Universiteit Leiden

Master Computer Science

Evaluating the Costs of Constraints in University Course Timetabling Problems

Name: Isaac Braam
Student ID: s2715252

Date: [31/03/2025]

Specialisation: Artificial Intelligence

1st supervisor: Arno Knobbe
2nd supervisor: Hao Wang

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

The University Course Timetabling Problem (UCTP) is an NP-hard problem, making it empirically difficult to get a feasible solution, and furthermore, difficult to assess which constraints make the problem strenuous. Truly determining constraint costs requires an exhaustive search with a factorial complexity of $\mathcal{O}(n!)$. This thesis introduces a novel Weighted Scheduling Algorithm (WSA) with a computational complexity of $\mathcal{O}(n^2)$, to accurately estimate constraint costs. Experimental results show that WSA outperforms prior methods in terms of accuracy. WSA enables fair evaluation of constraint costs and reduces reliance on uninformed iterative human input. WSA supports quantitatively informed decision-making and streamlines the scheduling process. This work bridges theoretical advancements with practical needs, by addressing the complexities of UCTP with a focus on fair distribution, efficiency, and accuracy.

Contents

1 Introduction **5**
- 1.1 Motivation . . . 5
- 1.2 Research Question . . . 7

2 Preliminary **8**
- 2.1 University Course Timetabling Problem (UCTP) . . . 8
  - 2.1.1 Scheduling . . . 8
  - 2.1.2 Types of Timetabling . . . 8
  - 2.1.3 School Timetabling . . . 8
  - 2.1.4 Course Timetabling . . . 9
  - 2.1.5 Examination Timetabling . . . 10
  - 2.1.6 Timetabling Workflows . . . 10
  - 2.1.7 Practical Implications . . . 11
- 2.2 Constraint Cost Evaluation Techniques . . . 11
  - 2.2.1 Shapley Values . . . 12
  - 2.2.2 Bucket Comparison . . . 12

3 Related Work **13**
- 3.1 Multi-Objectivity . . . 13
- 3.2 Alternative constraint cost evaluation methods . . . 13
  - 3.2.1 Graph Coloring . . . 13
  - 3.2.2 Shadow Prices . . . 14
  - 3.2.3 Flexibility Scores . . . 14

4 Limitations of other Constraint Cost Evaluation Methods **15**
- 4.1 Heuristic Approach . . . 15
- 4.2 Random Scheduling . . . 16
- 4.3 Same Algorithm Constraint Cost Evaluation . . . 16
- 4.4 Neutral Networks . . . 16
- 4.5 Conclusion on Prior Approach . . . 17

5 Algorithm Description: Weighted Scheduling Algorithm (WSA) **18**
- 5.1 Step 1: Initialization of the Scheduling Matrices . . . 18
  - 5.1.1 Illustrative Example of Initialization . . . 19
- 5.2 Step 2: Iterative Probabilistic Assignment of Activities . . . 19
  - 5.2.1 Illustrative Example for Iterative Probabilistic Assignment of Activities . . . 22
- 5.3 Step 3a: Stand-Alone Constraint Cost Evaluation . . . 24
  - 5.3.1 Illustrative Example for Constaint Cost Calculation . . . 25
- 5.4 Step 3b: Shapley Value Constraint Cost Evaluation . . . 26
  - 5.4.1 Illustrative Example of Shapley Value Constraints Cost Calculation . . . 27
- 5.5 Why the Order of Activities Does Not Matter . . . 28

6 Results **29**
- 6.1 Exhaustive Search Backtest . . . 29
  - 6.1.1 WSA versus a Linear Method . . . 30
  - 6.1.2 Shapley versus Stand-Alone Constraint Cost Evaluation Method . . . 31

6.2 Bucket Comparison 32
6.3 Time Complexity 32

7 Discussion **35**
7.1 Implications of Findings 35
7.2 Limitations and Challenges 35

8 Conclusion **36**
8.1 Key Contributions 36
8.2 Future Work 36
8.3 Final Remarks 36

A Mathematical Proof of Order Invariance **40**

B Most Constraining Constraint Types **40**

C Algorithm Steps Example **40**

1 Introduction

The University Course Timetabling Problem (UCTP) has long been a logistical challenge, evolving as universities have expanded and increased in complexity. Historically, constructing the university timetable was often the responsibility of individual professors. However, this paradigm becomes infeasible with the growth of universities, as the complexity of the UCTP problem increases correspondingly [23]. As this NP-hard problem expands, the search space grows exponentially, making it increasingly difficult to identify near-optimal solutions within a feasible computational time [7]. Consequently, the generated solutions exhibit a greater deviation from the theoretical optimal. As a result of this, there is now greater incentive and available budget for developing more sophisticated and efficient methods to tackle this issue. This claim is substantiated by TimeEdit, a company specializing in addressing the UCTP and the primary benefactor of this thesis. TimeEdit utilizes a deterministic heuristic-based algorithm that is capable of managing UCTP instances with over 10,000 activities. However, generating a satisfactory schedule typically takes about four days, and requires extensive human guidance. A method for determining which constraints are most costly would be of great help to the scheduling problem.

1.1 Motivation

TimeEdit has identified that the most costly adjustment that universities are adopting is the expansion of free-choice course offerings. This significantly increases complexity, requiring courses to accommodate a much broader range of student schedules, making timetable optimization increasingly challenging. Such a single change comes with a list of constraints. This highlights the need for the possibility to make an assessment for a set of constraints, not merely assessing a stand-alone constraint.

The company constructs the schedule by having a university course coordinator decide which courses will be offered in the upcoming academic year and specify the associated requirements for each course (e.g., the room needs to have computers, or the professor has Mondays off.) The company subsequently executes the algorithm to generate a satisfactory schedule. Empirical evidence suggests that, in many cases, no feasible solution exists given all the hard-to-satisfy constraints of the preliminary schedule. Even if a feasible solution is possible, no polynomial-time algorithm can certify its existence, since the problem is NP-hard. Consequently, an iterative process begins, where discussions with university staff is needed to decide which constraints may be relaxed (or whether additional resources should be acquired to meet the requirements). This process falls into the third category of scheduling methodologies outlined in Section 2.1.6, which comes down to an algorithm, guided by human input throughout the process. While this hybrid approach can yield feasible schedules, it has notable downsides: the iterative process involves a significant amount of back and forth communication. To illustrate this, we refer to Figure 1, and focus specifically on Step 7: Human Adjustments to Constraints. This step may need to be repeated multiple times, depending on the complexity of the scheduling problem. The rerunning of the scheduler, and the human input in this step, are costly, and the people are often ill-suited to filter through the large amount of scheduling data. This thesis, therefore, aims to reduce reliance on iterative input from uninformed humans in the UCTP.

To achieve this goal, this thesis developed a method of estimating the most constraining constraints. By systematically assessing these constraints, the method identifies those that impose the greatest computational and operational costs within the scheduling process. These

constraints are identified as the best constraints to be foregone. An example of constraint costs in the UCTP can be found in Tables 7 and 8. The usefulness of this method is two-fold:

1. The course constructor or coordinator can determine the cost of a constraint, or a set of constraints, at the time they are initialized, this actions refers to Figure 1, Step 2. As a result, costly constraints are carefully considered before being added. This helps decision-makers set realistic constraints from the outset.

2. When the algorithm requires the user to relax certain constraints, the user can make informed decisions based on a clear estimation of what constraints are the most appropriate to relax, this actions refers to Figure 1, Step 7. This will speed up the entire scheduling process, as well as minimizing the amount of constraints that need to be relaxed to arrive at a feasible solution.

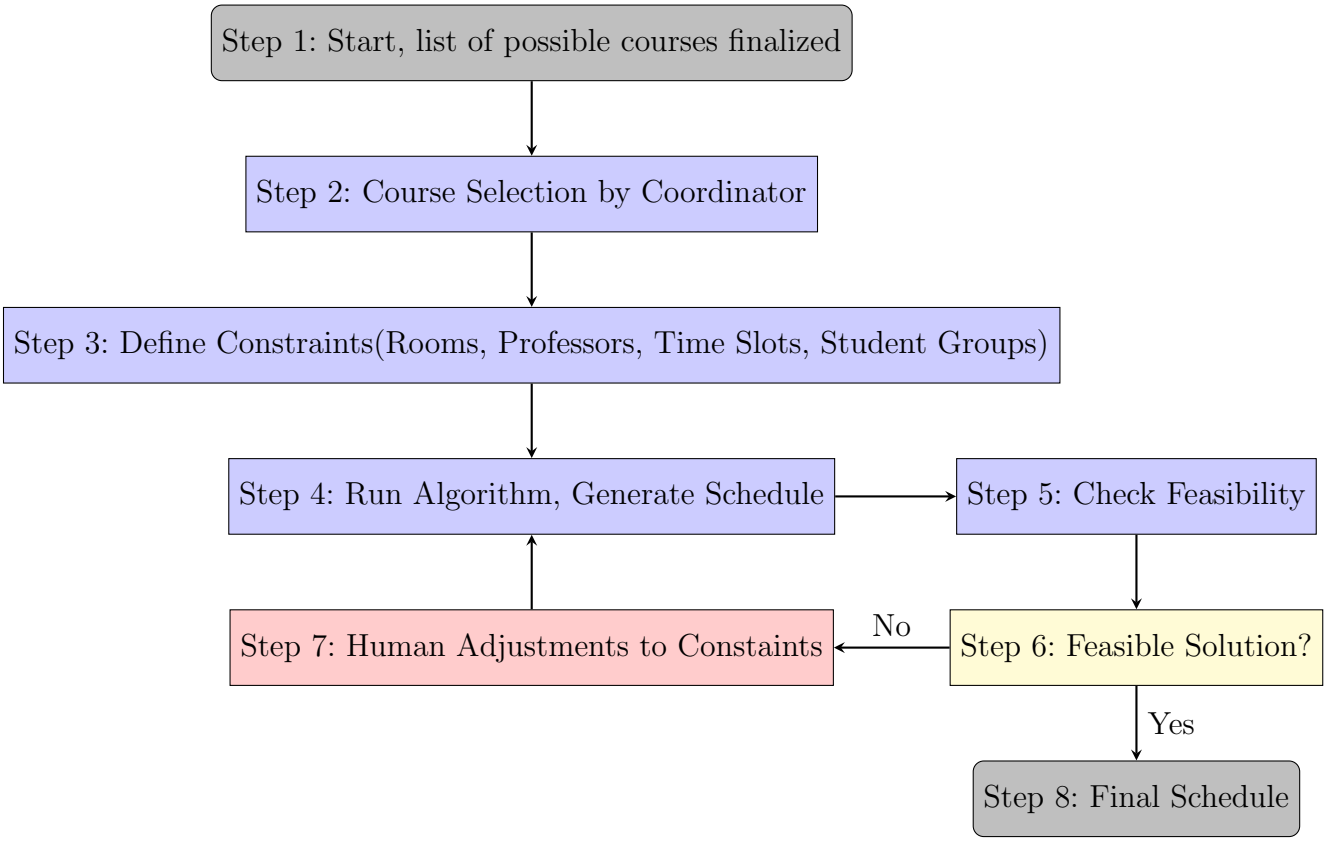


Figure 1: TimeEdit Scheduling Pipeline. The scheduling workflow begins with course selection and constraint definition, followed by the generation of an initial schedule. A feasibility check determines whether a valid schedule is found. If feasible, the schedule is finalized; otherwise, an iterative refinement process requires human intervention to adjust constraints before rerunning the algorithm.

1.2 Research Question

Effective UCTP is a complex optimization problem constrained by multiple factors, such as room availability and instructor schedules. The challenge lies in determining which constraints most significantly hinder the creation of feasible schedules; these quantified hindrances are referred to as constraint costs. Understanding constraint costs can help universities make informed decisions about which constraints to relax or modify to improve feasibility and efficiency.
The primary research question addressed in this thesis is:

> **What method(s) can be used to efficiently estimate the cost of constraints and sets of constraints in the University Course Timetabling Problem (UCTP)?**

To further refine this research, the following sub-questions are formulated:

1. **What are the limitations of existing constraint evaluation techniques in the context of UCTP?**
2. **How can constraint cost estimation improve the efficiency of university course scheduling algorithms?**
3. **How do different types of constraints impact scheduling feasibility?**
4. **Can Shapley values provide a reliable addition to the assessment of constraint costs?**
5. **What is the computational complexity of the constraint cost evaluation methods?**

2 Preliminary

Before delving into the core methodology of this thesis, it is essential to establish a foundational understanding of the UCTP, and the techniques used to evaluate constraint costs. This chapter provides the necessary background by outlining the key components of UCTP, including its structure, types, and scheduling workflows. In addition, we introduce different methods for evaluating constraint costs, which play a crucial role in this thesis' constraint costs assessing algorithm.

2.1 University Course Timetabling Problem (UCTP)

2.1.1 Scheduling

Scheduling can be defined as the organization of entities (such as individuals, tasks, vehicles, lectures, exams, or meetings) within a specific space and time framework, to ensure that all constraints are met [24]. Constraints are defined as relationships, either among entities or between entities and their spatial or temporal arrangements, which restrict the possible configurations of the schedule. In the UCTP context, spatial constraints refer to restrictions related to the physical arrangement of entities, such as rooms, buildings, or campus facilities, while temporal constraints relate to the scheduling of events over time, such as specific time slots or days. To simplify, we will address these spatial and temporal arrangements as **room-time slots**.
Large multi-objective combinatorial scheduling problems are especially challenging because the size of the search space grows exponentially as the problem size increases.

2.1.2 Types of Timetabling

Research into timetabling can be traced back to early foundational works. Schmidt and Schaerf highlighted three main classes of timetabling problems [17]:

1. **School timetabling**
2. **Course timetabling**
3. **Examination timetabling**

2.1.3 School Timetabling

School timetabling primarily involves assigning teachers and students to classes without scheduling conflicts, ensuring that no teacher or class is involved in more than one lecture at any given time. Van Den Broek proved that a solution always exists for this basic version of the school timetabling problem, provided that no teacher or class is scheduled for more lectures than there are available time slots [23].

School timetabling can be formulated mathematically in the following manner:
Mathematical Formulation of the school timetabling problem:
We assume that we have:

- A set of c classes.

- A set of t teachers.
- A total of p available time periods.
- A set of n activities to be scheduled.
- A predefined number of required sessions m_{ij} between each class i and teacher j.
- X_{ijk} is a binary decision variable that represents whether class (or course) i is assigned to teacher j during time period k

The problem is to find:

$$X_{ijk} \quad (i = 1, \ldots, c;\ j = 1, \ldots, t;\ k = 1, \ldots, p)$$

subject to

$$\sum_{k=1}^{p} X_{ijk} = m_{ij} \tag{1}$$

$$\sum_{j=1}^{t} X_{ijk} \leq 1 \tag{2}$$

$$\sum_{i=1}^{c} X_{ijk} \leq 1 \tag{3}$$

where $X_{ijk} = 1$ if class i and teacher j meet at period k and 0 otherwise.

2.1.4 Course Timetabling

The course timetabling problem entails scheduling a series of lectures for each course within a limited number of rooms and available time periods. The key difference between school and course timetabling lies in the overlap of student enrollments. In course timetabling, students may enroll in multiple courses that overlap, whereas school classes consist of distinct, non-overlapping groups of students. While the school timetabling problem is tractable, the course timetabling problem is intractable [2].

Course timetabling can be formulated mathematically in the following manner:
Mathematical Formulation of the course timetabling problem:
The problem is to find:

$$X_{ijk} \quad (i = 1, \ldots, c;\ j = 1, \ldots, r;\ k = 1, \ldots, p)$$

subject to

$$\sum_{k=1}^{p} X_{ijk} = m_{ij} \tag{4}$$

$$\sum_{j=1}^{r} X_{ijk} \leq 1 \tag{5}$$

$$\sum_{i=1}^{c} X_{ijk} \leq 1 \tag{6}$$

Students cannot be scheduled in overlapping courses:

$$\sum_{s \in S} X_{sjk} \leq 1, \quad \forall s \in S \tag{7}$$

where S represents the set of students enrolled in multiple courses.

2.1.5 Examination Timetabling

Examination timetabling differs from course timetabling in other significant ways. It typically involves a single exam per course, enforces a strict time conflict constraint for the students, and allows multiple student groups to have exams concurrently within a single room.

Examination timetabling can be formulated mathematically in the following manner:
Mathematical Formulation of the examination timetabling problem:
The problem is to find:

$$X_{ik} \quad (i = 1, \ldots, c;\ k = 1, \ldots, p)$$

subject to

$$\sum_{k=1}^{p} X_{ik} = 1, \quad \forall i \tag{8}$$

$$\sum_{s \in S_i} X_{ik} \leq 1, \quad \forall s, \forall k \tag{9}$$

$$\sum_{i=1}^{c} X_{ik} \leq R, \quad \forall k \tag{10}$$

where $X_{ik} = 1$ if exam i is scheduled in period k, S_i is the set of students enrolled in exam i, and R is the number of available rooms.

Among the three timetabling problems, the course timetabling problem is by far the most challenging, primarily due to its high degree of non-determinism driven by overlapping student enrollments. The UCTP under consideration in this thesis is functionally identical to the course timetabling problem, described in Section 2.1.4.

2.1.6 Timetabling Workflows

Next to the three types of timetabling, the three different workflows, as defined by Dario [20] must be distinguished.

1. **Decision-Making Then Search:**

   In this approach, decisions about the problem constraints or preferences are made upfront, and the search process then operates within the boundaries defined by these decisions.

2. **Search, Then Decide Between Options:**

   This method involves conducting a broad search over potential solutions, generating a range of viable options, and then choosing one of these for the final schedule. Further described in Section 3.1.

3. **Guided Algorithm Search:**

   Here, the algorithm is guided during the search process, often with human intervention to steer the search toward promising regions of the solution space.

The process of interest in this thesis, and where the proposed algorithm is useful, is the Guided Algorithm Search, as the benefactor of this thesis uses this approach. Additionally, the proposed algorithm demonstrates utility in making decisions about constraints prior to implementing any of the three workflows, although this is most frequently observed in the first workflow, Decision-Making Then Search.

2.1.7 Practical Implications

Meta studies by Zampieri and Schaerf emphasize that overly simplified models often fail to capture the complexity required by real-world educational institutions, rendering many of the theoretical approaches in scientific papers impractical in operational settings [25].
Furthermore, Mccollum notes that the limited adoption of research-based timetabling methods within educational institutions indicates a persistent gap between academic research and practical implementation [14]. This thesis aims to narrow the gap between theoretical approaches and application by applying the proposed methods to complex real-world scheduling problems.
A successful real-world endeavor has been achieved by researchers at TU Eindhoven, where a real-world student scheduling problem (SSP) has been successfully implemented [23].
In scenarios like the SSP, where all courses on a student's preference list must be accommodated, the problem becomes so constrained that feasible solutions are often non-existent. This is the same problem that TimeEdit runs into. This persistent issue underscores the need for alternative strategies to manage constraints effectively.

2.2 Constraint Cost Evaluation Techniques

This thesis is not directly concerned with the UCTP problem, but rather with finding the constraint costs of a given UCTP problem. In the existing literature, constraint costs are frequently evaluated by examining their impact on the convex hull of the solution space. Specifically, constraints are assessed based on the extent to which their inclusion reduces the size of the convex hull [21]. The convex hull, in this context, represents the smallest convex set that encompasses all feasible solutions to the problem. By imposing constraints, portions of this solution space are effectively eliminated, thereby shrinking the convex hull and reducing the range of viable solutions.
In UCTP, we have a discrete solution space, by definition this makes the space non-convex. We are also unable to calculate the convex hull, as the restrictions are not expressible in a function. We can, however, sample the solution space, yet this introduces complications, as will be discussed in the next section. For much smaller instances of UCTP, the size of the solution spaces can be assessed through exhaustive search.
However, as the problem scales, this approach quickly becomes infeasible due to exponential growth in the number of possibilities. We will show this empirically in Section 6.1.

2.2.1 Shapley Values

Shapley values [18] are originally a concept from cooperative game theory used to fairly and fully distribute the total payoff among participants based on their contributions. Each participant's value is determined by calculating their marginal contributions across all possible coalitions. The method ensures fairness, as it considers both the participant's standalone value and their interactions with others. Exact computation of Shapley values has a complexity of $\mathcal{O}(n!)$. This makes them infeasible for large n, although various sampling techniques have been shown to retain high accuracy [15]. Shapley values are increasingly used in fields like economics and machine learning, and they have many utilities for accessing performance or costs in incalculable complicated processes, as demonstrated in my previous work [1].
In this thesis, Shapley values will be utilized for assessing constraint costs. This approach allows for a fair allocation of each constraint's contribution to the overall cost by evaluating their effects when multiple constraints are measured simultaneously during the "sampling" process. These coalitions are by most methods incalculable, as they all have overlapping sets. By using Shapley values, it is possible to ensure a fair evaluation of each constraint's impact. This method avoids the overestimation that can occur with standalone assessments and the underestimation that often arises from marginal approaches [8].
In the original paper [18], the Shapley value for player i is given by:

$$\varphi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!\,(n - |S| - 1)!}{n!} \left(v(S \cup \{i\}) - v(S)\right) \tag{11}$$

Where:

- N is the set of all players in the game.
- $n = |N|$ is the total number of players.
- v is the characteristic function, which assigns a value $v(S)$ to each coalition $S \subseteq N$.
- S is a subset of players that does not include player i.

2.2.2 Bucket Comparison

Bucket comparisons are a concept mostly used in financial settings. In finance, a bucket is a way of grouping data points into ranges based on specific criteria [9]. A bucket comparison involves breaking down data into these predefined ranges (buckets) and comparing the metrics of one bucket to another, analogous to the binning process employed in histograms. This comparison is typically used to identify patterns or anomalies across different segments of the data. Bucket comparisons will, however, only matter for a specific test in Section 6.2. Where we bucket varying constraint costs ranges, to have less volatile results.

3 Related Work

The preliminaries introduced the required background knowledge for this thesis project. The following literature review gives further information on UCTP and introduces related concepts that are not necessary to understand this thesis. The first part addresses multi-objectivity, whereas the second part mentions alternative constraint evaluation methods.

3.1 Multi-Objectivity

Scheduling for university courses is inherently multiobjective, as multiple, often conflicting, criteria must be balanced to evaluate schedule quality. Traditionally, many UCTP solutions reduce the problem to single objective optimization by weighting criteria, yet recent multi-objective meta-heuristic techniques are demonstrating effectiveness without such reduction [20]. Despite promising results in benchmark scenarios, applications of these techniques to real-world scheduling remain limited [10]. Multi-objective scheduling often entails trade-offs among criteria, where improvement in one criterion (e.g., minimizing student congestion) may negatively impact another (e.g., shortening the timetable duration). To navigate these trade-offs, it is common to employ Pareto optimization, which presents a set of different solutions from which decision-makers can select an adequate schedule [22]. To make sure that the solutions are different enough, Tabu search is often implemented when aiming for a diverse pareto front. Tabu search ensures the solutions span a different neighbourhood in the solution space [6]. Additionally, "peckish-uniform initialization" is used for the same means. This strategy resembles a greedy algorithm that occasionally makes mistakes and supports greater diversification in timetabling problems. Corne and Ross stated that a high proportion of highly fit initial solutions is not desirable; instead, a higher proportion of slightly fit initial solutions seems more preferable [5]. They presented evidence suggesting that peckish-uniform initialization is beneficial across timetabling problems, whereas greedy-feasible strategies tend to be counterproductive.
In a related work, Corne and Ogden employed a direct representation for their problem and compared three algorithms: hill-climbing, simulated annealing, and evolutionary algorithms. Their findings indicated that the performance of each technique varied depending on the representation scheme used. This further emphasizes the importance of selecting appropriate initialization and search strategies [4].

3.2 Alternative constraint cost evaluation methods

When evaluating constraint costs in UCTP, several alternative methodologies exist, including Graph Coloring, Shadow Prices, and Flexibility Scores. However, these methods are not considered in this research due to their computational intractability, or lack of accuracy when applied to large-scale UCTP instances.

3.2.1 Graph Coloring

Another innovative approach in the literature is graph coloring, which we could utilize to estimate the complexity of scheduling activities in UCTPs. This approach involves representing UCTP as a constraint satisfaction problem (CSP), in which constraints are visualized within a constraint graph [16]. Here, nodes represent courses or events, and edges denote shared resources (e.g., rooms, teachers) between these nodes. By analyzing the "tightest" or most interconnected

nodes within this graph, schedulers can identify critical constraints that significantly restrict scheduling.
This method, referred to as graph coloring, looks promising for evaluating the constraint costs. Yet, current methodologies exhibit excessive oversimplifications, which render them insufficient for addressing real-world UCTPs. One key issue is that all edges should be treated in the same manner. For instance, all activities in the UCTP would typically share a certain resource: the available rooms at the university. However, activities with similar group sizes would conflict more with each other than those with differing group sizes. Since this is a scalar value rather than a binary cutoff, graph coloring does not account for this distinction, thus making it incapable of the required precision.

3.2.2 Shadow Prices

Shadow prices originate from linear programming and optimization theory [19]. They represent the marginal cost of relaxing a constraint by one unit. In the context of UCTP, shadow prices can be interpreted as the cost or benefit associated with modifying a specific constraint, such as increasing room capacity or relaxing scheduling restrictions. By assigning a numerical value to these changes, shadow prices provide a quantitative measure of how individual constraints influence the overall feasibility and quality of the solution space. Shadow prices are crucial to UCTP, as they highlight constraints that may be overly restrictive or disproportionately impactful.

3.2.3 Flexibility Scores

Flexibility scores, in contrast, are designed to assess the degree of freedom or adaptability within the solution space given a set of constraints [19]. These scores quantify how much leeway exists when attempting to satisfy constraints without compromising the feasibility of the solution. In the UCTP context, flexibility scores can be particularly useful for identifying bottlenecks or rigid areas in the solution space. For example, if certain courses or time slots have extremely low flexibility scores, this indicates that the associated constraints are highly restrictive, and potentially warrant further investigation or relaxation.

4 Limitations of other Constraint Cost Evaluation Methods

In this section, we evaluate why other existing methods for constraint cost evaluation are ill-suited for large-scale UCTPs. Several approaches have been proposed, including heuristic-based methods, random search, constraint cost evaluation using the same scheduling algorithm, and neural networks. However, these methods either introduce biases, are computationally infeasible, or fail to provide accurate constraint cost estimations. Our goal in this section is to critically assess these approaches and highlight their limitations, ultimately motivating the need for an alternative method, such as the proposed Weighted Scheduling Algorithm (WSA).
A well-designed algorithm for the UCTP, like TimeEdit's, achieves near-optimal solutions, but operates within a specific, narrow solution space. When using constraint cost evaluation techniques, it is often necessary to construct a multitude of schedules in order to conduct accurate measurements. Rerunning the same algorithm to evaluate improvements when we relax constraints could be a viable approach, if not for its high computational cost. This underscores the necessity of a relatively low-cost evaluation method.
Each subsection will analyze a specific category of constraint evaluation methods and outline the reasons for why they fall short. By understanding these limitations, we can better appreciate the advantages of a more systematic and computationally feasible approach to constraint cost evaluation.

4.1 Heuristic Approach

Schedules are typically generated by heuristic algorithms. Often, the bigger UCTPs take days to run on high-end algorithms. Using a different, less computationally expensive algorithm than the one originally used to create the schedule would inevitably shift the search into an entirely different solution space. As a result, there is no guarantee that the costs derived from these measurements are accurate.
Furthermore, heuristic-based methods are inductively biased when evaluating constraint costs. A heuristic-based approach means that certain constraints are prioritized when scheduling, such as a greedy algorithm that schedules big classes first. While this can be effective for quickly generating solutions, the bias arises because the heuristic systematically addresses the constraints related to large classes and big rooms early in the process, when resources (e.g., large rooms) are still plentiful.
As a result, the constraint of for example needing a large room appears less costly in the final analysis, as the algorithm has already ensured that these constraints were satisfied early. The bias comes from the fact that the heuristic actively shapes the solution space to favor some constraints over others. As a consequence, the heuristic skews the solution space in favor of certain constraints, creating the illusion that they are easier to satisfy than they would be under a more neutral or alternative scheduling method.
This selective prioritization skews the evaluation of constraint costs. For a true understanding of how costly each constraint is, it is important to recognize that the apparent "ease" of satisfying some constraints is a direct consequence of the heuristic's influence, not an inherent property of the constraint itself.
Consequently, using simpler or even different heuristic measurements for evaluations does not meet our desideratum of achieving a fair constraint cost comparison. Therefore, sampling with a mismatched algorithm would be insufficient for accurate constraint cost evaluation.

4.2 Random Scheduling

Random scheduling involves randomly assigning activities to room-time slots. Although this method seems intuitively unbiased, it is still biased towards some constraints in the context of UCTP. This is because the probability of random search reaching the specific solution space of a sophisticated algorithm is infinitesimally small for a realistic UCTP problem. Random search is not likely to visit these spaces, as it often produces unrealistic results, such as assigning 30 students to a room designed for a 1000 students, a scenario that a well-constructed algorithmic scheduler would avoid. However, random search can serve as a diagnostic tool to identify the most demanding constraints, as courses with the most stringent constraints are likely to fail scheduling first, allowing the proposed tool to gauge difficulty; however, this measurement will be significantly biased as well.
To explain this in more detail, it is necessary to first define the types of constraints we are dealing with:

1. Static Constraints: These are predetermined and unchanging, such as a teacher being unavailable on Mondays.

2. Dynamic Constraints: These become more constraining as scheduling progresses, such as a teacher's availability decreases when their personal schedule fills.

Random scheduling would result in suboptimal timetables compared to a more advanced scheduling approach, often failing to accommodate a significant portion of activities. Since dynamic constraints generally impose higher costs in a fully packed schedule compared to a partially filled one, it becomes evident that the dynamic constraints are positively biased into costing less when using random scheduling.

4.3 Same Algorithm Constraint Cost Evaluation

What if the same algorithm were used? Running a heuristics-based algorithm, like the one TimeEdit uses, requires significant computing power, and still takes several days to run. The algorithm would need to run a multitude of times to see how the schedule changes when relaxing constraints to be measured. Even with the necessary resources, a definitive or converged cost would not be reached, as merely changing the constraints could lead to a "butterfly effect," where the dynamics of the solution space are fundamentally altered [3]. Therefore, to fully understand constraint costs, it becomes essential to evaluate the entire solution space, rather than relying on sampled trajectories.

4.4 Neutral Networks

Recent advances in deep reinforcement learning have demonstrated the potential of neural networks (NNs) for combinatorial optimization problems, such as the Covering Salesman Problem (CSP), which shares structural similarities with UCTP [12]. Specifically, NNs can be trained to construct near-optimal schedules by learning from historical data and dynamically adjusting to new constraints. Multi-head attention mechanisms and dynamic embeddings allow for efficient constraint satisfaction and scalable decision-making.
To adapt this approach to UCTP, we begin by translating scheduling constraints, such as time slots, rooms, teachers, and student groups, to nodes in a graph, as proposed in Section

3.2.1. The NNs encoder extracts spatial-temporal relationships between these elements using attention mechanisms, while a decoder sequentially assigns courses to time slots.
However, generating an initial schedule is only the first step. Constraint evaluation remains challenging because the NNs would introduce additional challenges. While NNs can generate their own heuristics, their "black-box" nature obscures the reasoning process, making it difficult to interpret constraint interactions or identify potential biases [18]. Furthermore, NNs approaches in the current literature for UCTPs rely on simplifications, like the ones mentioned in Section 3.2.1, that limit their applicability to real-world situations [14].

4.5 Conclusion on Prior Approach

Neither random search, NNs, graph coloring, nor alternative algorithms are sufficient due to their inductive biases and disjoint solution spaces to the original algorithm. A nuanced approach that considers the interplay of constraints within the chosen algorithm's context is needed for a robust analysis and meaningful comparisons. The proposed algorithm in this thesis does meet all these desiderata.

5 Algorithm Description: Weighted Scheduling Algorithm (WSA)

It has been concluded in previous sections that no known scheduling method is suitable for constraint cost measurements, which is why it is essential to develop a new method. The proposed scheduling algorithm, WSA, is a novel approach to scheduling for the purpose of analyzing constraint costs in a UCTP. Traditional scheduling methods that produce a single deterministic schedule can make it difficult to evaluate the true cost of each constraint, since any small change in the schedule may drastically affect constraint costs. To address these issues, we propose the Weighted Scheduling Algorithm (WSA), a *probabilistic* approach designed to fairly measure the cost of constraints without being influenced by the order in which scheduling decisions are made.

Key Idea. Rather than choosing a single deterministic time slot (and other resources) for each activity, WSA distributes every activity across *all feasible* choices using probability distributions. In much the same way that a quantum state can occupy multiple positions simultaneously, each activity is considered to be "superimposed" in multiple valid allocations. This ensures that constraints are evaluated against all possible scheduling outcomes, yielding a robust cost estimate that remains stable even as constraints are altered.

High-Level Overview. WSA determines constraint costs in three main steps:

- Step 1 initializes the resource matrices needed for the scheduling and constraint cost evaluations.
- Step 2 iteratively assigns activities to the probabilistic schedule, in a random order, as the order of assignment does not affect the final result, see Section 5.5 for a more detailed explanation.
- Step 3 evaluates the constraint costs using the probabilistic schedule which is constructed in Step 2, either through the Stand-alone method (Step 3a) or using the Shapley method (Step 3b).

5.1 Step 1: Initialization of the Scheduling Matrices

We first define the sets and indices used in this method:

- $\mathcal{T} = \{1, \ldots, t_{max}\}$: the set of all discrete time periods.
- $\mathcal{R} = \{1, \ldots, r_{max}\}$: the set of all rooms.
- $\mathcal{G} = \{1, \ldots, g_{max}\}$: the set of all student groups.
- $\mathcal{P} = \{1, \ldots, p_{max}\}$: the set of all professors.
- $\mathcal{A} = \{1, \ldots, n_{max}\}$: the set of activities to be scheduled.

In a traditional (deterministic) scheduling problem, like the one described in Section 2.1.4, we defined $X_{ijk} = 1$ if class i and professor j meet during period k, and 0 otherwise. In WSA, however, we consider a continuous relaxation of the original problem; rather than $X_{ijk} \in \{0, 1\}$, we allow continuous values in $[0, 1]$ to represent allocation *probabilities*.

Resource Matrices. Let $\mathcal{R}$ be a set of rooms, $\mathcal{P}$ a set of professors, and $\mathcal{G}$ a set of student groups. For each resource type, we define a matrix indexed by the appropriate resource set and the set of time slots $\mathcal{T}$:

- *Room Matrix*, $M_{\text{room}}(r,t) \in [0,1]$, where $r \in \mathcal{R}$ and $t \in \mathcal{T}$. The entry $M_{\text{room}}(r,t)$ represents the probability that room r is occupied at time t.
- *Student Matrix*, $M_{\text{student}}(g,t) \in [0,1]$, where $g \in \mathcal{G}$ and $p \in \mathcal{P}$. The entry $M_{\text{student}}(g,t)$ indicates the probability that student group g is occupied at time t.
- *Professor Matrix*, $M_{\text{professor}}(p,t) \in [0,1]$, where $p \in \mathcal{P}$ and $t \in \mathcal{T}$. The entry $M_{\text{Professor}}(p,t)$ denotes the probability that professor p is occupied at time t.

Initially, all these matrices are set to 0, indicating no scheduled activities. As activities are added, these entries increase based on the algorithm described below.

5.1.1 Illustrative Example of Initialization

Before any activities are scheduled, one might visualize a simple room-time slot matrix with all entries set to 0, indicating a 0% occupancy probability. Table 1 shows an example room-time matrix with two rooms and two time slots, all initialized to zero. Table 2 shows an example student group-time matrix with two student groups and two time slots, also all initialized to zero. And finally, Table 3 shows an example professor-time matrix with two professors and two time slots, all initialized to zero.

Room/Time	**01-01-2024 8:00**	**01-01-2024 9:00**
Room 1	0	0
Room 2	0	0

Table 1: An example of an empty room-time slot matrix M_{room} before the algorithm starts. Each entry is the probability that a given room is occupied at a particular time slot.

Student Groups	**01-01-2024 8:00**	**01-01-2024 9:00**
Group A	0	0
Group B	0	0

Table 2: An example of an empty student group-time slot matrix M_{student} before the algorithm starts. Each entry is the probability that a given student group is occupied at a particular time slot.

5.2 Step 2: Iterative Probabilistic Assignment of Activities

Iteratively we add one activity $a \in \mathcal{A}$, which may require a specific professor (or a set of candidate professors), a specific class (or multiple classes), and a feasible set of room-time slots. The WSA algorithm processes each activity as follows:

Professors	01-01-2024 8:00	01-01-2024 9:00
Professor A	0	0
Professor B	0	0

Table 3: An example of an empty professor-time slot matrix $M_{\text{professor}}$ before the algorithm starts. Each entry is the probability that a given professor is occupied at a particular time slot.

1. **Compute Availability.** For every feasible room-time slot (r, t), calculate a raw "availability score" by multiplying the complementary probabilities of each required resource being occupied. Formally, let $P_{\text{occupied}}(k, t)$ be the current probability that resource k is occupied at time t. Then for each (r, t):

$$\text{availability}(r, t) \;=\; \prod_{R,G,P} \big[\, 1 \;-\; P_{\text{occupied}}(k,\, t)\big] \tag{12}$$

   where the symbols R, G, and P respectively indicate that the resource is a Room, a Student Group, or a Professor.

   If there is a hard constraint forbidding (k, t), we set availability(k, t) to 0.

2. **Normalize Across Feasible Slots.** Sum these availability values over all (k, t) and normalize so the total sum is 1:

$$\hat{P}(r, t) \;=\; \frac{\text{availability}(r, t)}{\displaystyle\sum_{r',\, t'} \text{availability}(r', t')} \tag{13}$$

   This $\hat{P}(r, t)$ is the fraction of the activity assigned to room r at period t.

3. **Update Resource Matrices.** Once the normalized probability $\hat{P}(r, t)$ is computed for each room-time slot (r, t), we incorporate this partial assignment into *all relevant resource matrices*. In particular, for each (r, t) where $\hat{P}(r, t) > 0$:

$$M_{\text{room}}(r, t) \;\leftarrow\; M_{\text{room}}(r, t) \;+\; \hat{P}(r, t),$$

$$M_{\text{professor}}(p, t) \;\leftarrow\; M_{\text{professor}}(p, t) \;+\; \hat{P}(r, t) \quad \forall\, p \in P_a,$$

$$M_{\text{student}}(g, t) \;\leftarrow\; M_{\text{student}}(g, t) \;+\; \hat{P}(r, t) \quad \forall\, g \in G_a$$

   where P_a is the set of professors required by activity a and G_a is the set of student groups involved in a. If any entry of these matrices exceeds 1, one may rescale the entire matrix accordingly, ensuring that probabilities remain valid (i.e., no resource is assigned more than 100% at any given time). Emperically exceding a 100% occupation for a room-time slot is very rare and only found in small or atypical UCTPs.

```
procedure ASSIGN_ACTIVITY(activity a):

    # Step 1: Compute availability
    for each feasible (room r, time slot t):
        availability[r, t] = get_availability(M_room,
                                              M_professor,
                                              M_student,
                                              r, t)
        # Function get_availability() according to Formula (11)
        # If a constraint forbids (r, t), set availability[r, t] = 0

    # Step 2: Normalize, (as we devide 1.0 activity over the room-time slots)
    total_avail = sum( availability[r, t] for all r, t )
    for each (r, t):
        prob[r, t] = availability[r, t] / total_avail

    # Step 3: Update resource matrices
    for each (r, t):
        M_room[r, t] += prob[r, t]
        M_professor[p, t] += prob[r, t]
        M_student[g, t]  += prob[r, t]
        # rescale matrices if needed to keep all entries in [0,1]

end procedure
```

Figure 2: Pseudo Code for scheduling an activity with WSA

Pseudo Code. Performing the procedure described in Figure 2 iteratively over all activities in $\mathcal{A}$ yields a probabilistic schedule, that accurately reflects all constraints fairly.

AND and OR Constraints Until now we have only considered a constraint containing a single entity, yet constraints can involve multiple. For instance the choice of 2 professors to teach a given activity. These entities each have their own occupation matrix, so how do we combine these scores? There are two variations of combining thes scores, depending on if we are faced with an AND or an OR case. These operations are described in detail in the next two paragraphs.

- **AND Operations.** If an activity constraint involves multiple **required** entities simultaneously (e.g., a lesson for two student groups in the same room), the combined occupation probability for that slot, denoted as O_{combined}, is found by multiplying the complementary availabilities, effectively:

$$O_{\text{combined}} \;=\; 1 - \prod_{\text{entities}} \Big[1 \;-\; P(\text{occupied})\Big], \tag{14}$$

  Where $P(\text{occupied})$ is the probability that a given set of recourses within an entity is occupied at that time slot t. Since all resources must be available at the same time, a slot becomes infeasible if one resource is occupied Hence, we combine their occupancy probabilities in this manner.

- **OR Operations.** In cases where an activity constraint involves multiple **optional** entities simultaneously, such as when any one of several qualified professors can teach a given course, we denote the set of candidate resources as $\{E_1, E_2, \ldots, E_m\}$. Each resource E_i has an associated occupancy probability O_i in time slot t. To model the likelihood of each candidate being selected, we first compute a selection probability $P(E_i)$ for each resource, weighted by its availability:

$$P\big(E_i\big) \;=\; \frac{\big(1 - O_i\big)\, O_i}{\sum_{j=1}^{m} \big(1 - O_j\big)}, \tag{15}$$

and then combine these into an combined occupancy probability:

$$O_{\text{combined}} \;=\; \sum_{i=1}^{m} \Big[P\big(E_i\big) \;\times\; O_i \Big]. \tag{16}$$

In this way, exactly *one* of the optional resources is chosen for the activity, but each resource's occupancy and availability contributes probabilistically. In practice, if O_i is small (meaning E_i is often free), it has a larger chance of being selected, and vice versa.

5.2.1 Illustrative Example for Iterative Probabilistic Assignment of Activities

To illustrate how activities are assigned iteratively in WSA, consider the following example. Assume that after initialization, the resource matrices for rooms, professors, and student groups are all set to 0 (i.e., no occupancy). Now, suppose an activity $a \in \mathcal{A}$ is to be scheduled, and based on its constraints, only certain room-time slots are feasible.

Step 1: Compute Availability.
For every feasible room-time slot (r, t), the raw availability is calculated using Equation (12). where $P_{\text{res}}(\text{occupied}, t)$ is the current probability that the resource (Room, professor, or Student Group) is occupied at time slot t. Suppose that after evaluating these probabilities, we find that:

- For Room 1, the availability at 8:00 and 9:00 is a 100%, as no resource is occupied at the first iteration.
- For Room 2, the availability is 0 because a hard constraint forbids scheduling there. (This constraint could be that Room 2 is too small to host all the students.)

Step 2: Normalize Availability.
Using Equation (13), the raw availability values are normalized.

In our example, suppose that the normalization yields:

$$\hat{P}(\text{Room 1, 8:00}) = 0.5, \quad \hat{P}(\text{Room 1, 9:00}) = 0.5,$$

and

$$\hat{P}(\text{Room 2, 8:00}) = 0, \quad \hat{P}(\text{Room 2, 9:00}) = 0$$

Thus, the activity is equally likely to be scheduled in Room 1 at 8:00 or 9:00, and Room 2 is excluded.

Step 3: Update Resource Matrices.
The normalized probabilities are then added to the corresponding entries in each resource matrix. For each room-time slot (r, t).

- The room-time matrix M_{room} for Room 1 is updated to have a value of 0.5 at both 8:00 and 9:00. An example of this matrix is found in Table 4.
- Similarly, the student group-time matrix M_{student} is updated so that the relevant student groups (e.g., Group A) each receive a 0.5 probability at 8:00 and 9:00. An example of this matrix is found in Table 5.
- The professor-time matrix $M_{\text{professor}}$ is updated such that the professors involved (say, Professor A) have an occupancy probability of 0.5 at 8:00 and 9:00. An example of this matrix is found in Table 6.

Room/Time	**01-01-2024 8:00**	**01-01-2024 9:00**
Room 1	0.5	0.5
Room 2	0	0

Table 4: A probability matrix indicating that Room 1 is equally likely to host the activity at either 8:00 or 9:00, while Room 2 is unavailable due to a constraint.

Student Groups	**01-01-2024 8:00**	**01-01-2024 9:00**
Group A	0.5	0.5
Group B	0	0

Table 5: Illustrative student group-time matrix showing Group A is likely to be scheduled at either 8:00 or 9:00, while Group B is not scheduled in these slots for this activity.

Professors	**01-01-2024 8:00**	**01-01-2024 9:00**
Professor A	0.5	0.5
Professor B	0	0

Table 6: Illustrative professor-time matrix with a single professor (professor A) likely to occupy the 8:00 or 9:00 slot. Professor B is fully free during these times for this activity.

5.3 Step 3a: Stand-Alone Constraint Cost Evaluation

After the final probabilistic schedule has been constructed, which incorporates all constraints (as described in Steps 1 and 2), we can measure the cost of each constraint by looking at how much of the free room-time slots in M_{room} are effectively blocked by that constraint. This section shows how these costs are computed. The calculated value is technically deemed a stand-alone score [8]. We solely use M_{room} for constraint cost calculations since the other matrices are projectable onto it, while this is not the case the other way around.

1. Room Constraints (Binary Cut).
Suppose constraint c enforces that a given activity can only be scheduled in a subset of rooms $\mathcal{R}(c) \subseteq \mathcal{R}$. Then any room $r \notin \mathcal{R}(c)$ is effectively disallowed. The cost of c is then:

$$\text{cost}(c) \;=\; \sum_{r \in \mathcal{R} \setminus \mathcal{R}(c)} \; \sum_{t \in \mathcal{T}} \Big[1 - M_{\text{room}}(r,t)\Big].$$

Intuitively, if a room-time pair (r,t), disallowed by c, is often used ($M_{\text{room}}(r,t)$ close to 1), then $\big(1 - M_{\text{room}}(r,t)\big)$ is small, so it contributes little to the cost. On the other hand, if (r,t) is nearly free ($M_{\text{room}}(r,t) \approx 0$), then $\big(1-0\big) = 1$ inflates the cost, because the disallowed slot could otherwise, probabilistically, have been used.

2. Time Slot Constraints (Binary Cut).
Next, suppose constraint c imposes that an activity can only occur in a subset of time slots $\mathcal{T}(c) \subseteq \mathcal{T}$. Then all $t \notin \mathcal{T}(c)$ become disallowed. The cost of c is found by summing across all rooms in these disallowed slots:

$$\text{cost}(c) \;=\; \sum_{t \in \mathcal{T} \setminus \mathcal{T}(c)} \; \sum_{r \,\in\, \mathcal{R}} \Big[1 - M_{\text{room}}(r,t)\Big].$$

Intuitively, if rooms at slot t are nearly fully occupied, then the cost contributed by disallowing t is relatively small. On the other hand, if the room-time slots are nearly empty, the cost is relatively high.

3. Professor Constraints (Matrix-Based).
If constraint c relates to one or more professors in $\mathcal{P}(c) \subseteq \mathcal{P}$, then we combine multiple professors (an OR-like operation, described in Section 5.2) by computing:

$$\text{professor\_occupation\_adjusted}(c,t) \;=\; \prod_{p \,\in\, \mathcal{P}(c)} M_{\text{professor}}(p,t).$$

Separately we define

$$\text{freeRoom}(t) \;=\; \sum_{r \in \mathcal{R}} \Big(1 - M_{\text{room}}(r,t)\Big),$$

representing the total unused capacity across all rooms at slot t. The professor cost is then:

$$\text{cost}(c) \;=\; \sum_{t \,\in\, \mathcal{T}} \Big[\text{freeRoom}(t) \times \text{professor\_occupation\_adjusted}(c,t)\Big].$$

Intuitively, if professors in $\mathcal{P}(c)$ are collectively busy (high occupancy) while many rooms at slot t are free, the term $\text{professor\_occupation\_adjusted}(c,t) \times \text{freeRoom}(t)$ is large, showing that c is blocking a potentially good scheduling slot. If professors are mostly free, this product is low.

4. Student Group Constraints (Matrix-Based).
Finally, if c involves student groups $\mathcal{G}(c) \subseteq \mathcal{G}$, we use the AND-like combination (described in Section 5.2) of their occupation probabilities. We define:

$$\text{group\_occupation\_adjusted}(c,t) \;=\; 1 \;-\; \prod_{g \in \mathcal{G}(c)} \Big[1 - M_{\text{student}}(g,t)\Big].$$

Separately we define

$$\text{freeRoom}(t) \;=\; \sum_{r \in \mathcal{R}} \Big(1 - M_{\text{room}}(r,t)\Big),$$

representing the total "unused" capacity across all rooms at slot t. The student-group cost is:

$$\text{cost}(c) \;=\; \sum_{t \in \mathcal{T}} \Big[\text{freeRoom}(t) \times \text{group\_occupation\_adjusted}(c,t)\Big].$$

Intuitively, if at least one group in $\mathcal{G}(c)$ is busy while numerous rooms are free at t, this raises the cost because c is blocking a potentially good scheduling slot.

Conclusion. Each constraint c results in a scalar $\text{cost}(c)$, describing how heavily it disallows potentially viable room-time slots under the final probabilistic schedule. By separating the cost calculations for all constraints, one can easily assess which constraints are the primary drivers of scheduling difficulty.

5.3.1 Illustrative Example for Constaint Cost Calculation

Suppose a constraint c says an activity can be taught by either professor T_1 or T_2 . Let there be two time slots, $\mathcal{P} = \{1, 2\}$, and two rooms, $\mathcal{R} = \{R_1, R_2\}$. From the final *professor-time* matrix $M_{\text{professor}}(p,t) \in [0,1]$ (the probability that professor p is busy at slot t), suppose:

$$M_{\text{professor}}(P_1, 1) = 0.9, \quad M_{\text{professor}}(P_1, 2) = 0.5,$$

$$M_{\text{professor}}(P_2, 1) = 0.6, \quad M_{\text{professor}}(P_2, 2) = 0.4.$$

Because this constraint has an *OR* requirement, its combined professor occupancy for slot t is:

$$\text{professor\_occupation\_adjusted}(c,t) \;=\; 1 \;-\; \Big(1 - M_{\text{professor}}(P_1,t)\Big) \;\times\; \Big(1 - M_{\text{professor}}(P_2,t)\Big).$$

Hence:

$$\text{professors\_occupation\_adjusted}(c,1) = 1 - (1 - 0.9)\,(1 - 0.6) = 0.96,$$

$$\text{professors\_occupation\_adjusted}(c,2) = 1 - (1 - 0.5)\,(1 - 0.4) = 0.70.$$

Next, consider the final *room-time* matrix $M_{\text{room}}(r,t)$, which might look like:

	$t=1$	$t=2$
$M_{\text{room}}(R_1,t)$	0.8	0.2
$M_{\text{room}}(R_2,t)$	0.7	0.9

Define

$$\text{freeRoom}(t) \;=\; \sum_{r \in \mathcal{R}} \big[\,1 - M_{\text{room}}(r,t)\big],$$

the total "unused" capacity across all rooms at slot t. Thus:

$$\mathsf{freeRoom}(1) = \big(1 - 0.8\big) + \big(1 - 0.7\big) = 0.5,$$

$$\mathsf{freeRoom}(2) = \big(1 - 0.2\big) + \big(1 - 0.9\big) = 0.9.$$

The final cost for this constraint c is:

$$\mathsf{cost}(c) \;=\; \sum_{t \in \{1,2\}} \Big[\mathsf{freeRoom}(t) \;\times\; \mathsf{professors\_occupation\_adjusted}(c,t) \Big].$$

When we fill in the numbers we get:

$$\mathsf{cost}(c) = \big(0.5 \times 0.96\big) \;+\; \big(0.9 \times 0.70\big) = 1.11.$$

Thus, the total professor cost is 1.11. This indicates that, due to this constraint, we effectively lose 1.11 feasible room–time slot combinations from the solution space when attempting to schedule this activity.

5.4 Step 3b: Shapley Value Constraint Cost Evaluation

The stand-alone cost allocation method we described in Section 5.3 does not take into account the interplay between constraint costs within the activity. To fairly allocate constraint costs, while keeping into account the interplay, the Shapley value approach can be used. It considers every possible subset (coalition) of constraints and calculates the marginal contribution of each constraint. The results in Section 6 will show whether the Shapley values or the stand-alone constraint cost method will be better.

To calculate constraint costs using the Shapley method, we have applied the formula presented in Section 2.2.1, let $\mathcal{S}$ represent a subset of constraints and $Cost(\mathcal{S})$ the cost for the subset. The Shapley value ϕ_i for constraint i is computed as:

$$\phi_i = \sum_{\mathcal{S} \subseteq \mathcal{C} \setminus \{i\}} \frac{|\mathcal{S}|!(|\mathcal{C}| - |\mathcal{S}| - 1)!}{|\mathcal{C}|!} \left[Cost(\mathcal{S} \cup \{i\}) - Cost(\mathcal{S}) \right] \tag{17}$$

Pseudo-Code for Cost Evaluation The pseudo-code in Figure 3 clarifies the evaluation and allocation of constraint costs using Shapley values.

```
procedure EVALUATE_CONSTRAINT_COSTS(
    M_room, M_professor, M_student, activities, constraints
):
    costs = {}
    for activity in activities:
        total_occupancy = sum(1 - M_room)
        coalition_costs = {}
        for coalition in powerset(constraints):
            occupancy_without_coalition = recompute_occupancy(
                M_room, M_professor, M_student, activity, coalition
            )
            coalition_costs[coalition] = total_occupancy - occupancy_without_coalition

        shapley_values = {}
        for constraint in constraints:
            shapley_values[constraint] = compute_shapley_value(
                constraint, coalition_costs, constraints
            )
        costs[activity] = shapley_values

    return costs
end procedure
```

Figure 3: Psuedo code for the Shapley constraint cost evaluation method.

Here, 'recompute_occupancy' recalculates occupancy probabilities after removing the constraint we are measuring, and 'compute_shapley_value' calculates the marginal contributions according to the Shapley formula, Formula 5.4 .

5.4.1 Illustrative Example of Shapley Value Constraints Cost Calculation

Consider a small scenario with:

- One room, Room 1.
- Two time slots, $\{8{:}00,\ 9{:}00\}$.
- One activity containing two constraints, C_1 and C_2, of which each partially disallows usage of Room 1 in different time slots:
    1. C_1: "Room 1 is partially unavailable at 8:00."
    2. C_2: "Room 1 is partially unavailable at 9:00."

Let $M_{\text{room}}(\text{Room } 1, t) \in [0, 1]$ be the *final* occupancy probability under any chosen subset of constraints. We measure *cost* by

$$\text{Cost}(\mathcal{S}) \;=\; \sum_{t \,\in\, \{8:00,\, 9:00\}} \Big[\, 1 \;-\; M_{\text{room}}\big(\text{Room } 1, t; \mathcal{S}\big) \Big],$$

where $M_{\text{room}}(\cdot; \mathcal{S})$ is the occupancy after applying the constraint set $\mathcal{S}$.

Occupancy Under Each Subset of Constraints. We examine all subsets of $\{C_1, C_2\}$:

1. $\varnothing$ (no constraints): $M_{\text{room}}(\text{Room 1}, 8{:}00) = 0.6,\ M_{\text{room}}(\text{Room 1}, 9{:}00) = 0.7$. Hence,
$$\text{Cost}(\varnothing) = \big[1 - 0.6\big] + \big[1 - 0.7\big] = 0.7.$$
2. $\{C_1\}$ (only the first constraint): $M_{\text{room}}(\text{Room 1}, 8:00) = 0.2,\ M_{\text{room}}(\text{Room 1}, 9:00) = 0.7$. Thus,
$$\text{Cost}(\{C_1\}) = \big[1 - 0.2\big] + \big[1 - 0.7\big] = 1.1.$$
3. $\{C_2\}$ (only the second constraint): $M_{\text{room}}(\text{Room 1}, 8{:}00) = 0.6,\ M_{\text{room}}(\text{Room 1}, 9{:}00) = 0.4$. Hence,
$$\text{Cost}(\{C_2\}) = \big[1 - 0.6\big] + \big[1 - 0.4\big] = 1.0.$$
4. $\{C_1, C_2\}$ (both constraints): $M_{\text{room}}(\text{Room 1}, 8{:}00) = 0.2,\ M_{\text{room}}(\text{Room 1}, 9{:}00) = 0.1$. Then,
$$\text{Cost}\big(\{C_1, C_2\}\big) = \big[1 - 0.2\big] + \big[1 - 0.1\big] = 1.7.$$

Computing Shapley Values. We denote $\mathcal{C} = \{C_1, C_2\}$. For each constraint $i \in \mathcal{C}$, the Shapley value is

$$\phi_i \ = \sum_{\mathcal{S} \subseteq \mathcal{C} \setminus \{i\}} \frac{|\mathcal{S}|!\,\big(|\mathcal{C}| - |\mathcal{S}| - 1\big)!}{|\mathcal{C}|!} \Big[\text{Cost}\big(\mathcal{S} \cup \{i\}\big) - \text{Cost}\big(\mathcal{S}\big)\Big].$$

For $\boldsymbol{C_1}$:

- $\mathcal{S} = \varnothing$: $|\varnothing| = 0,\ (|\mathcal{C}| - |\varnothing| - 1)! = 1!,\ \frac{0!\times 1!}{2!} = \frac{1\times 1}{2} = 0.5,\ \text{Cost}(\{C_1\}) - \text{Cost}(\varnothing) = 1.1 - 0.7 = 0.4$. Weighted contribution: $0.5 \times 0.4 = 0.20$.
- $\mathcal{S} = \{C_2\}$: $|\mathcal{S}| = 1,\ (2 - 1 - 1)! = 0! = 1,\ \frac{1!\times 1}{2!} = \frac{1}{2} = 0.5,\ \text{Cost}(\{C_1, C_2\}) - \text{Cost}(\{C_2\}) = 1.7 - 1.0 = 0.7$. Weighted contribution: $0.5 \times 0.7 = 0.35$.

Hence, $\phi_{C_1} = 0.20 + 0.35 = 0.55$.
For $\boldsymbol{C_2}$:

- $\mathcal{S} = \varnothing$: $\frac{0!\times 1!}{2!} = 0.5,\ \text{Cost}(\{C_2\}) - \text{Cost}(\varnothing) = 1.0 - 0.7 = 0.3$. Weighted contribution: $0.5 \times 0.3 = 0.15$.
- $\mathcal{S} = \{C_1\}$: $\frac{1!\times 1}{2!} = 0.5,\ \text{Cost}(\{C_1, C_2\}) - \text{Cost}(\{C_1\}) = 1.7 - 1.1 = 0.6$. Weighted contribution: $0.5 \times 0.6 = 0.30$.

Hence, $\phi_{C_2} = 0.15 + 0.30 = 0.45$.
Note that $\phi_{C_1} + \phi_{C_2} = 1.0$, which matches the total increase in cost from $\varnothing$ to $\{C_1, C_2\}$ $(1.7 - 0.7 = 1.0)$. This is because the sum of individual Shapley values always gives full contribution of the whole coalition score. Constraint C_1 has a slightly higher Shapley value, $\phi_{C_1} = 0.55$ versus $\phi_{C_2} = 0.45$, indicating that C_1 is more restricting.

5.5 Why the Order of Activities Does Not Matter

Because WSA relies on probability distributions and normalizations (rather than deterministic sequence-based decisions), the final outcome does not depend on the order in which activities are processed. A detailed mathematical proof can be found in Appendix A, but the intuition is straightforward: Probability distributions do not carry inherent sequence information, so reordering the addition of activities will ultimately result in the same final occupancy matrices.

6 Results

In theory, after considering all requirements fairly, the presented method should indicate well how costly the constraints are on the whole solution space. However, to determine whether this holds true emperically, it is necessary to set up test cases.

6.1 Exhaustive Search Backtest

To validate the proposed method we use use an Exhaustive Search (ES) method, where we compare by counting the total number of possible schedules. This allows us to precisely measure how the solution space expands when we relax a constraint. However, because ES is computationally expensive as it grows exponentially with the number of activities added, we can only apply this method to tiny instances, as is shown in Figure 7a. Despite this limitation, the ES method provides an accurate and reliable indication of whether our results align with the entire solution space shrinkage.
That being said, generalizing the results from a sub-problem to the entire problem is incorrect. This generalization has been the recent downfall of Shapley values utility in determining the singled out input feature effects from the coalition effects [13]. This finding highlights the importance of running estimation methods on the entire problem set, to understand how much the solution space shrinks when a constraint is relaxed.

Table 7: Example of a Problem description of the UCTP

activity	room	slots	groups	teachers
Activity 1	a,b	1,2,3	a,b	A,B
Activity 2	b	1,2,3	b	B
Activity 3	a	3	b	A,B
Activity 4	a,b	2,3	a	A, B
Activity 5	a	1,2	b	A, B

Table 8: Cost matrix when evaluated by ES method, costly constraints are more red.

activity	room	slots	groups	teachers
Activity 1	32	32	32	32
Activity 2	64	32	136	64
Activity 3	64	64	80	32
Activity 4	32	48	32	32
Activity 5	64	32	104	32

Table 7 provides an overview of the problem constraints, including room assignments, slots, groups, and teachers for various activities. Table 8 presents the number of schedules available when relaxing constraints using the ES method. Table 9 displays the Stand-Alone cost assignment by our algorithm.
Although these values are not directly comparable, they are ordinally significant, allowing for the calculation of the Spearman rank correlation. The current rank correlation between Matrix 8 and Matrix 9 is 0.79.

Table 9: Cost matrix when evaluated by the WSA method (using the Stand-Alone cost evaluation), costly constraints are more red.

activity	room	slots	groups	teachers
Activity 1	0.00	0.00	0.61	0.00
Activity 2	0.90	0.00	1.80	0.01
Activity 3	1.56	1.79	1.80	0.00
Activity 4	0.00	1.30	0.79	0.00
Activity 5	1.56	0.67	1.80	0.00

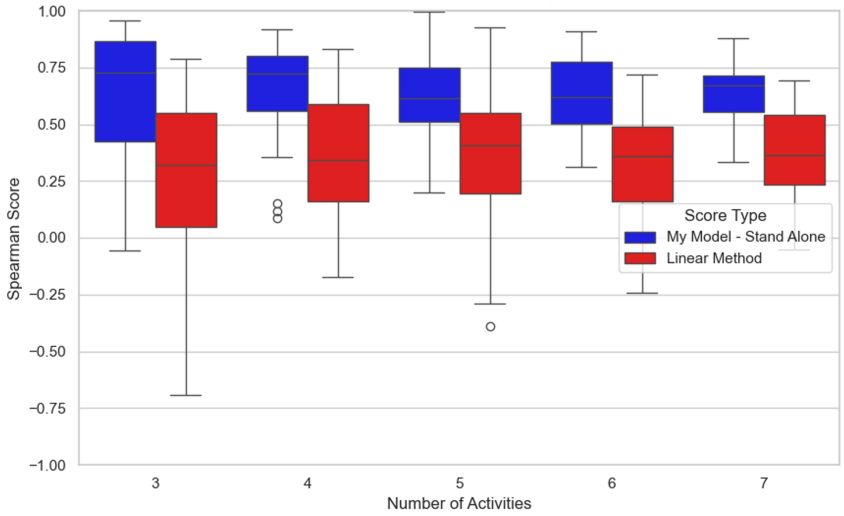


Figure 4: Spearman rank correlation for a 100 runs of each problem size. These rank correlations are calculated as explained in Table 7. 8, and 9. Ultimately we compare the rank correlation of WSA (using the Stand-Alone method) with ES, against the rank correlation of a liniear method with ES.

.

6.1.1 WSA versus a Linear Method

The results in Figure 4 indicate a generally positive rank correlation, demonstrating that our method effectively estimates constraint costs (i.e., solution space shrinkage). To better understand the significance of this score, we compared it to the current analytic method employed by TimeEdit, which utilizes a linear model. The primary heuristic used to determine which constraints to relax is based on a linear demand for resources (e.g., identifying whether a particular teacher is assigned an excessive number of lessons). A challenge arises in this process when aggregating individual resource occupations into a singular cost value. This kind of procedure was also applied to the WSA method, as described in Section 5.2. Since this step is non-trivial, we conducted an empirical analysis to identify the most effective approach. Our findings indicate that the optimal linear method for this problem set involves computing the *maximum* cost of the resource set when faced with an OR operation, while employing the *additive* cost function for the set under an AND operation.

6.1.2 Shapley versus Stand-Alone Constraint Cost Evaluation Method

The results in Figure 5 Show us that the Shapley method follows a trend to outperfrom the Stand Alone method on bigger instances. As the problem size increases, the observed variability in the outcomes declines in accordance with the principle commonly attributed to the law of large numbers [11]. The average Spearman score of 0.81 for the Shapley method shows our method, on the biggest tested problem, gives a good indication of what constraints restrict the solution space the most.

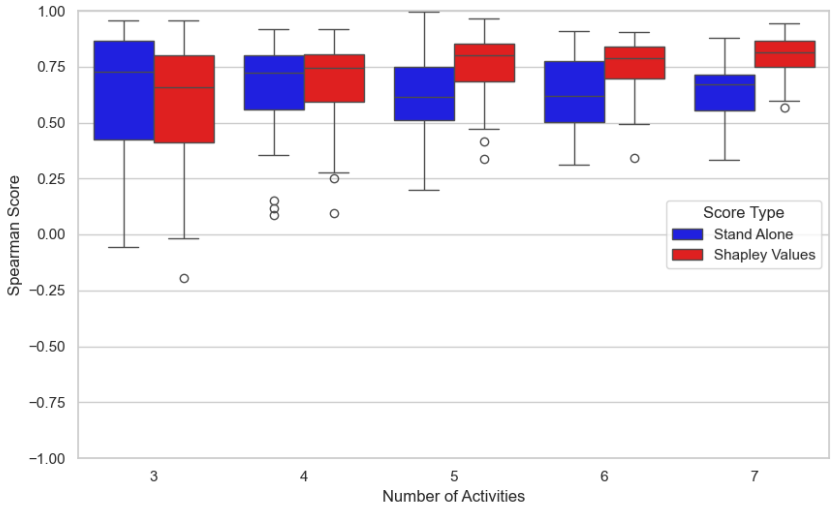


Figure 5: Spearman rank correlation for a 100 runs of each problem size. These rank correlations are calculated as explained in Table 7. 8, and 9. Ultimately we compare the rank correlation of WSA, while using the Stand-Alone method, with ES, against the rank correlation of WSA, while using the Shapley method, with ES.

6.2 Bucket Comparison

In this section, we analyze the effect of bucketed constraints on scheduling performance. The Bucketing method is introduced in Section 2.2.2. Specifically, we categorize the ranking of cost constraints determined by our algorithm into distinct buckets. The goal is to observe whether relaxing constraints in the higher-cost buckets leads to increasingly easier scheduling problems. To estimate the impact of these buckets, we evaluate their performance with a heuristic scheduling algorithm. This methods, while useful, can introduce certain biases, as discussed in Section 4. By analyzing the average scheduling scores across different buckets, we aim to validate whether WSAs ranking of constraint costs correctly identifies the most costly constraints (i.e., those that make the problem significantly harder when enforced).
For this experiment, we utilized data provided by TimeEdit, representing a real-world scheduling problem for the 2024/2025 academic year at one of their university clients. The dataset spans a total of 57 weeks, where each week is considered a separate scheduling problem.
On average, a single week in this dataset consists of:

- 3,276 activities,
- 93 rooms,
- 653 teachers,
- 122 time slots,
- 89 student groups.

This dataset provides a comprehensive and realistic basis for evaluating scheduling methodologies. The downward trend in Figure 6 confirms that the constraints identified as more costly make the problem harder, because they result in significantly better scheduling scores when relaxed. In Appendix B we show which type of constraints are most constraining for this specific UCTP.

6.3 Time Complexity

We have performed emperical tests on the computational time in Figure 7 and Figure 8.These tests align with theoretical analyses.

Exhaustive Search The Exhaustive Search validation method, where all possible combinations of activities, rooms, time slots, teachers, and student groups are generated and validated, has a complexity of:

$$O((RTPG)^N) \tag{18}$$

where R, T, P, and G denote rooms, time-slots, professors, and groups, respectively, and N is the number of activities. This confirms the rapid growth of possible schedules with increasing N.

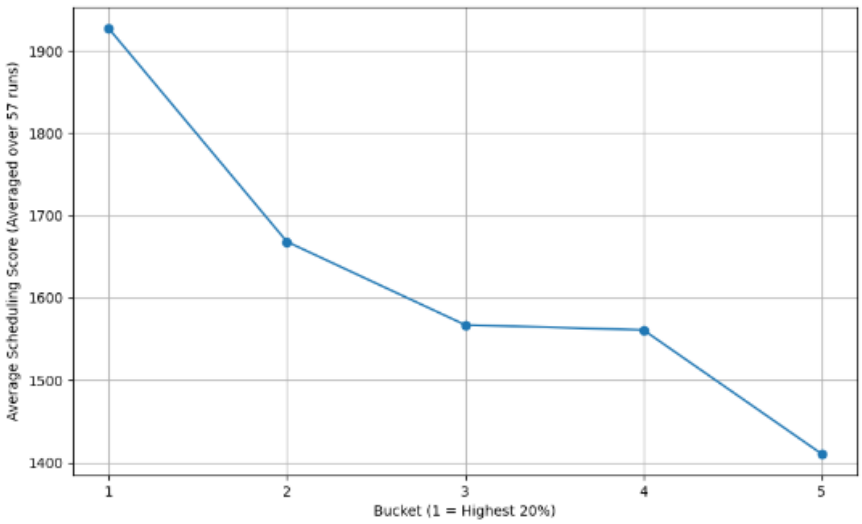


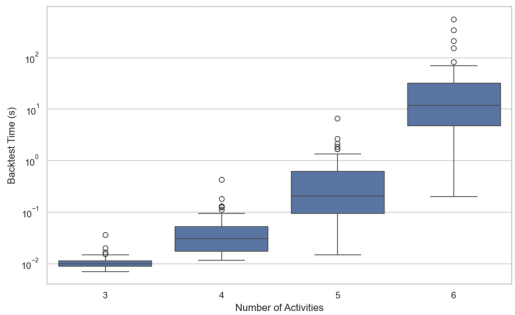
Figure 6: Average scheduling scores across constraint buckets. The x-axis represents the constraint bucket (1 = highest 20%), while the y-axis shows the average scheduling score over all 57 weeks in the dataset. The score is calculated as the amount of activities that could be scheduled when using a greedy heuristic schedular, which schedules most difficult activities first. The activities are ranked by the formula: x $= \|slots\| + \|teachers\| + \|rooms\| - \|groups\|$

WSA The WSA algorithm comprises three steps, with the second step, Scheduling, being the most computationally intensive by far. The other two steps contribute negligibly to the overall complexity and are therefor excluded from this analysis. The complexity of the Scheduling step is expressed as:

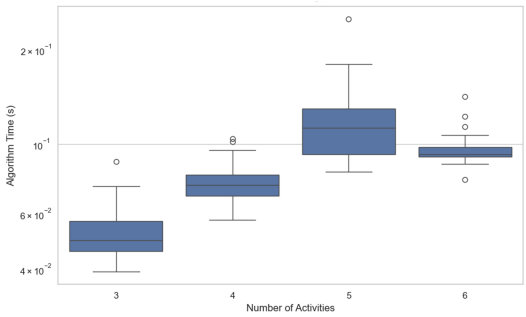
$$O(N \cdot R \cdot T \cdot P \cdot G) \tag{19}$$

In a UCTP, when N grows, the other variables do not grow as hard, as for each activity, we do not necessarily need a new room, time-slot, profesor, or student group. Figure 8 shows that the algorithm best fits a quadratic time complexity, modeled as:

$$T(n) = an^2 + b, \quad a = 0.000537, \quad b = -6.507427 \tag{20}$$



(a) Computational time for ES vs. number of activities in UCTP.



(b) Computational time for WSA vs. number of activities in UCTP.

Figure 7: (a) ES runtimes and (b) WSA runtimes for UCTP. The exponential increase in (a) confirms theoretical complexity of the ES method, making exhaustive search infeasible for large problems. These scores are averaged over 50 runs.

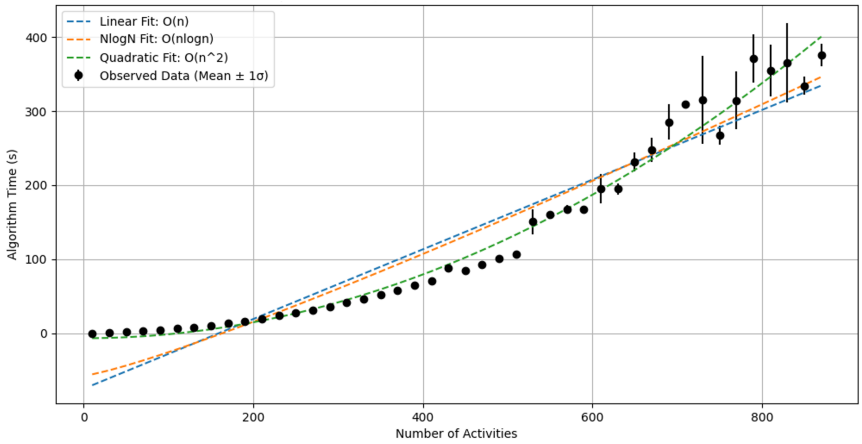


Figure 8: Computational time for our algorithm vs. number of activities in UCTP, averaged over 10 runs. The best-fit complexity function follows $O(n^2)$.

7 Discussion

The results of this study provide significant insights into the constraint cost evaluation problem within the UCTP. The proposed method WSA offers a novel and relatively computationally efficient approach for assessing constraint costs.

7.1 Implications of Findings

The empirical validation demonstrates that WSA achieves a high correlation with the exhaustive search method in evaluating constraint costs. This strongly suggests that WSA provides a reliable approximation of constraint costs without requiring an impractical computational burden. Unlike traditional heuristic-based methods used in industry (e.g., TimeEdit's linear heuristic model), WSA provides a more balanced and theoretically grounded approach to constraint evaluation. Traditional heuristics often rely on domain expertise and human intuition, which lead to inconsistencies and inefficiencies. By contrast, WSA provides a more systematic and quantitative measure of constraint costs.
Additionally, the results indicate that the bucket comparison method successfully categorizes constraints into meaningful cost tiers. This finding is particularly relevant for real-world implementations, as it enables decision-makers to distinguish costly constraints more effectively.

7.2 Limitations and Challenges

While the proposed algorithm significantly enhances constraint cost evaluation, certain limitations must be acknowledged:

- **Complexity:** The algorithm is still in the order of $O(n^2)$, which is not bad, given that we are representing the problem as matrices. However, some improvements in matrix calculations could be found here.
- **Consistent Room-Time-Slot Constraint** We do not have sufficient computational resources to execute the WSA algorithm on the complete university dataset in a single run. To overcome this, we treat each week as a separate instance of the UCTP. This approach does not consider the soft constraint of assigning the same course to the same room and time slot each week. This leads to a slight loss in accuracy with respect to real-world applicability.
- **Traveling Ability:** In the real world, students and teachers have the ability to travel to different campuses, given that the break between activities is long enough. In this thesis, we ignored this ability as we did not have access to the traveling times between campuses. This leads to a slight loss in accuracy with respect to real-world applicability.

8 Conclusion

This thesis addressed the challenge of efficiently estimating constraint costs in the UCTP, a critical issue faced by academic institutions and scheduling software providers. We introduced the Weighted Scheduling Algorithm (WSA), a novel approach that fairly evaluates constraint costs through probabilistic scheduling and Shapley value attribution.

8.1 Key Contributions

This thesis has the following contributions to the existing UCTP literature:

- Developed WSA, a scheduling model-agnostic approach that assigns probabilistic weights to constraint costs, avoiding biases inherent in heuristic-based methods.
- Demonstrated through empirical testing that WSA strongly correlates with exhaustive search-based constraint cost evaluations, validating its effectiveness.
- Demonstrated through empirical testing that Shapley constraint cost estimations outperform Stand-Alone constraint cost estimations while using WSA.
- Showed that constraint cost rankings obtained using WSA align well with real-world scheduling scenarios, as demonstrated in the bucket comparison analysis.
- Reasoned that a lot of approaches are inaccurate or inefficient for evaluating constraint costs.

8.2 Future Work

Several promising directions for future research arise from this study:

- **More Benchmarks on Real-World Problems:** A lot of the experimental validation was performed on simulated datasets, with one test on a real-world problem set. While the results strongly support the effectiveness of WSA, additional validation on real-world university timetabling datasets would further strengthen confidence in its applicability.
- **Benchmark on a Real-World Pipeline for the UCTP:** To determine the effectiveness of WSA in aiding to reach a feasible schedule, we should conduct a comparative study using real scheduling scenarios. This would involve creating two versions of the schedule, one where the team utilizes WSA and another where they use their regular methods. We would evaluate them by comparing the feasibility and quality of the resulting schedule, as well as measuring how quickly the team arrives at a feasible solution.

8.3 Final Remarks

By bridging the gap between theoretical scheduling research and practical implementation, this thesis provides a significant step toward improving constraint cost estimation in UCTP. The introduction of WSA opens a novel method for reducing reliance on iterative human input and aiding to more informed decision-making in (academic) scheduling. As universities continue to expand and scheduling complexities increase, these kinds of algorithmic advancements will play

a crucial role in ensuring efficient and reliable course timetabling.

In the near future, this work will enable TimeEdit to more effectively determine which constraints to relax, ultimately leading to improved and faster scheduling for universities worldwide, as our empirical results demonstrate that, across a wide range of scheduling scenarios, WSA outperforms the linear method previously employed by TimeEdit.

References

[1] Isaac Braam, Jasper Kousen, and Koen Ripping. Shapley attribution in machine learning trading models.

[2] Edmund Kieran Burke and Sanja Petrovic. Recent research directions in automated timetabling. 140(2):266–280.

[3] Mei Ching Chen, San Nah Sze, Say Leng Goh, Nasser R. Sabar, and Graham Kendall. A survey of university course timetabling problem: Perspectives, trends and opportunities. *IEEE Access*, 9:106515–106529, 2021.

[4] David Corne and John Ogden. Evolutionary optimisation of methodist preaching timetables. In Edmund Burke and Michael Carter, editors, *Practice and Theory of Automated Timetabling II*, volume 1408, pages 142–155. Springer Berlin Heidelberg. Series Title: Lecture Notes in Computer Science.

[5] David Corne, Peter Ross, and Edmund Burke. Peckish initialisation strategies for evolutionary timetabling. Lecture Notes in Computer Science, pages 227–240.

[6] Luca Di Gaspero and Andrea Schaerf. *Multi-Neighbourhood Local Search with Application to Course Timetabling*. Pages: 275.

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

[8] Alexandre Fréchette, Lars Kotthoff, Tomasz Michalak, Talal Rahwan, Holger Hoos, and Kevin Leyton-Brown. Using the shapley value to analyze algorithm portfolios. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30, 03 2016.

[9] FTSE Russell. Russell growth and value indexes: The enduring utility of style. Technical report, FTSE Russell, 2021. Accessed: March 16, 2025.

[10] Amin Jamili, Mahdi Hamid, Hassan Gharoun, and Sajjad Khoshnoudi. *Developing a Comprehensive and Multi-Objective Mathematical Model for University Course Timetabling Problem: A Real Case Study*.

[11] Andrey Nikolaevich Kolmogorov. *Grundbegriffe der Wahrscheinlichkeitsrechnung*. Springer Berlin Heidelberg, 1933.

[12] Kaiwen Li, Tao Zhang, Rui Wang, Yuheng Wang, Yi Han, and Ling Wang. Deep reinforcement learning for combinatorial optimization: Covering salesman problems. *IEEE Transactions on Cybernetics*, 52(12):13142–13155, December 2022.

[13] Joao Marques-Silva and Xuanxiang Huang. Explainability is not a game, 2024.

[14] Barry Mccollum. University timetabling: Bridging the gap between research and practice.

[15] Christopher Musco and R. Teal Witter. Provably accurate shapley value estimation via leverage score sampling. *arXiv preprint arXiv:2410.01917*, 2024.

[16] Siddhartha Roy and Runa Ganguli. A study on course timetable scheduling using graph coloring approach. 12(2):469–485.

[17] A. Schaerf. A survey of automated timetabling. 13(2):87–127.

[18] L. S. Shapley. 17. a value for n-person games. In Harold William Kuhn and Albert William Tucker, editors, *Contributions to the Theory of Games (AM-28), Volume II*, pages 307–318. Princeton University Press.

[19] J. Silva, Edmund Burke, and Sanja Petrovic. An introduction to multiobjective metaheuristics for scheduling and timetabling. pages 91–129.

[20] J. Silva, Edmund Burke, and Sanja Petrovic. *An Introduction to Multiobjective Metaheuristics for Scheduling and Timetabling*, pages 91–129. 01 2004.

[21] Rostislav Staněk, Peter Greistorfer, Klaus Ladner, and Ulrich Pferschy. Geometric and lp-based heuristics for angular travelling salesman problems in the plane. *Computers Operations Research*, 108:97–111, 2019.

[22] Thatchai Thepphakorn, Pupong Pongcharoen, and Srisatja Vitayasak. A new multiple objective cuckoo search for university course timetabling problem. In Chattrakul Sombattheera, Frieder Stolzenburg, Fangzhen Lin, and Abhaya Nayak, editors, *Multi-disciplinary Trends in Artificial Intelligence*, pages 196–207, Cham, 2016. Springer International Publishing.

[23] John Van Den Broek, Cor Hurkens, and Gerhard Woeginger. Timetabling problems at the TU eindhoven. 196(3):877–885.

[24] Anthony Wren, Peter Ross, and Edmund Burke. Scheduling, timetabling and rostering — a special relationship? Lecture Notes in Computer Science, pages 46–75.

[25] Andrea Zampieri and Andrea Schaerf. Modelling and solving the italian examination timetabling problem using tabu search.

A Mathematical Proof of Order Invariance

Mathematical Proof of order Invariance in Weighted Scheduling Algorithm (WSA)
Let:

- $A = \{A_1, A_2, ..., A_n\}$ be the set of n activities.
- $T = \{T_1, T_2, ..., T_m\}$ be the set of m available room-time slots.
- $P(A_i \to T_j)$ be the probability of assigning activity A_i to room-time slot T_j.

In a deterministic scheduling model, we seek a permutation σ of A, meaning an ordered assignment $(\sigma(A_1), \sigma(A_2), ..., \sigma(A_n))$. However, in WSA, assignments are made probabilistically without enforcing a sequence.

Step 2: Independence from Order in Probability Theory
WSA operates under the assumption that each activity A_i is assigned to a time slot according to some probability set. This can be written as:

$$P(A_1 \to T_{j_1}, A_2 \to T_{j_2}, ..., A_n \to T_{j_n}) = P(A_1 \to T_{j_1})P(A_2 \to T_{j_2})...P(A_n \to T_{j_n}) \quad (21)$$

This holds under the assumption that individual assignment probabilities are weighted according to constraints. The key observation here is that the joint probability remains the same regardless of the order in which activities are assigned, meaning that reordering the sequence does not change the resulting probability set.
For any two permutations σ and π of $\{1, 2, ..., n\}$:

$$P(A_{\sigma(1)} \to T_{j_1}, A_{\sigma(2)} \to T_{j_2}, ..., A_{\sigma(n)} \to T_{j_n}) = P(A_{\pi(1)} \to T_{j_1}, A_{\pi(2)} \to T_{j_2}, ..., A_{\pi(n)} \to T_{j_n}) \quad (22)$$

Since probabilities are independent of order, we conclude that the scheduling process in WSA does not depend on the sequence in which activities are considered.

B Most Constraining Constraint Types

In Figure 9 we find that student groups are overall the most costly constraint; this is in line with TimeEdits prediction. Rooms are also a costly constraint, this is mostly due to the fact that big groups can't get a lecture in a small room, of which there are a lot of in the dataset. Slots and teacher constraints are not that constraining in this UCTP, as the slot constraints weren't that narrow, and there were a lot of teachers.

C Algorithm Steps Example

In this example we have divided every activity's weight by 10 to avoid overly normalizing the matrices. This is not needed for larger UCTPs.

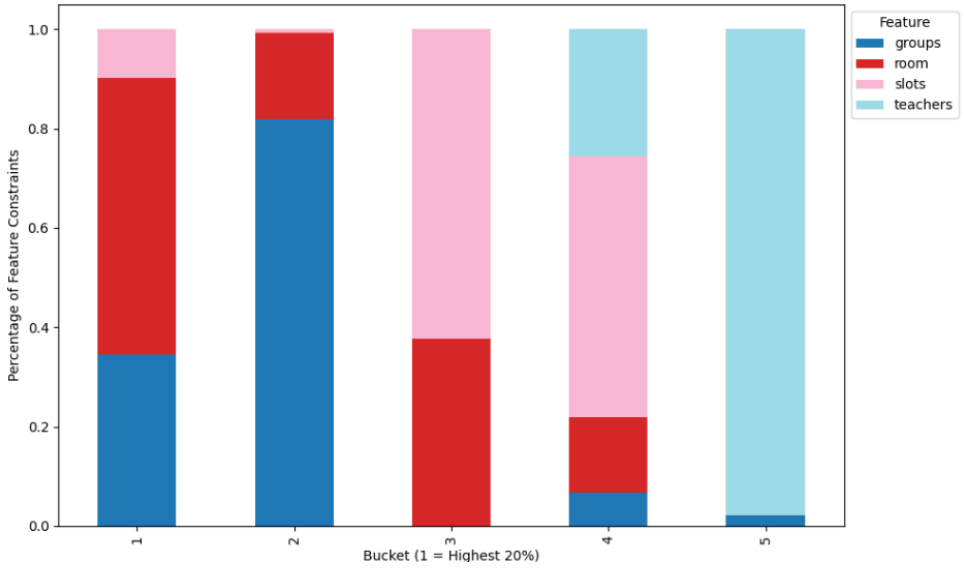


Figure 9: Distribution of types of constraint, binned by their scheduling costs across constraint buckets. The x-axis represents the constraint bucket (1 = highest 20%), while the y-axis shows the distribution of type of constraint averaged over all 57 weeks in the dataset. The costs are determined by WSA.

Activity	Room	Slots	Groups	Teachers
Activity 1	[b]	[s1]	[a, b]	[A]
Activity 2	[b]	[s1, s2]	[a, b]	[A, B]
Activity 3	[a, b]	[s1, s2]	[a]	[A, B]

Table 10: Activity Data

	s1	s2
a	0.0	0.0
b	0.0	0.0

Table 11: Initial Room Time Matrix

	s1	s2
A	0.0	0.0
B	0.0	0.0

Table 12: Initial Teacher Matrix

	s1	s2
a	0.0	0.0
b	0.0	0.0

Table 13: Initial Student Matrix

	s1	s2
a	0.0	0.0
b	0.1	0.0

Table 14: Room Time Matrix After Activity 1

	s1	s2
A	0.1	0.0
B	0.0	0.0

Table 15: Teacher Matrix After Activity 1

	s1	s2
a	0.1	0.0
b	0.1	0.0

Table 16: Student Matrix After Activity 1

	s1	s2
a	0.00000	0.00000
b	0.14158	0.05842

Table 17: Room Time Matrix After Activity 2

	s1	s2
A	0.12079	0.02921
B	0.02079	0.02921

Table 18: Teacher Matrix After Activity 2

	s1	s2
a	0.14158	0.05842
b	0.14158	0.05842

Table 19: Student Matrix After Activity 2

	s1	s2
a	0.024808	0.027759
b	0.162875	0.084558

Table 20: Room Time Matrix After Activity 3

	s1	s2
A	0.143842	0.056158
B	0.043842	0.056158

Table 21: Teacher Matrix After Activity 3

	s1	s2
a	0.187683	0.112317
b	0.141580	0.058420

Table 22: Student Matrix After Activity 3

	s1	s2
a	0.024808	0.027759
b	0.162875	0.084558

Table 23: Final Room Time Matrix

	s1	s2
A	0.143842	0.056158
B	0.043842	0.056158

Table 24: Final Teacher Matrix

	s1	s2
a	0.187683	0.112317
b	0.141580	0.058420

Table 25: Final Student Matrix

Activity	Room	Slots	Teachers	Groups
Activity 1	0.242884	0.353653	0.500060	0.624940
Activity 2	0.492937	0.000000	0.228616	1.178277
Activity 3	0.000000	0.000000	0.304821	1.446041

Table 26: Final constraint costs