THE UNIVERSITY *of* EDINBURGH

Topics in Applied Operational Research

Timetabling Optimization Project for the School of Mathematics

by

Haoyuan Zhang, Yuxin Li, Yuting Zhang, Anagha Indulal Nair

April 2024

Contents

Introduction

The optimization of university schedules is crucial for the seamless operation of educational institutions. This report explores the intricate task of optimizing university-level timetables through a blend of Mixed Integer Programming (MIP) models and sophisticated optimization algorithms to tackle a variety of scheduling constraints. Our analysis is grounded in a dataset comprising detailed course information, student enrollment numbers, and room capacities, all of which undergo thorough preprocessing to guarantee precision and applicability. For details please see A

Our methodology leverages an innovative amalgamation of algorithms designed to adeptly navigate both hard and soft constraints, crafting a schedule that is both realistic and efficient. Given the pivotal role of lectures and workshops in the academic schedule, our optimisation process is bifurcated into two distinct models: one for optimising lectures and another for workshops. This division allows for targeted consideration of specific constraints such as room capacity, class timing without overlaps, and the need for lecture recording facilities. Our primary objective is to minimise both the incidence and quantity of course conflicts. To this end, we employ Particle Swarm Optimization and Simulated Annealing algorithms for the lecture model, applying these to a preliminarily generated timetable to effectively reduce course conflicts.

Conversely, the workshop model focuses on accommodating multiple sessions within constrained classroom capacities, enabling students to select class times that best suit their schedules. This aims to decrease the overall frequency of instructional sessions, thereby alleviating the workload on faculty and staff. Here, we predominantly utilise Genetic Algorithms, iterating through numerous mutations to meet our optimisation targets.

Remarkably, our strategy has proven both successful and replicable. We have managed to decrease the number of scheduling conflicts from 25 to 8 and reduce the courses involved in conflicts from 39 to 21. For the workshop, we reduced the total number of classes from 361 to 182, demonstrating the efficacy of our approach in enhancing the scheduling framework of university timetables. For detail code and output are shown in H

1.1 Mixed Integer Programming Model

1.1.1 Variables

- **Set of Courses** Let $\mathcal{C}$ be the set of courses, where:

$$\mathcal{C} = \{\text{MATH07004}, \text{MATH08057}, \text{MATH10010}, \ldots\}$$

- **Number of Students per Course** For each course $c \in \mathcal{C}$, let st_c be the number of students attending the course. For example:

$$st_{\text{MATH07004}} = 248, \quad st_{\text{MATH08051}} = 395, \ldots$$

- **Set of Rooms** Let $\mathcal{R}$ be the set of rooms available for scheduling, where

$$\mathcal{R} = \{MH_{1.19}, JCMB_{2901}, \ldots, HBBclassroom_4\}$$

- **Room Capacities** For each room $r \in \mathcal{R}$, let cap_r be the capacity of the room.
- **Lecture Duration** For courses $c \in \mathcal{C}$, let C_1 be duration for lectures that are 1 hours and C_2 is for 2 hours. $C_1, C_2 \in \mathcal{C}$

- **Set of Time Periods** Let $\mathcal{H}$ be the set of time slots from 9:00-18:00, where:

$$\mathcal{H} = \{1, 2, \ldots, 9\}$$

$\mathcal{D}$ be the set of school days which is Monday to Friday, where:

$$\mathcal{D} = \{1, 2, \ldots, 5\}$$

Objective function Minimising the number of time conflicts and the number of time conflicts:

$$\min \quad \sum_{c\in\mathcal{C}} \sum_{h\in\mathcal{H}} \sum_{d\in\mathcal{D}} z_{chd}, \tag{1.1}$$

where

$$z_{c_1c_2hd} \geq x_{c_1hdr} + x_{c_2hdr} - 1 \quad \forall c_1, c_2 \in \mathcal{C}, c_2 > c_1, \forall h \in \mathcal{H}, \forall d \in \mathcal{D}, \forall r \in \mathcal{R} \tag{1.2}$$

$z_{dhc}, x_{chdr}, x_{rhc}, y_{rhc} \in B$ and
z_{dhc} equals 1 if the class at the same time h, on the same day d, and in the same classroom r was scheduled $c \in \mathcal{C}$. For courses $c \in \mathcal{C}$, each room $r \in \mathcal{R}$, hours $h \in \mathcal{H}$, days $d \in \mathcal{D}$ $x_{rhc}, x_{chdr} \in \mathcal{B}$ equals 1 if course c allocated in day d in hour h in room r. For courses $c_1 \in \mathcal{C}$, each room $r \in \mathcal{R}$, hours $h \in \mathcal{H}$, $y_{rhc} \in \mathcal{B}$ equals 1 if course c allocated in day d in hour h and $h+1$ in same room r.

1.1.2 Constraint

Detail explanation of each constraint in B

$$\sum_{c\in\mathcal{C}} x_{chdr} \leq 1 \quad h \in \mathcal{H},\ r \in \mathcal{R},\ d \in \mathcal{D} \tag{1}$$

$$\sum_{r\in\mathcal{R}} x_{chdr} \leq 1 \quad c \in \mathcal{C},\ h \in \mathcal{H},\ d \in \mathcal{D} \tag{2}$$

$$\sum_{r\in\mathcal{R}} \sum_{c\in\mathcal{C}} x_{chdr} \leq 1 \quad h \in \mathcal{H},\ d \in \mathcal{D} \tag{3}$$

$$cap_r \geq st_c \quad r \in R, c \in C \tag{4}$$

$$y_{rhc_2} \leq x_{rhc_2} \tag{5}$$

$$y_{rhc_2} \leq x_{rh+1c_2} \tag{6}$$

$$y_{rhc_2} \geq x_{rhc_2} + x_{r,h+1,c_2} - 1 \tag{7}$$

$$x_{chdr} = 0 \quad \forall c \in C2, \forall h = |\mathcal{H}|, \forall d \in \mathcal{D}, \forall r \in \mathcal{R} \tag{8}$$

Methodology

2.1 Timetable for Lectures

2.1.1 Algorithm

In our approach, we implemented three distinct models to address the challenges: random timetable generator, simulated annealing, and PSO optimiser. Our objective is twofold: to minimise both the incidence of scheduling conflicts and the total number of courses involved in these conflicts. This dual focus stems from our initial goal of reducing scheduling conflicts. Initially, our efforts were solely concentrated on decreasing the frequency of these conflicts. However, we observed that this approach led to an undesirable outcome where classes were predominantly scheduled on Mondays and Tuesdays to minimise conflict occurrences, which was impractical for real-life scheduling. To overcome this, we introduced an additional objective function to reduce the overall number of courses in conflict, aiming to create a more practical and functional timetable. This adjustment was necessary because a high number of conflicts would restrict students to attending only one lecture at a time, if another lecture was scheduled during the same time, negatively impacting their educational experience. Therefore, we assigned equal importance to both aspects—conflict occurrences and the number of conflicting courses—in our objective function to generate a timetable that is truly beneficial in real-life scenarios.

2.1.2 Auxiliary functions

We set the timetable as a list of 5-tuples, (*course*, *room*, *day*, *start_hour*, *end_hour*). This representation facilitates the scheduling of courses in a structured manner. The following utility functions have been developed to support the process of optimising these timetables, addressing both conflict resolution and schedule efficiency

- **calculate_conflict_count:** This function calculates the total number of conflicts within the timetable. A conflict occurs when two or more courses are scheduled in the same time slot, potentially causing students to be double-booked. By iterating over the timetable and checking for overlaps in course times, it provides a numerical value representing the total count of such conflicts.
- **calculate_conflict_occurrences:** While closely related to the $calculate\_conflict\_count$, this function focuses on identifying the number of time slots that contain conflicts, regardless of how many courses are involved in each conflict.
- **find_conflicting_courses:** This function identifies the specific courses that have scheduling conflicts by examining the timetable for courses scheduled at the same time but in different rooms. It returns a set of courses involved in such overlaps.
- **print_timetable_by_room:** Aimed at improving the usability and readability of the timetable, this function organises and displays the timetable sorted by room, day, and starting hour.
- **check_unassigned_courses:** This function checks whether all courses have been successfully assigned a time slot in the optimised timetable. It identifies any courses that have not been scheduled.
- **find_unused_time_slots1:** Find unscheduled periods by traversing the schedule and printing them as day, start hour, end hour.

Random timetable generator

This function is designed to randomly create a timetable from our dataset, adhering to the stipulation that two distinct courses cannot be scheduled in the same classroom simultaneously. Additionally, courses must be assigned within the time frame of 9 AM to 6 PM. The generate random sample function is designed to randomly generate a course schedule by assigning time periods and classrooms to the course, subject to specific constraints. It traverses each course, selecting a day of the week, a room, and the start time of the course from a list of available rooms. This feature ensures that the selected room has sufficient capacity to accommodate the number of students attending the course. It also checks that the duration of the course does not exceed the available time period of the day, avoiding scheduling the course outside the allowed time.

A key component of this function is the 'is time slot available' subroutine, which checks whether the selected time period for a course in a particular room has no scheduling conflict with an existing course in the schedule. This includes verifying that two classes are not scheduled in the same classroom at the same time on the same day. This function handles classes of different duration, ensuring that even classes that require multiple hours can be scheduled in one day without exceeding the maximum available time for the day. Through a combination of random selection and careful verification, "generate random sample" effectively produces a random initial schedule with no classroom conflicts. And then we will use other two function to optimize the initial timetable.

2.1.3 Simulated Annealing

Simulated annealing is a probabilistic technique for approximating the global optimum of a given optimisation problem([1]).In the context of optimisation, this method starts with a random solution to the problem and then explores the solution space by making incremental changes to the solution. At each step, changes that improve the objective function are accepted, and changes that do not are accepted with a probability that decreases with temperature. Over time, the system "cools" and becomes less likely to accept worse solutions, allowing it to converge on a global optimum or a solution close to it([2]).In creating a timetable, simulated annealing is a useful method since these problems often have a large search space and may contain many local optima that a more naive optimisation approach might get trapped in([3]).

The code of the simulated annealing algorithm has 5 parts. The 'fitness' function assesses the quality of an academic timetable by calculating and penalising the number of scheduling conflicts within it([3]). Specifically, it tallies the total occurrences of time slot conflicts and the overall count of such conflicts, applying penalties to both. The fitness score is inversely proportional to the combined penalty, meaning timetables with fewer conflicts are deemed more suitable. The fitness score determines the likelihood of accepting a new timetable solution over the current one. A higher fitness score (indicating fewer scheduling conflicts) makes a solution more favourable, guiding the algorithm towards more optimal timetable arrangements with minimal conflicts.

The 'generate sample' function creates an initial timetable by systematically attempting to schedule each course into an available time slot and room that meets the course's requirements for duration and student capacity. It iterates over all courses, checking each possible day and room combination to find a match where:

1. The room can accommodate the number of students enrolled in the course.
2. The course fits within the available hours for that day, without exceeding the maximum lecture hours.

If such a slot is found, the course is added to the timetable, marking it as successfully scheduled. This initial generation of a timetable provides a starting point for further optimisation processes, ensuring that all courses are initially placed in a feasible, if not yet optimised, schedule.

The 'mutate' function is designed to introduce variations into an existing timetable, serving as a mechanism for exploring the solution space in search of more optimal configurations. This function:

1. Selects a subset of courses to be re-scheduled, aiming to improve the overall timetable by altering these courses' allocations.
2. Utilizes information about unused time slots to identify potential new slots for these courses, attempting up to 100 re-scheduling efforts per course.

3. Ensures that any new scheduling adheres to the constraints of room capacity and maximum lecture hours, similar to the initial timetable generation.

The mutation process is crucial for the Simulated Annealing algorithm, as it allows the exploration of new timetable configurations beyond the local neighbourhood of the current solution. By randomly altering parts of the timetable, it increases the chance of escaping local optima and moving towards a globally optimal solution. The ‘simulated annealing‘ function orchestrates the optimisation process. It begins with a high initial temperature, set at 100,000,000, and gradually cools down to a final temperature of 0.0000001, following a cooling rate (alpha) of 0.99([4]). This gradual cooling schedule allows the algorithm to shift smoothly from an exploration of the solution space to a more focused exploitation of the best solutions encountered. A unique feature of our methodology is the incorporation of the Metropolis criterion for dynamically accepting new solutions([5]). This aspect, in conjunction with the temperature-dependent behaviour of the algorithm, allows for the occasional acceptance of suboptimal solutions. Such a strategy enhances the exploration of the solution space by preventing premature convergence on local optima and encouraging a more comprehensive search for the optimal timetable configuration. In every iteration, use the function mentioned before to calculate the occurrences of conflicts and the count of conflicting courses, adding these metrics to our records. Subsequently, we graph the progression of these variables as iterations advance, providing a visual representation of the optimisation process. This data visualisation enables us to observe how effectively the optimisation progresses and whether it converges towards a satisfactory outcome.

2.1.4 Particle Swarm Optimization Algorithm

The Particle Swarm Optimization (PSO) algorithm, inspired by the social behaviour of birds flocking or fish schooling, optimises solutions by moving a group of particles through the solution space. Each particle adjusts its trajectory towards its own best position and the global best among all particles, effectively searching for optimal solutions([6]). PSO is well-suited for optimising school timetables because it efficiently navigates large, complex search spaces typical of scheduling problems, addressing constraints like scheduling conflicts and resource limitations([7]).

There are 8 parts in this algorithm. The fitness function, Particle Class, Update velocity Function, Reinitialize particle Function, Reinitialize particle Function, Update position Function, Find unused time slots Function, Local search Function, Update particle Function and The PSO optimise function. Here, we focus on the PSO optimisation function since it is the caller of the PSO algorithm. The remaining functions will be detailed in C.

The PSO optimisation function leverages Particle Swarm Optimization to refine school timetables. With 30 particles, the function strikes a balance between diverse solutions and computational efficiency, ensuring a thorough exploration of the solution space. The limit of 10,000 iterations provides ample opportunity for the algorithm to escape local optima and refine towards the best solution. The inertia weight (w) is 0.4, encouraging wide-ranging search initially([8]), then focusing on solution refinement. Both cognitive and social coefficients are set at 2, equally weighting personal achievements and the swarm's knowledge, promoting a balanced approach to individual and collective learning([9]). The maximum velocity cap of 5 prevents drastic timetable changes, ensuring modifications remain practical and manageable. Additionally, within the PSO optimise function, we incorporate a plotting feature to visualise the changes in conflict occurrences and the count of conflicting courses at each iteration.

2.1.5 Analysis and Results

The final result of the best timetable for the semester 1 lecture is shown in F.1 and for the semester 2 lecture is shown in D.2b

To pinpoint the optimal timetable solution, we deploy a two-pronged algorithmic approach, initiating from a randomly generated schedule. Our method involves an initial optimisation with the Simulated Annealing algorithm, followed by a secondary refinement through the Particle Swarm Optimization algorithm, enhancing the timetable previously adjusted by Simulated Annealing the result is shown in D.1 . In an alternate strategy, we reverse the order, applying PSO first, then SA as shown in D.2 . By comparing the outcomes of these sequential applications, we aim to discern the most effective combination for achieving the best timetable solution. From these two figures, we can find that when we first use the simulated annealing algorithm, the final result is we have 8 conflict occurrences and 21 courses that conflict. However, the alternate strategy has 8 conflict occurrences and 22 courses that conflict. We can find that the simulated annealing algorithm has a better output than PSO. The reason why this might be because SA methodically cools the system, allowing it to accept worse solutions initially, thereby navigating out of local optima towards a more globally optimal solution. Additionally, SA's simpler parameterise makes it easier to tune and apply to a wide range of problems, including those with discrete variables and strict constraints typical of timetabling issues([4]). In contrast, PSO's reliance on parameter tuning and its original design for continuous spaces might not be as effective in the discrete and highly constrained domain of timetable optimisation ([10]). The sub-optimal results from the PSO algorithm may stem from our limited exploration of its possible parameters. Additionally, given the stochastic nature of both algorithms, achieving improved outcomes necessitates extensive experimentation with parameter adjustments and multiple runs. This approach can potentially enhance the performance of the PSO algorithm. Given the stochastic nature of these two algorithms, the outcomes vary with each execution, occasionally resulting in unallocated free time slots. Addressing this issue could involve increasing the number of iterations within the code. However, due to the significant time and computational resources required for such adjustments, we have chosen not to implement this solution.

The inability to achieve a conflict-free timetable primarily arises from the constraint of having only 45 available time slots—derived from 5 working days multiplied by 9 available slots each day—while needing to schedule more than 45 courses. This discrepancy between the number of courses and available slots is the fundamental challenge in attaining a timetable without scheduling conflicts.

Additionally, by introducing a constraint to the timetable generation function to avoid scheduling courses on Wednesday afternoons, we observed an increase in both the occurrences of conflicts and the number of courses involved in conflicts as shown in D.3.

Applying the same approach to the Semester 2 lectures yielded comparable outcomes, where initiating the optimisation process with the Simulated Annealing algorithm before employing PSO proved to produce superior results compared to starting with PSO.

2.2 Timetable for Workshops

2.2.1 Background and Setup

Workshop, as a highly interactive and practice-oriented teaching and learning activity, occupies an important position in teaching and learning. However, due to the limitation of classroom capacity, especially when the capacity does not match the students who enrolled in this course, frequent workshops can lead to conflicts in classroom arrangement and waste of resources, such as the energy of teachers. Reducing the number of workshops and allowing more students to be accommodated in each workshop through optimal arrangement not only makes effective use of classroom resources but also reduces fragmentation of students' and teachers' time and improves teaching efficiency.

Therefore, we have the same constraints on the lecture model, except that classroom capacity must be greater than or equal to the number of students enrolled in a course. If a classroom cannot accommodate all students who have chosen the course, we do not immediately discard this option; instead, we allocate the surplus students to the available time slot randomly.

2.2.2 Introduction to Genetic Algorithm

The genetic algorithm is a search algorithm that simulates natural selection and genetic mechanisms. It searches for the optimal solution in the solution space through selection, crossover, and mutation operations during the iterative process [11]. In reducing the number of workshop problems, genetic algorithms can be used to find the optimal or near-optimal solution by generating a wide range of possible timetable scenarios and improving them iteratively. Initial reasons for using genetic algorithms include their flexibility, adaptability and successful application experience in solving similar problems, especially the ability to deal with the ‘reducing the number of workshop’ problems and to achieve the goal and optimising the use of resources by intelligently adjusting the course allocation [12].

2.2.3 Model for Workshop

Generise a random model

In our model, we focus on three principal elements: courses, rooms, and the timetable, each designed to facilitate effective workshop scheduling. Courses are represented as a list, with each course identified by a unique identifier, i.e. the course code. In addition to the identifier, each course is associated with specific attributes including the number of students enrolled in the course and the total hours required for the course. Each classroom also has a unique identifier and is equipped with a corresponding capacity.

The timetable is constructed in detail as a comprehensive dictionary. Each entry contains a course identifier as well as a detailed list outlining all the classroom hours scheduled for that course. Each entry in the list specifies the specific time slot assigned to the course, showing the classroom and day of the week for each session. This structure provides a clear overview of room allocations and availability.

To initiate the scheduling process, we generate a list of all conceivable time slots, represented as tuples pairing room numbers with weekdays. Critical to our model is considering each course’s specific requirements, such as the total instructional hours needed and the maximum number of sessions permissible. These parameters are determined by student enrollment numbers and the minimum capacity of available rooms, aiming to meet instructional requirements efficiently while minimising the fragmentation of sessions. In laying the groundwork for the timetable, our *generate_sample* function employs a random strategy for allocating time slots, utilising the *random.sample* method to inject an element of flexibility and random strategy into the model in order to explore a variety of scheduling scenarios.

As a result, a preliminary randomised timetable model has been developed that adapts to the curriculum’s needs, providing the basis for subsequent optimisation.

Optimisation by Genetic Algorithm

In our genetic algorithm, mutation and exchange operations are key steps for exploring the solution space and optimising the schedule. The implementation details of these operations and their role in the algorithm are described below:

- **Mutation:** The goal of the mutation operation is to increase the diversity of the solution space in order to avoid early convergence of the algorithm to a local optimum. In this process, a course is randomly selected and a scheduled time slot is randomly selected for that course and then replaced with an unoccupied time slot. This randomisation strategy has the potential to lead to new, more efficient timetable configurations. The mutation operation takes into account course-specific needs, such as the number of instructional hours required for the course (i.e. 1 hour and 2 hours), ensuring that each mutation does not violate the basic time requirements of the course while exploring the possible room for optimisation.
- **Exchange:** The core of the exchange operation is to facilitate the swapping of time slots between two courses, a strategy that is not limited to exchanges between courses of the same duration, but also allows for exchanges between courses of different durations (i.e. 1 hour and 2 hours), greatly increasing the flexibility of the adjustment. This approach provides a wider scope for optimisation, allowing the algorithm to find the optimal timetable structure through a more diverse combination of courses.

After each mutation or exchange operation, we evaluate the newly generated timetable scheme and assess the effectiveness of the scheme by calculating the number of classes for each course. If the new scheme is able to reduce the total number of classes, we accept the change, and in this way, the genetic algorithm explores and optimises the timetable through a continuous iterative process that seeks to minimise the number of classes in a course while meeting all teaching and learning needs.

2.2.4 Analysis and Results

During schedule optimisation in the workshop, we control the number of schedules generated in each mutation or exchange attempt by setting *mutation_time* and the total number of mutation and exchange operations by setting *total_mutation*. These parameters have a significant impact on the algorithm's ability to find an optimal solution.

The *mutation_time* parameter directly affects the breadth of solutions explored by the algorithm in each operation. We find that increasing *mutation_time* improves the diversity of solutions, thus increasing the chances of finding a better schedule configuration. However, it also implies an increase in the amount of computation per operation, leading to a corresponding increase in the algorithm's runtime. A higher value of *total_mutation* means that the algorithm has more opportunities to explore and adapt, thus potentially reaching a better schedule configuration. However, again, this will increase the total computation time.

Comparative Experiments for the Parameter "*total_mutation*"

We are exploring the impact of different *total_mutation* settings on the final optimisation results. To gain a deeper understanding of the impact of parameter settings on the algorithm's performance, we designed a series of comparative experiments, including the method of gradually increasing the value of *total_mutation* versus the method of directly setting a larger value of *total_mutation*.

The experiment is divided into three main phases. In phase 1, we initialise *total_mutation* to 150 and run the algorithm. Then, using the results of the previous optimisation as a new starting point, the setting *total_mutation*=150 is applied again until the total number of mutations reaches 2400. In phase 2, we set *total_mutation* to 1200 once and run the algorithm once. In phase 3, we set *total_mutation* directly to the maximum value of 2400 to test the effectiveness of performing large-scale optimisations in a single run.

This allows us to observe how the algorithm performs in a larger single optimisation and how efficient and effective it is compared to the staged optimisations. The results are shown in Figure E.3. It can be observed from the figure E.3 that the staged optimisation (phase 1) is able to gradually improve the optimisation results, whereas the results obtained by directly setting the maximum value in a one-time large-scale optimisation (phase 3) do not show the expected significant improvement.

Comparative Experiments for the Parameter "*mutation_time*"

Similarly, we can perform a detailed tuning and analysis of *mutation_time*, aiming to evaluate its specific impact on the optimisation results. The experiment unfolds in three main phases, each exploring different scales of *mutation_time* adjustments: incremental increases, a moderate one-time increase, and a substantial one-time increase, to gauge their impacts on optimisation outcomes.

In Phase 1, we initialise *mutation_time* to 10 and run the algorithm. Then, using the results of the previous optimisation as a new starting point, the setting *mutation_time*=10 is applied again until the total number of mutations reaches 100. In phase 2, we directly set *mutation_time* to 50 and 100, evaluating the effects of these one-time increases. In phase 3, *mutation_time* is directly set to its maximum value of 100, testing the impact of a substantial one-time adjustment.

The experimental results, Figure E.4, show that the gradual increase of *mutation_time* can significantly improve the optimisation results. Especially in the first phase, when *mutation_time* is gradually increased from 0 to 100, the optimisation results are gradually improved, showing the positive effect of increasing *mutation_time* on the optimisation process. Whereas in the second and third phases, although increasing *mutation_time* to a higher value at once can quickly improve the optimisation results, the best optimisation results achieved by gradually increasing *mutation_time* are more significant.

Results for Workshop: Semester 1 and 2

We explored different strategies for optimising the Workshop schedule via genetic algorithms, particularly around the settings of the *mutation_time* and *total_mutation* parameters. During our experiments, we observed two interesting phenomena: while better results were eventually obtained when a gradual increase in *total_mutation* was applied in smaller steps, the first step tended to give better results if a larger value of *total_mutation* was set directly at the initial stage. Similarly, experiments on *mutation_time* show a similar trend. This finding suggests that multiple optimisations with larger parameter values may be a more effective strategy in the application of genetic algorithms.

In the above discussion, We set three different sets of data: *mutation_time*= 500, 400, 200 and *total_mutation*= 8000, 6000, 4000, respectively for workshops in semester 1. The experimental results, Figure E.1, show that as the values of these parameter values increase, the optimisation effect gradually increases, but the computational cost grows accordingly.

With a wider search space and more iterations, the algorithm has a better chance of exploring superior solutions in the solution space. However, this does not mean that an infinite increase in parameter values will always lead to better results, as there is a trade-off, which is computational resource limitations. In addition, iterations may only lead to diminishing marginal returns after a certain level of optimisation has been reached. The stopping condition we set, which terminates when the results of two consecutive iterations are the same, avoids invalid calculations and ensures the stability of the results. We obtained the optimum situation in each situation; the number of teaching for a random generalised timetable is 361; after optimising by Genetic Algorithm, we can get 182, 230, and 233 for three pairs of parameters.

For the semester 2 workshop, we repeated the strategy we used in the semester 1 workshop, we can get Figure E.2. To save significant computational time, we pick the lower values for *mutation_time* and *total_mutation*. They are (300,4000), (150,2000) and (50,1000). We can also find a similar pattern to the model for a workshop in semester 1. This can also verify our conclusion in the semester 1 model, which is a bigger pair of *mutation_time* and *total_mutation*, the more optimised outcome we get. After optimising, we reduced the number of teachings to 162, 174 and 237, corresponding to different pairs.

We explore different strategies for Workshop schedule optimisation by genetic algorithms. Through our experiments, we not only observed the phenomenon that the optimisation effect improves with the increase of parameter values, but also discovered potential strategies to achieve efficient optimisation in the presence of limited computational resources. Based on these findings, we make the following recommendations to the university with the aim of further improving the optimisation efficiency of the Workshop schedule and reducing the number of unnecessary sessions:

- Utilise spare classrooms: Consider allocating spare general-purpose classrooms for Workshop use, particularly for courses with large numbers of students. Our research has found that as the number of students on some courses far exceeds the capacity of the largest Workshop classroom (e.g. over 400, whereas the largest Workshop classroom has a capacity of 84), the number of classes required can be significantly reduced by flexibly allocating spare general classrooms to provide more space for Workshop.
- Utilising high-performance computing resources: although our experimental results demonstrate optimal solutions within a selected range of parameters, there is still the possibility of finding a globally optimal solution. We encourage schools to utilise more powerful computer resources by setting larger values for *mutation_time* and *total_mutation* and performing more iterative loops. This will not only explore a wider solution space but may also find solutions that are closer to the global optimum, further improving the quality of schedule optimisation.

Conclusion

In this report, we provide a comprehensive review of the optimising course timetabling problem through the Particle Swarm Optimization (PSO) algorithm, the Annealing algorithm, and the Genetic Algorithm. Allocating time slots and rooms efficiently for various types of classes, such as lectures and workshops, is a critical issue within educational institutions, particularly for universities.

For lectures, we have tried many times to optimise the model and there are still a small number of time conflicts. For the whole school timetable there are five days and nine time slots in each day so there are forty five time slots in total. Because of the limited time slots, the model have reached to the minimized the number of time conflict although not decreased to zero. We also have tried some assumptions for example add some fake rooms.

Timetabling workshops are much more complex than timetabling lectures because they need many different time slots in a week for one course so that students can satisfy every student's studying requirements. By keeping iterations in the model, the model progressively approaches the optimal situation for the pair of parameters, which means the number of teaching classes has decreased and converged in the end. This process takes a lot of time and needs higher technology facilities to reach. The part of results for the workshop is shown in I.1 with five iterations for a pair of parameters (300,4000). From the bottom of Figure I.1, we notice that the number of teaching decreases from 365 to 222. The part of results for workshops in semester 2 is also shown in J.1, and we can see the number of teaching decreases from 268 to 171 with six iterations for a pair of parameters (300,2000).

3.1 Further Improvements for school

While certain hard constraints have been appropriately incorporated into the model, there remains a scope for the integration of additional soft constraints to augment the model's efficacy and comprehensiveness.

First, the concept of a daily maximum course load should be considered. It is hard for people to fully concentrated in class is they already have many compact classes. It will help to foster a balanced schedule to prevent anyone from encountering an overwhelming or sporadic timetable. Such equilibrium is vital for sustaining high levels of energy and engagement throughout the duration of the classes.

Another critical aspect to address is the challenge posed by different campus locations. The school have two campuses, students have transport time to attend the classes between different campuses. For example, change from Kings Building to George Square takes about fifteen minutes by bus, excluding waiting time for buses. Many students are majoring in Mathematics and Economics, which means they have classes in the School of Mathematics and the School of Economics, but there is no economics class in Kings Building, so they have to travel between two campuses. When students have continuous classes on different campuses, they may miss some important points in the lecture. Students travelling between campuses must be allotted sufficient time to commute. Therefore, trying to avoid continuing classes becomes helpful for students who have continuing classes in different campus courses.

For model algorithms, we can keep adapting the parameters of our model till we find the best parameters which show the best results in the model, and keep running the more optimised model, which will help. Also, there are different algorithms that can be used, such as the hybrids algorithm [13].

As is clear from the review, the optimization of school timetables is a critical component of educational strategy. Through the application of optimization research, we can craft timetables considering many factors. In doing so, we pave the way for an educational environment that is efficient in fostering the growth of our students and teachers alike.

Appendix A

Data preprocessing

A.1 Data preprocessing

Before timetabling, it is necessary to meticulously collating data, this section will show how the measures were taken to taken to integrate courses more effectively to make it easier for further coding.

The data preprocessing stage involved a systematic approach to organizing and preparing data sources. The primary datasets utilized in this process included the ”Anon Enrollment Data,” ”School of Mathematics - Timetable Data,” and ”Timetabling KB Rooms.”
The data was initially segmented by academic term, distinguishing between Semester 1 and Semester 2, in order to facilitate a more structured and coherent analysis. It was then refined by categorizing the academic sessions into lectures and workshops. These lectures and workshops were further divided based on their duration, specifically segregating sessions that lasted 1 hour from those that extended to 2 hours. This stratification enabled a more nuanced approach to scheduling and resource allocation.
Online lectures and workshops were not factored into the timetabling since they did not necessitate the allocation of physical spaces and lectures lasting more than 2 hours were excluded from the schedule as their continuous slot allocation complicated the timetabling process.

A.1.1 Course Integration and Organisation

Firstly, weeks 9-19 were designated as Semester 1, while Weeks 26-37 were categorized as Semester 2, thereby dividing the academic year into two distinct segments. For the ease of coding within the course scheduling system, all lectures and workshops with a duration of 1 hour and 30 minutes were uniformly treated as 2 hours.

Secondly, to ensure the continuity of weekly lectures, courses were grouped strategically in such a way that the course pairs of the same duration were grouped together under a single code to be treated as one course, as exemplified by the combination of MATH11132 and MATH10010, represented as MATH11132 + MATH10010. When combining courses in this manner, the enrollment number attributed to the unified course is determined by taking the higher of the two individual course enrollments, ensuring that adequate space and resources are allocated to accommodate all students.

In cases, where the lectures happen more than once a week, unique identification were given by appending letters to the course code, to treat each class as unique, resulting in codes such as MATH11132A and MATH11132B. Another strategy that was taken was that the courses whose student enrollment number exceeded the maximum capacity of any given room, has been divided into two separate classes to manage size effectively. For example, a course has 620 students enrolled in total and the maximum capacity of the room given is 500, so we decided to split them into two classes, one class accommodates 300 students under the code MATH08058, while the other, with 320 students, is identified as MATH08058_1. These adjustments facilitates smoother operation and scheduling.

A.2 Changes made to the Original Plan

During the course of the academic scheduling process, certain obstacles emerged that required a re-working of the original objectives and constraints. To address these difficulties, adjustments were made, ensuring that the system remained flexible and responsive to the needs of both the institution and its students.
According to the mid-project report, it was initially planned to ”*provide a comprehensive solution that optimizes(maximizes) classroom usage and minimizes scheduling conflicts*”. But as the project progressed we had to rethink our goals and redefine them. The objective functions and a few constraints were changed, as will be explained in the next chapter.

Appendix B

Explanation for constraints

- (1) ensures that at any given time h, in any room r, on any day d, at most one course c can be scheduled. This prevents the overlap of courses in the same room and time slot.
- (2) ensures that a that each course c can only be scheduled in one room r at a particular time slot h on any day d.
- (3) ensures that that across all rooms r and all courses c, only one course can be scheduled in any time slot h on any day d.
- (4) ensures that the capacity cap_r of each room r is greater than or equal to the number of students st_c in any course c. Because each room must have enough seats to accommodate all the students enrolled in the course.
- (5) ensures that $yrhc_2$ can only be 1 if the first lecture in same course c_2 is scheduled in the beginning of the consecutive time slot.
- (6) ensures that $yrhc_2$ can only be 1 if the second lecture in same course c_2 is also scheduled in the consecutive time slot.
- (7) ensures for a 2 hours lecture, in the non-last time slot, if the lesson starts at time h, then the second lesson should also take place at time $h + 1$.
- (8) ensures for two hours lecture(requiring consecutive slots) are not scheduled in the last time slot of any day.

Appendix C

PSO functions

Fitness function is the same as simulated annealing algorithm since we have the same optimization goal. Particle Class is defined as a particle in the swarm, representing a potential solution (timetable). Each particle maintains its current position (timetable configuration), velocity (change tendency), and the best position it has found (optimal timetable it has discovered). The best fitness attribute holds the fitness of the best position, guiding the particle's movement towards more promising areas of the solution space([8]).

Update velocity Function adjusts each particle's velocity based on its past movement, the difference between its current and best positions, and the difference between its position and the global best position. The parameters w (inertia weight), c1 (cognitive coefficient), and c2 (social coefficient) balance the influence of a particle's momentum, personal best, and the swarm's best discoveries([14]). The max velocity parameter limits the maximum change, preventing overly drastic modifications that could destabilize the search process.

Reinitialize particle Function occasionally reinitializes a particle to a completely new timetable to introduce additional diversity into the swarm and prevent stagnation([8]). This process, triggered with a 5% chance, ensures that the swarm can explore widely separated regions of the solution space.

Update position Function moves each particle according to its velocity, updating its timetable while respecting the constraints of classroom capacities and scheduling hours([8]). This function ensures that each particle iteratively adjusts its solution in search of a more optimized timetable.

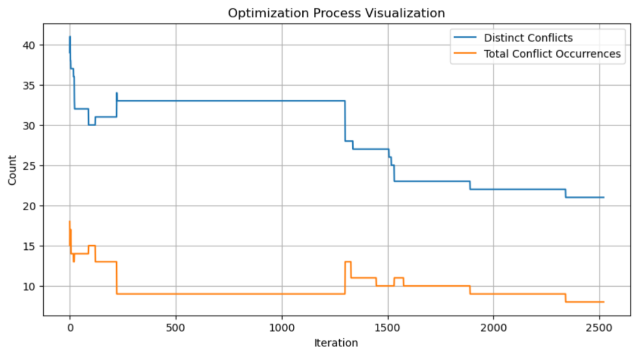
Find unused time slots Function identifies time slots within the current timetable that are not utilized, providing an opportunity to reallocate courses and potentially reduce conflicts. This function supports the optimization process by highlighting areas where improvements can be made.

Local search Function performs a local search on a particle's position, making minor adjustments to course timings to explore nearby configurations. This search can lead to incremental improvements in the timetable's fitness by fine-tuning the scheduling details([9]).

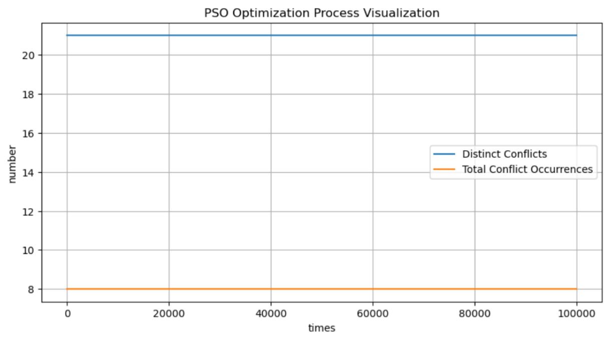
Update particle Function integrates the velocity update and position update processes, including local search, to comprehensively refine a particle's solution. This holistic update process ensures that each particle's movement is deliberate and geared towards finding a more optimal timetable([8]).

Appendix D

Result figure for Lecture

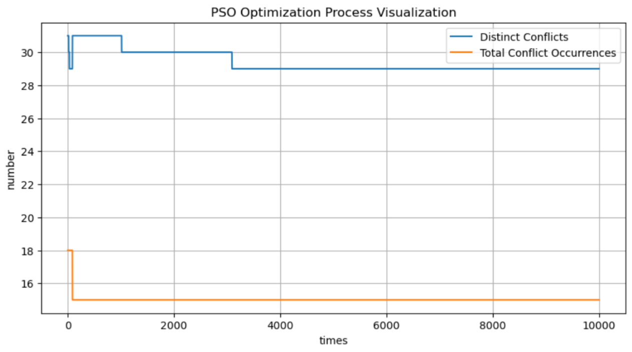


(a) Simulated annealing algorithm

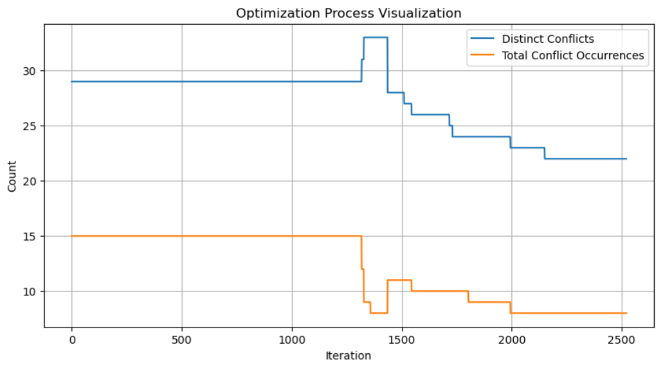


(b) PSO algorithm

Figure D.1: First use SA algorithm then use PSO algorithm



(a) PSO algorithm



(b) Simulated annealing algorithm

Figure D.2: First use PSO algorithm then use SA algorithm

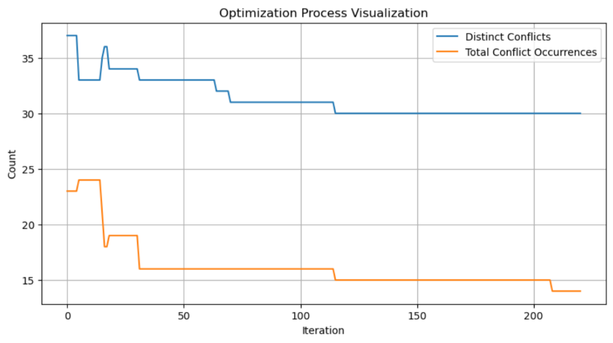


Figure D.3: Avoid allocating courses on Wednesday afternoon

Appendix E

Result figure for workshop

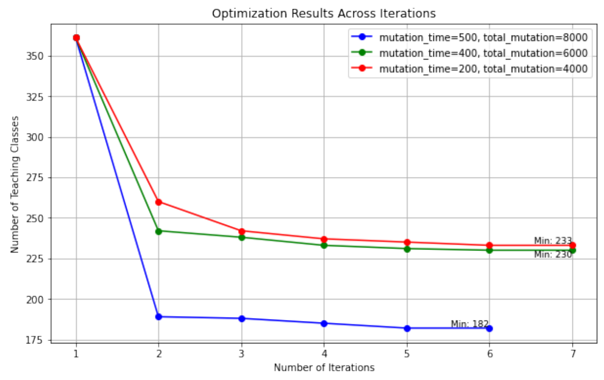


Figure E.1: Optimization results for Semester 1 workshops

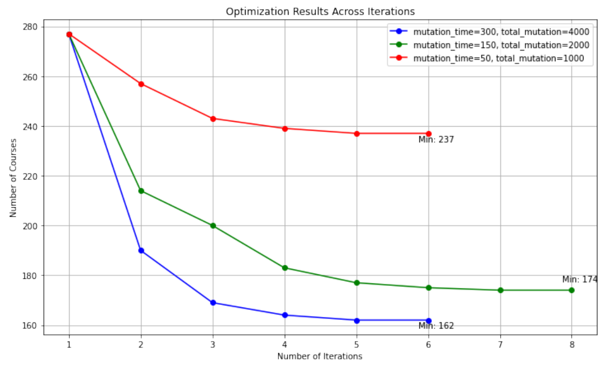


Figure E.2: Optimization results for Semester 2 workshops

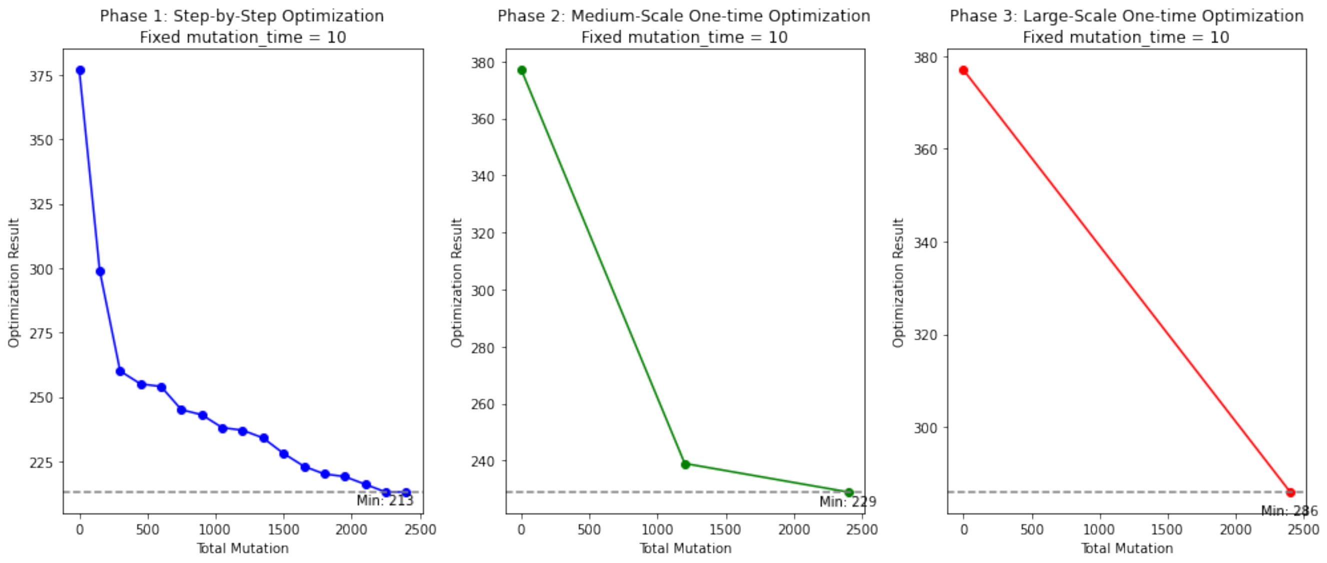


Figure E.3

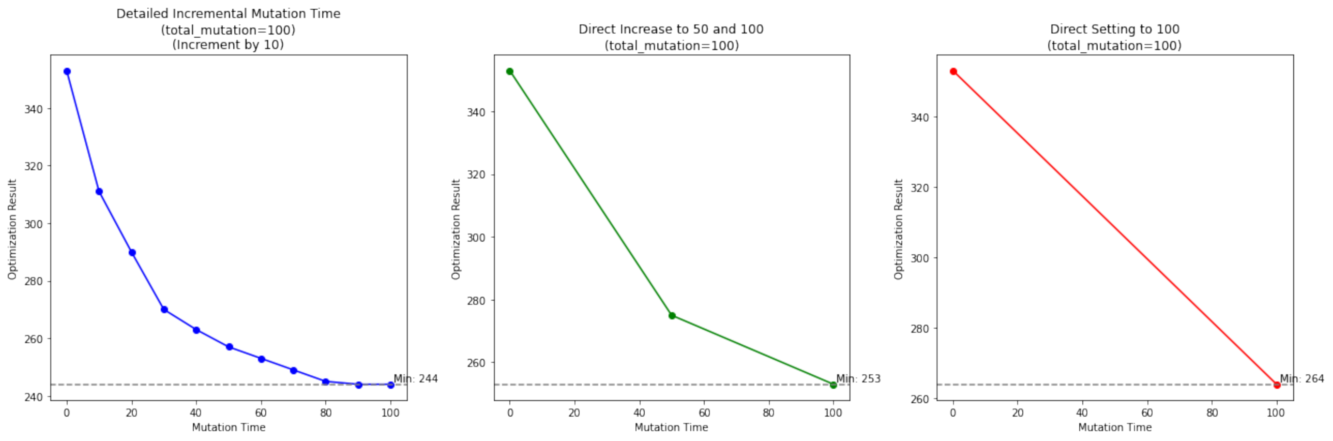
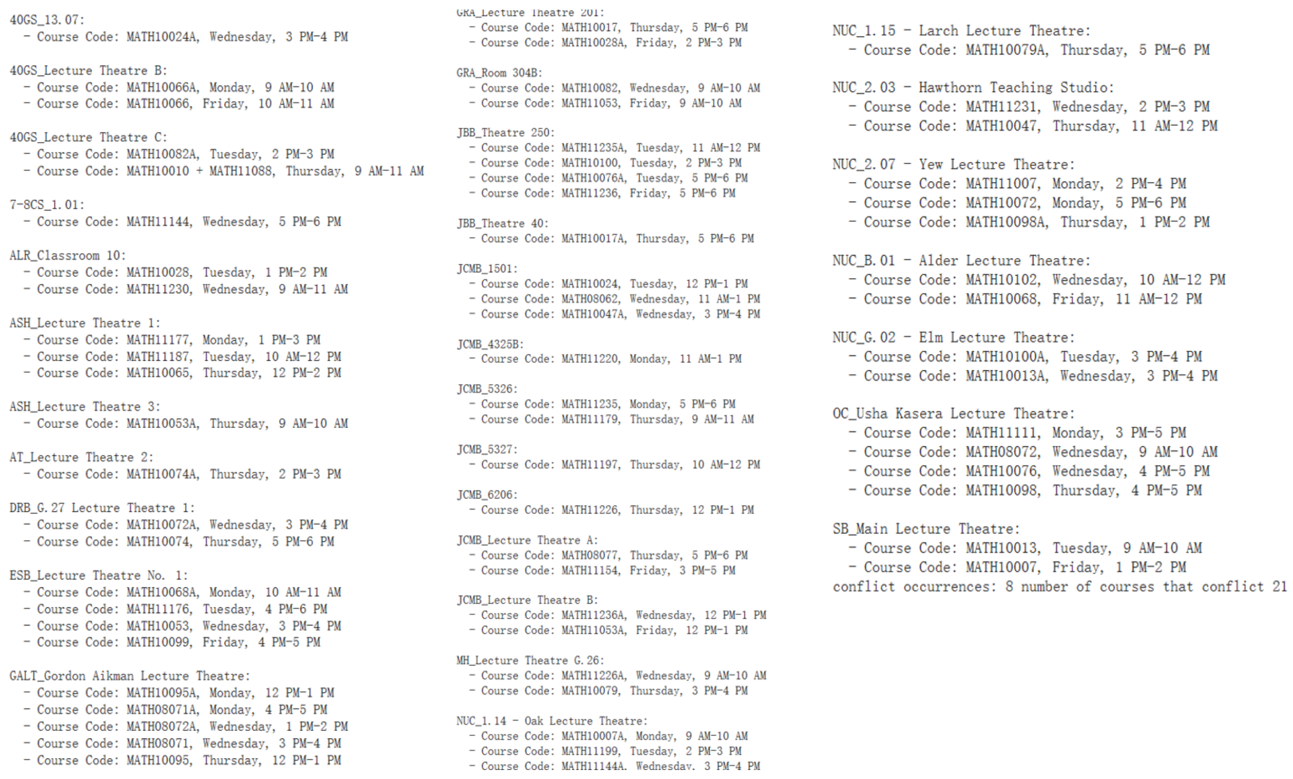

Detailed Incremental Mutation Time
(total_mutation=100)
(Increment by 10)
Optimization Result
Mutation Time
Min: 244
Direct Increase to 50 and 100
(total_mutation=100)
Optimization Result
Mutation Time
Min: 253
Direct Setting to 100
(total_mutation=100)
Optimization Result
Mutation Time
Min: 264

Figure E.4

Appendix F

Result of SEM1 lecture timetable

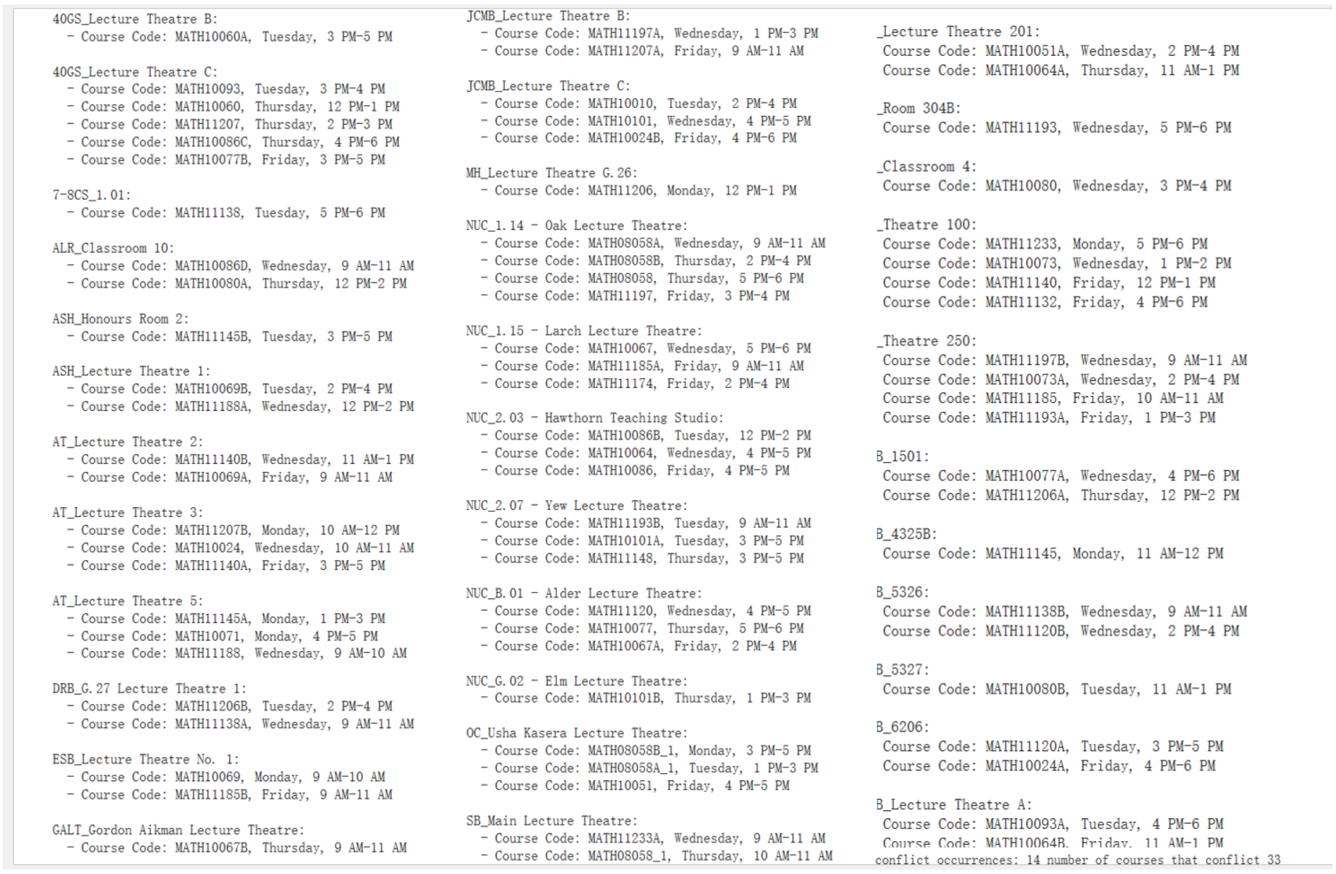


```
40GS_13.07:
  - Course Code: MATH10024A, Wednesday, 3 PM-4 PM

40GS_Lecture Theatre B:
  - Course Code: MATH10066A, Monday, 9 AM-10 AM
  - Course Code: MATH10066, Friday, 10 AM-11 AM

40GS_Lecture Theatre C:
  - Course Code: MATH10082A, Tuesday, 2 PM-3 PM
  - Course Code: MATH10010 + MATH11088, Thursday, 9 AM-11 AM

7-8CS_1.01:
  - Course Code: MATH11144, Wednesday, 5 PM-6 PM

ALR_Classroom 10:
  - Course Code: MATH10028, Tuesday, 1 PM-2 PM
  - Course Code: MATH11230, Wednesday, 9 AM-11 AM

ASH_Lecture Theatre 1:
  - Course Code: MATH11177, Monday, 1 PM-3 PM
  - Course Code: MATH11187, Tuesday, 10 AM-12 PM
  - Course Code: MATH10065, Thursday, 12 PM-2 PM

ASH_Lecture Theatre 3:
  - Course Code: MATH10053A, Thursday, 9 AM-10 AM

AT_Lecture Theatre 2:
  - Course Code: MATH10074A, Thursday, 2 PM-3 PM

DRB_G.27 Lecture Theatre 1:
  - Course Code: MATH10072A, Wednesday, 3 PM-4 PM
  - Course Code: MATH10074, Thursday, 5 PM-6 PM

ESB_Lecture Theatre No. 1:
  - Course Code: MATH10068A, Monday, 10 AM-11 AM
  - Course Code: MATH11176, Tuesday, 4 PM-6 PM
  - Course Code: MATH10053, Wednesday, 3 PM-4 PM
  - Course Code: MATH10099, Friday, 4 PM-5 PM

GALT_Gordon Aikman Lecture Theatre:
  - Course Code: MATH10095A, Monday, 12 PM-1 PM
  - Course Code: MATH08071A, Monday, 4 PM-5 PM
  - Course Code: MATH08072A, Wednesday, 1 PM-2 PM
  - Course Code: MATH08071, Wednesday, 3 PM-4 PM
  - Course Code: MATH10095, Thursday, 12 PM-1 PM

GRA_Lecture Theatre 201:
  - Course Code: MATH10017, Thursday, 5 PM-6 PM
  - Course Code: MATH10028A, Friday, 2 PM-3 PM

GRA_Room 304B:
  - Course Code: MATH10082, Wednesday, 9 AM-10 AM
  - Course Code: MATH11053, Friday, 9 AM-10 AM

JBB_Theatre 250:
  - Course Code: MATH11235A, Tuesday, 11 AM-12 PM
  - Course Code: MATH10100, Tuesday, 2 PM-3 PM
  - Course Code: MATH10076A, Tuesday, 5 PM-6 PM
  - Course Code: MATH11236, Friday, 5 PM-6 PM

JBB_Theatre 40:
  - Course Code: MATH10017A, Thursday, 5 PM-6 PM

JCMB_1501:
  - Course Code: MATH10024, Tuesday, 12 PM-1 PM
  - Course Code: MATH08062, Wednesday, 11 AM-1 PM
  - Course Code: MATH10047A, Wednesday, 3 PM-4 PM

JCMB_4325B:
  - Course Code: MATH11220, Monday, 11 AM-1 PM

JCMB_5326:
  - Course Code: MATH11235, Monday, 5 PM-6 PM
  - Course Code: MATH11179, Thursday, 9 AM-11 AM

JCMB_5327:
  - Course Code: MATH11197, Thursday, 10 AM-12 PM

JCMB_6206:
  - Course Code: MATH11226, Thursday, 12 PM-1 PM

JCMB_Lecture Theatre A:
  - Course Code: MATH08077, Thursday, 5 PM-6 PM
  - Course Code: MATH11154, Friday, 3 PM-5 PM

JCMB_Lecture Theatre B:
  - Course Code: MATH11236A, Wednesday, 12 PM-1 PM
  - Course Code: MATH11053A, Friday, 12 PM-1 PM

MH_Lecture Theatre G.26:
  - Course Code: MATH11226A, Wednesday, 9 AM-10 AM
  - Course Code: MATH10079, Thursday, 3 PM-4 PM

NUC_1.14 - Oak Lecture Theatre:
  - Course Code: MATH10007A, Monday, 9 AM-10 AM
  - Course Code: MATH11199, Tuesday, 2 PM-3 PM
  - Course Code: MATH11144A, Wednesday, 3 PM-4 PM

NUC_1.15 - Larch Lecture Theatre:
  - Course Code: MATH10079A, Thursday, 5 PM-6 PM

NUC_2.03 - Hawthorn Teaching Studio:
  - Course Code: MATH11231, Wednesday, 2 PM-3 PM
  - Course Code: MATH10047, Thursday, 11 AM-12 PM

NUC_2.07 - Yew Lecture Theatre:
  - Course Code: MATH11007, Monday, 2 PM-4 PM
  - Course Code: MATH10072, Monday, 5 PM-6 PM
  - Course Code: MATH10098A, Thursday, 1 PM-2 PM

NUC_B.01 - Alder Lecture Theatre:
  - Course Code: MATH10102, Wednesday, 10 AM-12 PM
  - Course Code: MATH10068, Friday, 11 AM-12 PM

NUC_G.02 - Elm Lecture Theatre:
  - Course Code: MATH10100A, Tuesday, 3 PM-4 PM
  - Course Code: MATH10013A, Wednesday, 3 PM-4 PM

OC_Usha Kasera Lecture Theatre:
  - Course Code: MATH11111, Monday, 3 PM-5 PM
  - Course Code: MATH08072, Wednesday, 9 AM-10 AM
  - Course Code: MATH10076, Wednesday, 4 PM-5 PM
  - Course Code: MATH10098, Thursday, 4 PM-5 PM

SB_Main Lecture Theatre:
  - Course Code: MATH10013, Tuesday, 9 AM-10 AM
  - Course Code: MATH10007, Friday, 1 PM-2 PM
conflict occurrences: 8 number of courses that conflict 21
```

Figure F.1

Appendix G

Result for SEM2 lecture timetable



```
40GS_Lecture Theatre B:
  - Course Code: MATH10060A, Tuesday, 3 PM-5 PM

40GS_Lecture Theatre C:
  - Course Code: MATH10093, Tuesday, 3 PM-4 PM
  - Course Code: MATH10060, Thursday, 12 PM-1 PM
  - Course Code: MATH11207, Thursday, 2 PM-3 PM
  - Course Code: MATH10086C, Thursday, 4 PM-6 PM
  - Course Code: MATH10077B, Friday, 3 PM-5 PM

7-8CS_1.01:
  - Course Code: MATH11138, Tuesday, 5 PM-6 PM

ALR_Classroom 10:
  - Course Code: MATH10086D, Wednesday, 9 AM-11 AM
  - Course Code: MATH10080A, Thursday, 12 PM-2 PM

ASH_Honours Room 2:
  - Course Code: MATH11145B, Tuesday, 3 PM-5 PM

ASH_Lecture Theatre 1:
  - Course Code: MATH10069B, Tuesday, 2 PM-4 PM
  - Course Code: MATH11188A, Wednesday, 12 PM-2 PM

AT_Lecture Theatre 2:
  - Course Code: MATH11140B, Wednesday, 11 AM-1 PM
  - Course Code: MATH10069A, Friday, 9 AM-11 AM

AT_Lecture Theatre 3:
  - Course Code: MATH11207B, Monday, 10 AM-12 PM
  - Course Code: MATH10024, Wednesday, 10 AM-11 AM
  - Course Code: MATH11140A, Friday, 3 PM-5 PM

AT_Lecture Theatre 5:
  - Course Code: MATH11145A, Monday, 1 PM-3 PM
  - Course Code: MATH10071, Monday, 4 PM-5 PM
  - Course Code: MATH11188, Wednesday, 9 AM-10 AM

DRB_G.27 Lecture Theatre 1:
  - Course Code: MATH11206B, Tuesday, 2 PM-4 PM
  - Course Code: MATH11138A, Wednesday, 9 AM-11 AM

ESB_Lecture Theatre No. 1:
  - Course Code: MATH10069, Monday, 9 AM-10 AM
  - Course Code: MATH11185B, Friday, 9 AM-11 AM

GALT_Gordon Aikman Lecture Theatre:
  - Course Code: MATH10067B, Thursday, 9 AM-11 AM

JCMB_Lecture Theatre B:
  - Course Code: MATH11197A, Wednesday, 1 PM-3 PM
  - Course Code: MATH11207A, Friday, 9 AM-11 AM

JCMB_Lecture Theatre C:
  - Course Code: MATH10010, Tuesday, 2 PM-4 PM
  - Course Code: MATH10101, Wednesday, 4 PM-5 PM
  - Course Code: MATH10024B, Friday, 4 PM-6 PM

MH_Lecture Theatre G.26:
  - Course Code: MATH11206, Monday, 12 PM-1 PM

NUC_1.14 - Oak Lecture Theatre:
  - Course Code: MATH08058A, Wednesday, 9 AM-11 AM
  - Course Code: MATH08058B, Thursday, 2 PM-4 PM
  - Course Code: MATH08058, Thursday, 5 PM-6 PM
  - Course Code: MATH11197, Friday, 3 PM-4 PM

NUC_1.15 - Larch Lecture Theatre:
  - Course Code: MATH10067, Wednesday, 5 PM-6 PM
  - Course Code: MATH11185A, Friday, 9 AM-11 AM
  - Course Code: MATH11174, Friday, 2 PM-4 PM

NUC_2.03 - Hawthorn Teaching Studio:
  - Course Code: MATH10086B, Tuesday, 12 PM-2 PM
  - Course Code: MATH10064, Wednesday, 4 PM-5 PM
  - Course Code: MATH10086, Friday, 4 PM-5 PM

NUC_2.07 - Yew Lecture Theatre:
  - Course Code: MATH11193B, Tuesday, 9 AM-11 AM
  - Course Code: MATH10101A, Tuesday, 3 PM-5 PM
  - Course Code: MATH11148, Thursday, 3 PM-5 PM

NUC_B.01 - Alder Lecture Theatre:
  - Course Code: MATH11120, Wednesday, 4 PM-5 PM
  - Course Code: MATH10077, Thursday, 5 PM-6 PM
  - Course Code: MATH10067A, Friday, 2 PM-4 PM

NUC_G.02 - Elm Lecture Theatre:
  - Course Code: MATH10101B, Thursday, 1 PM-3 PM

OC_Usha Kasera Lecture Theatre:
  - Course Code: MATH08058B_1, Monday, 3 PM-5 PM
  - Course Code: MATH08058A_1, Tuesday, 1 PM-3 PM
  - Course Code: MATH10051, Friday, 4 PM-5 PM

SB_Main Lecture Theatre:
  - Course Code: MATH11233A, Wednesday, 9 AM-11 AM
  - Course Code: MATH08058_1, Thursday, 10 AM-11 AM

_Lecture Theatre 201:
 Course Code: MATH10051A, Wednesday, 2 PM-4 PM
 Course Code: MATH10064A, Thursday, 11 AM-1 PM

_Room 304B:
 Course Code: MATH11193, Wednesday, 5 PM-6 PM

_Classroom 4:
 Course Code: MATH10080, Wednesday, 3 PM-4 PM

_Theatre 100:
 Course Code: MATH11233, Monday, 5 PM-6 PM
 Course Code: MATH10073, Wednesday, 1 PM-2 PM
 Course Code: MATH11140, Friday, 12 PM-1 PM
 Course Code: MATH11132, Friday, 4 PM-6 PM

_Theatre 250:
 Course Code: MATH11197B, Wednesday, 9 AM-11 AM
 Course Code: MATH10073A, Wednesday, 2 PM-4 PM
 Course Code: MATH11185, Friday, 10 AM-11 AM
 Course Code: MATH11193A, Friday, 1 PM-3 PM

B_1501:
 Course Code: MATH10077A, Wednesday, 4 PM-6 PM
 Course Code: MATH11206A, Thursday, 12 PM-2 PM

B_4325B:
 Course Code: MATH11145, Monday, 11 AM-12 PM

B_5326:
 Course Code: MATH11138B, Wednesday, 9 AM-11 AM
 Course Code: MATH11120B, Wednesday, 2 PM-4 PM

B_5327:
 Course Code: MATH10080B, Tuesday, 11 AM-1 PM

B_6206:
 Course Code: MATH11120A, Tuesday, 3 PM-5 PM
 Course Code: MATH10024A, Friday, 4 PM-6 PM

B_Lecture Theatre A:
 Course Code: MATH10093A, Tuesday, 4 PM-6 PM
 Course Code: MATH10064B, Friday, 11 AM-1 PM
conflict occurrences: 14 number of courses that conflict 33
```

Figure G.1

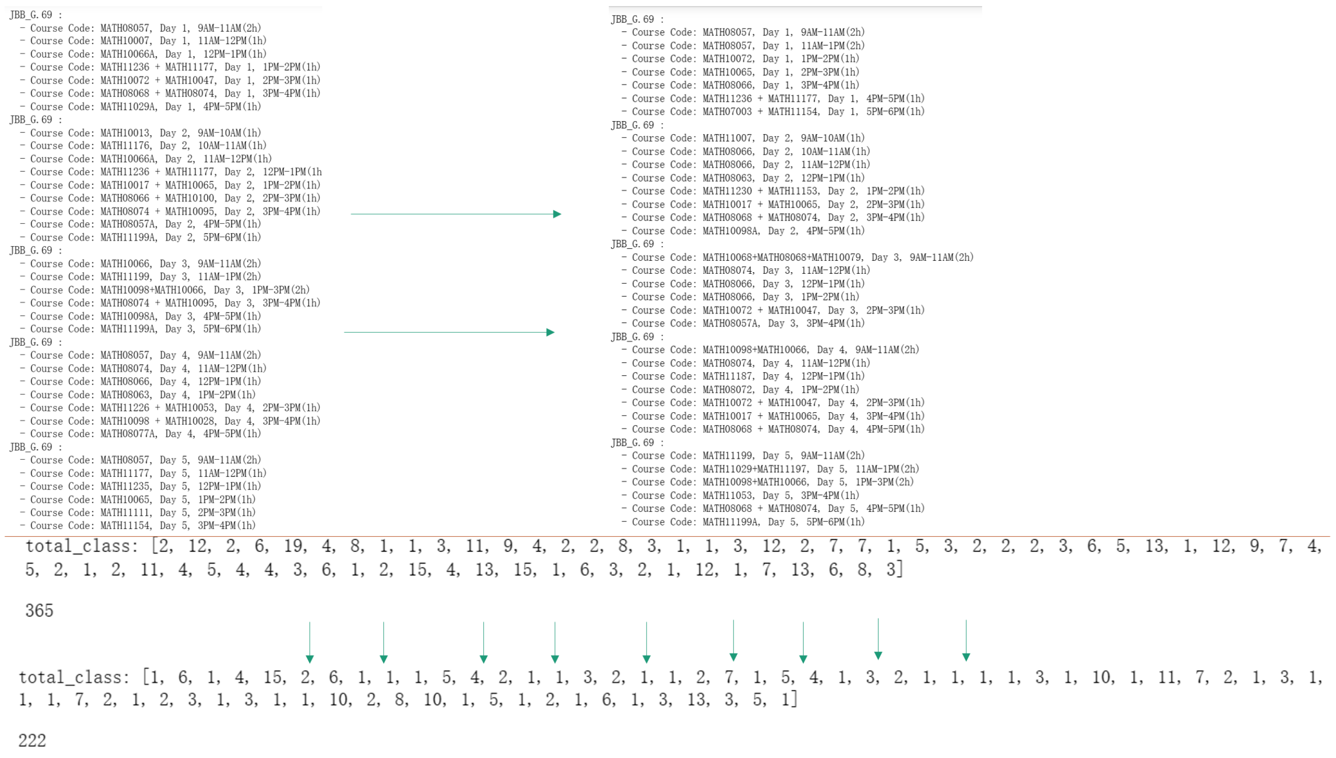
Appendix H

Code

https://github.com/ZacZhang-H/University-timetable-OR.git

Appendix I

Workshop result for SEM1



```
JBB_G.69 :
  - Course Code: MATH08057, Day 1, 9AM-11AM(2h)
  - Course Code: MATH10007, Day 1, 11AM-12PM(1h)
  - Course Code: MATH10066A, Day 1, 12PM-1PM(1h)
  - Course Code: MATH11236 + MATH11177, Day 1, 1PM-2PM(1h)
  - Course Code: MATH10072 + MATH10047, Day 1, 2PM-3PM(1h)
  - Course Code: MATH08068 + MATH08074, Day 1, 3PM-4PM(1h)
  - Course Code: MATH11029A, Day 1, 4PM-5PM(1h)
JBB_G.69 :
  - Course Code: MATH10013, Day 2, 9AM-10AM(1h)
  - Course Code: MATH11176, Day 2, 10AM-11AM(1h)
  - Course Code: MATH10066A, Day 2, 11AM-12PM(1h)
  - Course Code: MATH11236 + MATH11177, Day 2, 12PM-1PM(1h
  - Course Code: MATH10017 + MATH10065, Day 2, 1PM-2PM(1h)
  - Course Code: MATH08066 + MATH10100, Day 2, 2PM-3PM(1h)
  - Course Code: MATH08074 + MATH10095, Day 2, 3PM-4PM(1h)
  - Course Code: MATH08057A, Day 2, 4PM-5PM(1h)
  - Course Code: MATH11199A, Day 2, 5PM-6PM(1h)
JBB_G.69 :
  - Course Code: MATH10066, Day 3, 9AM-11AM(2h)
  - Course Code: MATH11199, Day 3, 11AM-1PM(2h)
  - Course Code: MATH10098+MATH10066, Day 3, 1PM-3PM(2h)
  - Course Code: MATH08074 + MATH10095, Day 3, 3PM-4PM(1h)
  - Course Code: MATH10098A, Day 3, 4PM-5PM(1h)
  - Course Code: MATH11199A, Day 3, 5PM-6PM(1h)
JBB_G.69 :
  - Course Code: MATH08057, Day 4, 9AM-11AM(2h)
  - Course Code: MATH08074, Day 4, 11AM-12PM(1h)
  - Course Code: MATH08066, Day 4, 12PM-1PM(1h)
  - Course Code: MATH08063, Day 4, 1PM-2PM(1h)
  - Course Code: MATH11226 + MATH10053, Day 4, 2PM-3PM(1h)
  - Course Code: MATH10098 + MATH10028, Day 4, 3PM-4PM(1h)
  - Course Code: MATH08077A, Day 4, 4PM-5PM(1h)
JBB_G.69 :
  - Course Code: MATH08057, Day 5, 9AM-11AM(2h)
  - Course Code: MATH11177, Day 5, 11AM-12PM(1h)
  - Course Code: MATH11235, Day 5, 12PM-1PM(1h)
  - Course Code: MATH10065, Day 5, 1PM-2PM(1h)
  - Course Code: MATH11111, Day 5, 2PM-3PM(1h)
  - Course Code: MATH11154, Day 5, 3PM-4PM(1h)
```

```
JBB_G.69 :
  - Course Code: MATH08057, Day 1, 9AM-11AM(2h)
  - Course Code: MATH08057, Day 1, 11AM-1PM(2h)
  - Course Code: MATH10072, Day 1, 1PM-2PM(1h)
  - Course Code: MATH10065, Day 1, 2PM-3PM(1h)
  - Course Code: MATH08066, Day 1, 3PM-4PM(1h)
  - Course Code: MATH11236 + MATH11177, Day 1, 4PM-5PM(1h)
  - Course Code: MATH07003 + MATH11154, Day 1, 5PM-6PM(1h)
JBB_G.69 :
  - Course Code: MATH11007, Day 2, 9AM-10AM(1h)
  - Course Code: MATH08066, Day 2, 10AM-11AM(1h)
  - Course Code: MATH08066, Day 2, 11AM-12PM(1h)
  - Course Code: MATH08063, Day 2, 12PM-1PM(1h)
  - Course Code: MATH11230 + MATH11153, Day 2, 1PM-2PM(1h)
  - Course Code: MATH10017 + MATH10065, Day 2, 2PM-3PM(1h)
  - Course Code: MATH08068 + MATH08074, Day 2, 3PM-4PM(1h)
  - Course Code: MATH10098A, Day 2, 4PM-5PM(1h)
JBB_G.69 :
  - Course Code: MATH10068+MATH08068+MATH10079, Day 3, 9AM-11AM(2h)
  - Course Code: MATH08074, Day 3, 11AM-12PM(1h)
  - Course Code: MATH08066, Day 3, 12PM-1PM(1h)
  - Course Code: MATH08066, Day 3, 1PM-2PM(1h)
  - Course Code: MATH10072 + MATH10047, Day 3, 2PM-3PM(1h)
  - Course Code: MATH08057A, Day 3, 3PM-4PM(1h)
JBB_G.69 :
  - Course Code: MATH10098+MATH10066, Day 4, 9AM-11AM(2h)
  - Course Code: MATH08074, Day 4, 11AM-12PM(1h)
  - Course Code: MATH11187, Day 4, 12PM-1PM(1h)
  - Course Code: MATH08072, Day 4, 1PM-2PM(1h)
  - Course Code: MATH10072 + MATH10047, Day 4, 2PM-3PM(1h)
  - Course Code: MATH10017 + MATH10065, Day 4, 3PM-4PM(1h)
  - Course Code: MATH08068 + MATH08074, Day 4, 4PM-5PM(1h)
JBB_G.69 :
  - Course Code: MATH11199, Day 5, 9AM-11AM(2h)
  - Course Code: MATH11029+MATH11197, Day 5, 11AM-1PM(2h)
  - Course Code: MATH10098+MATH10066, Day 5, 1PM-3PM(2h)
  - Course Code: MATH11053, Day 5, 3PM-4PM(1h)
  - Course Code: MATH08068 + MATH08074, Day 5, 4PM-5PM(1h)
  - Course Code: MATH11199A, Day 5, 5PM-6PM(1h)
```

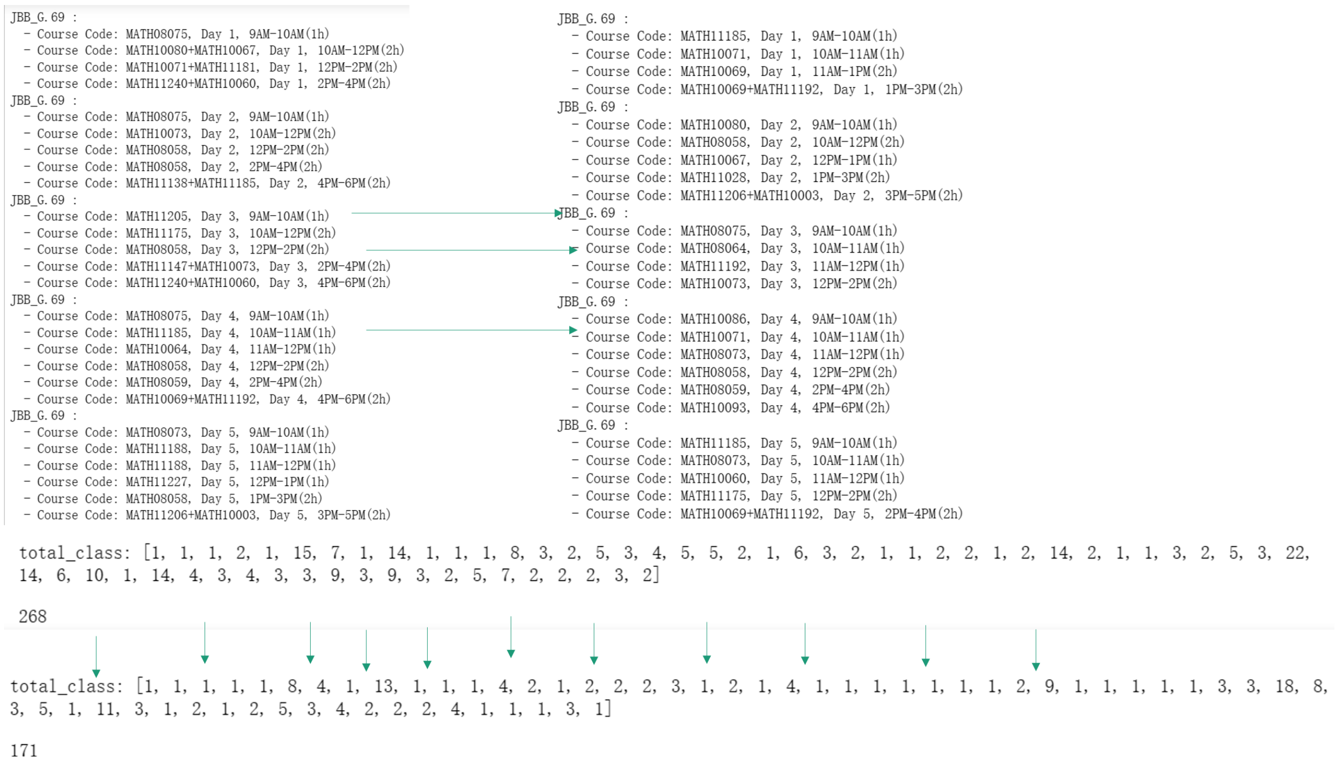
```
total_class: [2, 12, 2, 6, 19, 4, 8, 1, 1, 3, 11, 9, 4, 2, 2, 8, 3, 1, 1, 3, 12, 2, 7, 7, 1, 5, 3, 2, 2, 2, 3, 6, 5, 13, 1, 12, 9, 7, 4,
5, 2, 1, 2, 11, 4, 5, 4, 4, 3, 6, 1, 2, 15, 4, 13, 15, 1, 6, 3, 2, 1, 12, 1, 7, 13, 6, 8, 3]

365
```

```
total_class: [1, 6, 1, 4, 15, 2, 6, 1, 1, 1, 5, 4, 2, 1, 1, 3, 2, 1, 1, 2, 7, 1, 5, 4, 1, 3, 2, 1, 1, 1, 1, 3, 1, 10, 1, 11, 7, 2, 1, 3, 1,
1, 1, 7, 2, 1, 2, 3, 1, 3, 1, 1, 10, 2, 8, 10, 1, 5, 1, 2, 1, 6, 1, 3, 13, 3, 5, 1]

222
```

Figure I.1

Appendix J

Workshop result for SEM2



```
JBB_G.69 :
  - Course Code: MATH08075, Day 1, 9AM-10AM(1h)
  - Course Code: MATH10080+MATH10067, Day 1, 10AM-12PM(2h)
  - Course Code: MATH10071+MATH11181, Day 1, 12PM-2PM(2h)
  - Course Code: MATH11240+MATH10060, Day 1, 2PM-4PM(2h)
JBB_G.69 :
  - Course Code: MATH08075, Day 2, 9AM-10AM(1h)
  - Course Code: MATH10073, Day 2, 10AM-12PM(2h)
  - Course Code: MATH08058, Day 2, 12PM-2PM(2h)
  - Course Code: MATH08058, Day 2, 2PM-4PM(2h)
  - Course Code: MATH11138+MATH11185, Day 2, 4PM-6PM(2h)
JBB_G.69 :
  - Course Code: MATH11205, Day 3, 9AM-10AM(1h)
  - Course Code: MATH11175, Day 3, 10AM-12PM(2h)
  - Course Code: MATH08058, Day 3, 12PM-2PM(2h)
  - Course Code: MATH11147+MATH10073, Day 3, 2PM-4PM(2h)
  - Course Code: MATH11240+MATH10060, Day 3, 4PM-6PM(2h)
JBB_G.69 :
  - Course Code: MATH08075, Day 4, 9AM-10AM(1h)
  - Course Code: MATH11185, Day 4, 10AM-11AM(1h)
  - Course Code: MATH10064, Day 4, 11AM-12PM(1h)
  - Course Code: MATH08058, Day 4, 12PM-2PM(2h)
  - Course Code: MATH08059, Day 4, 2PM-4PM(2h)
  - Course Code: MATH10069+MATH11192, Day 4, 4PM-6PM(2h)
JBB_G.69 :
  - Course Code: MATH08073, Day 5, 9AM-10AM(1h)
  - Course Code: MATH11188, Day 5, 10AM-11AM(1h)
  - Course Code: MATH11188, Day 5, 11AM-12PM(1h)
  - Course Code: MATH11227, Day 5, 12PM-1PM(1h)
  - Course Code: MATH08058, Day 5, 1PM-3PM(2h)
  - Course Code: MATH11206+MATH10003, Day 5, 3PM-5PM(2h)
```

```
JBB_G.69 :
  - Course Code: MATH11185, Day 1, 9AM-10AM(1h)
  - Course Code: MATH10071, Day 1, 10AM-11AM(1h)
  - Course Code: MATH10069, Day 1, 11AM-1PM(2h)
  - Course Code: MATH10069+MATH11192, Day 1, 1PM-3PM(2h)
JBB_G.69 :
  - Course Code: MATH10080, Day 2, 9AM-10AM(1h)
  - Course Code: MATH08058, Day 2, 10AM-12PM(2h)
  - Course Code: MATH10067, Day 2, 12PM-1PM(1h)
  - Course Code: MATH11028, Day 2, 1PM-3PM(2h)
  - Course Code: MATH11206+MATH10003, Day 2, 3PM-5PM(2h)
JBB_G.69 :
  - Course Code: MATH08075, Day 3, 9AM-10AM(1h)
  - Course Code: MATH08064, Day 3, 10AM-11AM(1h)
  - Course Code: MATH11192, Day 3, 11AM-12PM(1h)
  - Course Code: MATH10073, Day 3, 12PM-2PM(2h)
JBB_G.69 :
  - Course Code: MATH10086, Day 4, 9AM-10AM(1h)
  - Course Code: MATH10071, Day 4, 10AM-11AM(1h)
  - Course Code: MATH08073, Day 4, 11AM-12PM(1h)
  - Course Code: MATH08058, Day 4, 12PM-2PM(2h)
  - Course Code: MATH08059, Day 4, 2PM-4PM(2h)
  - Course Code: MATH10093, Day 4, 4PM-6PM(2h)
JBB_G.69 :
  - Course Code: MATH11185, Day 5, 9AM-10AM(1h)
  - Course Code: MATH08073, Day 5, 10AM-11AM(1h)
  - Course Code: MATH10060, Day 5, 11AM-12PM(1h)
  - Course Code: MATH11175, Day 5, 12PM-2PM(2h)
  - Course Code: MATH10069+MATH11192, Day 5, 2PM-4PM(2h)
```

```
total_class: [1, 1, 1, 2, 1, 15, 7, 1, 14, 1, 1, 1, 8, 3, 2, 5, 3, 4, 5, 5, 2, 1, 6, 3, 2, 1, 1, 2, 2, 1, 2, 14, 2, 1, 1, 3, 2, 5, 3, 22,
14, 6, 10, 1, 14, 4, 3, 4, 3, 3, 9, 3, 9, 3, 2, 5, 7, 2, 2, 2, 3, 2]

268
```

```
total_class: [1, 1, 1, 1, 1, 8, 4, 1, 13, 1, 1, 1, 4, 2, 1, 2, 2, 2, 3, 1, 2, 1, 4, 1, 1, 1, 1, 1, 1, 1, 2, 9, 1, 1, 1, 1, 1, 3, 3, 18, 8,
3, 5, 1, 11, 3, 1, 2, 1, 2, 5, 3, 4, 2, 2, 2, 4, 1, 1, 1, 3, 1]

171
```

Figure J.1

Bibliography

[1] F Melício, P Caldeira, and A Rosa. Solving the timetabling problem with simulated annealing. *Enterprise information systems*, pages 171–178, 2000.

[2] Diana Sánchez-Partida, Enrique Gabriel Baquela, Jaime Mora-Vargas, and Neale R Smith. Case study: Simulated annealing for improving the educational timetable. *Nova scientia*, 8(17):340–360, 2016.

[3] Nuno Leite, Fernando Melício, and Agostinho C Rosa. A fast simulated annealing algorithm for the examination timetabling problem. *Expert Systems with Applications*, 122:137–151, 2019.

[4] Dimitris Bertsimas and John Tsitsiklis. Simulated annealing. *Statistical science*, 8(1):10–15, 1993.

[5] Isabel Beichl and Francis Sullivan. The metropolis algorithm. *Computing in Science & Engineering*, 2(1):65–69, 2000.

[6] Dongshu Wang, Dapei Tan, and Lei Liu. Particle swarm optimization algorithm: an overview. *Soft computing*, 22:387–408, 2018.

[7] Olusola Abayomi-Alli, Adebayo Abayomi-Alli, Sanjay Misra, Robertas Damasevicius, and Rytis Maskeliunas. Automatic examination timetable scheduling using particle swarm optimization and local search algorithm. *Data, Engineering and Applications: Volume 1*, pages 119–130, 2019.

[8] Qinghai Bai. Analysis of particle swarm optimization algorithm. *Computer and information science*, 3(1):180, 2010.

[9] Yan Jiang, Tiesong Hu, ChongChao Huang, and Xianing Wu. An improved particle swarm optimization algorithm. *Applied Mathematics and Computation*, 193(1):231–239, 2007.

[10] Zi Chao Yan and Yang Shen Luo. A particle swarm optimization algorithm based on simulated annealing. *Advanced materials research*, 989:2301–2305, 2014.

[11] Vijay Kanade. What are genetic algorithms? working, applications, and examples. 2023. URL https://www.spiceworks.com/tech/artificial-intelligence/articles/what-are-genetic-algorithms/.

[12] Kartika Gunadiand Denny Alexander Wibowo Gregorius Satia Budhi. Genetic algorithm for scheduling courses. 2015. URL https://core.ac.uk/download/pdf/32452919.pdf.

[13] Andrea Bettinelli, Valentina Cacchiani, Roberto Roberti, and Paolo Toth. An overview of curriculum-based course timetabling. *Top*, 23:313–349, 2015.

[14] Ahmed G Gad. Particle swarm optimization algorithm and its applications: a systematic review. *Archives of computational methods in engineering*, 29(5):2531–2561, 2022.