timetables to ensure a more compact timetable.

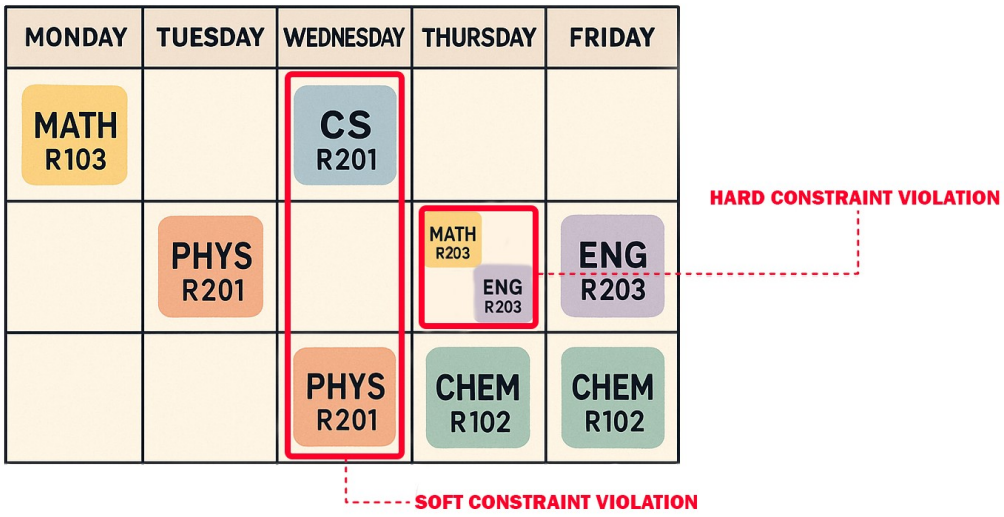

FIGURE 2.1: Hard and soft constraints example

2.3. International Timetabling Competition (ITC)

Since 2002, the International Conference on the Practice and Theory of Automated Timetabling (PATAT)* has supported timetabling competitions to encourage research in this field. There have already been five editions of the International Timetabling Competition (ITC), with three focusing on university timetabling. The third edition, held in 2011, centered on high school timetabling.

The first ITC in 2002 focused on the PE-CTT problem and utilized artificially generated instances. Over successive editions, the competition introduced problem instances that increasingly reflected real-world constraints and complexities.

The Second International Timetabling Competition (ITC-2007)† was structured into three distinct tracks: Examination Timetabling Problem, Post-enrollment Course Timetabling, and Curriculum-based Course Timetabling. The second one is an extended version of the previous competition. This dissertation focuses on the Curriculum-Based Course Timetabling track (track 3) of the ITC-2007, as it aligns with the problem under investigation. Even after 18 years, these benchmark datasets remain widely used in academic research and optimization studies, demonstrating their relevance in the field.

ITC-2019 has a more realistic and complex focus, reflecting real problems collected from universities around the world. However, because ITC-2019 has a higher entry curve, ITC-2007 was deemed sufficient for validating and testing the effectiveness of our approach.

*https://patatconference.org/communityService.html
†https://www.eeecs.qub.ac.uk/itc2007/

2.4. Monte Carlo Tree Search

MCTS is a decision-making algorithm that has been successfully applied in a variety of optimization problems with a huge search space. The algorithm has proven to be particularly effective in games such as Go and Chess, where it can outperform even the best human players [2].

2.4.1 Phases

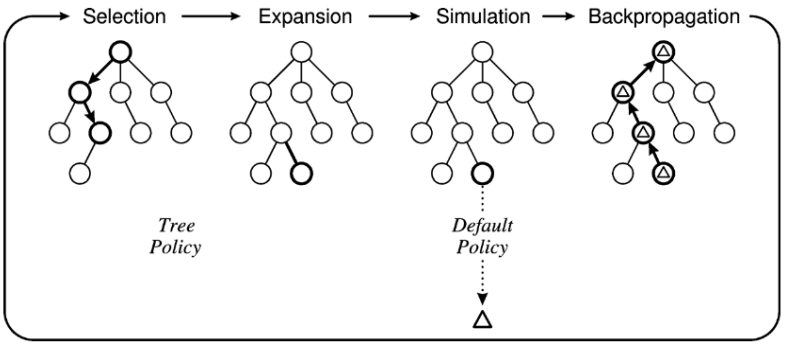

FIGURE 2.2: One MCTS iteration [3]

The algorithm consists of four phases, as illustrated in Figure 2.2 [3]. These phases are repeated for a predefined number of iterations or until a computational budget (such as time or memory) is reached. The phases are as follows:

1. **Selection:** The tree is traversed from the root node until it finds a node that is not completely expanded, i.e., a non-terminal node with unvisited children. The selection is guided by a policy that balances exploration and exploitation. Typically, the policy used is Upper Confidence Bounds for Trees (UCT), which selects nodes that maximize formula 2.1.

$$UCT = \frac{w_i}{n_i} + 2C\sqrt{\frac{2\ln n}{n_i}}, \tag{2.1}$$

   where w_i is the total reward of all playouts through this state, n_i is the number of visits of child node i, C is a constant greater than zero (typically $\sqrt{2}$), and n is the number of visits of the parent node.

2. **Expansion:** One or more child nodes are added to the node previously reached in the selection phase.

3. **Simulation:** From the newly added node(s), a simulation is performed according to a default policy, which may include random moves until a terminal node is reached.

4. **Backpropagation:** The simulation result is then propagated back through the traversed nodes, where the number of visits and the average reward for each node are updated, until it reaches the root.

2.4.2 Properties

MCTS has several characteristics that make it especially suitable for solving complex problems [3]:

- **Aheuristic:** Does not require domain-specific heuristics to be effective, which is specially useful in complex problems. While adding some domain knowledge can improve performance, MCTS can still yield good results with simple random simulations.
- **Anytime:** Can return a valid solution at any point during its execution. Since each iteration immediately updates the search tree, the current best move is always available.
- **Asymmetric:** Builds its tree asymmetrically, focusing more on promising areas of the search space. This leads to more efficient exploration and better use of computational resources.

2.5. Local Search

Local search is an optimization technique that iteratively improves a solution within a given search space by making small changes and retaining those that yield better results. It is effective for large-scale problems where the search is computationally expensive or unfeasible. Popular local search algorithms include HC, Simulated Annealing (SA), and Tabu Search (TS). While local search algorithms are powerful, they can sometimes get stuck and fail to find the global best solution. Techniques such as simulated annealing help escape local optima and explore a broader range of solutions.

2.5.1 Hill Climbing

HC is a local search optimization algorithm that begins with an initial solution, which can be randomly generated or specifically chosen depending on the problem. From this starting point, HC evaluates neighbouring solutions, which are generated by making small modifications to the current solution. If a neighbouring solution is found to be better than

the current one, the algorithm moves to that new solution. The algorithm finishes when there is no better neighbouring solution that offers an improvement over the current one, indicating that a local optimum has been attained.

While HC is effective and easy to implement, it has limitations. The algorithm can get stuck in:

- **Local optimum:** A solution state that is better than its neighbours but not necessarily the global optimum, i.e., the best possible solution in the entire search space, restricting further improvement.
- **Plateaus:** Mostly flat regions in the search space where neighbouring solutions have the same value, making it challenging for the algorithm to determine a direction of improvement.
- **Ridges:** A region in the search space where moving in all directions appears to lead downhill, and reaching a better solution often requires a sequence of non-improving moves, which HC does not explore.

To address these issues, variations of the algorithm have been developed:

- **Simple Hill-Climbing:** Evaluates one neighbour at a time and moves to the first improvement found, making it efficient but susceptible to local maxima.
- **Steepest-Ascent Hill-Climbing:** Evaluates all neighbours of the current state and chooses the best among them. This algorithm is a variation of the simple hill-climbing algorithm but takes more time.
- **Stochastic Hill-Climbing**: Randomly selects a neighbour, without evaluation, and moves to it if it improves the solution. This approach is less commonly used than the other two.

3. Related Work

The literature on the University Course Timetabling Problem (UCTTP) is vast, so this chapter will cover various approaches to address this problem, mostly based on surveys.

According to Lewis' survey [4], algorithms can be divided into three categories:

- **One-Stage** algorithms use a weighted sum function of constraints to identify solutions that satisfy both hard and soft constraints at the same time;
- **Multi-Stage** algorithms aim to optimize soft constraints while ensuring feasibility;
- **Multi-stage with relaxation** is divided into multiple stages, with some constraints being relaxed while satisfying others.

Other surveys, such as Abdipoor et al. [5], categorized UCTTP into five groups: operational research, metaheuristics, hyperheuristics, multi-objective, and hybrid approaches.

3.1. Operational Research (OR)

Operational Research (OR) techniques provide mathematically rigorous solutions. Despite their complexity, these methods are simple to implement and can be considered if time and memory constraints are not a concern [6]. However, if time is limited, the other types of approaches provide better quality results.

As Babaei et al. and Chen et al. detailed in their surveys [6, 7], OR includes Graph Coloring (GC), Integer and Linear Programming (IP/LP) and Constraint Satisfaction(s) Programming (CSPs) based techniques.

3.2. Metaheuristics

Metaheuristics *provide "acceptable" solutions in a reasonable time for solving hard and complex problems* [8].

Metaheuristics are similarly defined by Du et al. as *a class of intelligent self-learning algorithms for finding near-optimum solutions to hard optimization problems* [9].

Metaheuristics, while not guaranteeing global optimality, have become one of the most used solution strategy for UCTTP and can be classified into two categories: Single solution-based and Population-based.

3.2.1 Single Solution-Based

Single solution-based metaheuristics, also known as local search algorithms, focus on modifying a single solution throughout the search in order to improve that solution.

This metaheuristic approach includes algorithms such as Hill Climbing (HC), Simulated Annealing (SA), Tabu Search (TS), and Iterated Local Search (ILS). SA is perhaps the most effective single solution-based metaheuristic, particularly in benchmark datasets, but TS has also been shown to be successful in minimizing hard constraints [5].

3.2.2 Population-Based

Population-based meta-heuristics iteratively improve a population of solutions by generating a new population based on the selection, recombination and/or mutation of individuals in the current population. This method can be subdivided into Evolutionary Algorithms, such as Genetic Algorithms, and Swarm Intelligence, such as Ant Colony Optimization [5, 9].

Population-based metaheuristics, as opposed to single-solution-based metaheuristics, are more focused on exploration rather than exploitation [8, 9]. Despite promising results on real-world datasets, population-based approaches were rarely applied to benchmark datasets and performed poorly in International Timetabling Competition (ITC) competitions compared to single solution-based methods [5].

3.3. Hyperheuristics

Heuristic methods have been highly effective in solving a wide range of problems. However, their application to new problems is often challenging due to the vast number of parameter tuning and the lack of clear guidance [10]. To address this problem, hyperheuristics aim to generalize methods by selecting or combining the most suitable heuristic(s) for a specific problem, rather than explicitly solving the problem.

Hyperheuristic approaches are less common, possibly due to their performance [7].

3.4. Multi-Objective Optimization

Multi-objective or multi-criteria approaches aim to optimize multiple conflicting objectives simultaneously. These methods are frequently used to find the optimal Pareto front, which

is a set of compromise optimal solutions such that one cannot improve one objective without deteriorating the other. However, the disadvantage of this approach lies in the execution effort [7]. Several multi-criteria algorithms can also be included in other categories, such as metaheuristics.

3.5. Hybrid Approaches

Hybrid approaches mix algorithms from two or more of the previously mentioned types of approaches.

Hybrid approaches can indeed be divided into two main categories [5]:

- **Collaborative hybrids:** Involve running different algorithms sequentially, intertwined, or in parallel, sharing information without structural integration. This collaboration enables each component to focus on its area of the problem, with occasional communication guiding the overall search process.
- **Integrative hybrids:** Integrate components of different algorithms into a single framework, creating a cohesive and interdependent method.

In particular, Tomáš Müller, winner of the Second International Timetabling Competition (ITC-2007) tracks 1 and 3, developed a collaborative hybrid approach* for all three tracks [11], finding feasible solutions for all instances of tracks 1 and 3 and most instances of track 2.

The algorithms used included Iterative Forward Search (IFS), HC, Great Deluge (GD), and optionally SA. During the construction phase, IFS is employed to find a complete solution, followed by HC to find the local optimum. When a solution cannot be improved further using HC, GD is used to iteratively decrease, based on a cooling rate, a bound that is imposed on the value of the current solution. Optionally, SA can be used when the bound reaches its lower limit.

The solver's techniques have been integrated into the UniTime† system, which is widely used for scheduling in academic institutions. Furthermore, the principles behind Müller's hybrid methodology are still relevant in modern optimization methods.

*https://github.com/tomas-muller/cpsolver-itc2007
†https://www.unitime.org/

While Müller's approach was highly successful in 2007, subsequent research has revealed that other methods may outperform it in some instances. However, it remains a benchmark against which new approaches are often compared.

3.6. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) application to Combinatorial Optimization Problems (COPs) remains relatively unexplored. In the context of UCTTP, Goh's investigation [12] is the only known study that explores the potential of MCTS in solving the Post-Enrollment Course Timetabling (PE-CTT).

Goh's investigation employs a two-stage approach, first utilizing MCTS to find initial feasible solutions and then applying local search techniques to enhance the quality of these solutions.

The research introduces several enhancements to the standard MCTS algorithm, including heuristic-based simulations and tree pruning techniques, which improved the MCTS performance and effectiveness. The study also provides a comprehensive comparison of MCTS against traditional methods such as Graph Coloring heuristic and TS, with TS emerging as the most effective method.

One major limitation of MCTS for timetabling is, as highlighted by Goh, its rigorous decision-making process. Events are assigned sequentially and cannot be reassigned, limiting the algorithm's flexibility. In contrast, local search methods, such as TS, allow dynamic reassignment, enabling a more efficient exploration of the search space. Furthermore, the tree-based structure of MCTS restricts its hybridization with local search techniques, which have proven essential in achieving significant results in other learning-based algorithms. Another drawback of MCTS is its high computational cost, making it less effective under the strict time constraints imposed by timetabling problems.

This dissertation aims to demonstrate that, despite its limitations, MCTS still holds potential for timetabling problems with the appropriate modifications and hybridization.

3.7. Summary

We analyzed various surveys [4–7] and categorized UCTTP approaches based on their problem-solving strategies:

- Metaheuristics emerged as one of the most promising, particularly single-solution-based algorithms like SA and TS.
- Hybrid approaches, like Müller's ITC-2007 approach [11], were also shown to be effective.
- Operational research methods are simple to implement but are often outperformed due to scalability limitations.
- Hyperheuristics, while still emerging, aim to generalize methods by dynamically selecting among heuristics.
- Multi-objective methods optimize multiple conflicting objectives simultaneously, but they are computationally expensive.
- Goh [12] study on the PE-CTT found that TS outperformed MCTS, despite improvements to the standard MCTS algorithm.
- MCTS and its hybridization remain unexplored in the context of Curriculum-based Course Timetabling (CB-CTT), making this the novel contribution of our work.

4. Development

This chapter describes the development of the timetable system, formulating the problem, discussing the implementation of the Monte Carlo Tree Search (MCTS) and Hill Climbing (HC) hybrid approach, detailing the design choices, and algorithmic improvements made throughout the project.

The source code for the hybrid timetabling system is available at `https://github.com/DanielaTomas/FCUP-SCHEDULE/tree/main/schedule-backend/FlaskAPI/mcts/`. The repository includes detailed instructions for installation, configuration, and running the system.

4.1. Problem Formulation

The formulation presented is based on the model proposed in the Second International Timetabling Competition (ITC-2007) track 3. The entities in the problem are listed below:

- **Periods,** $P = \{P_0, P_1, ..., P_{|P|-1}\}$**:** Days are divided into fixed timeslots, with periods consisting of a day and a timeslot.
- **Rooms,** $R = \{R_0, R_1, ..., R_{|R|-1}\}$**:** Each room has a capacity and a type.
  - Capacity (number of seats)
- **Lecturers,** $L = \{L_0, L_1, ..., L_{|L|-1}\}$**:** We assume lecturers are pre-assigned to events.
  - Availability
- **Events,** $E = \{E_0, E_1, ..., E_{|E|-1}\}$**:** Represent the events to be scheduled (usually lectures). Each event has associated the following attributes:
  - Capacity (number of students);
  - Number of lectures;
  - Minimum working days (days over which the events of the same course should be spread);
  - Available periods;
  - Priority;
  - Lecturer

- **Blocks,** $B = \{B_0, B_1, ..., B_{|B|-1}\}$**:** A block may represent a group of related events. Usually represents all events from the same curriculum, which is a set of courses in a study program.

4.1.1 Assignment Attributes

Although not intrinsic properties of events, the following elements are essential in the assignment process:

- **Period (day, timeslot):** Each event is to be scheduled into exactly one period.
- **Room:** Each event must be assigned to a room that satisfies its capacity.

4.1.2 Constraints

There are several hard and soft constraints that affect the creation of timetables. The following hard and soft constraints were selected, drawing inspiration from those of ITC-2007:

Hard constraints:

- **H1:** Events belonging to the same course must be scheduled and must be assigned to distinct periods.
- **H2:** Two events can be scheduled in the same room, but only if they are in different periods.
- **H3:** Events of the same curricula, or taught by the same lecturer, must be scheduled in different periods.
- **H4:** If a lecturer is unavailable in a given period, then no events can be taught by this lecturer in that period.

Soft constraints:

- **S1:** Events of the same course should be spread into the given minimum number of days.
- **S2:** Events belonging to the same curriculum should be in consecutive periods.

- **S3:** The capacity of the room should be higher or equal than the capacity of the event.
- **S4:** All events of a course should be given in the same room.

The problem is typically defined as a minimization problem, with the algorithm attempting to minimize soft constraint violations while strictly adhering to hard constraints. However, in this implementation, the signals of the evaluation functions were inverted. As a result, the algorithm is designed as a maximization problem, where higher scores indicate better solutions.

4.2. Monte Carlo Tree Search

4.2.1 Search Space

The search space, S, for this problem is large as it involves a product of all possible event-period-room combinations:

$$S = E \times P \times R,$$

where E is the set of events, P is the set of periods (day, timeslot), and R is the set of rooms.

To reduce the search space, unavailable periods and unavailable rooms for a chosen period are discarded immediately in the MCTS $expansion$ phase.

4.2.2 MCTS Tree

Figure 4.1 illustrates the basic structure of the MCTS tree. The root represents the initial state, where no events have been assigned. The first level corresponds to assigning the first event, E_0, to various P and R. Each child node represents a specific assignment $((E_0, P_0, R_0), ..., (E_0, P_{|P|-1}, R_{|R|-1}))$. Subsequent levels correspond to the sequential assignment of events $E_1, E_2, ..., E_{|E|-1}$.

4.2.2.1 Tree Nodes

A node is composed of the following attributes:

- A **parent** is a reference to the parent node, allowing the tree structure to be navigated upwards. The root is the only node that does not have a parent.

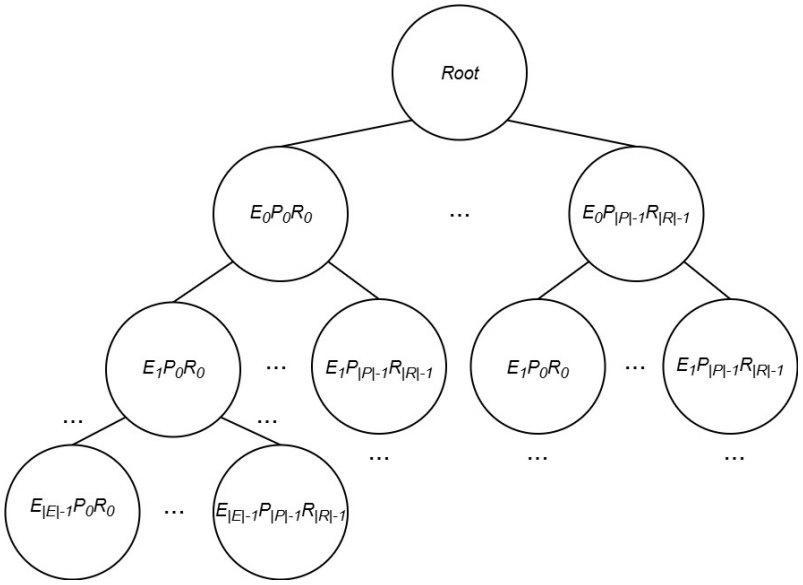

FIGURE 4.1: MCTS tree

- **Children** is initialized as an empty list at first but will later hold references to the node's child nodes.
- The **expansion limit** specifies the maximum number of children that a node can have, which helps control the expansion process by limiting dynamically how many possible event-period-room assignments can be explored from that node. The expansion limit is calculated based on the number of available rooms per period for the next event. A value of zero indicates that the node cannot be expanded further.
- **Visits** is initialized to 0, tracking the number of times a node has been visited.
- The **hard and soft scores** are both initialized to 0, representing the cumulative back-propagated hard and soft scores for a node, which are used to evaluate the quality of the node's state.
- The **hard and soft simulation results** represent the result of a single simulation performed for a node. These values are used as $best\_penalty\_n$ in the normalization equation (section 4.2.6 Equation 4.2) to scale results between 0 and 1.

4.2.3 Events Allocation

Events are sorted in advance so that the most difficult event to place is placed first in the tree. We propose to determine the priority of an event (Equation 4.1) based on:

- The difference between the number of lectures and minimum working days to prioritize events with more lectures spread over fewer days;

- The number of available periods, as having fewer available periods indicates more scheduling restrictions;
- The capacity of an event, which implies that events with greater capacity have higher priority;
- The number of curricula in which the event is, as events that appear in more curricula are more difficult to allocate.

If two or more events have the same priority score, they are ordered randomly.

The order of the events has a significant impact on the results. We tried other alternatives, but this one worked the best.

$$\begin{aligned} Priority = (\#lectures - \#min\_working\_days) \cdot 4 \\ -\#available\_periods \cdot 3 \\ +capacity \cdot 2 \\ +\#blocks \end{aligned} \tag{4.1}$$

4.2.4 Periods and Rooms Allocation

Initially, our approach to allocating periods and rooms in the MCTS tree followed a structured sequence. This method aimed to explore all possible (period, room) combinations based on the number of children of the node being expanded. However, as we progressed, this approach started to fail when the availability of rooms varied across different periods, leading to duplicate assignments.

To overcome these limitations more robustly, a refined approach was then developed. Instead of computing the (period, room) combinations via index math as we first thought, we pre-compute all valid combinations once. This method produces a flat list of unique and valid assignments.

Moreover, periods that are less frequently available across all events are given higher priority. In this way, the algorithm reduces the risk of future conflicts, guiding the tree toward less constrained decisions early. For each period, available rooms are also sorted by how closely their capacity matches the event's requirements. This reduces wasted space and improves flexibility in room usage.

4.2.5 Methodology

The MCTS algorithm consists of four phases: *selection*, *expansion*, *simulation*, and *backpropagation*. Each phase plays a critical role in exploring the search space and updating the tree structure with the simulation outcomes. Figure 4.2 provides a high-level overview of these four phases in the MCTS process.

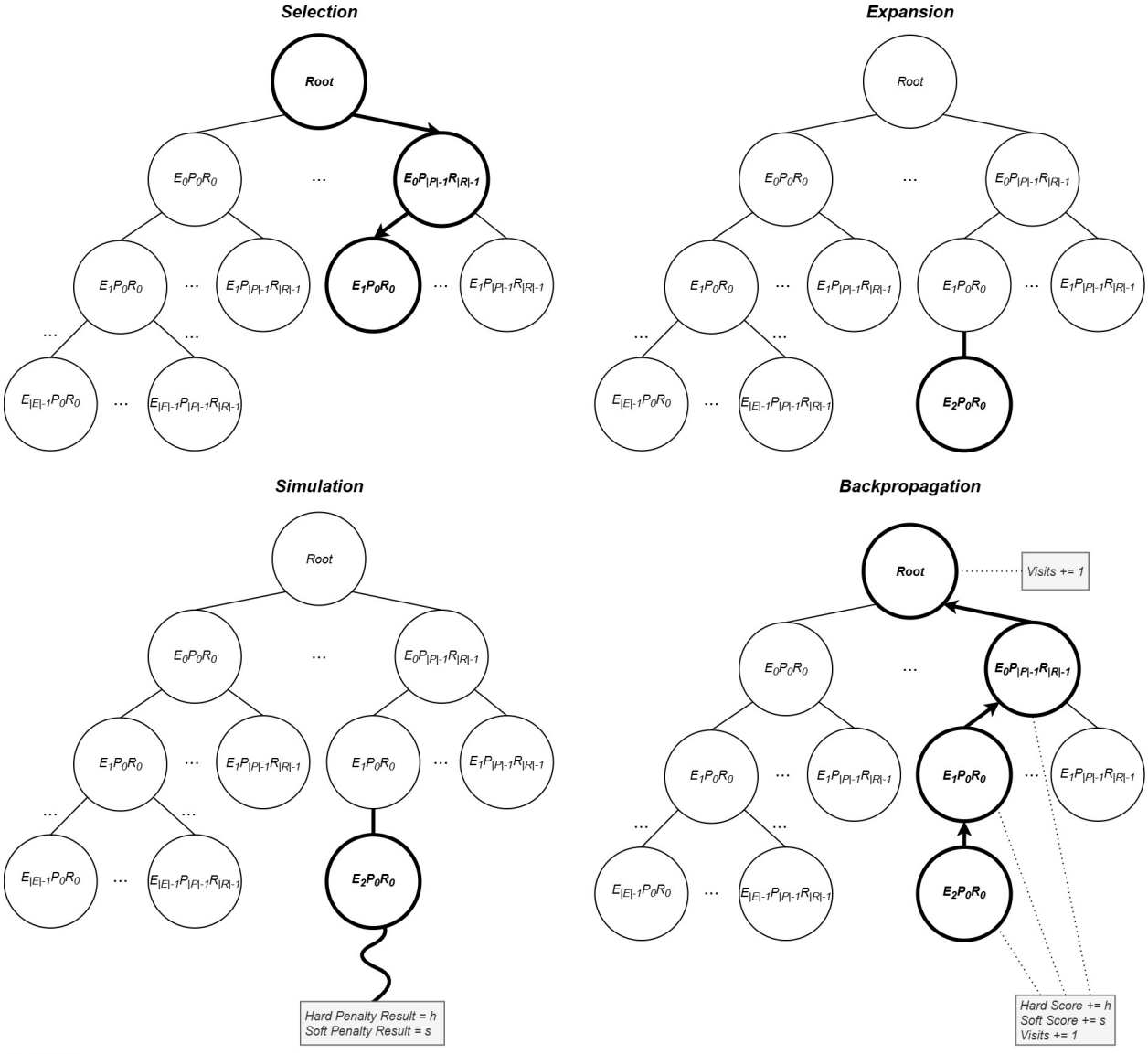

FIGURE 4.2: MCTS steps

4.2.5.1 Selection

The *selection* phase (Algorithm 1) begins at the root and traverses the tree by iteratively selecting the best child nodes according to Equation 2.1 until a terminal (leaf) node is

reached. A node is considered terminal if it cannot be expanded further (i.e., its expansion limit is zero). Conversely, a node is deemed fully expanded if either its expansion limit is zero or if the number of its children equals its expansion limit. If the traversal encounters a node that is not fully expanded, the selection stops, and that node is selected for further expansion.

The best child node (Algorithm 2) is determined by applying the Upper Confidence Bounds for Trees (UCT) formula (Equation 2.1). However, since the problem has two goals, minimizing hard constraints' violations first and then the soft constraints, the UCT formula was applied differently. Initially, the algorithm computes the weights for each child node using the hard score. Among the children with the highest hard score, the weights are recalculated based on the soft score. The child node with the highest weight in this second calculation is selected as the best child.

Algorithm 1 Selection

```
1:  function SELECTION
2:      Start from the root node
3:      while current node is not terminal do
4:          if current node has unexplored children then
5:              break
6:          else
7:              if no valid children remaining then
8:                  return False                          ▷ Fully expanded tree
9:              else
10:                 BEST_CHILD(children, c_param)
11:             end if
12:         end if
13:     end while
14:     Store the selected node for expansion
15:     return True
16: end function
```

Algorithm 2 Best Child

```
1: function BEST_CHILD(children, c_param = 1.4)
2:     Evaluate candidates based on hard constraints using UCT (Equation 2.1)
3:     Filter those with the highest hard score
4:     Among them, evaluate based on soft constraints using UCT (Equation 2.1)
5:     return child with the highest overall score
6: end function
```

4.2.5.2 Expansion

The $expansion$ phase (Algorithm 3) is responsible for expanding the node previously selected by assigning the next event to a valid (period, room) combination and creating a

corresponding child node.

The next event to schedule is determined by the depth of the current node. For the selected event, the algorithm gathers the available (weekday, timeslot) combinations and filters suitable rooms for each period (Algorithm 4), based on the event's capacity requirements and current room occupation.

A list of valid (period, room) combinations is generated. If no such combinations exist or the node has already been expanded for all possibilities, expansion is terminated, and the node is marked as fully expanded.

A combination is selected based on the number of already-expanded children.

Before proceeding, the algorithm evaluates the partial timetable (i.e., the new child) against a global hard penalty threshold. Since the problem is formulated as a maximization problem (i.e., higher hard scores indicate fewer constraint violations), this threshold defines the minimum acceptable hard score for a solution. If the resulting hard score for the partial timetable with the new assignment is lower than the threshold, the expansion stops for that branch. However, if the penalty exceeds or equals the threshold, the node remains in the tree. This pruning strategy was introduced to focus computational effort on promising and potentially feasible solutions. Notably, we use a threshold rather than immediately discarding all infeasible nodes because, in real-world timetabling problems, a fully feasible solution might not always be possible. Completely eliminating infeasible nodes would make the algorithm unsuitable for hard instances, as it would prevent the search from distinguishing between degrees of infeasibility.

If the expansion continues, the algorithm calculates the number of valid (period, room) combinations for the next event. This value is stored as the child's expansion_limit to guide future branching.

Finally, a new child node is created with the updated path and expansion limit. This new node is added to the tree, and the current node is updated to this newly created node.

4.2.5.3 Simulation

The $simulation$ phase estimates the value of each action, in this case, each event allocation, which will guide the selection and expansion steps in future iterations.

Algorithm 3 Expansion

```
function EXPANSION
    Identify the next event based on current node's depth
    FIND_AVAILABLE_ROOMS(...)  ▷ Retrieve available periods and corresponding valid rooms

    if no valid (period, room) combinations exist then
        Mark current node as fully expanded
        return False
    end if

    Select the next untried (period, room) combination
    Assign this to the event

    Evaluate partial assignment's hard score

    if hard score is below threshold then
        Pruning
        return False
    end if

    Create a new child node
    Add the child node to the current node's children
    Update current node to the newly created child
    return True
end function
```

Algorithm 4 Find Available Rooms

```
function FIND_AVAILABLE_ROOMS(event_capacity, rooms, events, available_periods)
    Initialize period_room_availability with all rooms available per period
    for each event already scheduled do
        Remove occupied rooms from the corresponding periods
    end for

    Identify rooms that satisfy the event's capacity requirement

    for each available period do
        if rooms are still available then
            Find intersection between available and suitable rooms
            if intersection is not empty then
                Sort intersected rooms by (room capacity - event capacity)
            else
                Sort available rooms by |room capacity - event capacity|
            end if
            Update period_room_availability with sorted list
        end if
    end for

    return period_room_availability
end function
```

A random event allocation approach was initially explored but produced suboptimal results. Therefore, a more structured method was implemented.

The *simulation* function (Algorithm 5) starts by retrieving the events already scheduled. It then sorts the remaining events, i.e., events that have not been assigned yet, based on two criteria:

1. Whether they were previously unassigned in prior simulations, ensuring that more critical events are allocated first.
2. Their priority (Equation 4.1), to prioritize harder-to-schedule events.

For each unvisited event, the algorithm searches for the best available period and room combination (Algorithm 6). This involves iterating through all the available periods and rooms, calculating the hard and soft penalties for each combination. During the calculation of the soft penalties, the importance of compactness (i.e., minimizing the time between related events) is adjusted, with a higher weight applied as the algorithm progresses, encouraging more compact schedules in later stages. The combination with no hard penalties and the lowest soft penalty score is selected as the best option. If multiple equally optimal choices exist, one is randomly selected. The event is then updated with the selected room, weekday, and timeslot. If no valid assignment is found, the event is left unassigned and its priority is increased for future simulations.

After scheduling all events, the algorithm evaluates the overall timetable, computing its hard and soft penalty scores. The best and worst penalty scores are then updated based on these results. If a new best solution is found, the timetable is saved to a file. Moreover, if a new best solution is found and the hard penalty is zero, indicating a feasible solution, the HC algorithm (section 4.3 Algorithm 8) is used to further optimize the timetable.

4.2.5.4 Backpropagation

The *backpropagation* phase (Algorithm 7) updates the path in the tree from the currently expanded node up to the root based on the simulation results.

It starts from the current node, which is the leaf node where the simulation occurred, and moves up until reaching the root node. During this process, it performs updates for each node in the path, including incrementing the number of visits and updating the hard and soft scores with the simulation results.

Algorithm 5 Simulation

```
1:  function SIMULATION(start_time, time_limit)
2:      Get current partial assignment
3:      Sort remaining events by unassignment history, priority, and randomness
4:      Initialize unassigned events set
5:
6:      for each remaining event do
7:          FIND_BEST_ROOM_AND_PERIOD()
8:          if assignment found then
9:              Assign event to selected period and room
10:         else
11:             Mark event as unassigned
12:         end if
13:     end for
14:
15:     Evaluate overall timetable to compute hard and soft penalties
16:     Update global penalty trackers
17:
18:     if new solution improves global best then
19:         Save new best hard and soft penalties
20:         Write best assignment to output file
21:         if all events assigned and hard penalty is zero but soft is non-zero then
22:             HILL_CLIMBING(...)
23:             Update global soft penalty
24:         end if
25:     end if
26:
27:     Normalize penalties
28:     return normalized hard and soft penalties
29: end function
```

4.2.6 Normalization

The normalization equation 4.2 is applied to both hard and soft constraints to standardize the simulation results [13]:

$$N(n) = \frac{e^a - 1}{e - 1}, \quad \text{with } a = \frac{best\_penalty_n - worst\_penalty}{best\_penalty - worst\_penalty}, \tag{4.2}$$

where $best\_penalty$ and $worst\_penalty$ represent the best and the worst simulation results in the entire tree, and $best\_penalty_n$ is the best simulation result under node n.

This formulation ensures that the best simulation result is mapped to a value close to one, while the worst is mapped to zero.

Algorithm 6 Find Best Room and Period

```
function FIND_BEST_ROOM_AND_PERIOD
    Initialize min_soft_penalty ← ∞
    Initialize candidates ← empty list
    Compute compactness_weight based on search depth

    for each (weekday, timeslot) in event's available periods do
        FIND_AVAILABLE_ROOMS(...)
        for each available room do
            Compute hard_penalty for this assignment
            if hard_penalty == 0 then
                Compute soft_penalty as:
                    room capacity penalty
                    + compactness_weight × compactness penalty
                    + minimum working days penalty
                    + room stability penalty
                if soft_penalty < min_soft_penalty then
                    Update min_soft_penalty
                    Set candidates to [(room, weekday, timeslot)]
                else if soft_penalty == min_soft_penalty then
                    Add (room, weekday, timeslot) to candidates
                end if
            end if
        end for
    end for
    if candidates not empty then
        return random choice from candidates
    else
        return None
    end if
end function
```

Algorithm 7 Backpropagation

```
procedure BACKPROPAGATION(simulation_result_hard, simulation_result_soft)
    Start at the current node
    while node is not None do
        Increment visit count
        Update hard score with simulation hard result
        Update soft score with simulation soft result
        Move to the parent node
    end while
end procedure
```

4.3. Hill climbing

The HC algorithm (Appendix 6 Algorithm 8) is applied as a local search method to further improve the timetable obtained from the MCTS simulation phase. This approach refines an initially feasible solution by iteratively exploring small modifications to enhance overall quality.

To maintain the structure established during the tree search, HC only modifies events that are not assigned in the tree. This ensures that the global search strategy remains intact while improving suboptimally placed events by the simulation phase.

4.3.1 Neighborhoods

After finding a feasible solution with MCTS, HC refines the solution by exploring neighboring states. The algorithm explores the solution space using six different types of neighbors inspired by the ITC-2007 track 3 winner [11]:

- **Period move:** Change the timeslot and weekday of an event while keeping the same room.
- **Room move:** Change the room of an event while keeping the same timeslot and weekday.
- **Event move:** Change both the room and timeslot of an event.
- **Room stability move:** Assign all events of the same course to the same room if possible.
- **Compactness move:** Move an event to a timeslot adjacent to another event of the same curriculum.
- **Minimum working days move:** Spread a course's events across more days to avoid teaching all lectures in a short period.

Each move has a weight, which affects how often it is chosen. Period, room and event moves appear to be the most effective, consequently they have a higher weight than the others.

4.3.2 Methodology

The HC approach follows the following steps:

1. Starts with an initial complete timetable with no hard constraints violated.
2. Repeatedly selects a random neighborhood move, based on the assigned weights.
3. Applies the move and evaluates the new timetable.
4. If the new timetable is better, it keeps the changes; otherwise, it reverts them.
5. Stops when:
    - A timetable satisfying all hard and soft constraints is found.
    - No improvements are identified after a predefined number of idle iterations.
    - The time limit is reached.

Algorithm 8 Hill Climbing

```
1: function HILL_CLIMBING(best_timetable, start_key, best_result_soft, start_time, time_limit)
2:     Define neighborhood moves with associated probabilities
3:     Initialize idle iteration counter
4:
5:     while within idle limit and time limit do
6:         Select a neighborhood move based on weighted probability
7:         Extract events not fixed in the tree (unscheduled events)
8:         Apply neighborhood move to generate a modified timetable
9:
10:        if no change was made then
11:            Increment idle counter and continue
12:        end if
13:
14:        Evaluate the modified timetable
15:        if result improves best soft score then
16:            Update best soft score and best timetable
17:            Write updated result to file
18:            if soft penalty is zero then
19:                return 0
20:            end if
21:            Reset idle counter
22:        else
23:            Revert changes and increment idle counter
24:        end if
25:
26:        Revert changes and increment idle counter
27:
28:    end while
29:
30:    return best soft result
31: end function
```

4.4. Diving

To improve the effectiveness of timetable generation, we also implemented a diving strategy in MCTS (Algorithm 9 and 10). This diving strategy enables the algorithm to follow and deepen promising paths, aiming to improve the convergence speed and solution quality.

Algorithm 9 Selection with Diving

```
1: procedure SELECTION
2:     if queue is not empty then
3:         DIVE()
4:     end if
5:
6:     ...                          ▷ Perform standard MCTS traversal from the root
7:
8: end procedure
```

Algorithm 10 Dive

```
1: function DIVE
2:     selected_node ← Retrieve next node from the queue
3:     simulation_path ← Retrieve associated simulation path
4:     if selected_node parent exists and is expandable then
5:         Set selected_node to the parent
6:         return True
7:     else if selected_node is fully expanded then
8:         Follow the simulation_path:
9:         for each scheduled assignment in the path do
10:            if a child of the node matches the assignment then
11:                Set selected_node to this child
12:                Set current node to this child
13:                Remove the assignment from the simulation_path
14:                return True
15:            end if
16:        end for
17:    else
18:        Set current node to selected_node
19:        return True
20:    end if
21: end function
```

4.4.1 Diving Approach Example

Figure 4.3 illustrates the MCTS process using the diving strategy over six iterations. Initially, the algorithm begins with the root node and an empty queue. Whenever the queue is empty, the UCT formula is applied to select a child for expansion as described in 4.2.5.

During the early MCTS iterations, child nodes $A1$, $A2$, and $A3$ are sequentially added under the root. Each of these nodes undergoes a simulation that yields a penalty score. In this example, we assume that there are no hard constraint violations; hence, the score directly reflects the soft constraint penalty.

While expanding the root, the algorithm tracks the node with the best simulation result and stores it in the queue, along with the corresponding complete timetable generated during the simulation. Therefore, in the first iteration, node $A1$ produces the best score and is therefore added to the queue. In the second iteration, node $A2$ does not outperform $A1$, so the queue remains unchanged. However, once the root is fully expanded in the third iteration, node $A3$ yields a better score than $A1$, resulting in $A1$ being replaced by $A3$ in the queue.

As a result, in the fourth iteration, node $A3$ is the only element in the queue and, is directly

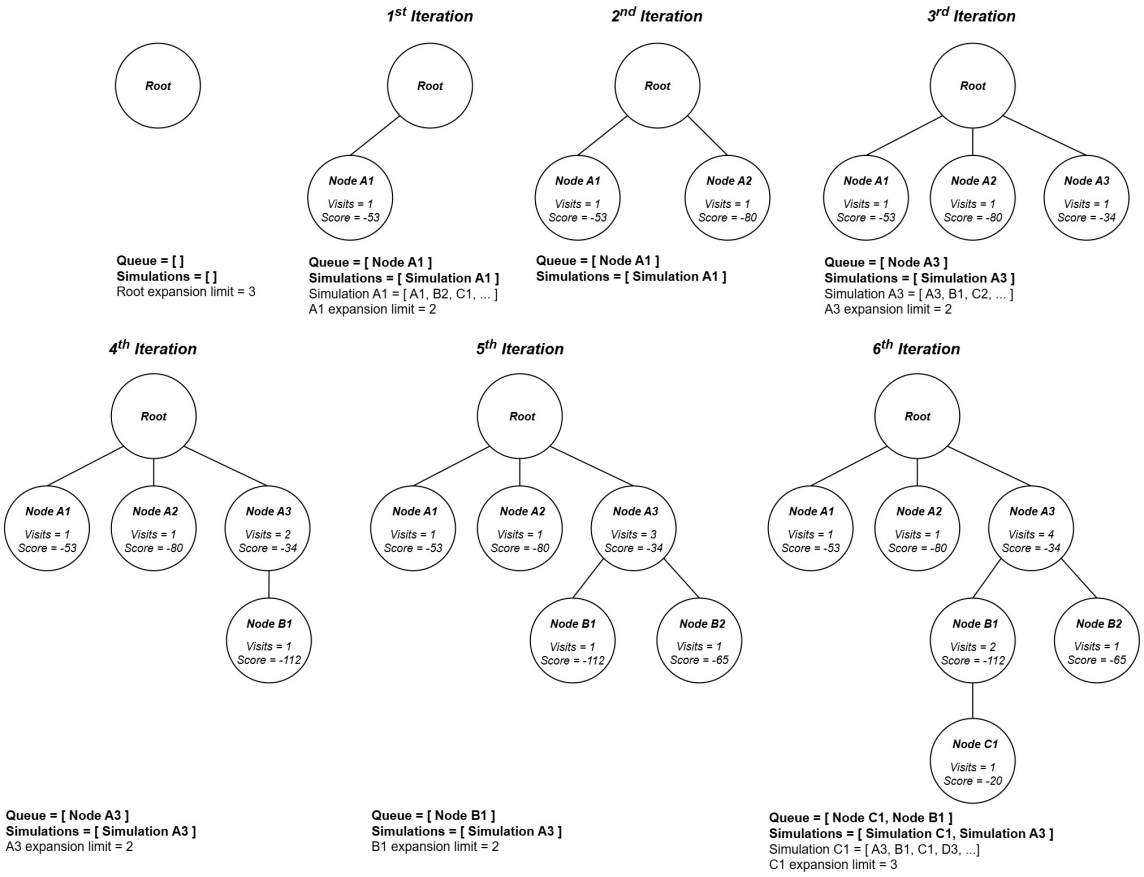

FIGURE 4.3: Diving approach

selected for further expansion. This continues through the fifth iteration. During the expansion process, whenever a child node is found with a better simulation result than the current best, it is placed at the front of the queue.

In the fifth iteration, although $A3$ is fully expanded, none of its children surpass its score of -34. Therefore, the next node selected is $B1$ because was part of the simulation path associated with $A3$. Upon expanding $B1$, the algorithm discovers node $C1$, which yields a better score than the previous best from $A3$. Consequently, $C1$ is placed at the front of the queue.

The expansion of $B1$ continues, but once $B1$ is fully expanded, the algorithm selects $C1$ for expansion next, since it now holds the highest priority in the queue.

5. Tests

This chapter outlines the numerous tests used to evaluate the performance of the final algorithm on instances from the Second International Timetabling Competition (ITC-2007) benchmark.

5.1. Test Instances

The evaluation used all 21 instances from the ITC-2007 track 3 dataset (`comp01.ctt` to `comp21.ctt`). Each instance includes detailed information about about curricula, courses, rooms, unavailability constraints, lectures, periods and lecturers. The key features of each instance are summarized in Table 5.1.

Instance	Curricula	Courses	Rooms	Constraints	Lectures	Periods	Lecturers
comp01	14	30	6	53	160	30	24
comp02	70	82	16	513	283	25	71
comp03	68	72	16	382	251	25	61
comp04	57	79	18	396	286	25	70
comp05	139	54	9	771	152	36	47
comp06	70	108	18	632	361	25	87
comp07	77	131	20	667	434	25	99
comp08	61	86	18	478	324	25	76
comp09	75	76	18	405	279	25	68
comp10	67	115	18	694	370	25	88
comp11	13	30	5	94	162	45	24
comp12	150	88	11	1368	218	36	74
comp13	66	82	19	468	308	25	77
comp14	60	85	17	486	275	25	68
comp15	68	72	16	382	251	25	61
comp16	71	108	20	518	366	25	89
comp17	70	99	17	548	339	25	80
comp18	52	47	9	594	138	36	47
comp19	66	74	16	475	277	25	66
comp20	78	121	19	691	390	25	95
comp21	78	94	18	463	327	25	76

TABLE 5.1: Key features of the ITC-2007 track 3 instances.

All instances guarantee at least one feasible solution, which means that in the optimum there are no violations of hard constraints. Additionally, to validate the correctness of a solution, the competition organizers provided a C++ source code that identifies the type of violation and calculates the corresponding penalty.

5.2. Test Setup

The tests were conducted for different values of the parameter C (Equation 2.1), which affects the exploration-exploitation trade-off in the Monte Carlo Tree Search (MCTS) algorithm. A higher C value increases the weight of the second term, allowing the algorithm explore less-visited nodes with more intensity. On the other hand, a lower C value favors nodes with a higher average reward, resulting in increased exploitation of known favorable choices. Specifically, we tested the algorithm with some values of C ranging from 0.1 to 1000 (0.1, 0.2, 0.5, 1, 1.4, 2, 5, 10, 20, 50, 100, 200, 500, 1000).

Additionally, an alternative Upper Confidence Bounds for Trees (UCT) formula was evaluated, modifying the exploitation term to use the accumulated reward instead of the average (Equation 5.1), giving an advantage to the nodes that were exploited first.

$$UCT = w_i + 2C\sqrt{\frac{2\ln n}{n_i}}, \tag{5.1}$$

where w_i is the total reward of all playouts through this state, n_i is the number of visits of child node i, C is a constant (typically $\sqrt{2}$ in game-playing scenarios), and n is the number of visits of the parent node.

For optimization problems, the value of C is problem-dependent and usually needs to be empirically tuned rather than set to $\sqrt{2}$ [2].

In the Hill Climbing (HC) algorithm, the parameter HC_IDLE was set to 5000, indicating the maximum number of consecutive iterations allowed without improvement before the search is terminated.

5.3. Performance Metrics

Each test corresponds, therefore, to a different C value, and the performance measurements are recorded for 21 problem instances, denoted as `comp01` to `comp21`. For each instance, the method iterates numerous times until a stopping condition is met, which is in this experiment a time constraint.

For each test, the following key performance metrics were considered:

- **Best hard penalty:** The lowest number of hard constraint violations found during the execution of the algorithm for a given instance. A value of zero indicates a feasible solution that satisfies all hard constraints.

- **Worst hard penalty:** The highest number of hard constraint violations found during the execution of the algorithm for a given instance.
- **Best soft penalty:** The lowest soft constraint penalty achieved during the search process for a given instance.
- **Worst soft penalty:** The highest soft constraint penalty achieved during the search process for a given instance.
- **Iteration of best solution:** The specific iteration at which the best solution (in terms of hard or soft penalties) was found.
- **Total number of iterations:** The overall count of iterations performed during the algorithm's execution for each problem instance.
- **Time to best solution:** The elapsed time required to reach the best solution for a given instance.

5.4. Testing Procedure

To thoroughly evaluate the algorithm's robustness and consistency, two different testing approaches were employed:

1. **1-hour and 24-hours runs:** The algorithm was executed for 1 hour and 24-hours to assess its ability to converge towards high-quality solutions given sufficient runtime.
2. **10-minute runs:** The algorithm was run for 10 minutes using ten different random generator seeds (1-10). Using fixed seeds ensured that the results could be reproduced, allowing for a fair comparison of different parameter settings and configurations. This approach also helped to evaluate the consistency and variability of results across different initializations.

6. Results

This chapter presents the main outcomes of the proposed Monte Carlo Tree Search (MCTS)-based approach for Curriculum-based Course Timetabling (CB-CTT). The following sections explore aspects such as computational performance, feasibility of generated timetables, analysis of the exploration constant C, the impact of pruning, effectiveness of the diving strategy, and a comparison against benchmark competition results.

6.1. Python vs PyPy Performance

During the development and testing of the method, both standard Python and PyPy* were evaluated for performance. While both correctly executed the MCTS algorithm, the performance difference was substantial. PyPy called over three times more functions compared to Python.

Given the compute-intensive nature of MCTS, PyPy was ultimately chosen for running all the experiments and benchmarks.

6.2. Feasibility

One of the fundamental requirements of CB-CTT is the ability to produce feasible schedules, i.e., timetables that strictly adhere to hard constraints such as avoiding room clashes, overlapping events, and unavailable periods.

Throughout all experiments, the proposed method consistently produced feasible solutions across all tested datasets from the Second International Timetabling Competition (ITC-2007) benchmark. This demonstrates the robustness of the method's constraint handling mechanisms, regardless of instance size or structural complexity, confirming the general applicability of the approach.

6.3. Random Simulation Performance

To establish a baseline for the effectiveness of guided search, a random simulation test was conducted. In this configuration, event, period, and room assignments were selected at random, without considering constraint satisfaction or search tree statistics. The goal of this experiment was to evaluate the ability of a purely stochastic method to find feasible solutions and to compare its performance against the guided MCTS approach.

*A Just-In-Time (JIT) python compiling alternative. Link: `https://pypy.org/`

Across multiple runs, as expected, the random simulation consistently failed to produce feasible solutions (Table 6.1). Although random simulations can execute a high number of iterations due to its simplicity and lack of tree maintenance overhead, the majority of these iterations were ineffective, repeatedly exploring infeasible parts of the search space.

These results highlight the importance of incorporating domain knowledge information to ensure feasibility in a realistic time frame.

6.4. C Parameter Behaviour

The exploration constant C in the Upper Confidence Bounds for Trees (UCT) formula (Formula 2.1) was varied across several orders of magnitude (from 0.1 to 1000), including the modified variant incorporating accumulated rewards (Equation 5.1).

Surprisingly, the results showed that varying C had minimal impact on solution quality, which indicates a lower-than-expected sensitivity to node selection. No clear or consistent pattern emerged across different benchmark instances or configurations.

6.5. Pruning

The application of pruning during MCTS plays a crucial role in the efficiency of the search process.

Figure 6.1 illustrates the behavior of the method without pruning. The search frequently explores infeasible branches of the tree, which is evident from the large number of iterations with hard constraint violations. Although some feasible solutions are eventually discovered, the overall search efficiency is compromised due to the continuous evaluation of solutions that will be ultimately discarded.

Conversely, when pruning is applied (Figure 6.2), the search is restricted to branches that satisfy hard constraints. As a result, the hard constraint value remains consistently at zero throughout the iterations, reflecting that pruning effectively excludes infeasible regions from the search space. This restriction allows MCTS to focus on high-quality and valid solutions, preventing wasteful exploration. The cleaner progression and stable constraint satisfaction not only accelerate convergence but also improve the overall robustness of the approach.

While not depicted in Figure 6.2, it is worth noting that simulations may still experience occasional hard constraint violations because pruning is applied to branches within the tree rather than across the entire search space.

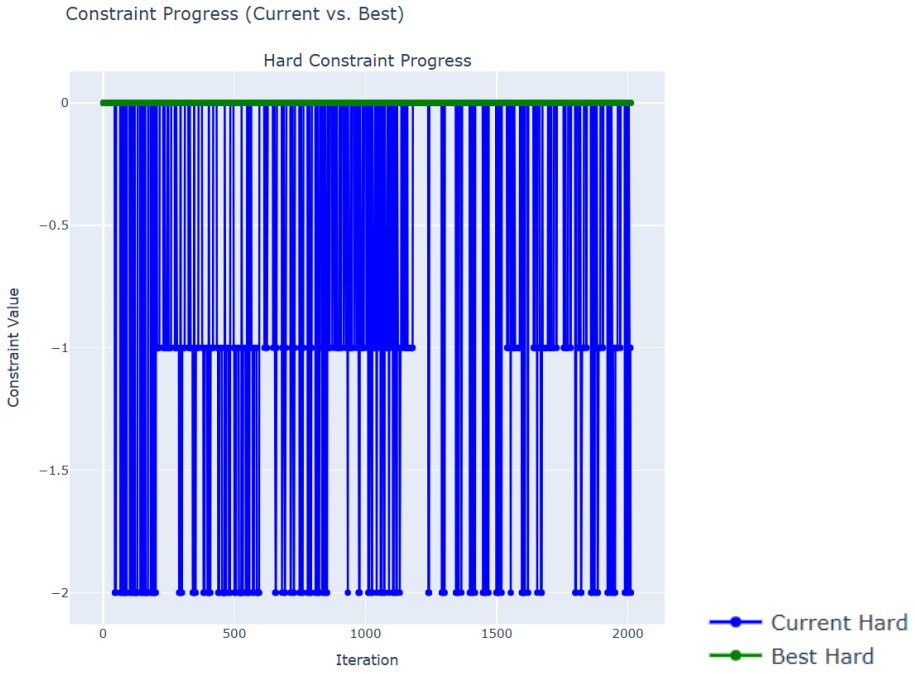

FIGURE 6.1: Hard constraint progress without pruning during 10 minutes on the `comp01` instance from ITC-2007.

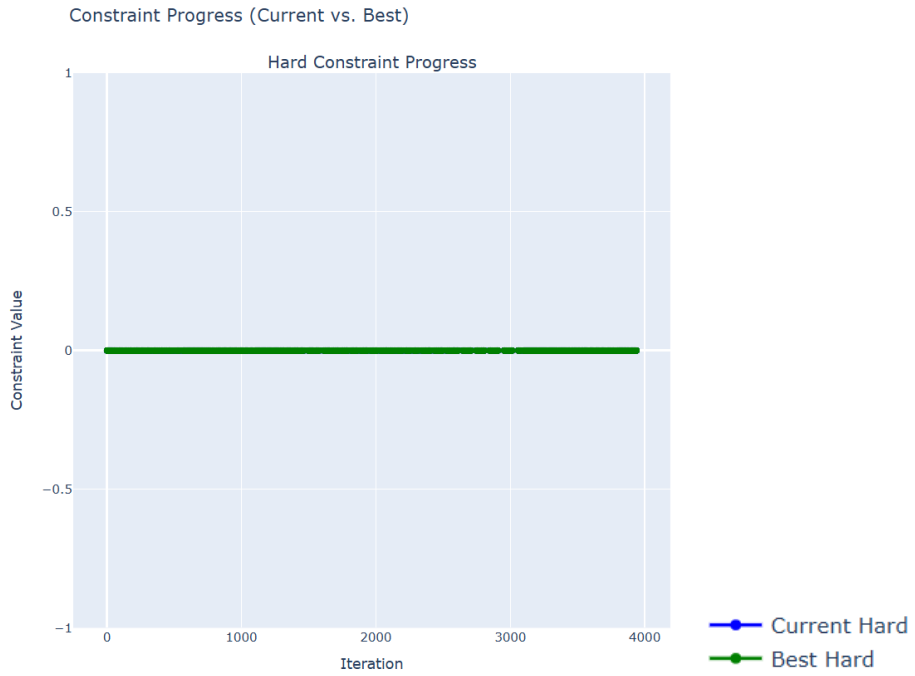

FIGURE 6.2: Hard constraint progress with pruning during 10 minutes on the `comp01` instance from ITC-2007.

6.6. Hill Climbing

To improve solution quality, Hill Climbing (HC) was combined with the MCTS algorithm (chapter 4 section 4.3). Importantly, HC operates exclusively on complete feasible solutions returned by MCTS, ensuring that hard constraint validity is maintained throughout the optimization process.

However, the integration of HC did not consistently yield lower soft constraint penalties across all tested instances within the 10-minute time limit (Table 6.1). In some cases, the baseline MCTS approach and the version with the diving strategy achieved comparable or even superior results.

Despite the results and the additional computational overhead introduced by HC, it still demonstrates potential to provide meaningful improvements in solution quality.

6.7. Diving

To improve the convergence speed and solution quality during timetable generation, the MCTS algorithm incorporates a diving strategy (chapter 4 section 4.4). Moreover, the diving strategy complements pruning by directing attention to already feasible branches. This method helps concentrate computational resources on areas of the search space that are both valid and high quality.

Figures 6.4 and 6.3 show the soft constraint penalty progression over time on the `comp01` instance from the ITC-2007 benchmark, using and not using the diving strategy.

The results demonstrate that the diving mechanism contributes positively to the search effectiveness:

- **Faster convergence:** The algorithm reaches lower penalty regions more rapidly compared to a standard breadth-oriented MCTS approach. By tracking and expanding the most promising nodes first, the diving queue helps avoid wasting iterations on less promising parts of the tree.
- **Improved solution quality:** As observed in the soft penalty trend, the use of diving leads to an improvement compared to the approach without diving (Figure 6.3). This improvement is only noticeable when we extend the run time. This shows that the diving strategy not only accelerates convergence but also contributes to reaching higher-quality solutions.

- **Focused exploitation:** The queue-based selection mechanism ensures that the algorithm deepens the most promising paths. In practice, this limits unnecessary expansion of under-performing nodes and encourages building upon solutions with known potential.

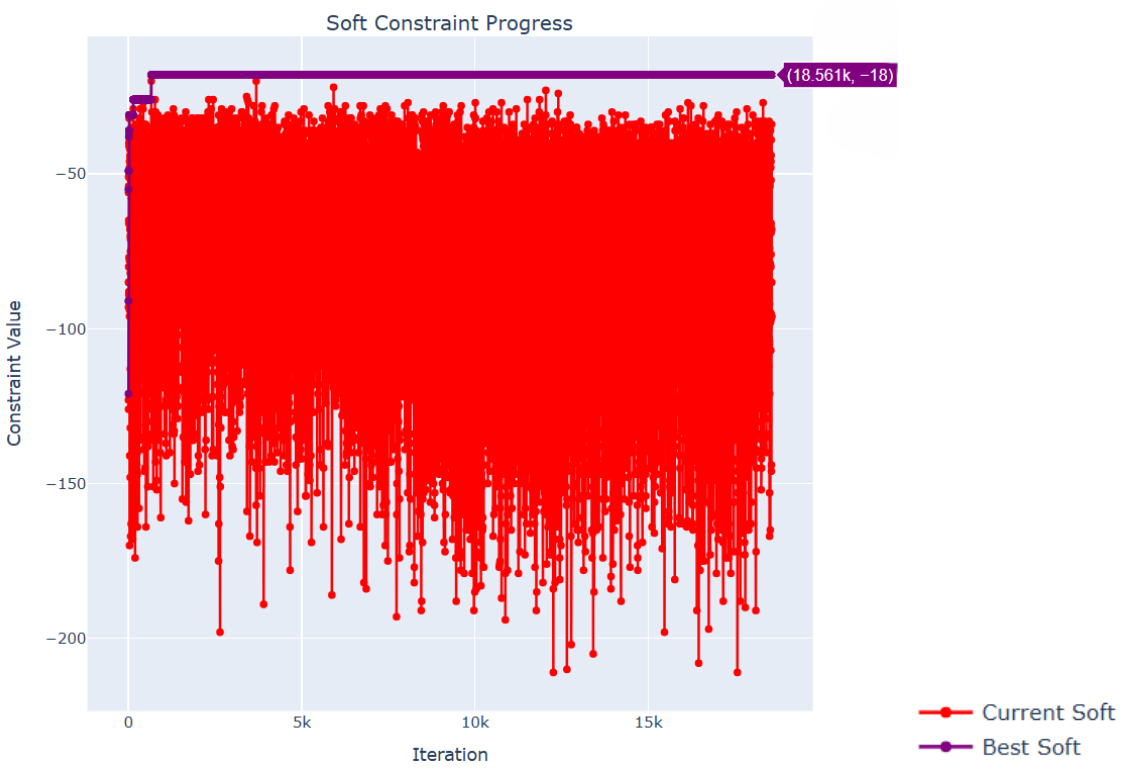

FIGURE 6.3: Soft constraint progress without using the diving approach on the `comp01` instance from ITC-2007 for 1 hour and C=1.4.

In both approaches (with and without diving) it is noticeable that the algorithm eventually stagnates, reaching a point where no further improvement is observed. After an initial phase of rapid improvement, the soft penalty score typically plateaus without finding an optimal solution. However, with the diving strategy, the soft constraint penalties at each iteration tend to remain closer to the best solution found, indicating a more stable and focused search trajectory.

Although the diving strategy yielded improved convergence, it may also reveal a potential drawback in the balance between exploration and exploitation. By aggressively deepening promising paths, the algorithm might prematurely narrow its search space, reducing its ability to discover alternative high-quality regions. This behavior suggests that while diving is effective in focusing the search, it might also limit broader exploration.

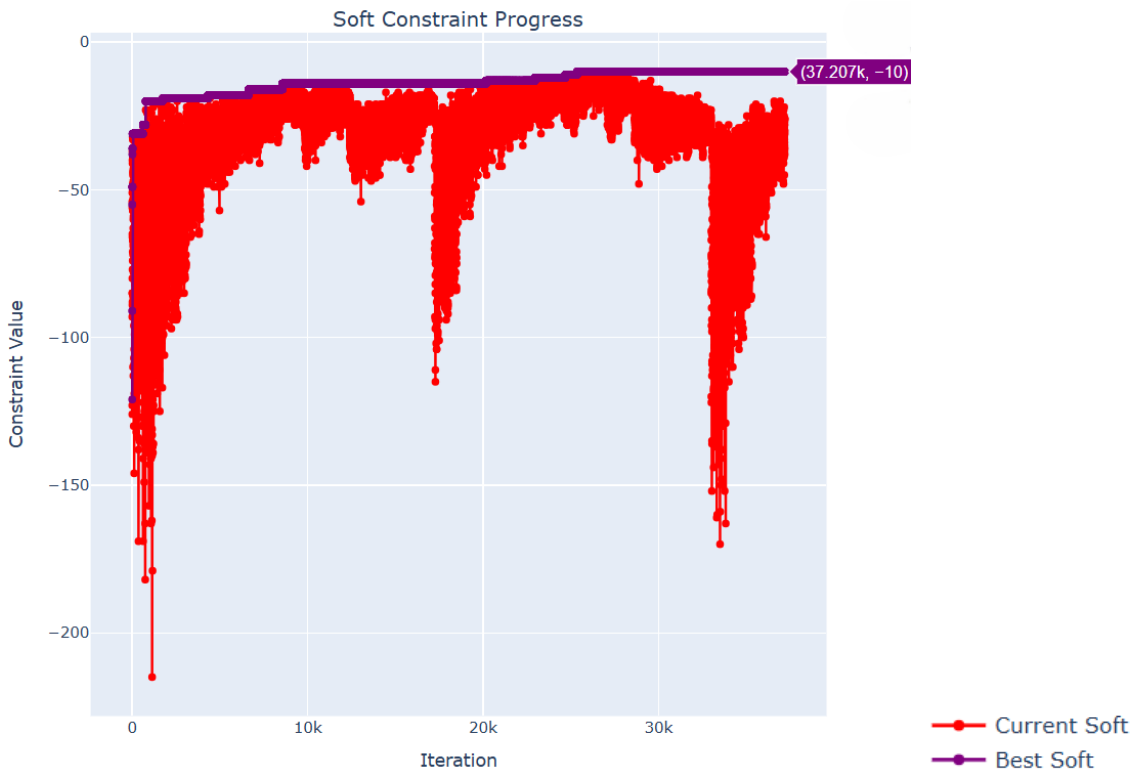

FIGURE 6.4: Soft constraint progress using the diving approach on the `comp01` instance from ITC-2007 for 1 hour and C=1.4.

6.8. Extended Runtime Evaluation

To determine whether the proposed approach benefits from longer execution times, additional experiments were conducted with extended time limits (1 hour and 24 hours) on selected benchmark instances. These experiments aimed to evaluate the long-term convergence behavior of the different algorithm variants.

The results showed that the algorithm tended to plateau early, but it slightly reduced soft constraint penalties. The number of iterations remained relatively low. With even longer times, we could have had better outcomes.

6.9. Competition Results Comparison

The effectiveness of our approach was evaluated on the official ITC-2007 track 3 benchmark datasets. Table 6.1 presents a comparison between the best results submitted by Müller (competition winner) and our approach (with C=0.1) for instances `comp01.ctt` to`comp05.ctt`, and `comp11.ctt`. Initially, all benchmark instances were considered for evaluation, but due to time constraints, the experiments were narrowed down to focus on only six instances. We chose the first five cases simply because they were the most often

utilized and thus the ones we became most familiar with. The `comp11.ctt` instance was chosen since it produces an optimal solution.

Although our approach does not yet outperform state-of-the-art methods in terms of soft constraint minimization, even with an extended time, it consistently finds feasible solutions within reasonable time limits. This confirms its robustness and potential as a foundation for further refinement.

It is also important to note that the differences in soft constraint penalties across the various method variants are small. This limited variability may be attributed to the small number of observations per instance and the fixed 10-minute time budget allocated to each run. Under such constraints, the first solution found by the algorithm can have a significant influence on the final outcome, as the MCTS has limited opportunity to explore alternative branches, resulting in relatively few iterations.

This effect may also help explain why HC and diving, although designed to improve solution quality, do not consistently outperform the base MCTS variant. Nonetheless, it was consistently observed that variants incorporating the diving strategy perform a greater number of iterations within the same time frame. This suggests that diving enhances search efficiency, even if the resulting improvement in solution quality is not always substantial.

Instance	Müller			Random Simulation			MCTS			MCTS+ HC			MCTS+Diving			MCTS+ HC+Diving		
	avg	max	min	avg	max	min	avg	max	min	avg	max	min	avg	max	min	avg	max	min
comp01	5 (0)	5 (0)	5 (0)	2136.8 (96.1)	2469 (99)	1844 (91)	23.2 (0)	26 (0)	17 (0)	21.5 (0)	24 (0)	18 (0)	18 (0)	22 (0)	12 (0)	17.4 (0)	23 (0)	12 (0)
comp02	61.3 (0)	70 (0)	51 (0)	8099.1 (231.9)	9351 (236)	6909 (228)	201.4 (0)	213 (0)	194 (0)	197.1 (0)	208 (0)	184 (0)	213.1 (0)	227 (0)	190 (0)	208.2 (0)	232 (0)	186 (0)
comp03	94.8 (0)	103 (0)	84 (0)	5941.1 (184.5)	6575 (187)	5299 (182)	243.5 (0)	253 (0)	238 (0)	243.5 (0)	254 (0)	227 (0)	251 (0)	262 (0)	243 (0)	241.6 (0)	260 (0)	215 (0)
comp04	42.8 (0)	48 (0)	37 (0)	5168.2 (179.8)	5552 (183)	4407 (176)	147.8 (0)	154 (0)	136 (0)	142 (0)	152 (0)	129 (0)	149.2 (0)	159 (0)	142 (0)	140.9 (0)	151 (0)	127 (0)
comp05	343.5 (0)	379 (0)	330 (0)	8752.1 (121.6)	9482 (125)	7922 (118)	657.7 (0)	685 (0)	621 (0)	665.7 (0)	702 (0)	638 (0)	633.3 (0)	700 (0)	546 (0)	639.6 (0)	766 (0)	587 (0)
comp11	0 (0)	0 (0)	0 (0)	2033.8 (68.8)	2211 (72)	1826 (64)	11.6 (0)	14 (0)	10 (0)	9.2 (0)	12 (0)	6 (0)	12.6 (0)	14 (0)	11 (0)	9.9 (0)	13 (0)	7 (0)

TABLE 6.1: Comparison of Müller and our method variants across six benchmark instances during 10 minutes and run 10 times. Soft constraint penalties are shown with hard constraint violations in parentheses.

6.10. Summary

This chapter evaluated the approach across several dimensions: implementation performance, feasibility, parameter tuning, pruning impact, and solution quality. Key findings include:

- **Feasibility:** The method consistently generated feasible timetables in the challenging ITC-2007 set of benchmark instances.
- **Performance:** PyPy significantly outperformed standard Python, justifying its use in all experiments.
- **Random Simulation:** Purely random simulations failed to produce feasible solutions, emphasizing the necessity of guided search and domain knowledge to navigate the complex solution space effectively.
- **Pruning:** Efficiently eliminated some infeasible branches, improving search focus and convergence.
- **Hill Climbing:** The inclusion of HC did not consistently outperform the base MCTS or diving versions within the 10-minute budget, but it demonstrated potential to refine certain solutions.
- **Diving:** Diving successfully reduced soft constraint penalties after MCTS found a feasible solution.
- **Competition Benchmarking:** Although solution quality was below the best-known results, the approach shows solid potential with room for improvement.

7. Conclusion

The proposed approach presents a novel application of Monte Carlo Tree Search (MCTS) to University Course Timetabling Problem (UCTTP), combined with Hill Climbing (HC) for local improvements. While MCTS has seen limited application in UCTTP, this dissertation demonstrates its potential.

The integration of HC significantly improves solution quality by exploiting feasible regions more effectively.

An additional innovation is the incorporation of a diving strategy into the MCTS process. This mechanism allows the algorithm to focus more deeply on promising branches of the search tree, accelerating convergence and reducing unnecessary exploration.

However, our findings indicate that both standard and diving-based MCTS tend to plateau after a certain number of iterations, with diminishing returns even under extended run times. While diving produces more consistent improvements and tends to stay closer to the best-found solutions, the problem of long-term stagnation remains and represents a key area for future optimization.

Our experimental results, including comparisons against the benchmark dataset from the ITC-2007 competition, show that the hybrid approach consistently produces feasible solutions. Although it does not yet outperform state-of-the-art solutions, this dissertation provides an effective and adaptive scheduling process for Faculty of Sciences of the University of Porto (FCUP) and can be extended to other institutions and help in other studies.

7.1. Future Work

Future work may focus on fine tuning the algorithm, improving the heuristic functions, enhancing computational performance, and integrating a visualization tool.

7.1.1 Performance

One promising direction is to postpone room allocation during both tree construction and simulation. In this approach, the fixed nodes in the tree represent only the $(weekday, timeslot)$ (Appendix 6 Figure 1) and, in the simulation, only allocate these two attributes for each

event. Room assignments are performed afterward, once the simulation reaches a complete assignment of periods. Exploring this variant could help reduce computational overhead and allow a deeper or broader search within the same time constraints.

However, this approach might be less effective when the number of available rooms is limited. In such cases, delaying room assignments could lead to difficulties in finding feasible allocations later in the simulation.

Although early testing of this approach did not yield strong results, it remains a promising idea that deserves further investigation and refinement.

7.1.2 Expand Hard and Soft Constraints

Expanding the set of hard and soft constraints may also be important. For instance, additional hard constraints could enforce requirements such as assigning specific room types based on the nature of the class (e.g., labs requiring specialized equipment), while soft constraints could aim to give both lecturers and students a free period during lunchtime. Integrating such constraints would make the system more aligned with the practical needs and expectations of real-world academic environments.

7.1.3 Visualization tool

A previous project* developed a timetabling visualization tool using Flask and reactive programming principles with the Elm language. Flask facilitated data management and communication between the front-end and the database (Appendix 6 Figure 3), ensuring that timetable updates were efficiently processed and displayed.

This tool allows users to manually construct and modify schedules while providing a responsive interface (Appendix 6 Figure 2). However, despite its usability, the tool has some limitations that this dissertation attempted to address:

- **No conflict detection support:** Users had to manually check for conflicts, increasing the risk of errors.
- **Lack of automated guidance:** It did not provide recommendations or suggestions to help users make optimal scheduling decisions.
- **No quality assessment:** The system lacked mechanisms to evaluate the quality of a given timetable.

*__Front-end:__ `https://github.com/luismdsleite/schedule` __Back-end:__ `https://github.com/luismdsleite/schedule-backend`

The development process involved several adaptations and refinements. Initially, the previous work was analyzed and made functional, serving as a baseline for the new system.

The MCTS algorithm was being developed while also being integrated into the front-end interface intended to showcase the project. However, as the focus shifted towards adapting it to the Second International Timetabling Competition (ITC-2007) track 3 benchmark for performance evaluation and fine-tuning the algorithm, the front-end development gradually lost priority. As the algorithm evolved and diverged significantly from its original form, aligning it with the prior work proved difficult. Consequently, continuing its development would be time-consuming and ultimately not worthwhile, as it was primarily intended for demonstration purposes. Instead, the system should be directly integrated into the sigarra* website.

*https://sigarra.up.pt/up/pt/web_page.inicial

References

[1] U. D. Porto, “Fcup em números.” [Online]. Available: https://www.up.pt/fcup/pt/a-fcup/institucional/fcup-em-numeros/ [Cited on page 1.]

[2] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk, “Monte carlo tree search: a review of recent modifications and applications,” *Artificial Intelligence Review*, vol. 56, 07 2022. [Cited on pages 7 and 33.]

[3] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, “A survey of monte carlo tree search methods,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012, conference Name: IEEE Transactions on Computational Intelligence and AI in Games. [Online]. Available: https://ieeexplore.ieee.org/document/6145622/?arnumber=6145622 [Cited on pages 7 and 8.]

[4] R. Lewis, “A survey of metaheuristic-based techniques for university timetabling problems,” *OR Spectrum*, vol. 30, no. 1, pp. 167–190, Jan. 2008. [Online]. Available: https://doi.org/10.1007/s00291-007-0097-0 [Cited on pages 10 and 13.]

[5] S. Abdipoor, R. Yaakob, S. L. Goh, and S. Abdullah, “Meta-heuristic approaches for the university course timetabling problem,” *Intelligent Systems with Applications*, vol. 19, p. 200253, Sep. 2023. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2667305323000789 [Cited on pages 10, 11, and 12.]

[6] H. Babaei, J. Karimpour, and A. Hadidi, “A survey of approaches for university course timetabling problem,” *Computers & Industrial Engineering*, vol. 86, pp. 43–59, Aug. 2015. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0360835214003714 [Cited on page 10.]

[7] M. C. Chen, S. N. Sze, S. L. Goh, N. R. Sabar, and G. Kendall, “A survey of university course timetabling problem: Perspectives, trends and opportunities,” *IEEE Access*, vol. 9, pp. 106 515–106 529, 2021, conference Name: IEEE Access. [Online]. Available: https://ieeexplore.ieee.org/document/9499056/?arnumber=9499056 [Cited on pages 10, 11, 12, and 13.]

[8] E. Talbi, “Metaheuristics: From design to implementation,” *John Wiley & Sons google schola*, vol. 2, pp. 268–308, 2009. [Cited on pages 10 and 11.]

[9] K.-L. Du, M. Swamy *et al.*, *Search and optimization by metaheuristics*. Springer, 2016, vol. 1. [Cited on pages 10 and 11.]

[10] E. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, “Hyper-heuristics: A survey of the state of the art,” *Journal of the Operational Research Society*, vol. 64, pp. 1695–1724, 07 2013. [Online]. Available: https://www.researchgate.net/publication/256442073_Hyper-heuristics_A_survey_of_the_state_of_the_art [Cited on page 11.]

[11] T. Müller, “Itc2007 solver description: a hybrid approach,” *Annals of Operations Research*, vol. 172, no. 1, pp. 429–446, Nov. 2009. [Online]. Available: http://link.springer.com/10.1007/s10479-009-0644-y [Cited on pages 12, 14, and 27.]

[12] S. L. Goh, “An investigation of monte carlo tree search and local search for course timetabling problems,” *University of Nottingham, Malaysia*, pp. 76–105, Jul. 2017. [Online]. Available: https://eprints.nottingham.ac.uk/43558/ [Cited on pages 13 and 14.]

[13] J. P. Pedroso and R. Rei, “Tree search and simulation,” in *Applied Simulation and Optimization: In Logistics, Industrial and Aeronautical Practice*, M. Mujica Mota, I. F. De La Mota, and D. Guimarans Serrano, Eds. Cham: Springer International Publishing, 2015, pp. 109–131. [Online]. Available: https://doi.org/10.1007/978-3-319-15033-8_4 [Cited on page 25.]

Appendix A

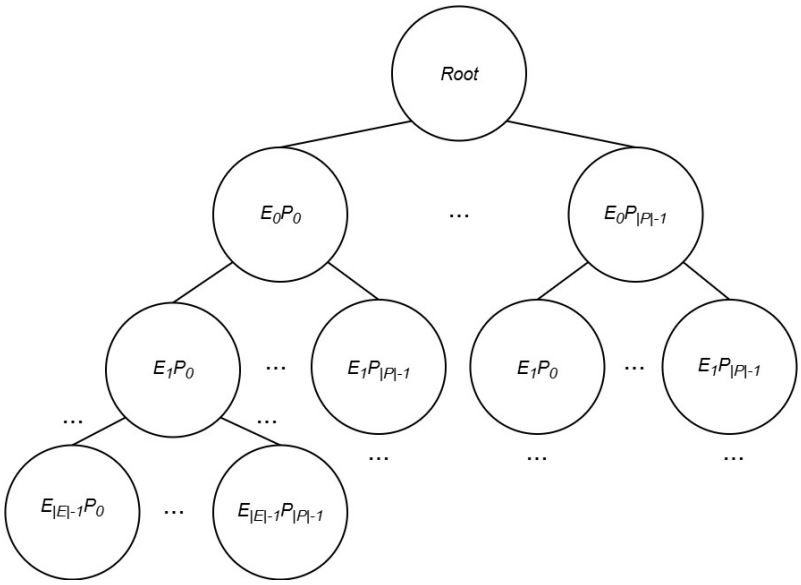

FIGURE 1: Proposed Monte Carlo Tree Search (MCTS) tree

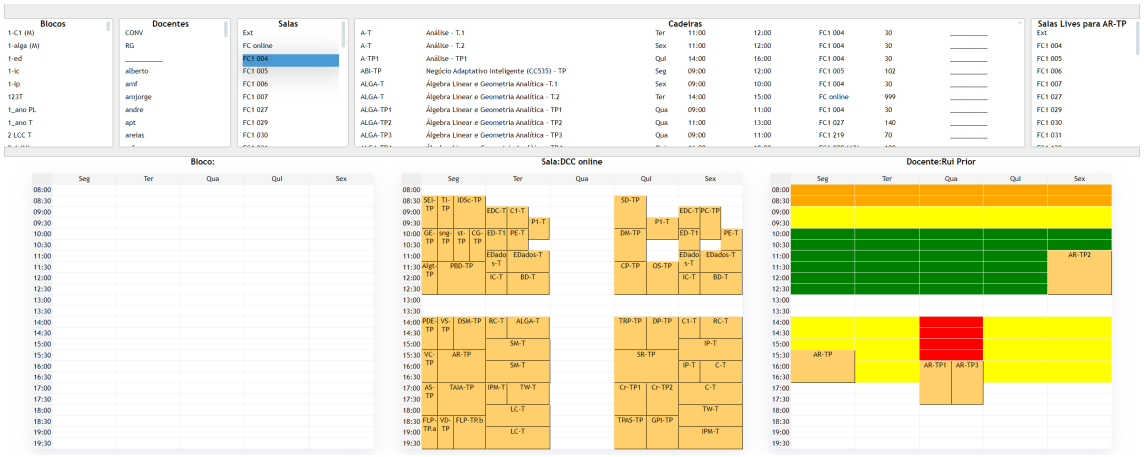

FIGURE 2: Previous work interface

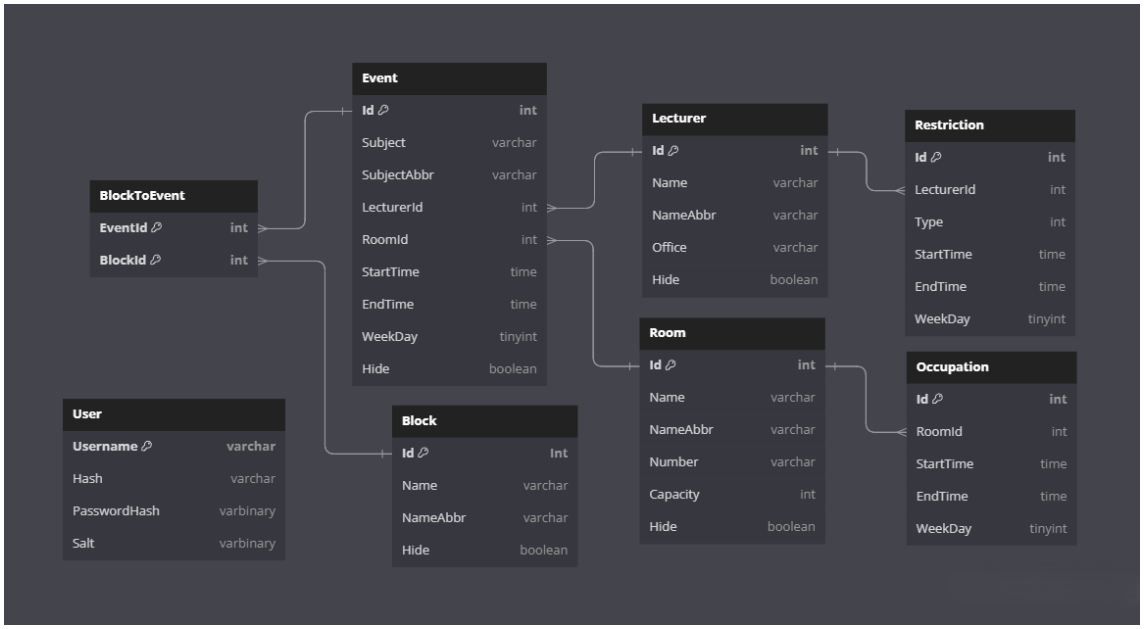

FIGURE 3: Previous work UML