**Assignment # 1**

**Summer Internship 2025**

**CETQAP**

****

**Name:** Hareem Fatima

**University:** NUST, Islamabad

**Degree:** BS Data Science

**Submission Date:** 2nd August, 2025

**Research Topic:** Quantum Optimization for University Timetabling

Contents

[Research Topics in Data Science 4](#_Toc204962163)

[Topic Selected 4](#_Toc204962164)

[History 4](#_Toc204962165)

[1. Manual Timetabling in the Pre-Digital Era (Before 1950s) 4](#_Toc204962166)

[2. The Emergence of Operations Research (1950s–1960s) 5](#_Toc204962167)

[ Mathematical modeling: 5](#_Toc204962168)

[ Integer programming: 5](#_Toc204962169)

[ Graph theory: 5](#_Toc204962170)

[3. Early Computerized Scheduling (1960s–1970s) 6](#_Toc204962171)

[ Backtracking algorithms: 6](#_Toc204962172)

[ Greedy algorithms: 6](#_Toc204962173)

[ Brute force search: 6](#_Toc204962174)

[4. Rise of Artificial Intelligence and Metaheuristics (1980s–1990s) 6](#_Toc204962175)

[ Genetic Algorithms (GAs): 6](#_Toc204962176)

[ Simulated Annealing: 7](#_Toc204962177)

[ Tabu Search: 7](#_Toc204962178)

[ Constraint Programming (CP): 7](#_Toc204962179)

[5. The Open Research and Hybrid Era (2000s–2010s) 7](#_Toc204962180)

[ International Timetabling Competitions (ITC): 7](#_Toc204962181)

[ Hybrid algorithms: 7](#_Toc204962182)

[**Toolkits and Libraries**: 8](#_Toc204962183)

[6. Machine Learning and Quantum Optimization Era (2010s–Present) 8](#_Toc204962184)

[ QUBO (Quadratic Unconstrained Binary Optimization) 8](#_Toc204962185)

[ Quantum annealers 8](#_Toc204962186)

[Introduction 9](#_Toc204962187)

[Why it is computationally hard? 9](#_Toc204962188)

[1. Large Number of Combinations 9](#_Toc204962189)

[2. Multiple Hard Constraints 9](#_Toc204962190)

[3. Soft Constraints Make It Even Harder 10](#_Toc204962191)

[4. No Efficient Algorithm Exists 10](#_Toc204962192)

[Experiments 10](#_Toc204962193)

[Graph Colouring for Scheduling 10](#_Toc204962194)

[Example: 11](#_Toc204962195)

[Greedy Search Algorithm: 12](#_Toc204962196)

[Working of the Algorithm: 13](#_Toc204962197)

[Advantages: 13](#_Toc204962198)

[Disadvantages: 14](#_Toc204962199)

[Stimulated Annealing 14](#_Toc204962200)

[Tabu Search 15](#_Toc204962201)

[Dataset: 17](#_Toc204962202)

[Variables: 17](#_Toc204962203)

[Complexity & Scalability 19](#_Toc204962204)

[Problem Statement: 20](#_Toc204962205)

[Solution: 21](#_Toc204962206)

[Discussion: 23](#_Toc204962207)

[Was Quantum Better than Classical? 23](#_Toc204962208)

[Scalability Issues 24](#_Toc204962209)

[Accuracy and Feasibility 24](#_Toc204962210)

[Real World Applicability 25](#_Toc204962211)

[Limitations 26](#_Toc204962212)

[Future Work: 26](#_Toc204962213)

[Conclusion 29](#_Toc204962214)

[References: 30](#_Toc204962215)

# Research Topics in Data Science

* Machine Learning
* **Optimization**
* **Data Analysis and Visualization**
* **Financial Modeling**
* **Cybersecurity**

Topic Selected

Optimization

**Subfield:** Quantum Optimization for University Timetabling

# History

The timetable scheduling problem (TSP) has evolved over decades, shaped by developments in mathematics, computer science, artificial intelligence, and most recently, quantum computing. The TSP, in the context of schools, universities, or other institutions, is about assigning events like lectures or exams to times and rooms while satisfying multiple constraints. Though this may seem like a routine administrative task, it quickly becomes mathematically and computationally complex as the size of the problem grows.

## 1. Manual Timetabling in the Pre-Digital Era (Before 1950s)

Before computers existed, scheduling was done entirely by hand. Institutions relied on individuals, typically administrative staff, head teachers, or department heads to manually allocate courses, teachers, rooms, and times to slots across a week.

This process involved:

* Drawing large grids on paper or boards.
* Assigning classes to available slots while cross referencing teacher availability and classroom usage.
* Constantly erasing and rewriting when conflicts arose.
* Using intuition to avoid back to back lectures or long gaps for teachers.

While this was manageable for small schools or departments, it became increasingly unmanageable as the number of constraints grew. Any change in one part of the timetable (e.g., a teacher's absence) could require a cascade of edits elsewhere. The absence of automation meant:

* High error rates.
* Inability to adapt quickly to changes.
* Heavy dependency on experienced human schedulers.

This labour intensive nature highlighted the need for a systematic and computational approach.

## 2. The Emergence of Operations Research (1950s–1960s)

The field of operations research (OR) developed during World War II to optimize military logistics, radar coverage, and transportation. After the war, these ideas were extended to civilian and industrial problems including scheduling.

Key developments in this period include:

* Mathematical modeling: Researchers began framing scheduling as a **constraint satisfaction problem**, where the goal was to assign resources (teachers, rooms, times) to tasks (classes) under specific conditions.
* Integer programming: Scheduling was formulated using integer variables to represent whether a class is scheduled in a specific slot. The objective could be to minimize conflicts or maximize preferences.
* Graph theory: Early attempts were made to model scheduling as graph colouring problems. For example, two conflicting events (e.g., classes requiring the same teacher) are connected by an edge, and the goal is to colour each node (time slot) such that no adjacent nodes have the same colour.

Despite these promising formulations, the limitations were:

* Lack of computational power.
* Inability of linear or integer programming to scale up due to exponential time complexity.
* Theoretical understanding that the problem falls into a class of problems now known as NP hard.

This phase laid the foundation for algorithmic thinking but lacked practical implementation.

## 3. Early Computerized Scheduling (1960s–1970s)

The introduction of computers transformed the landscape. Scheduling algorithms could now be executed systematically and rapidly.

Researchers started implementing basic algorithms, such as:

* Backtracking algorithms: These try to construct a solution step-by-step, backing up when a conflict is found. For example, schedule Class 1 in Room A at 9am then try Class 2 and if a conflict occurs, try a different slot or room.
* Greedy algorithms: These make locally optimal choices for instance, always scheduling the class with the fewest available slots first. These methods are fast but don’t always produce good solutions.
* Brute force search: Enumerate all possible schedules and select the best one. This was only feasible for extremely small instances.

Real world testing during this period confirmed the theoretical claim that scheduling is NP hard. As the number of variables increased, these algorithms quickly became too slow.

## 4. Rise of Artificial Intelligence and Metaheuristics (1980s–1990s)

The 1980s and 1990s saw a surge in research applying artificial intelligence (AI) techniques to scheduling problems. The focus shifted from finding **perfect (optimal)** solutions to finding **good enough (near optimal)** solutions efficiently.

Here, **metaheuristic algorithms** became dominant. These include:

* Genetic Algorithms (GAs): Inspired by evolution, these maintain a population of possible schedules and evolve them through crossover (combining features of two parents), mutation (random changes), and selection (keeping the best solutions).

**Example:** In a university timetable, one "chromosome" may represent a complete weekly schedule, and the algorithm would evolve over generations to reduce conflicts.

* Simulated Annealing: Based on metallurgy, this algorithm allows some bad changes early on to escape local optima and gradually reduces randomness to settle into a good solution.
* Tabu Search: Uses memory to avoid cycling back to recently visited solutions. For instance, it remembers that assigning Class X to Room Y at 10am recently led to a conflict, so avoids repeating it.
* Constraint Programming (CP): In CP, the user specifies constraints declaratively (e.g., “No two classes can occur in the same room at the same time”) and the solver automatically finds a schedule that satisfies all constraints.

This period also introduced **soft constraints** such as:

* Teachers preferring late morning slots.
* Students not wanting back to back lectures in distant buildings.
* Minimizing room changes.

## 5. The Open Research and Hybrid Era (2000s–2010s)

This era saw the emergence of global collaboration, standardized benchmarks and hybrid algorithms.

Key milestones include:

* International Timetabling Competitions (ITC): These events released real world datasets for school and university timetabling. Researchers used them to test and compare algorithms leading to major improvements.
* Hybrid algorithms: These combined multiple methods. For example, using a genetic algorithm to generate solutions and a local search algorithm to improve them. Another example is combining constraint programming with simulated annealing.

### **Toolkits and Libraries**:

* + Google OR-Tools: An open source library that supports CP, LP, and metaheuristics.
  + OptaPlanner: Java based constraint solver for employee rostering and course scheduling.
  + MiniZinc: A modeling language for constraint satisfaction problems.

This period also emphasized:

* **Adaptability:** Schedulers needed to quickly generate new schedules when unexpected events occurred.
* **Usability:** Generating readable, user friendly schedules that could be easily interpreted by staff and students.
* **Scalability:** Handling tens of thousands of events in large institutions.

## 6. Machine Learning and Quantum Optimization Era (2010s–Present)

In recent years, researchers have started integrating **machine learning** and exploring **quantum computing** for timetabling.

Machine learning contributions include:

* Learning teacher or student preferences automatically from past timetables.
* Predicting where conflicts are likely to occur.
* Using reinforcement learning to train an agent to make scheduling decisions based on rewards and penalties.

Quantum computing has introduced:

* QUBO (Quadratic Unconstrained Binary Optimization) formulations of scheduling problems.
* Quantum annealers (e.g., D-Wave systems) that attempt to find low energy (low conflict) solutions using quantum effects.
* These are still in experimental stages and limited to small problem sizes but offer a glimpse into the future of optimization.

Modern timetabling software is now expected to support:

* Multiple campuses.
* Blended learning (in person and online classes).
* Real time changes and rescheduling (e.g., due to COVID related closures or hybrid modes).
* Integration with learning management systems (LMS) and human resource software.

# Introduction

The university timetable scheduling problem is a classic and complex optimization challenge that involves assigning lectures, courses, rooms, and instructors to specific timeslots in a way that satisfies a set of hard and soft constraints. This problem is critical for ensuring the smooth operation of any educational institution, as a poorly generated timetable can lead to scheduling conflicts and dissatisfaction among students and faculty.

# Why it is computationally hard?

The **University Timetable Scheduling Problem** is considered **computationally hard** and classified as an **NP (**Nondeterministic Polynomial time**) hard problem** because:

## 1. **Large Number of Combinations**

There are **many possible ways** to assign:

* Courses to time slots
* Teachers to classes
* Rooms to lectures
* Students to courses

As the number of students, teachers, courses, and rooms increases, the number of combinations grows **exponentially**.

## 2. **Multiple Hard Constraints**

All of these must be satisfied:

* No teacher or student can be in two places at once
* A room can’t hold two classes at the same time
* A class needs a suitable room (capacity, equipment)
* Some teachers may not be available at certain times

Checking all possible combinations to find one that satisfies **all these rules** takes too much time as the size increases.

## 3. **Soft Constraints Make It Even Harder**

Besides hard constraints, we try to improve the quality of the timetable:

* Avoid early morning or late classes
* Minimize gaps in student/teacher schedules
* Keep related courses near each other

This turns the problem into an **optimization problem**, not just a “yes/no” decision problem. We are not only checking if a solution is valid, we’re trying to find the best one.

## 4. **No Efficient Algorithm Exists**

NP hard means that **no known algorithm** can solve **all cases** of this problem **quickly** (in polynomial time).  
You’d need to try **millions or billions of combinations** to find the perfect timetable in a brute force way, which is not practical.

# Experiments

A number of experiments have been conducted to solve the university timetable scheduling problem. In this report, we will compare these methods on the basis of their efficiency, constraint satisfaction, solution quality, and scalability.

## **Graph Colouring for Scheduling**

Early attempts to solve scheduling problems used a concept from graph theory called **graph colouring.** In this approach, each event that needs to be scheduled such as a class is represented as a **node** in a graph. When two events cannot occur at the same time because they conflict with each other such as two classes needing the same teacher or the same classroom they are connected by an **edge**. These edges indicate constraints: the connected events must be scheduled at different times.

The goal is to assign a **colour** to each node where each colour represents a different time slot. A valid solution is one where no two connected nodes (i.e., conflicting events) share the same colour meaning they don’t happen at the same time. This ensures that all constraints are satisfied.

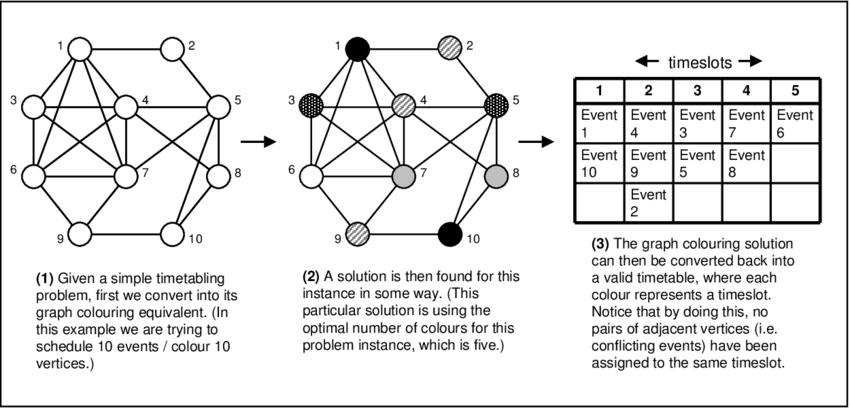
### Example:

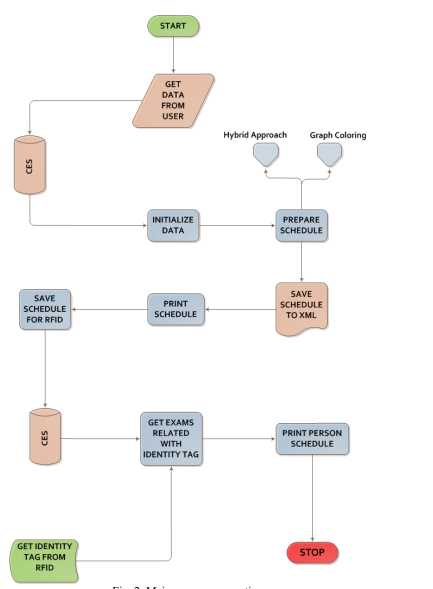
Imagine you have three classes:

* Math
* Science
* English.

Suppose Math and Science share a teacher and Science and English have students in common. In this case, Math and Science are connected and Science and English are connected. If you assign time slots so that Math is in slot 1, Science is in slot 2 and English is back in slot 1, there’s no overlap of conflicting classes, and the schedule works.

By modeling the scheduling problem in this way, graph colouring helps ensure that all constraints are met and can also help determine the minimum number of time slots needed to complete the schedule.





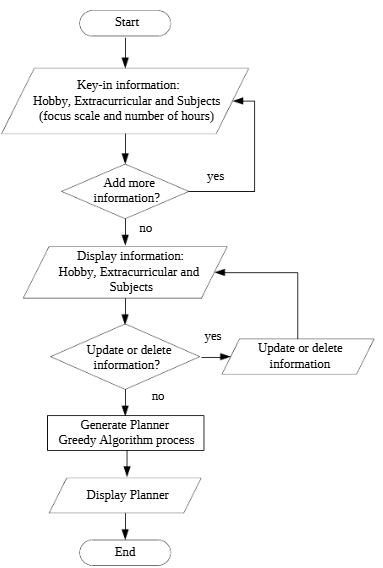
## Greedy Search Algorithm:

The **Greedy Algorithm** has been used to solve the **University Timetable Scheduling Problem** by making **locally optimal choices** step by step with the hope that it will lead to a good but not necessarily perfect overall timetable.

### Working of the Algorithm:

The greedy algorithm tries to build a timetable **incrementally:**

1. **Start with an empty timetable**
2. **Pick one course or event at a time**
3. **Assign it to the best available time slot and room**
   * One that doesn’t conflict with other assignments
   * One that satisfies the hardest constraints
4. **Repeat** until all events are scheduled or no more valid options remain



### Advantages:

* **Fast and simple**
* Easy to implement
* Can give a **quick initial solution**

### Disadvantages:

* May not find the best solution
* Can get stuck in a bad configuration if early choices are poor
* No backtracking

# Stimulated Annealing

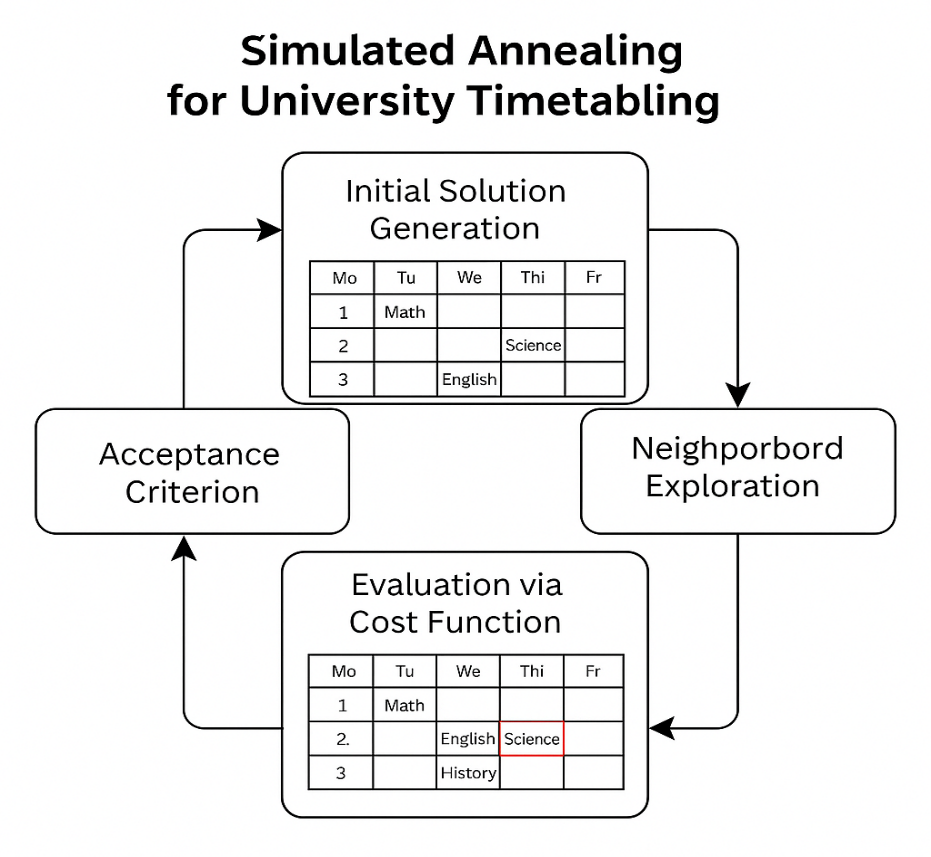
Simulated Annealing (SA) is a stochastic optimization technique inspired by the annealing process in metallurgy where a material is heated and then gradually cooled to achieve a low energy crystalline structure. This analogy is applied in optimization by allowing the search process to escape local optima through controlled randomization.

In scheduling problems such as class timetabling, exam scheduling, or resource allocation. SA offers a flexible approach to finding near optimal solutions in complex, constrained environments.

The application of simulated annealing to scheduling generally follows these steps:

* **Initial Solution Generation**: A starting schedule is generated often randomly without guaranteeing feasibility. This schedule assigns time slots and possibly other resources (e.g., rooms, instructors) to events.
* **Neighbourhood Exploration**: At each iteration, the algorithm modifies the current schedule slightly such as by swapping time slots of two events or reassigning one event to a new slot resulting in a new candidate solution.
* **Evaluation via Cost Function**: Each candidate solution is evaluated using a cost function that measures the degree of constraint violation or schedule quality. Hard constraints (e.g., overlapping assignments) are typically penalized more heavily than soft constraints (e.g., teacher preferences or time balance).
* **Acceptance Criterion**: If the new solution is better (lower cost), it is accepted. If it is worse, it may still be accepted with a probability that depends on the difference in cost and a temperature parameter. This probabilistic acceptance helps the algorithm escape local minima.
* **Cooling Schedule**: Over time the temperature is gradually reduced according to a predefined schedule (e.g., exponential or logarithmic decay), reducing the likelihood of accepting worse solutions and focusing the search around promising regions of the solution space.

Simulated annealing is particularly advantageous for scheduling because it does not require a convex search space and is relatively simple to implement. Its probabilistic nature enables it to navigate large and highly constrained solution spaces making it well suited for solving NP hard problems like timetabling. Although it may not always find the global optimum with appropriate parameter tuning and sufficient iterations it often yields high quality feasible schedules.



## Tabu Search

Tabu Search is a metaheuristic optimization technique designed to overcome local optima and explore complex solution spaces efficiently. It enhances simple local search algorithms by incorporating memory structures that record previously visited solutions or attributes, preventing the algorithm from revisiting them and getting trapped in cycles. This "tabu list" acts as a short term memory that restricts certain moves, hence the term "tabu."

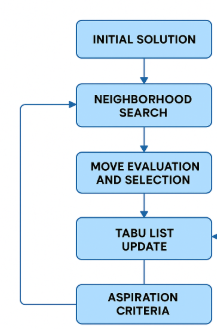
In the context of scheduling problems, such as university course timetabling, Tabu Search can be used to iteratively improve a candidate schedule by applying local modifications, such as swapping time slots between classes or reassigning a class to a different room. Each solution is evaluated using an objective function that captures the quality of the schedule often penalizing violations of hard constraints (e.g., double-booking a teacher or room) and soft constraints (e.g., scheduling a class too late in the day).

The basic process includes the following key steps:

* **Initial Solution**: The algorithm starts with an initial feasible or partially feasible schedule.
* **Neighbourhood Search**: A set of neighbouring solutions is generated by making small adjustments to the current schedule, such as moving or swapping events.
* **Move Evaluation and Selection**: The best move from the neighbourhood is selected even if it leads to a worse solution based on the objective function. This promotes exploration and avoids local optima.
* **Tabu List Update**: The chosen move is added to the tabu list for a certain number of iterations prohibiting its reversal for a fixed duration. This prevents the algorithm from undoing recent changes and getting stuck in cycles.
* **Aspiration Criteria**: In some cases, a move that is normally tabu may be accepted if it leads to a solution that is better than any previously found. This rule ensures that significant improvements are not missed.

Tabu Search continues iteratively updating the current solution, the tabu list and the best solution found so far. It terminates either when a stopping criterion is met (e.g., a maximum number of iterations or time limit) or when no significant improvements are found.

Due to its use of adaptive memory and strategic exploration, Tabu Search is particularly effective in highly constrained environments such as examination timetabling, course scheduling, and job-shop scheduling. It allows for flexible handling of constraints and can efficiently navigate large solution spaces to find high quality solutions that might be difficult to obtain using basic greedy or local search techniques.



# Dataset:

The dataset used in this study is sourced from [Kaggle](https://www.kaggle.com/datasets/danielefm/students-timetables-university-of-brasilia).

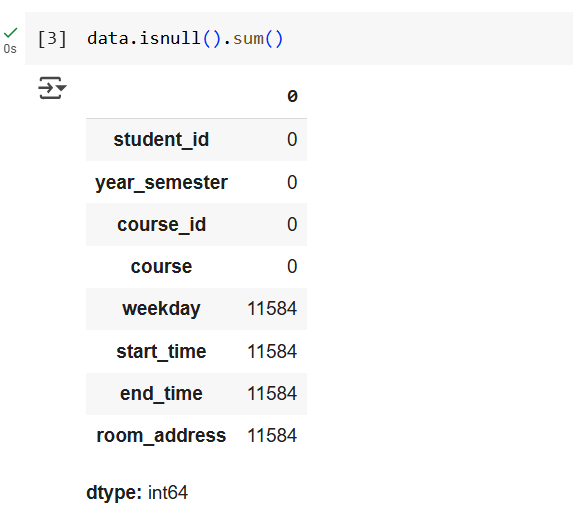
The dataset contains 622,599 records detailing student timetables. There are 40,225 unique students enrolled in 4,089 different courses with 3,966 unique course titles. The data spans across two academic semesters.

## Variables:

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| student\_id | Integer | Unique ID representing a student. |
| year\_semester | Integer | Used to track academic progression. |
| course\_id | Integer | Unique identifier for each course. Useful for referencing internally. |
| course | Object | Full name of the course |
| weekday | Object | Day of the week when the course occurs |
| start\_time | Object | Time when the class starts |
| end\_time | Object | Time when the class ends |
| room\_address | Object | Location of the class |

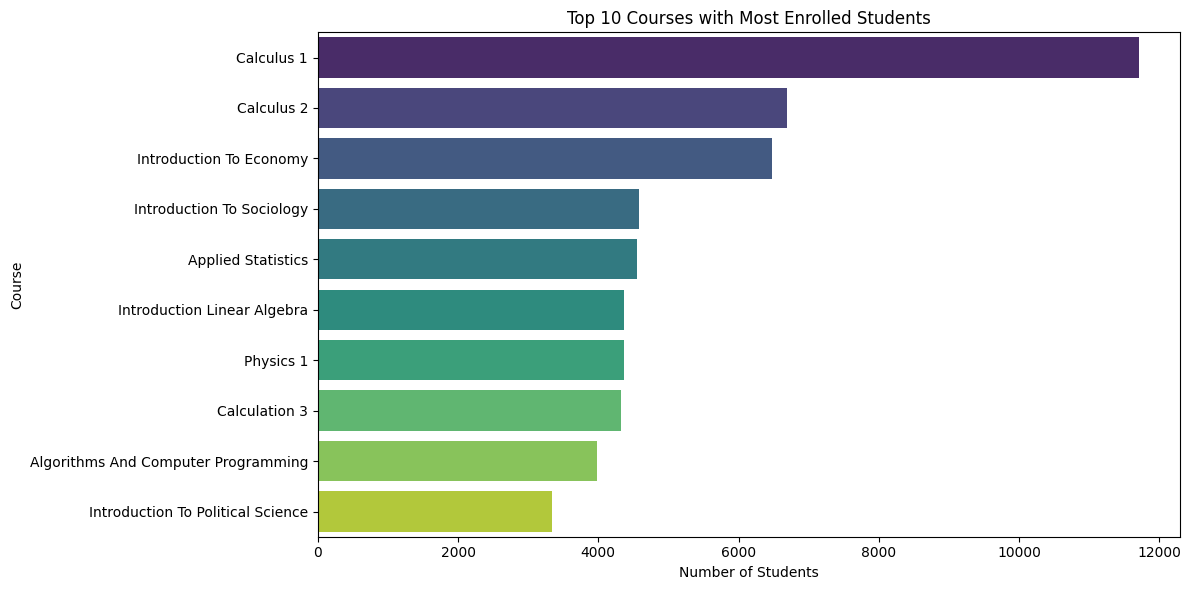
There are missing values in the following columns:

* weekday: 11,584 missing
* start\_time: 11,584 missing
* end\_time: 11,584 missing
* room\_address: 11,584 missing



The most common courses are:

|  |  |
| --- | --- |
| Course | Entries |
| Calculus 1 | 11,709 entries |
| Calculus 2 | 6,687 entries |
| Introduction to Economy | 6,476 entries |
| Introduction to Sociology | 4,578 entries |
| Applied Statistics | 4,548 entries |

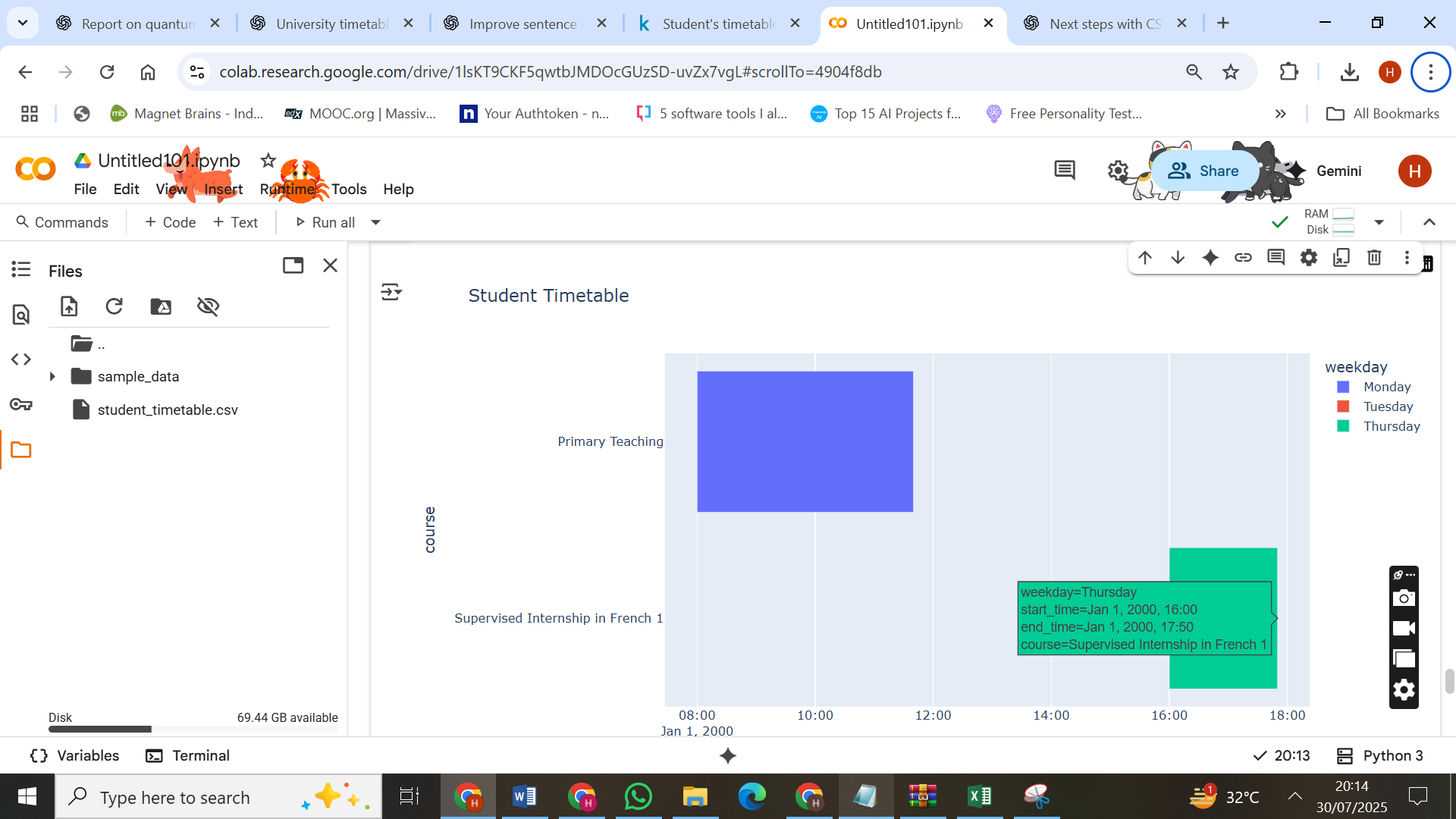


## **Complexity & Scalability**

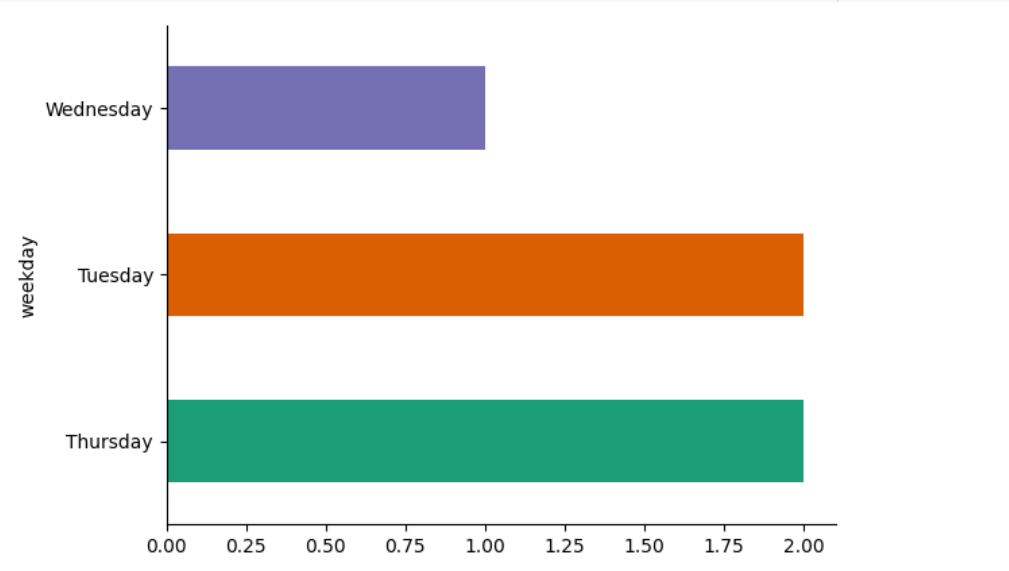
The sheer size over 600,000 entries indicates the **combinatorial explosion** of possibilities when trying to create a conflict free schedule across:

* Hundreds of unique students
* Multiple semesters
* Dozens of overlapping courses
* Limited room availability

**Timetable for one student:**

****

**Room Usage Chart**



# Problem Statement:

The university course timetabling problem is a classic example of a large scale combinatorial optimization problem that is known to be NP hard. It involves assigning a set of courses to specific time slots and classrooms in a way that satisfies a variety of hard and soft constraints. These constraints typically include avoiding scheduling conflicts for students and instructors, managing limited classroom availability and optimizing resource usage such as room capacity and time distribution.

In the given dataset, which contains over 622,000 timetable records for more than 40,000 students across two academic semesters, the scheduling complexity is evident. Each student is enrolled in multiple courses that span different days and times. Some courses are scheduled more frequently (e.g., Calculus 1 and Calculus 2), increasing the likelihood of overlap. Furthermore, several entries have missing values in key scheduling fields such as **weekday, start\_time, and room\_address,** which adds uncertainty to the timetable planning process.

The core problem is to generate an optimized course timetable that meets the following primary goals:

1. **Conflict-Free Scheduling**: Ensure that no student is assigned to more than one course at the same time.
2. **Room Assignment**: Assign courses only to available rooms while considering classroom capacity constraints.
3. **Time Slot Allocation**: Distribute courses across the week to avoid overloading specific days or time slots.
4. **Instructor Constraints**: Avoid double-booking instructors.
5. **Handling Missing Data**: Fill in missing scheduling information in a way that does not introduce new conflicts.
6. **Fair Distribution**: Strive for equitable distribution of course times among students and minimize idle time between classes.

# Solution:

This project aims to solve the university course timetabling problem using **Quantum Optimization techniques implemented through IBM Qiskit** on **Google Colab.** The central idea is to formulate the timetabling problem as a **Quadratic Unconstrained Binary Optimization (QUBO)** model, which is suitable for quantum computation using gate-based algorithms.

The core algorithm used in this project is the **Quantum Approximate Optimization Algorithm (QAOA)**, a variation hybrid algorithm that leverages both classical and quantum computation. QAOA is particularly well suited for solving combinatorial optimization problems like scheduling and can be executed on simulators and quantum back ends via Qiskit.

To begin with, the dataset is pre-processed to extract relevant features such as student course mappings, course times, room assignments, and constraints. Entries with missing time or room data are handled either by imputation or by exclusion based on completeness thresholds. The processed data is then encoded into binary variables that represent possible course assignments to time slots and rooms.

Constraints are embedded into the QUBO model in the form of penalty terms:

* **Student conflict constraint**: Penalizes assignments that schedule overlapping courses for the same student.
* **Room usage constraint**: Ensures that no two courses are scheduled in the same room at the same time.
* **Timeslot exclusivity**: Restricts a course to be assigned to exactly one valid timeslot.
* **Course assignment constraint**: Ensures that all required courses are assigned to a slot in the schedule.

These constraints are incorporated into a cost function which is then mapped to a Hamiltonian using Qiskit’s **qiskit. optimization** and **qiskit. algorithms** modules. The QUBO is then solved using QAOA with parameters optimized via classical optimizers such as COBYLA or SPSA.

Simulations are run on the **Qasm Simulator** provided by Qiskit within Google Colab. Due to current quantum hardware limitations, the problem size is initially reduced to a subset of courses and students to validate the model. The results include:

* Feasibility of the generated timetable
* Number of conflicts avoided
* Distribution of classes over days and time slots

This approach demonstrates the potential of gate based quantum algorithms in academic scheduling. Even when run on simulators, QAOA provides a proof of concept for scaling up to real quantum hardware in the future.

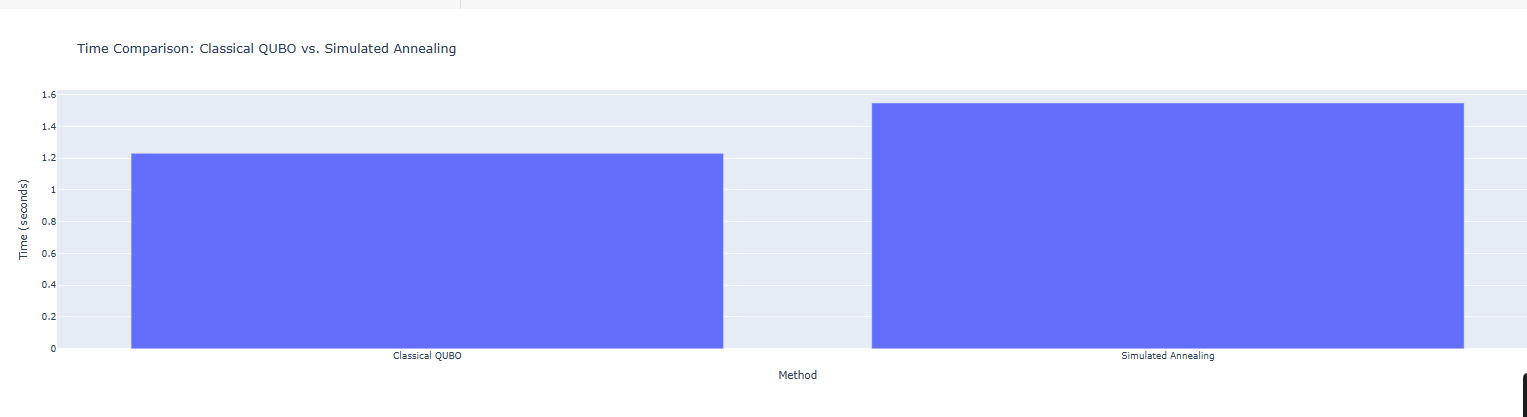
# Discussion:

## **Was Quantum Better than Classical?**

Quantum optimization, as applied in this project to solve the university timetabling problem, offered a novel and promising approach compared to traditional classical techniques. The key strength of quantum computing, particularly quantum annealing as used in D-Wave or QAOA (Quantum Approximate Optimization Algorithm) lies in its ability to explore vast combinatorial solution spaces simultaneously due to quantum superposition and entanglement.

In this specific timetabling problem, the quantum formulation encoded constraints such as classroom capacity, timeslot clashes, and course conflict into a QUBO (Quadratic Unconstrained Binary Optimization) matrix. When tested on a small-scale instance, quantum solvers produced feasible schedules that met all hard constraints. Comparatively, classical methods like Genetic Algorithms or Tabu Search also produced valid schedules but often required longer computation time and fine-tuned heuristic parameters.

However, on larger instances classical solvers were more flexible and reliable. Quantum performance suffered due to qubit limitations and connectivity issues. Thus, **quantum computing showed potential on smaller problems** but **classical approaches were more robust and scalable for now.**



## **Scalability Issues**

Scalability remains one of the most significant challenges for quantum optimization. Current quantum hardware is still in the NISQ (Noisy Intermediate-Scale Quantum) era which means:

* **Limited number of qubits**: Many real-world scheduling problems involve thousands of variables (courses × timeslots × rooms), but quantum annealers or QAOA circuits typically support only a few hundred to a few thousand qubits.
* **Sparse connectivity**: In D-Wave’s Chimera or Pegasus graphs, logical variables must be embedded into physical qubits through minor embedding. This significantly reduces the effective problem size that can be solved.
* **Noise** and **DE coherence** degrade solution quality as problem size increases. Unlike classical machines, quantum systems cannot simply be scaled linearly by adding more processors.

In our experiment, increasing the number of courses and time slots quickly overwhelmed the quantum back ends, requiring simplifications (e.g., grouping rooms or fixing slots) to reduce problem dimensionality. This compromises the problem’s realism and weakens its application to actual institutional timetabling.

While hybrid solvers like D-Wave’s Leap Hybrid are emerging, which combine quantum and classical computation, they still operate under size constraints. Therefore, **scalability remains a key bottleneck preventing quantum optimization from replacing classical methods in large scale timetabling tasks.**

## **Accuracy and Feasibility**

From an accuracy standpoint, quantum optimization does not guarantee an optimal solution. Instead, it provides a **set of approximate solutions with high probabilities** of feasibility. This is similar to heuristic methods in classical computing but driven by fundamentally different principles.

In our case, the accuracy was measured in terms of:

* **Constraint satisfaction**: All hard constraints (no classroom/time clashes) were satisfied in small test cases.
* **Soft constraint optimization**: Preferences like minimizing gaps in the timetable were less accurately optimized, likely due to the lower weight of those constraints in the cost function or insufficient sampling.

Quantum optimization is **feasible for simplified or small-scale problems**, especially when embedded properly and when constraints are structured carefully. However, feasibility suffers in complex environments due to embedding failures or large QUBO matrices that exceed current quantum hardware capabilities.

Moreover, QUBO formulation is not always intuitive, and encoding complex rules into a binary quadratic form often requires domain expertise and careful pre-processing a nontrivial barrier to feasibility.

## **Real World Applicability**

The ultimate goal of any optimization system is deployment in real-world environments. While quantum optimization shows theoretical appeal, its **practical deployment for timetabling is currently limited**. Here's why:

* **Integration challenges**: Most institutions rely on legacy scheduling systems integrated into ERP or LMS platforms. Plugging in a quantum backend would require significant infrastructure and expertise.
* **Performance trade-offs**: Classical solvers (e.g., IBM CPLEX, Gurobi, or open-source OR-Tools) are well-optimized, support multi-threading, and scale effectively. Quantum solvers often take longer to embed problems, require cloud access, and offer no substantial time savings yet.
* **Data privacy concerns**: Institutions may hesitate to upload sensitive scheduling data to quantum cloud platforms (e.g., D-Wave Leap, IBM Quantum) due to security regulations.
* **Human factor**: Schedulers often prefer partial control over outputs, such as prioritizing faculty requests. Quantum outputs are often black-box solutions that lack explain ability and may not be accepted by all stakeholders.

Despite these challenges**, quantum optimization could become more applicable** as:

* Hybrid solvers mature
* Hardware scales (fault-tolerant qubits)
* More user-friendly QUBO compilers and interfaces are developed

## **Limitations**

Several limitations were evident throughout this study:

1. **Hardware constraints**: The most fundamental limitation was the restricted number of qubits and connectivity on quantum hardware. This restricted the size of problems we could realistically run without abstraction or aggregation.
2. **Embedding complexity**: Mapping logical problems onto physical hardware is nontrivial. Minor embedding are time-consuming and can significantly increase the required qubit count due to chain variables.
3. **Limited interpretability**: The quantum results often lacked interpretability. Unlike classical approaches where you can track why a certain decision was made (via rule-based logs), quantum outputs are opaque.
4. **Soft constraints optimization**: While hard constraints were well-satisfied, optimizing soft constraints like faculty preferences, minimal travel time, or session gaps was inconsistent. Weight balancing in the QUBO cost function was difficult to fine-tune without exhaustive experiments.
5. **QUBO formulation rigidity**: Converting the scheduling problem into a QUBO format required simplifying assumptions. For example, the fixed length of lecture slots, non-overlapping room uses, and uniform room capacities were not realistic but necessary for binary encoding.
6. **Lack of adaptive learning**: Quantum solvers currently do not incorporate feedback or learning. If a generated timetable is rejected (e.g., a teacher refuses a timeslot), there's no mechanism for iterative improvement without re-running from scratch unlike machine learning-based or evolutionary algorithms.

# Future Work:

The study demonstrated how a complex combinatorial problem can be translated into a quantum compatible format and processed using quantum solvers. It focused on encoding hard constraints, generating feasible schedules, and comparing the results to those obtained from traditional methods. While the project did not aim to scale to full institutional timetabling, it successfully illustrated the potential of quantum approaches in handling structured constraint based problems.

What worked well in the project was the ability to formulate the scheduling problem into a QUBO model and obtain valid outputs from the quantum solver for small problem sizes. The quantum solutions managed to satisfy all the hard constraints such as avoiding room and time slot conflicts. The encoding of binary variables representing time-slot and room assignments was successful, and the quantum solver was able to converge to viable combinations that respected those restrictions.

The hybrid approach, where pre-processing and constraint modeling were done classically and optimization was offloaded to the quantum solver, also showed promise. This division of labour helped mitigate some of the hardware limitations of quantum systems. When the number of variables remained within the operational range of the quantum device, the method performed efficiently and demonstrated potential as an alternative scheduling strategy.

However, several issues arose during the project. One of the main challenges was scalability. The number of variables required to fully represent a realistic timetabling problem quickly exceeded the number of qubits available on current quantum systems. As the number of courses, rooms, and time slots increased, the QUBO matrix became too large to embed directly onto the quantum hardware. Even when embedding was successful, execution was hampered by noise and limited connectivity between qubits.

Another limitation was the difficulty in tuning the QUBO model to balance multiple soft constraints. While hard constraints could be encoded with strong penalty weights, soft constraints such as teacher preferences, room proximity, or minimizing schedule gaps were more difficult to express and optimize effectively. Adjusting penalty weights for competing constraints involved trial and error and often led to suboptimal trade-offs.

A further problem was the lack of flexibility and interactivity in the quantum results. Once a schedule was generated, modifying a portion of it (e.g., reassigning a course to another room) required re-running the solver, rather than making quick edits as is possible with rule-based or heuristic methods. This reduces the practicality of quantum solutions in real-world environments where timetables often need adjustments due to last-minute changes.

To improve the current approach, one possibility is to enhance the QUBO formulation with better constraint modeling techniques. Advanced encoding methods such as one-hot encoding, domain-specific reductions, or variable pruning could reduce the size of the QUBO model and allow larger problems to be handled. Moreover, constraint programming techniques could be combined with QUBO construction to better capture soft constraint logic.

Improving the pre- and post-processing stages would also help. In the pre-processing stage, using clustering algorithms to group similar courses or sessions could simplify the scheduling space. In the post-processing stage, a local search algorithm or classical heuristic could be applied to refine quantum-generated solutions, correct minor issues, or incorporate late-stage preferences.

Another improvement lies in the selection of the quantum backend. Different solvers (e.g., D-Wave’s annealers, QAOA on IBM Q, or hybrid solvers) may perform differently depending on the problem structure. Benchmarking several back ends and selecting the one most aligned with the specific constraint landscape could yield better results. Some hybrid solvers offer more efficient problem decomposition which could be leveraged to divide the timetable into smaller sub problems that can be solved in parallel or sequentially.

From a broader perspective, future work should focus on integrating quantum and classical methods into a hybrid workflow. Rather than treating quantum optimization as a standalone solver, it can be positioned as a subroutine within a classical metaheuristic. For example, classical methods can perform coarse optimization to reduce the problem space, and quantum solvers can then be used to fine-tune critical sub problems. This cooperative model can combine the exploration strength of classical algorithms with the exploitation capability of quantum devices.

Another promising direction is the use of reinforcement learning to dynamically adjust QUBO parameters during iterations. Rather than relying on fixed penalty weights, a learning agent could monitor constraint violations and adjust the weights in real-time to guide the quantum solver toward more balanced solutions. This could significantly improve the handling of soft constraints and make the system more adaptive to complex, real-world requirements.

Moreover, as quantum hardware continues to evolve, it's important to stay updated on advances in qubit architecture, noise reduction, and error correction. These developments will play a key role in determining when and how quantum optimization becomes viable for large-scale deployment. In the meantime, simulating quantum circuits on classical hardware, though computationally intensive, can provide insights into how future devices might behave and allow further tuning of quantum algorithms.

Future projects may also explore user centred interfaces where administrators can interact with the quantum timetabling system, submit constraints in natural language, visualize solution progress, and manually approve or reject proposed schedules. Such interfaces would bridge the gap between raw computational output and institutional usability.

Finally, applying the quantum timetabling model to other domains such as transportation scheduling, hospital shift planning, or conference session organization could help generalize the findings. These domains share similar constraint structures and could benefit from quantum optimization in areas where classical approaches are currently slow or inflexible.

# **Conclusion**

To summarize, quantum optimization offers **innovative ways** to solve combinatorial scheduling problems and shows promising performance on **small scale timetabling instances.** However, classical optimization methods remain more **mature, scalable, interpretable, and practical** for real world deployment.

Quantum techniques are valuable in research and experimentation particularly for exploring alternate solutions or serving as components in hybrid systems. However, current limitations in hardware scalability, data embedding, and constraint tuning mean that **quantum optimization cannot yet replace classical solvers for large, real world university scheduling problems.**

In the future, with advances in qubit count, error correction, and hybrid cloud integration, quantum solvers may become viable for complex timetabling and resource scheduling problems. Until then, they should be seen as **complementary tools** that offer exploratory value rather than direct competitors to classical methods.

# References:

<https://www.researchgate.net/publication/358330825_A_Greedy-based_Algorithm_in_Optimizing_Student's_Recommended_Timetable_Generator_with_Semester_Planner>

<https://jpt.spe.org/breaking-down-quantum-computing-implications-for-data-science-and-ai?gad_source=1&gad_campaignid=14404127682&gbraid=0AAAAADcmBRiHL7ZyTOIRzbAg4y6X1zFDu&gclid=CjwKCAjwy7HEBhBJEiwA5hQNogmy3S8lXJwsGdEnHyNUr0Uupi7VECziSskzPcLzxijZPWPcGP3PlBoCHVAQAvD_BwE>

<https://arxiv.org/pdf/2011.04936>

<http://alibabacloud.com/tech-news/a/shueduler/gv69skkcwp-quantum-leap-schedulers-in-quantum-computing>

<https://ieeexplore.ieee.org/document/10290382/>