EVALUATE THE PERFORMANCE OF UNIVERSITY COURSE TIMETABLING PROBLEM WITH DIFFERENT COMBINATIONS OF GENETIC ALGORITHM

BY

FOO YAO HENG

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2025

COPYRIGHT STATEMENT

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Ts Dr Ku Chin Soon, who has given me a precious opportunity to involve in a timetable scheduling project. Besides, he has provided me a lot of guidance to complete this project. When I was facing problems, his advice always assists me in overcoming the challenges. Again, a million thanks to my supervisor.

Finally, I must say thank you to my family and friends for their love, support, and continuous encouragement throughout the project.

ABSTRACT

University course timetabling problem (UCTP) is a scheduling problem that requires courses to be assigned to the limited time slots, classrooms, and lecturers, while adhering to a set of predefined constraints. Due to the effectiveness of genetic algorithm (GA) in optimisation problems, it has been widely discussed in numerous research to address UCTP. Nonetheless, the performance of GA in terms of operation techniques has not been studied enough, as the researchers have often focused on using a single GA combination or hybrid approaches to solve UCTP case studies. Therefore, this project aims to analyse the performance of different combinations of GA operation techniques and identify the best GA model. A flexible GA framework is developed, which allows alternative techniques to be integrated and executed easily. 64 combinations, involving 4 selection, 4 crossover, 1 mutation, and 4 replacement techniques, are evaluated on a partial mock dataset. In addition, this project proposes a new soft constraint, which requires consecutive classes for a student to be held in the same building. This constraint targets to reduce students' travel distance, thus producing a more student-friendly timetable. Experimental results shows that GA44 model which comprises of binary tournament selection, uniform crossover, swap mutation, and weak chromosome replacement is the best GA combination. In conclusion, the proposed constraint demonstrates clear benefits to student experience on campus and offers a fresh idea for future research with alternative approaches.

Area of Study: Scheduling Problem

Keywords: Optimisation, Combinatorial Optimisation Problem, University Course Timetabling Problem, Course Scheduling, Metaheuristics, Genetic Algorithm

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

bw	pitch bandwidth

LIST OF ABBREVIATIONS

UCTP	University Course Timetabling Problem
GA	Genetic Algorithm
NP-HARD	Non-Deterministic Polynomial Hard
OR	Operational Research
EA	Evolutionary Algorithm
MDVRP	Multi Depot Vehicle Routing Problem
TSP	Travelling Salesman Problem
IDE	Integrated Development Environment
SA	Simulated Annealing
TS	Tabu Search
UCTP	University Course Timetabling Problem
GP	Goal Programming
ATS	Adaptive Tabu Search
PSO	Partial Swarm Optimisation
ACO	Ant Colony Optimisation
CS	Cuckoo Search
HS	Harmony Search
PE-CTT	Post-Enrolment Course Timetabling
TSPP	Tabu Search with Sampling and Perturbation
RW	Roulette Wheel
CP	Constraint Programming
GD	Great Deluge
ANOVA	Analysis of Variance
UCSP	University Course Scheduling Problem
PSOSS	Partial Swarm Optimisation with Selective Search
PSM	Producer-Scrounger Method
ACOSP	Ant Colony Optimisation with Selective Probability
SI	Swarm Intelligence
LF	Lévy Flight
BNCS	Best-Nests Cuckoo Search

HM	Harmony Memory
HMS	Harmony Memory Size
MC	Memory Consideration
PA	Pitch Adjustment
RC	Random Consideration
HCMR	Harmony Memory Consideration Rate
PAR	Pitch Adjustment Rate
MI	Maximum Number of Improvisations
CBCTT	Curriculum-Based Course Timetabling
NP-COMPLETE	Non-Deterministic Polynomial Complete
GVRP	Green Vehicle Routing Problem
RCPSPTT	Resource-Constrained Project Scheduling Problem with Transfer Times
SQL	Structured Query Language
JDBC	Java Database Connectivity
API	Application Programming Interface
RDBMS	Relational Database Management System
ACID	Atomicity, Consistency, Isolation, Durability
ERD	Entity Relationship Diagram
CSV	Comma-Separated Value
SSN	Simple Search Neighbourhood
SWN	Swap Search Neighbourhood
LNS	Large Neighbourhood Search

Chapter 1
Introduction

1.1 Project Inspiration

A timetable is a tabulation that shows multiple events and their schedules [38]. Timetable scheduling problem is an optimisation problem where events are allocated to the limited resources such as space and time, while adhering to a set of predefined constraints [9,12]. It is a well-known problem across various fields, including education, hospitalisation, and transportation [12]. This is because a timetable plays an important role in smoothening the operations of multiple parties and facilitating the cooperations between them. Therefore, timetable scheduling problem has been and is still an important subject in a wide range of research areas [7].

The university course timetabling problem (UCTP) is one of the scheduling problems in which it requires courses to fit well into the limited time slots, classrooms, and lecturers with no conflicts [2,12]. This problem is significant because it resurfaces each semester as universities plan their course offerings [1]. Traditionally, university course timetables are manually created by university's administration staff. This manual process is not only time-consuming but also prone to errors, especially when accommodating large numbers of students and faculty with varying preferences and requirements [38].

In the context of UCTP, the constraints are typically modelled around courses, classes, lecturers, students, and classrooms. These constraints are not fixed, but instead they are highly specific and vary from institution to institution [2,38]. Furthermore, the different roles of timetable practitioners, such as students and lecturers, may lead to shifting priorities over time, further increasing the number of constraints. Together, these constraints define the feasibility of a timetable, where a high-quality timetable is one that satisfies all constraints.

Generally, there are two types of constraints in UCTP, which are hard and soft constraints. Hard constraints are requirements that cannot be violated in order to produce a feasible timetable [2,38]. For example, hard constraints include ensuring that no student is assigned to more than one class at a time, that the classroom capacity is not exceeded, and that lecturers are not double-booked [38]. On the other hand, soft constraints are preferences that are not strictly necessary for timetable feasibility [2,38]. It aims to improve the quality of the timetable by considering factors such as minimising the number of consecutive hours for lecturers and students and ensuring a balanced distribution of courses throughout the week [2,12].

With these complexities in UCTP, this problem is considered a non-deterministic polynomial hard (NP-hard) problem [1,8,9,38]. This implies that there is no conventional algorithm that is able to find an optimal solution in the polynomial time as the problem size such as number of students and constraints grows exponentially [3,8,38]. In addition, there is no particular solution that satisfies every UCTP due to the unique requirements of each university, let alone solving it manually [38]. Therefore, various optimisation algorithms are applied to tackle this problem, including metaheuristics, hyper-heuristics, multi-objective, operational research (OR), and hybrid approaches [1].

Among these feasible approaches, this project focuses on metaheuristics, particularly the genetic algorithm (GA). GA is first introduced by John Holland in 1975. It is an evolutionary algorithm (EA) [2] inspired by the principle of "survival of the fittest" proposed by Charles Darwin, which emphasises on natural selection and genetics [22]. GA is widely used to solve complex optimisation problems such as UCTP [2,8,12,38], multi depot vehicle routing problem (MDVRP) [23], and travelling salesman problem (TSP) [25]. This is because of its ability to explore large search spaces and find nearly optimal solutions in a reasonable amount of time. The algorithm begins with an initial population of candidate solutions, which is also known as chromosomes. These chromosomes then iteratively evolve over generations using genetic operators that mimics the biological processes such as selection, crossover, mutation, and replacement, and finally come up with a good-enough solution [22].

This project aims to study the absence of a significant constraint related to students' comfort, which is the distance between classrooms for consecutive classes. Besides, this project seeks to investigate how effectively GA can handle both hard and soft constraints in the UCTP, particularly with the newly introduced constraints. There is a lack of research on GA applications in UCTP, especially involving different operator combinations. Therefore, this project focuses on providing perspectives on this unexplored area by applying various GA combinations.

1.2 Problem Statements

The primary goal of university course timetabling problems (UCTPs) is to produce schedules that satisfy institutional requirements while supporting students' learning experience. However, many schedules generated by existing timetabling systems place consecutive classes for the same student groups in different buildings. This forces them to spend valuable minutes walking between blocks and often arrive unprepared or miss the opening of the next lesson. It is obvious that such avoidable travel erodes attention and reduces effective studying time.

Genetic algorithm (GA) is frequently adopted to solve UCTPs because it is a type of metaheuristics, which can explore large search spaces more quickly than exact mathematical techniques [4]. Nonetheless, most GA implementations in UCTPs embed only the common constraints and rely on a single combination of operators. Timetable quality can shift significantly when alternative selection, crossover, mutation, and replacement techniques are combined, either amplify or dampen the performance [37]. Therefore, comprehensive testing across operator combinations is essential, yet many authors choose to settle on one default configuration. This might be due to limited research time and incomplete familiarity with GA design.

Consequently, existing work neither enforces the building-continuity constraint nor identifies which GA combination performs best when that constraint is present. This project closes both gaps by incorporating the same-building requirement into the problem model and by systematically benchmarking diverse combinations of genetic operators to discover a configuration that produces high-quality, student-friendly timetables.

1.3 Project Objectives

First, this project aims to formalise a comprehensive set of hard and soft constraints for the university course timetabling problem (UCTP). The new soft constraint requires a student's consecutive classes to be held in the same building. A quantitative penalty is applied whenever a timetable forces an inter-building move in between consecutive classes, allowing the effect of the constraint to be measured across experiments.

Second, it targets to design and develop a flexible genetic algorithm (GA) framework whose operators can be exchanged without modifying the core code. This project evaluates 64 unique operator combinations formed from four selection, four crossover, one mutation, and four replacement techniques, including two newly proposed replacement methods. There is also an immediate repair operator after crossover and mutation operations to ensure that the timetable represented by the chromosome satisfies the imposed constraints.

Third, it seeks to assess the performance of every GA combination on a partial mock dataset. Each combination is executed in several independent trials, and the number of generations, execution time, and initial penalty cost are recorded. Statistical analysis of the repeated runs highlights the strengths and weaknesses of the individual techniques and pinpoints the best overall combination.

1.4 Project Scope

This project delivers a flexible genetic algorithm (GA) framework built in Java associated with the MySQL database engine. The framework aims to solve university course timetabling problem (UCTP) only. It respects 15 hard constraints and 4 soft constraints, including the proposed same-building requirement for consecutive classes. Other than that, it focuses on four selection, four crossover, one mutation, and four replacement techniques, which are combined to form 64 GA combinations. A repair operator is also adopted to further enforce the applied constraints.

Besides, a partial mock dataset is constructed, which mirrors the real timetables of Computer Science (CS) programme at Universiti Tunku Abdul Rahman (UTAR). It reproduces room types and locations, weekly slot structure, offered courses and assigned lecturers, while student and class records are generated programmatically to complete the input. This dataset exercises every constraint and supports verification.

The experiments run each combination with a population size of 100, chromosome length of 250, crossover probability of 0.7, and mutation probability of 0.4, on the partial mock dataset. The crossover and mutation probabilities are determined using grid search across parameter combinations of 0.5, 0.6, 0.7, 0.8, 0.9 crossover rate and 0.01, 0.2, 0.3, 0.4, 0.5 mutation rate. A run only stops when it achieves a penalty cost of zero, which is the optimum. The number of generations, execution time, and initial penalty cost are recorded for performance analysis A new metric, fitness improvement per generation, is also proposed to coordinate with the termination criterion. It is defined as the initial penalty cost divided by the number of generations. This offers a fairer comparison by mitigating bias arising from differing initial penalty cost. Besides, each run generates feasible timetables, including course, lecturer, student, and room timetables. Every combination is executed 10 times, and the averages of the metrics are computed for performance comparison.

All coding work and experiments are conducted on a laptop equipped with an Intel i5-10200H processor, 16 GB of memory, and an RTX 3060 laptop GPU. Visual Studio Code (VS Code) serves as the integrated development environment (IDE).

To clarify, the study focuses exclusively on GA and does not investigate alternative optimisation algorithms. It also avoids both a fully real-world dataset and a fully synthetic dataset, opting instead for the balanced mock dataset described above.

1.5 Project Impact and Contribution

The project introduces the same-building soft constraint, producing timetables that minimise student travel between consecutive classes and therefore reduce lost learning time.

Besides, it delivers a modular genetic algorithm (GA) framework in which the genetic operators are fully interchangeable. By benchmarking 64 operator combinations on a partial mock dataset, the study supplies the first systematic comparison of these techniques in the context of university course timetabling. Moreover, the crossover and mutation probabilities applied in the project are the best parameter combination found by performing grid search. The resulting insights help researchers select more effective operator combinations and parameter settings as well as design more rigorous experiments in the university course timetabling problem (UCTP) domain.

Other than that, two replacement techniques are proposed in this project, which introduces different perspectives on the importance of replacement techniques.

This project also proposes a new performance metric that introduces a fairer comparison on combination performance, which is the fitness improvement per generation, computed by dividing the initial penalty cost with the number of generations. This metric aims to mitigate the bias in performance comparison due to the differences in initial penalty cost. It also suits the stopping condition adopted in this project, where GA models are let to run until a perfect timetable is generated, which has no penalty cost. Therefore, the final fitness value cannot be used as the performance metric, as they are all the same. On the other hand, the number of generations is not fair enough to measure the performance as the initial penalty cost is random for each experiment and higher penalty cost often requires higher number of generations.

Furthermore, this project provides a foundation for future work in UTPs. Researchers can make further studies by extending the constraint set, integrating alternative optimisation approaches, or measuring new performance indicators.

1.6 Chapter Summary

The details of this research are shown in the following chapters. Chapter 2 reviews prior work on university timetabling, beginning with single-solution metaheuristics such as simulated annealing and tabu search, then moving to population-based techniques that include genetic algorithms, particle swarm optimisation, and other nature-inspired methods. This chapter also highlights common hard and soft constraints and critiques gaps in earlier research. Chapter 3 explains the methodology adopted in this project, detailing data collection, the complete constraint model, particularly to the newly introduced soft constraint that requires a student's consecutive classes to remain in the same building, system requirements, and verification plans. Chapter 4 gives particular attention to input and output design, data storage design, and genetic algorithm design, including the chromosome encoding, fitness evaluation, repair mechanisms, operator combinations and verification plans that ensure rigorous testing. Chapter 5 presents system testing, such as experiments on constraints, GA models, and comparison among operation techniques, as well as experiment on resource utilisation. Chapter 6 focuses on discussion of objective evaluation, system novelties, system limitations, future enhancement and improvement. Lastly, Chapter 7 consolidates the conclusion.

Chapter 2
Literature Review

2.1 University Timetabling Techniques

There are various optimisation algorithms applied to tackle the university course timetabling problems (UCTPs), including metaheuristics, hyper-heuristics, multi-objective approaches, operational research (OR) techniques and hybrid methods [1]. This project focuses on genetic algorithm (GA), which is a well-known metaheuristic. Therefore, studying this class of methods provides insights that are directly aligned with the goals of this project. Figure 2.1.1 illustrates the hierarchy of metaheuristics for UCTPs that are studied in this project.

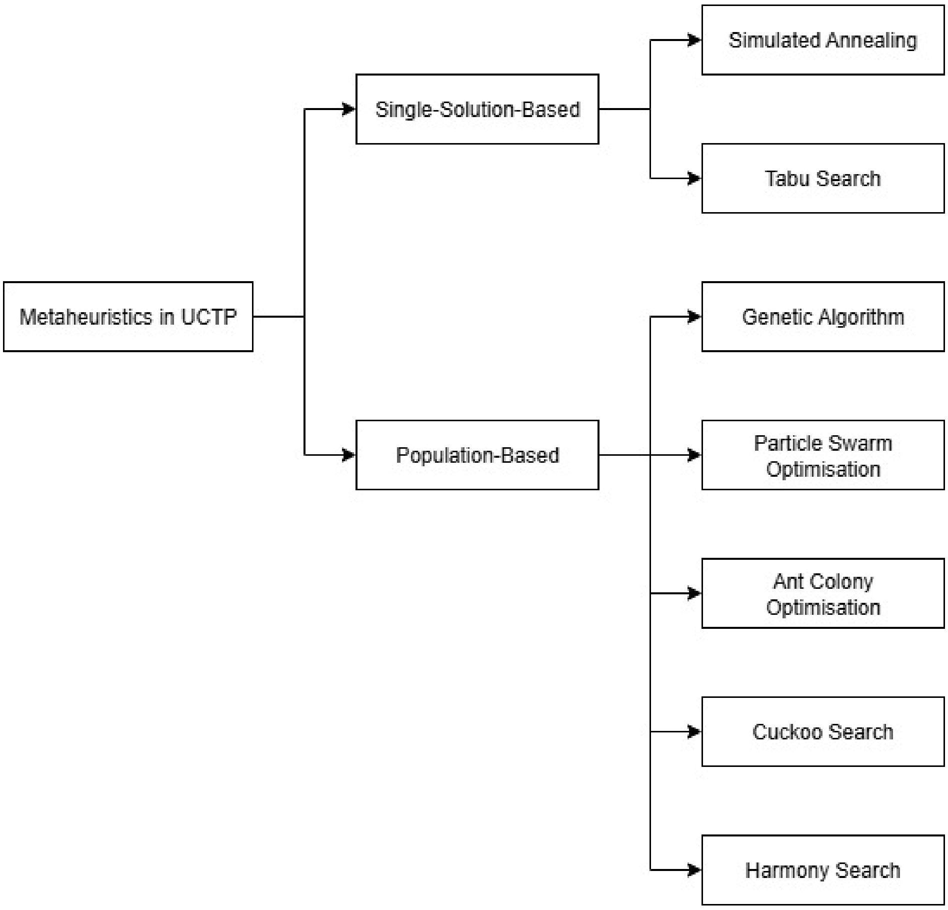


Figure 2.1.1 Metaheuristics in UCTPs

Metaheuristics are iterative processes that guide underlying heuristics so that they can explore and exploit the search space efficiently and locate near-optimal solutions at a reasonable computational cost. By operating at a higher level than heuristics (but lower than hyper-heuristics), they make few assumptions about the problem, cope well with incomplete information and limited resources, and can handle a wide variety of optimisation tasks by searching large solution spaces effectively [1].

2.1.1 Single-Solution-Based Metaheuristics

Single-solution-based metaheuristics focus on iteratively refining one candidate solution. These methods, often labelled as local search algorithms, start from a single solution that is chosen based on specific criteria. The solution then explores its neighbourhood to uncover improvements through repeated manipulation and relocation until a stopping condition is satisfied [2,12]. This family includes the techniques such as simulated annealing (SA) and tabu search (TS).

2.1.1.1 Simulated Annealing

Simulated annealing (SA) is a stochastic local search algorithm inspired by the annealing process, wherein a heated solid is slowly cooled to achieve a more stable state [1,3]. The algorithm begins with a randomly generated solution and iteratively replaces it with a neighbouring solution based on an acceptance probability criterion until a specified termination condition is met [1,2,3]. A temperature parameter guides the exploration of the search space. A high temperature in the early stages promotes exploration and helps the algorithm escape local optima. As the temperature is gradually decreased, the tendency for exploration reduces, encouraging convergence towards a global optimum [2,3]. SA is widely recognised for its ease of implementation and effective local search capability, although it is often limited by a slow convergence rate [1,3,4].

The research [4] applies the SA to address a faculty-level university course timetabling problem (UCTP). This research aims to ensure that students enrolled in double major and minor programmes can attend all necessary classes without time conflicts in an environment where classrooms are shared across faculties. This highlights its novelty, as it is the first to simultaneously take these constraints into consideration. The authors first formulate the problem as a goal programming (GP) model and then design an SA algorithm to overcome the GP model's scalability issues. Both proposed methods are tested with a sample dataset (2 departments with 35 courses and 4 student groups), two randomly generated datasets (3 departments with 57 courses and 4 departments with 77 courses), and a real-life engineering faculty dataset from a private university (5 departments with 107 courses, 53 lecturers, and 31 classrooms). The results show that GP reaches the optimum only for the sample dataset; it fails to find a feasible solution within the specified time limit of 3600 seconds for the other datasets. On the other hand, SA outperforms GP on the sample dataset, obtaining a feasible solution with 16 times less computational time. It also successfully generates solutions for the randomly generated datasets and improves the real-world dataset by 83 % in under 3 minutes.

2.1.1.2 Tabu Search

Tabu search (TS) is a memory-based metaheuristic built on local search [1,6]. The algorithm starts from an initial solution and iteratively explores the neighbourhood of the current solution until a termination criterion is satisfied [2]. It employs a short-term storage called a tabu list, which stores recently executed movements to prevent the search from cycling back to previously visited solutions. Nonetheless, this restriction can be overridden by an aspiration criterion when a movement leads to a solution better than any found so far, maintaining search flexibility. Furthermore, this storage is associated with a parameter known as tabu tenure, which defines how long a movement remains in the list [2,6]. By allowing non-improving movements while blocking revisits, this short-term memory helps the search avoid trapping in the local optima [1,2,6]. In addition, TS adopts intermediate and long-term memories that drive intensification and diversification, which offers a balance between exploitation and exploration as the search progresses [1,6].

Awad et al. [5] introduce an adaptive tabu search (ATS) framework to resolve the university course timetabling problem (UCTP) in large-scale scenarios. The approach is divided into two stages. First, the authors construct a feasible solution by employing a least saturation degree algorithm, supported by two neighbourhood structures. This construction stage focuses on satisfying all hard constraints without considering the soft constraints. Next, the approach enters the improvement stage. Two additional neighbourhood operators are applied within the ATS algorithm to minimise soft-constraint violations in the early solution. Specifically, the adaptiveness of the proposed TS is shown on the tabu list, where its length can be dynamically reduced. If the total penalty cost fails to decrease after 1000 iterations, the list length is shortened by 2, subject to a lower bound of 2; otherwise, it remains unchanged.

This mechanism allows the algorithm to counteract search stagnation. The method is evaluated on 11 benchmark datasets from Socha et al., which are categorised into 5 small, 5 medium, and 1 large instance. Its performance is assessed by comparing total penalty scores with those of 14 published approaches, where 7 of them are TS-based and 7 of them are not. The proposed ATS approach ranks second on the medium and the large datasets, surpassed only by a simulated annealing (SA) approach. This demonstrates its adaptability and effectiveness. Nevertheless, this research comes with a limitation, where the numerical results for the small datasets are not provided. The authors merely report that there is a minor performance slippage in those cases. Overall, the research shows that adaptive control of the tabu list can enhance TS performance on large-scale UCTP datasets.

2.1.2 Population-Based Metaheuristics

Population-based metaheuristics work on a collection of solutions that co-evolve through repeated cycles of selection, variation and replacement. In each iteration, the methods select high-quality individuals from the current population, apply problem-specific operators to produce improved variants, and substitute weaker members with these offspring until a termination criterion is met, typically an acceptable result is reached [2,12]. This family includes the techniques such as genetic algorithm (GA), partial swarm optimisation (PSO), ant colony optimisation (ACO), cuckoo search (CS), and harmony search (HS).

2.1.2.1 Genetic Algorithm

Genetic algorithm (GA) is an evolutionary algorithm inspired by Darwin’s theory of natural selection, in which fitter individuals have a higher chance of survival [1,2,8]. Typically, a GA proceeds through 5 stages, including initialisation, selection, crossover, mutation, and replacement. First, it generates an initial population of candidate solutions and evaluates their fitness. Based on those fitness values, the algorithm chooses parent solutions. A crossover operator then combines selected parents to produce offspring, which are subsequently mutated to maintain diversity. During replacement, the offspring substitute an equal number of individuals in the current population, thus forming the next generation. These steps, excluding the initialisation, repeat until a termination criterion is satisfied [2]. GA is highly customisable through parameters such as population size and mutation probability. For instance, a moderate mutation probability introduces randomness that prevents the search from getting trapped in local optima, while an excessively high value can cause the algorithm to degenerate into random search [7]. GA is widely favoured for tackling large-scale, non-linear optimisation problems due to its genetic operators offering a balance in exploration and exploitation [1,8].

The study [7] aims to automate the post-enrolment course timetabling (PE-CTT) process at University of Malaysia Sabah Labuan International Campus (UMSLIC). The authors propose a hybrid metaheuristic approach in which a tabu search with sampling and perturbation (TSPP) first builds a pool of feasible timetables, after which a GA repeatedly improves the solution quality. Particularly, the study focuses on GA. This is demonstrated by the conducted experiments, where the best parameter values for the GA under limited computational time are identified. The performance of two GA selection techniques, which are steady-state selection and roulette wheel (RW) selection, is also tested in a common environment with swap-transfer mutation and weak parent replacement. These experiments are run on a real-world dataset, which is the session 2018/2019 semester 1 student registration dataset that consists of 1993 students, 144 courses, 35 time slots, and 24 rooms. Performance is evaluated based on hard- and soft-constraint violations, comparing the best timetables generated by the proposed approach with those scheduled by the university's administrative staff. The former cuts the soft-constraint violations by 54 % relative to the latter. Moreover, the automated timetables have no hard-constraint violations, while the crafted timetables have 37 clashes that violate 2 hard constraints.

Another study [8] also focuses on tackling the university course timetabling problem (UCTP) for USMLIC using GA. The GA model employed in this study is configured as follows. First, a population of feasible solutions that satisfy all hard constraints is generated using constraint programming (CP). Then, the algorithm iteratively applies quinary tournament selection, one-point crossover, and random mutation to the population until 100,000 generations are reached or a 300-second cut-off time is met. Two experiments are conducted, in which the authors first assess the GA's capability to generate feasible timetables. Next, they compare the GA's performance with two other metaheuristics, which are great deluge (GD) and simulated annealing (SA). Both experiments use two real-world datasets from UMSLIC, which are the session 14/15 semester 2 instance (2248 students, 112 courses, 35 time slots, and 18 rooms) and the session 2015/16 semester 1 instance (2248 students, 112 courses, 35 time slots, and 18 rooms). Solution quality is evaluated according to the sum of penalties from hard-

and soft-constraint violations. Over 50 runs per dataset, GA lowers the average penalty cost of the initial solution by approximately 36 % for both datasets. On the other hand, GD improves the average quality of the initial solution by around 25 %, while SA achieves only an improvement of roughly 23 %. A one-way analysis of variance (ANOVA) confirms that these performance differences are statistically significant. The results indicate that GA outperforms both GD and SA, highlighting its effectiveness in the timetable scheduling domain.

2.1.2.2 Particle Swarm Optimisation

Particle swarm optimisation (PSO) is a stochastic optimisation technique that draws inspiration from the social behaviour of swarms, such as bird flocks and fish schools, particularly their coordinated movements [1,8,10]. The algorithm begins by initialising a swarm of particles, each representing a potential solution to the problem at hand [8]. Each particle is randomly placed in the search space and assigned an initial position and velocity [1,8]. Next, the algorithm enters an iterative cycle. During each iteration, a particle first updates its velocity based on three factors, which are its current velocity, the best position it has personally discovered, and the best position found by the entire swarm. Using this revised velocity, the particle then moves to a new position [1,8]. The personal best and global best values are refreshed whenever a particle or the swarm finds an improved solution, respectively. This cycle continues until a termination condition is met, at which point the global best position is returned as the final solution [8]. PSO is popular among researchers because it is simple to implement and requires only a few parameters to configure. However, in multi-dimensional or complex search spaces, the particles can easily stagnate, thus converging at a low speed and eventually affecting the solution quality. This stagnation occurs due to the swarm's liability to get trapped in local optima and the instabilities in particle velocities [10].

Hossain et al. [9] address the university course scheduling problem (UCSP) with a modified particle-swarm framework called PSO with selective search (PSOSS). They transform standard PSO operations by computing particle velocity with swap operators and swap sequences. On top of this, they propose two novel mechanisms. First, a forceful swap operator, combined with a repair technique, guarantees that every particle makes a feasible move. Second, a selective search operator retains the best intermediate timetable after each update. These innovations significantly enhance the algorithm's adaptability to the hard and soft constraints in UCSP. Experiments use a real-world dataset from the Computer Science and

Engineering (CSE) department of Khulna University of Engineering and Technology (KUET), comprising 38 courses, 27 lecturers, 5 student batches, and 13 rooms. In the experiments, PSOSS is compared against multiple metaheuristics, which are genetic algorithm (GA), traditional PSO, harmony search (HS), and producer-scrounger method (PSM). Performance evaluation proceeds in two stages, in which population size and iteration studies identify suitable parameters, after which solution quality is measured by a composite fitness score, calculated as the sum of lecturer-preference values minus consecutive-class penalties, and average lecturer-satisfaction percentage. On this benchmark, PSOSS achieves a best fitness of 471 and an 83 % average satisfaction, outperforming other approaches. Particularly, it outscores the nearest rival GA by approximately 10 % on both metrics. Furthermore, it converges to its optimum in 155 iterations, slower than HS that takes only 60 iterations, but significantly faster than GA, PSO, and PSM, which each require over 400 iterations. Together, these results demonstrate that PSOSS is able to deliver higher quality timetables with better efficiency than the competing metaheuristics.

2.1.2.3 Ant Colony Optimisation

Ant colony optimisation (ACO) is a swarm intelligence (SI) method inspired by how the ants communicate and interact with each other during foraging [1,2,11,12]. When searching for foods, the ants seek the shortest route between a food source and their nest to transport food efficiently [2,12]. They start by moving randomly and leave a pheromone trail along their paths [1,2,12]. When other ants encounter a trail, they are drawn to it and follow the same route. If they find the food, they return to the nest and lay an additional pheromone trail next to the original one. This makes the trail more attractive [2]. Since pheromones are volatile, the stronger trails, which correspond to the shorter routes, are continually amplified by many ants [11]. In contrast, the weaker trails gradually evaporate and disappear [2,11]. Eventually, only the best route remains [11]. When ACO is applied to optimisation problems, the artificial ants first construct individual solutions [11,12]. They then pass the information about the quality of these solutions to each other, guiding subsequent searches [11,12]. Through iterative reinforcement, the solution set converges towards the global optimum [11,13]. ACO excels at combinatorial optimisation thanks to its decentralised search and frequent feedback, but it can converge slowly or become trapped in local optima on complex, large-scale problems [13].

The research [11] seeks to tackle the university class scheduling problem (UCSP) for the Computer Science and Engineering (CSE) department of Khulna University of Engineering and Technology (KUET). The author focuses on two metaheuristics, which are a standard ACO and a proposed ACO with selective probability (ACOSP). Experiments are conducted in two different settings, which are a simple environment that uses a small dataset (10 courses, 8 lecturers, and 2 student batches) and a highly constrained environment that uses a larger dataset (37 courses, 35 lecturers, and 4 student batches). Both ACO-based approaches are tested in the two environments, whereas a genetic algorithm (GA) is evaluated only in the simple environment. Solution quality is measured with a fitness function that sums lecturer-preference scores, averaged over 10 trials, while the population size (or ant count) and the number of iterations are varied. The results show that both ACO variants outperform the GA in the simple setting and that ACOSP consistently achieves the highest fitness in both datasets. According to the author, this superiority arises because the ACO variants compute probabilities for every unassigned time slot during course assignment, whereas the GA does not. Moreover, ACOSP restricts each choice to a shortlist of promising time slots and considers only their probabilities, hence reducing search effort and enabling faster convergence.

2.1.2.4 Cuckoo Search

Cuckoo search (CS) is another swarm intelligence (SI) optimisation algorithm that employs a Lévy flight (LF) search mechanism to locate high quality solutions within large and complex search spaces [14]. It draws inspiration from the aggressive brood parasitic behaviour of certain cuckoo species, which discreetly lay their eggs in the nests of other birds [1,14]. Mimicking this strategy, CS treats each candidate solution as an egg placed in a host nest, while a fraction of nests is periodically abandoned or replaced to simulate the host bird's discovery and rejection of foreign eggs [15]. New solutions are then generated by LFs, a type of random walk featuring heavy-tailed step-length distributions. This mechanism enables the algorithm to balance extensive exploration with intensive exploitation of promising regions [14]. CS is widely applied to optimisation problems because it relies on only a few control parameters, offers a straightforward iteration that is easy to code, and yet maintains a strong random search capability [14].

Jebur and Abdullah [15] tackle the university course timetabling problem (UCTP) by proposing a best-nests CS (BNCS) variant designed to accelerate convergence and improve timetable quality relative to the traditional CS. Their workflow first ranks a population of candidate timetables and splits them into “best” and “normal” sub-groups. It then generates a new solution solely from the elite subset and compares it against a randomly chosen member of the inferior subset, where the better solution replaces the worse one. Both sub-groups are subsequently aggregated, after which a ratio of the worst solutions from the combined population is discarded. These steps repeat until a stopping criterion is reached. Key parameters, such as population size, LF λ value, and best-nest ratio, are tuned experimentally. Evaluation uses four datasets from KTH Royal Institute of Technology, which are labelled as small, medium, large, and extra-large size, varying from 70 to 293 class events and 160 to 540 time slots. Performance is measured as average fitness based on hard-constraint violations and compared with the traditional CS. The number of iterations scales with dataset size, ranging from 9500 to 57000. Across all datasets, BNCS converges faster and reaches higher fitness than CS, using a configuration that includes a population of 40, a λ of 1, and a best-nest ratio of 0.25. The newly introduced selection scheme clearly shows its capability in enhancing traditional CS, which helps accelerate exploitation without sacrificing exploration.

2.1.2.5 Harmony Search

Harmony search (HS) is an optimisation algorithm that frames each candidate solution as a musical harmony and seeks the best composition through iterative improvisation [9,16]. Inspired by how performers refine their instruments’ pitches to achieve an aesthetically pleasing sound, HS stores a population of solutions in a harmony memory (HM), whose size is termed the harmony memory size (HMS) [9,17]. It generates new harmonies through three operators, namely memory consideration (MC), pitch adjustment (PA) and random consideration (RC). These operators are regulated respectively by the harmony memory consideration rate (HMCR), pitch adjustment rate (PAR), and pitch bandwidth (bw). After setting these parameters and randomly filling the HM, the algorithm repeatedly improvises a new harmony, compares it with the current worst member of the HM, and replaces that worst harmony when an improvement is found. HM is thus continually updated until the maximum number of improvisations (MI) is reached, at which point the best harmony is returned [9,16]. Researchers have widely applied HS to real-world optimisation tasks due to its simple concept, limited tuneable parameters, and easy implementation, alongside its ability to balance exploration and exploitation. Nevertheless, HS can still exhibit low optimisation accuracy and suffer from premature convergence [17].

The study [16] aims to improve timetable quality for the College of Arts and Sciences (CAS) at Universiti Utara Malaysia (UUM) by eliminating hard-constraint clashes and cutting soft-constraint penalties. Therefore, the authors propose a hybrid HS algorithm combined with the great deluge (GD) heuristic. The algorithm begins by constructing 10 feasible timetables to populate the HM and then iteratively improvises new solutions using various operators. MC and PA operators refine the lectures stored in the HM, while moves generated by the RC operator are filtered through a GD acceptance test whose water level, representing solution quality, is reset to the current best solution at the start of every improvisation cycle. The search terminates and returns the best solution after the MI is reached. Experiments use the UUM CAS session 13/14 semester 1 undergraduate dataset, which consists of 247 courses, 850 lectures, 32 rooms, 350 lectures, and 20000 students. They tune the HMCR parameter of the proposed algorithm while comparing its output with the official timetable produced by commercial timetabling software. Timetable quality is evaluated using a curriculum-based course timetabling (CBCTT) validator algorithm that computes hard- and soft-constraint costs. The hybrid HS-GD approach achieves its best result when HMCR is set to 0.8, yielding a total cost of 708 with zero hard-constraint violations. It outperforms the university's software, which produces a timetable with a total penalty score of 1230 spanning both hard and soft constraints.

2.2 Genetic Algorithm

Genetic algorithm (GA), introduced by John Holland in 1975, is grounded in the concepts of genetics and natural evolution [22]. This metaheuristic searches for good solutions through the iterative application of selection, crossover and mutation operators. As these operators rely on random choices, the approach is fundamentally stochastic [8]. This indicates that the algorithm may produce varying best solutions across multiple runs for the same problem due to the different search behaviours of the operators in each execution [19].

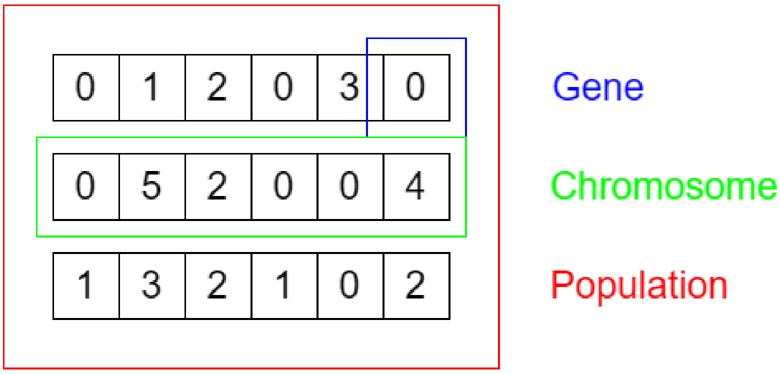


Figure 2.2.1 Genes, Chromosomes, and Population in GA

In the context of GA, a gene represents a single decision variable or attribute, a chromosome is an ordered collection of genes that encodes one candidate solution, and a population is the set of chromosomes that are evaluated together in the same generation [22]. Figure 2.2 illustrates these basic building blocks and their hierarchy within the algorithm.

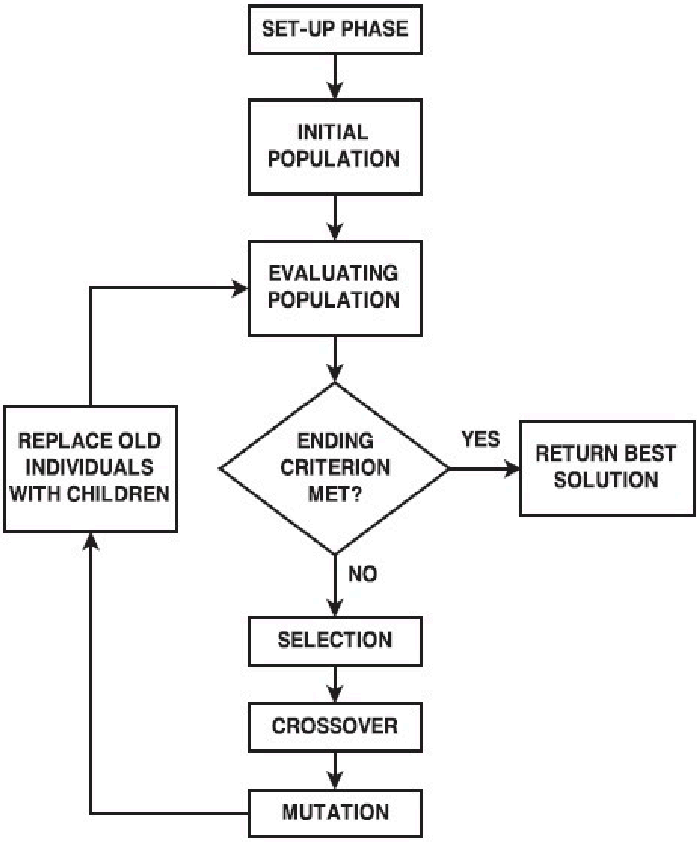


Figure 2.2.2 Flowchart of GA [23]

Figure 2.3 presents the flowchart of GA that guides this part of discussion. The algorithm starts with a setup phase in which parameters such as population size, number of generations, mutation rate and crossover probability are defined. An initial population of chromosomes is created at random, and each chromosome is evaluated with a problem-specific fitness function so that better solutions receive higher scores. If a termination criterion such as a maximum generation count, a time limit or satisfaction of a target fitness is met, the search stops, and the best chromosome is returned. Otherwise, two chromosomes are selected according to their fitness, typically favouring the stronger yet still giving weaker individuals a chance in order to preserve diversity. Those parents may undergo crossover which combines parts of both chromosomes, and the resulting offspring may then experience mutation which randomly alters one or more genes. Whether crossover or mutation occurs is governed by predefined probabilities, which allow offspring to be copied unchanged from one parent when crossover is skipped and to remain unmodified when the mutation probability test fails**.** After reproduction, the replacement strategy inserts the offspring into the population, the new generation is evaluated and the loop repeats until the stopping condition is satisfied.

GA is widely favoured for tackling large scale and combinatorial optimisation tasks, including the multi depot vehicle routing problem (MDVRP) [23], the travelling salesman problem (TSP) [25], and the university course timetabling problem (UCTP) [7,8]. Its popularity in these domains stems from its capacity to explore vast search spaces efficiently, while retaining the flexibility to incorporate domain specific constraints and objectives [22].

An important strength of GA lies in the number of tuneable parameters and operator variants that can be combined to suit different problems. For instance, several techniques are available for each stage of the algorithm process and their selection directly affects performance [37]. Besides, the population size in particular has a measurable impact on exploration because larger populations tend to provide greater coverage of the search space and thus a higher probability of reaching near optimal solutions. In addition, mutation helps the algorithm avoid premature convergence to local optima, though an excessive rate can cause the search to resemble random sampling [7].

Despite these advantages, GA also faces limitations. Crafting an appropriate representation and designing a meaningful fitness function can be challenging, and the practitioner must decide various parameter settings such as population size and operator probabilities. Moreover, even with careful tuning, the algorithm offers no guarantee of finding the global optimum, especially on problems with rugged or deceptive fitness landscapes [22].

2.2.1 Selection

The selection operator in genetic algorithm (GA) describes the process of selecting two chromosomes from the current population to act as parents. These two parents are responsible for breeding and producing offspring [18]. The main purpose of the selection operator is to ensure that the better genes are passed on to the next generation and progressively distributed among the population, thus increasing the overall fitness of the population [20]. Therefore, the selection of parents is based on the fitness values of the chromosomes, where fitter chromosomes have a higher probability of getting chosen [19].

However, favouring fitter chromosomes does not guarantee finding the global optimum. When the selection operator entirely depends on the best chromosome, there is a lack of variety in the mating pool. This results in the production of similar chromosomes in every generation, thereby reducing population diversity [21,22]. Consequently, the population prematurely converges and falls into the local optimum [20,21,22]. On the other hand, selecting unfit chromosomes slows down the convergence rate, resulting in a longer time to search for the global optimum. Hence, a good selection technique needs to maintain a balance between favouring fitter chromosomes and preserving population diversity to allow the solutions to converge to the global optimum within a reasonable time frame [21,22].

This section discusses two common selection techniques, which are roulette wheel (RW) selection, random selection, tournament selection, and linear ranking selection.

2.2.1.1 Roulette Wheel Selection

Roulette wheel (RW) selection is a selection technique in which all chromosomes in the population are distributed on a wheel proportionally based on their fitness [18,20]. The proportion assigned to each chromosome represents its chance of being selected as a parent. Hence, a fitter chromosome occupies a larger sector on the wheel and therefore has a higher chance of being chosen for the mating pool [20]. The wheel is then spun randomly. When the wheel stops, the chromosome occupying the sector pointed to by the pointer is selected as the parent [18,21]. Figure 2.2.3 shows that the "a2" chromosome, indicated by the pointer, is chosen as the parent.

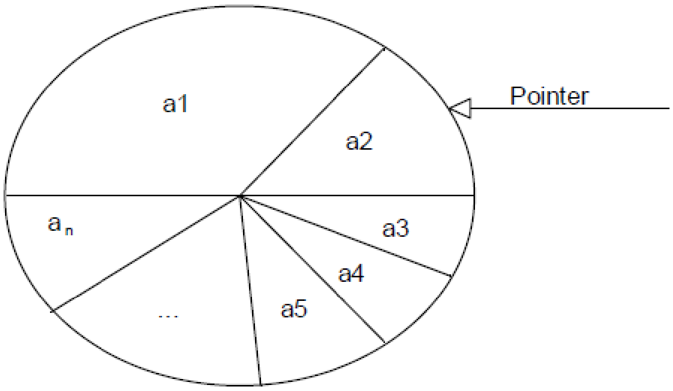


Figure 2.2.3 RW Selection [21]

Since better chromosomes are more likely to be chosen as parents, there is a risk of premature convergence [18]. Nevertheless, there is still a probability of selecting poorer chromosomes, which helps to avoid the solution being trapped in a local optimum [22,24]. Moreover, RW selection is widely used because it is easy to implement [18]. For instance, RW selection is used for optimising multi depot vehicle routing problem (MDVRP) with capacities and fixed endpoints in [23]. Besides, RW selection is applied in [25] to optimise travelling salesman problem (TSP), which is evaluated according to the minimum distance required to visit each city at least once and return to the starting city. This indicates that RW selection plays an important role in a genetic algorithm (GA) model for optimising non-deterministic polynomial complete (NP-complete) problems and combinatorial optimisation problems such as MDVRP and TSP.

2.2.1.2 Random Selection

Random selection is a selection technique in which all chromosomes in the population are sampled with equal probability, independent of fitness. Two parents are selected randomly from the population without any restriction.

Because no fitness-based weighting is required, random selection is straightforward to implement and computationally inexpensive, making it a common baseline or a diversity-preserving component in GA schemes. Furthermore, its unbiased sampling helps to maintain genetic diversity and reduces the risk of premature convergence, especially in noisy or deceptive fitness landscapes. However, the absence of selection pressure means that highly fit chromosomes are not preferentially propagated, which can slow convergence and increase the number of evaluations needed to reach a good solution. This has been proven by past research which compares the performance of different selection techniques against benchmark functions. In the experiment, random selection performs the worst among roulette wheel selection and tournament selection [20].

2.2.1.3 Tournament Selection

In tournament selection, several chromosomes are randomly selected to compete against each other [19,21] for a position as a parent. The winner of each tournament is evaluated by comparing the fitness of the participating chromosomes [23]. Hence, the fittest chromosome in the tournament is going to win and getting selected as the parent [19,21].

The number of chromosomes involved in each tournament is defined as the tournament size [21]. The larger the tournament size, the higher the chance that the best chromosome is selected and wins the tournament [19,23]. This increases the probability of losing diversity in the population [18,21]. Therefore, the tournament size needs to be carefully set to avoid premature convergence. The most commonly used variant is tournament selection with a tournament size of two, also known as binary tournament selection [21]. Figure 2.2.4 demonstrates the process of tournament selection with a tournament size of three.

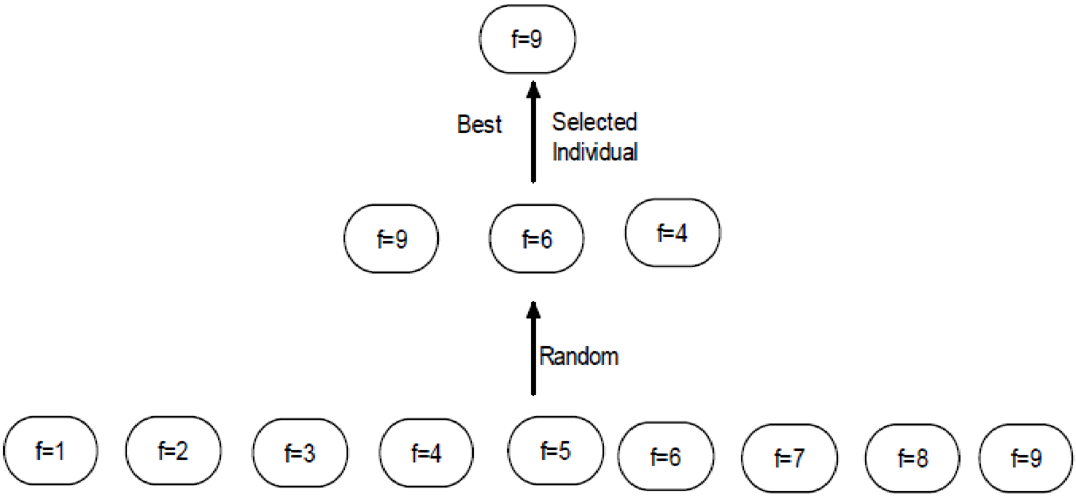


Figure 2.2.4 Process of Tournament Selection [21]

Tournament selection is the most popular selection technique in genetic algorithm (GA) due to several advantages [21,23,25]. One of the advantages is its high efficiency compared to other techniques. This is because it does not require a ranking process and therefore has low time complexity [21,25]. Furthermore, tournament selection is able to maintain population diversity with small or moderate tournament sizes. Yet, the population starts to lose its diversity when the tournament size is too large [18,21]. The high efficiency of tournament selection is supported by various research studies utilising it for optimisation problems. For example, tournament selection was applied in solving the multi depot vehicle routing problem (MDVRP) [19] and the travelling salesman problem (TSP) [25], as described in section 2.2.1.1.

2.2.1.4 Linear Ranking Selection

In Section 2.2.1.1, the disadvantage of Roulette Wheel Selection (RWS) is highlighted, where better chromosomes occupy disproportionately larger sectors on the selection wheel [23]. This provides them with higher selection chances while limiting opportunities for weaker chromosomes [22]. To address this imbalance, Ranking Selection, a method that applies the concept of normalisation to selection probabilities, has been introduced [23], with Linear Ranking Selection being one of its variations [18,23].

Linear Ranking Selection is essentially a modified version of RWS [18]. This technique involves four steps. First, the chromosomes are sorted based on their fitness, from best to worst. Second, they are ranked according to their order [21]. The best chromosome is assigned rank "1", whereas the worst chromosome receives rank "N", where "N" represents the population size [8]. Third, the chromosomes are distributed on a selection wheel depending on their ranks. The size of the sectors, which corresponds to the selection probability, increases linearly and uniformly from the lowest to the highest rank [23]. Lastly, the wheel is spun to select a chromosome as the parent.

By utilising ranking instead of fitness, the worst chromosome is able to maintain a relatively high selection probability [18]. This is illustrated in Figure 2.2.5, where chromosomes are more evenly distributed on the wheel when selection probabilities are assigned to them based on ranking rather than fitness. This approach preserves population diversity [22,23] and reduces the risk of premature convergence [18,23]. As a result, the likelihood of finding the global optimum is increased. However, this technique has several limitations. First, it is not efficient in terms of computational performance due to the necessity of sorting and ranking [18]. Second, the low tendency to favour the best chromosome results in a slow convergence rate [18,22,23].

Despite these drawbacks, the advantages of Linear Ranking Selection make it a balanced technique, favoured by researchers for optimisation problems. For instance, this technique was applied in tackling the Multi Depot Vehicle Routing Problem (MDVRP) [23] and the Travelling Salesman Problem (TSP) [25], which are described in Section 2.2.1.1. In these two experiments, Linear Ranking Selection outperformed other techniques, including RWS and Tournament Selection with a tournament size of five. This shows that this selection technique is a promising method for finding optimal solutions to optimisation problems.

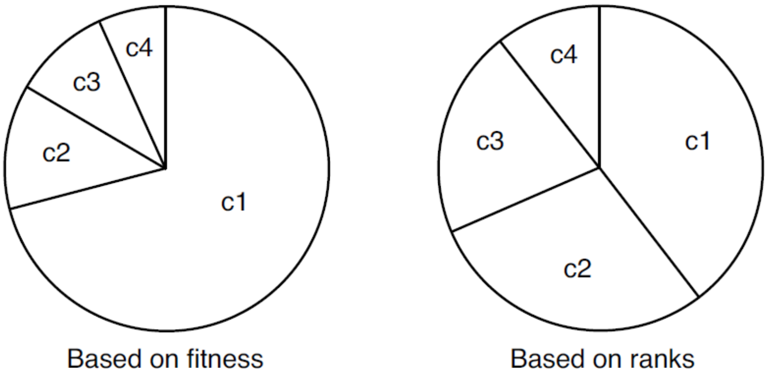


Figure 2.2.5 Selection Probabilities Based on Fitness and Ranking

2.2.2 Crossover

The crossover operator in genetic algorithm (GA) controls how two parent chromosomes are combined to create offspring. By exchanging segments of genetic material between selected parents, the operator introduces new gene combinations into the population and increases the chance that offspring inherit advantageous traits that improve overall fitness. This is supported by an experimental observation, which confirms that highly fit individuals often share specific genetic patterns that can be propagated to the next generation through crossover [27].

During crossover operation, giving too much emphasis to exploitation or exploration can hinder the search. Excessive exploitation, where parents are combined in very similar ways, limits diversity and increases the risk of premature convergence. On the other hand, excessive exploration, where chromosome segments are exchanged too randomly, slows down convergence and extends the searching time for the global optimum. Nonetheless, self-adaptive crossover rates can balance these tendencies by dynamically adjusting how aggressively genetic material is mixed, preserving diversity while still promoting fitter solutions [28].

This section discusses four common crossover technique, which are single-point crossover, two-point crossover, uniform crossover, and shuffle crossover.

2.2.2.1 Single-Point Crossover

Single-point crossover is a recombination operator where two parent chromosomes exchange genetic material at a single cut position to produce offspring [20,23]. After randomly choosing one crossover point along the genome, the first segment (from the start to the cut) is copied from first parent and the remaining segment (from the cut to the end) from second parent (and vice versa for the second child).

It is simple and fast, which can help propagate useful schemata and speed early progress. However, it assumes meaningful adjacency. If gene order does not reflect interacting features, the operator may disrupt dependencies or create invalid solutions and can reduce diversity if the cut often falls in similar places. Nonetheless, its simplicity makes it only offers low exploration compared to other crossover techniques [20], causing it infeasible [23].

2.2.2.2 Two-Point Crossover

Two-point crossover is a technique that selects two crossover positions at random on each parent chromosome, partitions the parents at those positions, and exchanges the genes that lie between the two points to create new offspring [29]. Because exactly two breakpoints are chosen, the procedure always produces two children whose middle segments come from opposite parents. Figure 2.2.6 depicts this process, showing the two randomly chosen cut points on both parents and the swapped middle segments that generate the resulting offspring.

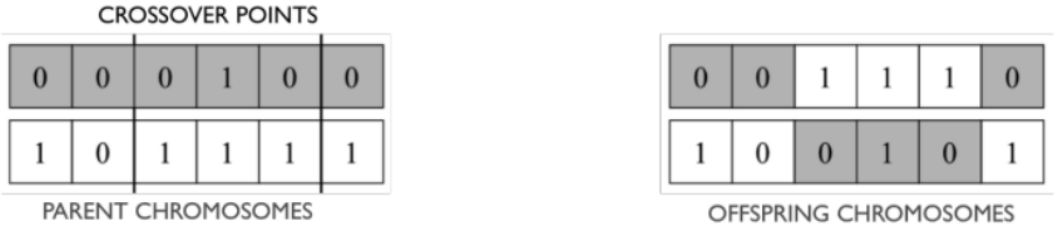


Figure 2.2.6 Process of Two-Point Crossover [30]

This operation technique is straightforward to code because it only requires picking two random indices and swapping the intervening genes. However, this simplicity also limits diversity. This is because the genes outside the selected segment remain unchanged, therefore the search may converge prematurely [18]. Despite these limitations, two-point crossover has yielded strong performance on several combinatorial optimisation problems. It has been incorporated into genetic algorithm (GA) solutions for the green vehicle routing problem (GVRP), where it helps generate high quality routes [31]. Besides, the GA applying two-point crossover outperforms several previously published methods such as tabu search (TS) and multi-pass in the resource-constrained project scheduling problem with transfer times (RCPSPTT) [32]. These successes demonstrate that two-point crossover can be an effective component of GA frameworks that tackle non-deterministic polynomial complete (NP-complete) tasks.

2.2.2.3 Uniform Crossover

Uniform crossover selects each gene independently from either parent with equal probability. Instead of cutting chromosomes into segments, it swaps individual bits by generating a random mask of 0s and 1s and copying genes accordingly: a mask bit of 1 takes the gene from the first parent, while 0 takes it from the second [29,39]. This produces two offspring whose genes are chosen uniformly from both parents. Because the mask is random, the effective number and location of crossover points are not predetermined, and inheritance is independent of position.

Its strengths include unbiased exploration, suitability for large gene subsets, and strong recombination capability. By examining every gene position, it can probe a wider solution space. The main drawback is a typically slower convergence rate, since many genes may be swapped at once, dampening exploitation. In practice, uniform crossover has been used for the travelling salesman problem (TSP), where it helps maintain genetic diversity and can improve the algorithm's speed in reaching optimal or near-optimal solutions [40].

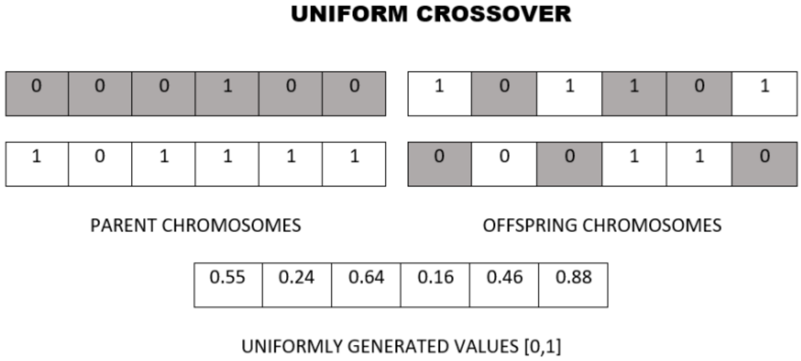


Figure 2.2.7 Process of Uniform Crossover [30

2.2.2.4 Shuffle Crossover

Shuffle crossover aims to eliminate positional bias so that offspring do not depend on where the crossover cut happens [42,43]. The method selects two parents and a random cut point, then first applies the same random permutation to both parents' genes. With the genes shuffled, the parents undergo a standard single-point crossover at the chosen position to create two children [43]. Finally, the inverse permutation unshuffles the offspring back to the original indexing.

In spirit it resembles uniform crossover, but the crucial difference is that shuffle crossover exchanges contiguous segments rather than individual bits. Because a fresh random shuffling is used for each crossover, the original gene positions have far less influence on recombination and on the resulting offspring's quality [41].

2.2.3 Mutation

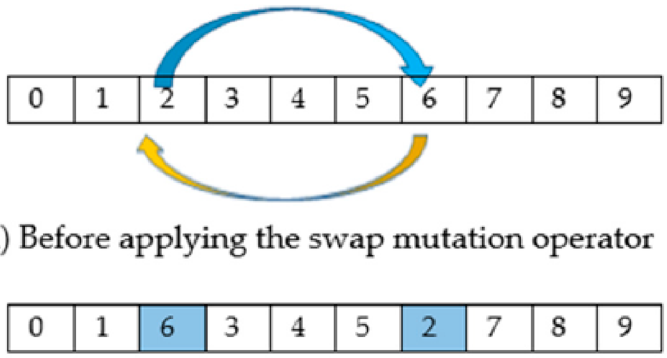
The mutation operator in genetic algorithm (GA) introduces small random changes into the chromosomes in order to preserve genetic diversity and broaden the search space. It typically alters one or two genes and is applied with a very low probability [23,33]. By injecting fresh variability after selection and crossover, the operator reduces the chance of premature convergence and helps the population escape local optima [28].

When tuning the mutation probability, it requires high awareness. If the rate is set too low, exploration is limited, promising genes may never be tested, and the search can stick in local optima. If the rate is set too high, offspring differ so greatly from their parents that the algorithm struggles to learn from past generations, which slows down convergence [33]. An effective setting must balance diversity and refinement, so that the search progresses toward the global optimum within a reasonable time [28,34].

This section discusses a common mutation technique, which is swap mutation.

2.2.3.1 Swap Mutation

Swap mutation is an operation technique that selects two genes within a chromosome and exchanges their positions [36]. Figure 2.2.8 shows an example in which the genes “2” and “6” are swapped. Because only two positions change, most neighbouring genes stay together. Therefore, population diversity is maintained, where successful gene combinations are not drastically disrupted.



(a) Before applying the swap mutation operator

(b) Before applying the swap mutation operator

Figure 2.2.8 Swap Mutation [35]

Although this technique preserves much of the original adjacency information, it inevitably breaks some links, which can be problematic in sequence-sensitive problems such as the travelling salesman problem (TSP), where the order of cities forms a path [36]. Even so, swap mutation is valued for its simplicity and its ability to inject diversity that helps the search escape local optima. These qualities explain its adoption in combinatorial optimisation tasks such as the multi-depot vehicle routing problem (MDVRP) [23] and the university course timetabling problem (UCTP) [7], where maintaining feasibility while exploring alternative arrangements is essential.

2.2.4 Replacement

The replacement operator in genetic algorithm (GA) determines which chromosomes leave the population and which new offspring join the population [22]. Since the population size must be kept constant, space must be created for the new offspring to join the population [26]. There are two types of replacement techniques, which are steady state updates and generational updates [22,26].

Generational update techniques replace the entire population with newly produced offspring [22,26]. This restricts chromosomes to mating only with those from the same generation [22]. On the other hand, steady state update techniques allow new offspring to join the population immediately after each reproductive process [22,26]. This involves the replacement of existing chromosomes. Typically, a tournament method is used to decide which chromosome to replace. Sometimes, the worst or oldest chromosomes are replaced by the offspring to accelerate population convergence [22].

According to [20] and [26], steady state update techniques generally perform better than generational update techniques. This is because the nature of the latter, which replaces the entire population, prevents the best chromosome from the previous generation from being carried over and inherited in future generations. Therefore, this section mainly discusses steady state update techniques, specifically two of them, which are weak parent replacement and tournament replacement.

2.2.4.1 Weak Parent Replacement

Weak parent replacement is a technique where parents and offspring compete for a spot in the population. In this technique, the two fittest chromosomes out of the four (two parents and two offspring) are retained in the population [22,26]. This allows the weaker parents to be replaced by the stronger offspring, thus gradually increasing the overall fitness of the population [20,22]. This technique performs best when the selection operator includes both fit and unfit chromosomes as parents. Nevertheless, if the selection operator consistently chooses only fitter chromosomes as the parents, the improvement in population fitness is limited [22].

Weak parent replacement was utilised in minimisation experiments on a test suite consisting of 6 benchmark functions and 3 real-world problems [26]. Besides, [20] used weak parent replacement in an experiment evaluating various combinations of operation techniques in GA with benchmark functions. The application of weak parent replacement in these research studies demonstrates its capability of maintaining population diversity, allowing the population to efficiently converge to the global optimum.

2.2.4.2 Tournament Replacement

Tournament replacement is a technique that replaces weaker chromosomes in the population with newly generated offspring. Unlike weak parent replacement and both parent replacement, which focus only on the parents and offspring for replacement decisions, tournament replacement selects chromosomes for replacement from the entire population.

The selection process in tournament replacement resembles tournament selection, which is discussed in Section 2.2.1.3, except that the worst chromosome in the tournament is chosen, rather than the best [22]. In this process, the size of the tournament, representing the number of participating chromosomes, needs to be carefully determined [21]. A larger tournament size has a higher probability of selecting the worst chromosome from the population each time to win the tournament, which can lead to premature convergence [19,23]. Following this setup, the process begins by randomly selecting several chromosomes for a tournament. During the contest, these chromosomes compete against each other based on their fitness. The worst among the chosen chromosomes wins the tournament and is selected to be replaced by the offspring [19,21].

Although this technique consistently targets weaker chromosomes, it does not guarantee an improvement in the population's overall fitness over successive generations. This is because the selected chromosomes are replaced with the newly generated offspring, regardless of their fitness. Consequently, there may be ineffective replacements if the offspring are not fitter than the chromosomes they replace. This results in a regression in the population's overall fitness, thus slowing down the convergence rate.

Despite these limitations, tournament replacement remains a valuable technique. However, the discussion and application of this technique in both academic research and practical scenarios are notably insufficient, highlighting a domain that has yet to be explored.

2.3 Constraints

2.3.1 Hard Constraints

No.	Hard Constraints	Previous Work
1	A student must attend at most one class per time slot.	[1], [2], [4], [5], [7], [8], [9], [12], [15], [38]
2	A lecturer must teach at most one class per time slot.	[1], [2], [4], [7], [9], [12], [15], [16], [38]
3	A room must host at most one class per time slot.	[1], [2], [4], [5], [7], [8], [12], [15], [16], [38]
4	A class must have exactly one lecturer.	[2]
5	A class must be assigned a room.	[2], [4], [38]
6	A class must use a room whose features meet its requirements.	[1], [5], [9], [12], [15]
7	A class must not enrol more students than the room's capacity.	[1], [2], [4], [5], [7], [8], [12], [15], [16], [38]
8	A class must be scheduled on weekdays only.	[7]
9	A class must be scheduled within the allowed daily time window.	[2], [38]

Table 2.3.1 Hard Constraints of Previous Work

2.3.2 Soft Constraints

No.	Soft Constraints	Previous Work
1	A student should not exceed the daily study-hours limit.	[4]

2	A student should not study beyond the consecutive-hours limit.	[1], [2], [7], [8], [12]
3	A student should attend more than one class when present on a day.	[1], [2], [7], [12]
4	A student’s courses should be concentrated into as few days as possible.	[4]
5	A student’s timetable should minimise idle gaps.	[4]
6	A lecturer should not exceed the daily teaching-hours limit.	[4]
7	A lecturer should not teach beyond the consecutive-hours limit.	[2], [9], [38]
8	A lecturer’s courses should be concentrated into as few days as possible.	[4]
9	A lecturer’s timetable should minimise idle gaps.	[2], [4]
10	A lecturer’s preferred time slots should be honoured where feasible.	[2], [4], [9]
11	A class should not occupy a lunch-break slot.	[2], [4], [9], [12]
12	A class should use a room whose capacity closely matches its size.	[8]
13	A departmental course should be held in its own department’s building.	[4]
14	A course’s classes should follow any required sequence.	[1], [12]
15	A course’s classes should span at least the specified minimum number of days.	[1], [12], [16]

Table 2.3.2 Soft Constraints of Previous Work

2.4 Critical Remarks of Previous Work

Existing studies on university course timetabling problems (UCTPs) demonstrate that a wide range of metaheuristics can deliver feasible and high-quality schedules. The research [4] applies goal programming (GP) with simulated annealing (SA) and performs well on a real institutional dataset. Besides, the study [5] introduces adaptive tabu search (ATS) and validates its competitiveness against 14 published approaches. Particle swarm optimisation (PSO) with selective search (PSOSS) in the research [9] and ant colony optimisation (ACO) with selective probability (ACOSP) in the research [11] both reports best-in-class solutions for their respective experiments. Other than that, Harmony search (HS) enhanced by great deluge (GD) in the paper [16] and the best-nests (BS) variant of cuckoo search (CS) in the research [15] further illustrate how hybrid metaheuristics can unlock performance gains in UCTPs with real-world datasets.

Despite these contributions, three recurring limitations emerge. First, comparative approaches are either absent or narrow. The study [4] does not contrast its results with other metaheuristics, while the study [11] measures performance against only one genetic algorithm (GA) implementation. Second, several studies operate under settings that may hide additional improvements. The research [8] enforces a short runtime limit and filters crossover operations for feasibility, which constrains exploration, whereas the research [15] models only hard constraints. Third, most evaluations overlook everyday travelling challenges on university campus, particularly the proximity of classrooms for consecutive classes.

Within the subgroup of GA papers, methodological diversity remains limited. The research [7] employs roulette wheel (RW) or steady-state selection with swap mutation and replaces the weakest individual with the fittest offspring yet omits crossover entirely. Other than that, the research [8] combines quinary tournament selection, one-point crossover, random mutation and conditional replacement, but does not investigate alternative operator combinations. Consequently, these studies vary parameters inside a narrow envelope rather than exploring how different combinations of selection, crossover, mutation and replacement interact.

Furthermore, no work reviewed models the constraint that consecutive classes for the same student should be held in the same building, even though this requirement directly affects student well-being and campus logistics. In addition, the small set of GA operator combinations examined so far is insufficient to establish which arrangements truly balance exploration and exploitation when both hard and soft constraints are present. These gaps motivate this project, which introduces the building continuity constraint and systematically benchmarks a broader set of selection, crossover, mutation, and replacement techniques to identify a configuration that delivers high-quality timetables while reducing student travel.

Chapter 3

System Methodology

3.1 Project Development

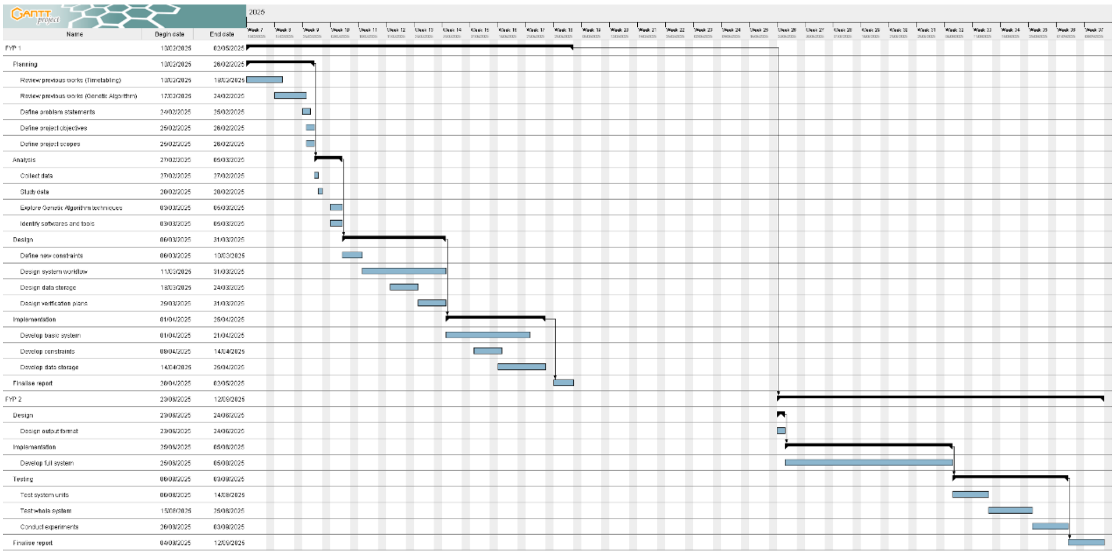


Figure 3..11 Gantt Chart for Project Development

Figure 3.1.1 illustrates the project development workplan in a Gantt chart, consisting of 2 phases (FYP1 and FYP2) and 5 stages, which include planning, analysis, design, implementation, and testing.

The project begins with a planning stage. Previous work on timetabling including variants, constraints, and approaches, is reviewed, followed by genetic algorithm (GA), in which various techniques of multiple GA operators are studied, to understand established optimisation strategies that best fit scheduling problems, specifically university timetabling course problems (UCTPs). Insights from these studies shape concise problem statements, particularly based on the shortcomings found in them. Besides, clear project scopes, such as datasets and constraints, are constructed. A set of achievable objectives are also defined to guide all subsequent work.

Next, attention shifts to analysis. Real timetable data are gathered and cleansed for consistency. Data such as programme structures and lecturer information are studied to strengthen dataset understanding. Partial mock data is created using the data collected to serve as the dataset used in the project. In parallel, GA operation techniques are compared in terms of their concepts, implementations, strengths, and weaknesses, in order to choose those worth-testing techniques. The programming languages, code libraries, and development tools are also analysed to select the most suitable combinations for rapid experimentation.

With requirements clarified, design tasks formalise how the system will operate. New constraints are enumerated to reflect fundamental focuses of the system. A high-level workflow is drafted to show how data flow from input to timetable generation and schedule validation in GA. Details such as chromosome encodings and operator designs are mentioned as well. Besides, a relational schema and data storage technology is designed to hold data used by the system, including courses, rooms, lecturers, and students. An entity relationship diagram (ERD) is drawn to show a clear structure. Lastly, verification plans specifying metrics and test scenarios that will later confirm whether generated timetables satisfy every constraint, are designed.

The implementation stage then delivers a working prototype on the Visual Studio Code (VS Code) development platform, as previously decided, using the selected Java programming language and MySQL database technology. Core programme structures are coded, laying out the chromosome representation and the genetic techniques for selection, crossover, mutation, and replacement operators. Constraint-checking modules are integrated to penalise infeasible timetables, and persistent data storage is set up to save datasets. However, no graphical user interface (GUI) is developed, and output format is yet to be finalised.

An interim report wraps up the first phase by documenting achievements, design decisions, and any limitations discovered along the way.

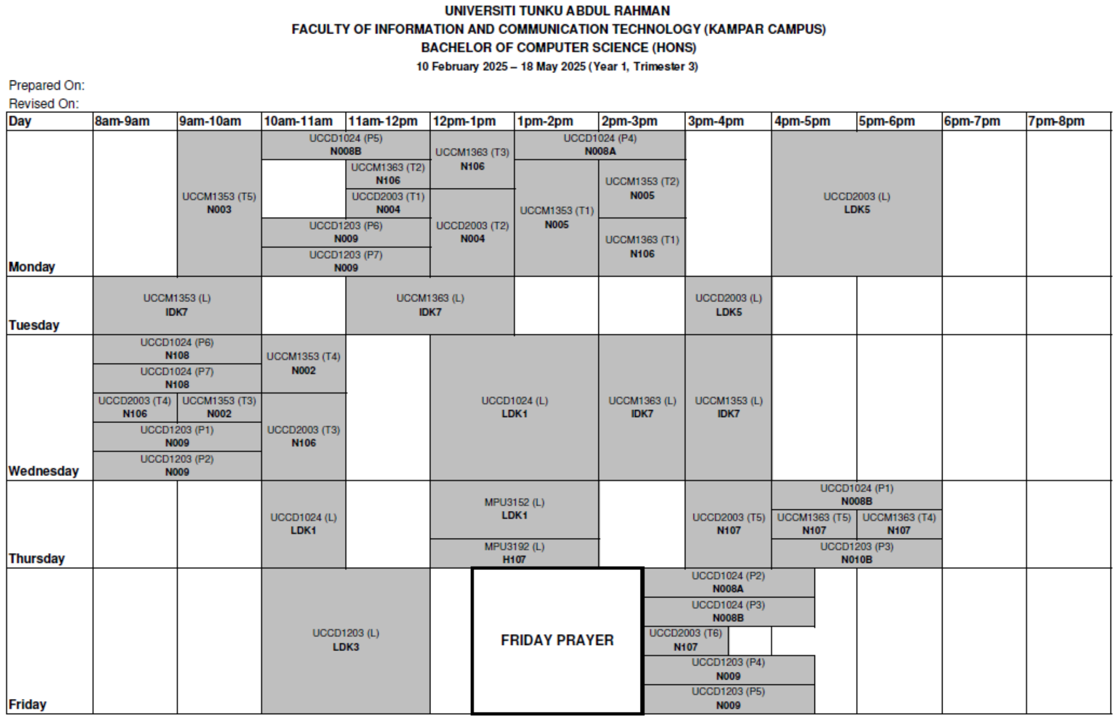
The second phase opens with a brief design refinement that focuses on output format. The presentation method of the final timetables is determined so that these results can be easily interpreted.

The stage is followed by the full-scale implementation. All remaining features such as GA operation techniques, are completed, including an experiment automation function. The system is expected to support end-to-end timetable generation and experimental result storage without manual intervention.

Continue with the testing stage, system tests verify end-to-end workflows under realistic workloads. Experiments are conducted to measure timetable quality and runtime efficiency against varying dataset sizes. Results are collected and analysed to report findings of this research.

Lastly, a detailed report is produced that captures the completed system, experimental results, and recommendations for future enhancements, marking the formal conclusion of the project.

3.2 Data Collection

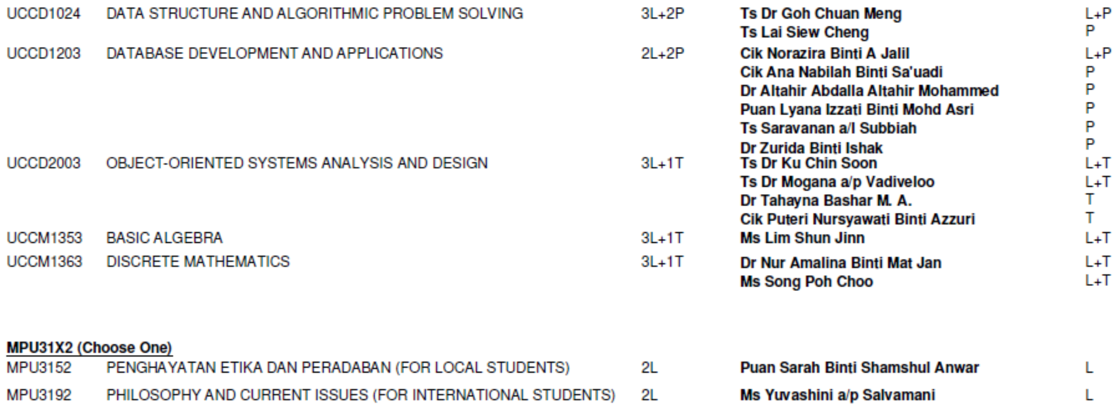


UNIVERSITI TUNKU ABDUL RAHMAN
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)
BACHELOR OF COMPUTER SCIENCE (HONS)
10 February 2025 – 18 May 2025 (Year 1, Trimester 3)

Prepared On:
Revised On:

Day	8am-9am	9am-10am	10am-11am	11am-12pm	12pm-1pm	1pm-2pm	2pm-3pm	3pm-4pm	4pm-5pm	5pm-6pm	6pm-7pm	7pm-8pm
Monday		UCCM1353 (T5) N003	UCCD1024 (P5) N008B UCCD1203 (P6) N009 UCCD1203 (P7) N009	UCCM1363 (T2) N106 UCCD2003 (T1) N004	UCCM1363 (T3) N106 UCCD2003 (T2) N004	UCCD1024 (P4) N008A UCCM1353 (T1) N005	UCCM1353 (T2) N005 UCCM1363 (T1) N106		UCCD2003 (L) LDK5			
Tuesday	UCCM1353 (L) IDK7			UCCM1363 (L) IDK7				UCCD2003 (L) LDK5				
Wednesday	UCCD1024 (P6) N108 UCCD1024 (P7) N108 UCCD2003 (T4) N106 UCCD1203 (P1) N009 UCCD1203 (P2) N009	UCCM1353 (T3) N002	UCCM1353 (T4) N002 UCCD2003 (T3) N106		UCCD1024 (L) LDK1		UCCM1363 (L) IDK7	UCCM1353 (L) IDK7				
Thursday			UCCD1024 (L) LDK1		MPU3152 (L) LDK1 MPU3192 (L) H107			UCCD2003 (T5) N107	UCCD1024 (P1) N008B UCCM1363 (T5) N107 UCCD1203 (P3) N010B	UCCM1363 (T4) N107		
Friday			UCCD1203 (L) LDK3		FRIDAY PRAYER		UCCD1024 (P2) N008A UCCD1024 (P3) N008B UCCD2003 (T6) N107 UCCD1203 (P4) N009 UCCD1203 (P5) N009					

Figure 3.2.1 Timetable for February 2025 CS Y1T3 Students



UCCD1024	DATA STRUCTURE AND ALGORITHMIC PROBLEM SOLVING	3L+2P	Ts Dr Goh Chuan Meng	L+P
			Ts Lai Siew Cheng	P
UCCD1203	DATABASE DEVELOPMENT AND APPLICATIONS	2L+2P	Cik Norazira Binti A Jalil	L+P
			Cik Ana Nabilah Binti Sa'uadi	P
			Dr Altahir Abdalla Altahir Mohammed	P
			Puan Lyana Izzati Binti Mohd Asri	P
			Ts Saravanan a/l Subbiah	P
			Dr Zurida Binti Ishak	P
UCCD2003	OBJECT-ORIENTED SYSTEMS ANALYSIS AND DESIGN	3L+1T	Ts Dr Ku Chin Soon	L+T
			Ts Dr Mogana a/p Vadiveloo	L+T
			Dr Tahayna Bashar M. A.	T
			Cik Puteri Nursyawati Binti Azzuri	T
UCCM1353	BASIC ALGEBRA	3L+1T	Ms Lim Shun Jinn	L+T
UCCM1363	DISCRETE MATHEMATICS	3L+1T	Dr Nur Amalina Binti Mat Jan	L+T
			Ms Song Poh Choo	L+T
MPU31X2 (Choose One)				
MPU3152	PENGHAYATAN ETIKA DAN PERADABAN (FOR LOCAL STUDENTS)	2L	Puan Sarah Binti Shamshul Anwar	L
MPU3192	PHILOSOPHY AND CURRENT ISSUES (FOR INTERNATIONAL STUDENTS)	2L	Ms Yuvashini a/p Salvamani	L

Figure 3.2.2 Programme Structure for February 2025 CS Y1T3 Students

Figures 3.2.1 and 3.2.2 present the official timetables and programme structures for February 2025 Computer Science (CS) year one, trimester three (Y1T3) students at Universiti Tunku Abdul Rahman (UTAR). These records are downloaded from the university's Faculty of Information and Communication Technology (FICT) website. By studying these data, it is found that each timetable specifies the daily allocation of classes, the assigned room, and its building. A clear pattern emerges, as rooms in building L host lectures, whereas rooms in building N host tutorials and practical sessions. Furthermore, the timetable spans Monday to Friday and reserves a period each Friday for Muslim prayer. Besides, the accompanying programme structure lists the course code, course name, offered class type, duration, and lecturer for every course. Together, these data provide the baseline for a partial mock dataset used in this project.

3.3 System Constraint

3.3.1 Hard Constraints

No.	Hard Constraint
1	A student must attend at most one class per time slot.
2	A student must enrol in every course required for the semester.
3	A lecturer must teach at most one class per time slot.
4	A room must host at most one class per time slot.
5	A class must enrol at least one student.
6	A class must have exactly one lecturer.
7	A class must be assigned a room once and only once.
8	A lecture class must be held in a lecture hall.
9	A tutorial class must be held in a tutorial room.
10	A practical class must be held in a computer lab.
11	A class must not enrol more students than the room's capacity.
12	A class must be scheduled on weekdays only.
13	A class must be scheduled between 08:00 and 18:00.
14	A class must not be scheduled on Friday between 12:00 and 14:00 (Muslim prayer time).
15	A class must not span across multiple days.

Table 3.3.1 Hard Constraints for Project

3.3.2 Soft Constraints

No.	Soft Constraint
1	A student should study no more than four consecutive hours.
2	A lecturer should teach no more than four consecutive hours.
3	A student's consecutive classes should be held in the same building.
4	A lecturer should receive at least one teaching hour.

Table 3.3.2 Soft Constraints for Project

3.4 System Requirements

3.4.1 Hardware

The hardware involved in this project is a laptop, which is used to develop the university timetabling system.

Description	Specification
Model	MSI GF65 Thin 10UE
Processor	Intel(R) Core(TM) i5-10200H CPU @ 2.40GHz
Operating System	Windows 11 Home Single Language 64-bit
Graphic	NVIDIA GeForce RTX 3060 Laptop GPU
Memory	16GB DDR4 RAM
Storage	512GB SSD

Table 3.4.1 Specifications of Laptop

3.4.2 Software

The software involved in this project is an integrated development environment (IDE), which is associated with a programming language and a database engine to develop the university timetabling system.

Descriptions	Specifications
IDE	Visual Studio Code (VS Code) version 1.99
Programming Language	Java OpenJDK version 21.0.5
Database Engine	MySQL version 8.0.40
Database Driver	MySQL Connector/J version 9.4.0

Table 3.4.2 Specifications of Software

3.5 Verification Plans

3.5.1 Hard Constraint Tests

No.	Hard Constraint	Verification
1	A student must attend at most one class per time slot.	Scan each student timetable; every slot must contain no more than one class.
2	A student must enrol in every course required for the semester.	Scan each student timetable; every compulsory course must appear.
3	A lecturer must teach at most one class per time slot.	Scan each lecturer timetable; every slot must contain no more than one class.
4	A room must host at most one class per time slot.	Scan each room timetable; every slot must contain no more than one class.
5	A class must enrol at least one student.	Count student timetable entries per class and confirm the total is at least one.
6	A class must have exactly one lecturer.	Scan each course timetable; every class must contain no more than one lecturer.
7	A class must be assigned a room once and only once.	Scan each course timetable; every class must be assigned a room once and only once.
8	A lecture class must be held in a lecture hall.	For each lecture in the courses timetable, verify its room in the rooms timetable is typed “lecture”.
9	A tutorial class must be held in a tutorial room.	For each tutorial in the courses timetable, verify its room in the rooms timetable is typed “tutorial”.

10	A practical class must be held in a computer lab.	For each practical in the courses timetable, verify its room in the rooms timetable is typed “practical”.
11	A class must not enrol more students than the room’s capacity.	Count student timetable entries per class and confirm the total is no more than room capacity.
12	A class must be scheduled on weekdays only.	This constraint is initially satisfied. Every class day should be Monday to Friday.
13	A class must be scheduled between 08:00 and 18:00.	This constraint is initially satisfied. Every class should start after 08:00 and end before 18:00.
14	A class must not be scheduled on Friday between 12:00 and 14:00 (Muslim prayer time).	Verify no class overlaps the Friday 12:00 to 14:00 time slots.
15	A class must not span across multiple days.	Scan each course timetable; every class must not span across multiple days..

Table 3.5.1 Tests for Hard Constraints

3.5.2 Soft Constraint Tests

No.	Soft Constraint	Verification
1	A student should study no more than four consecutive hours.	For each student timetable, locate every contiguous block of occupied hours; the length of every block must be no more than four.
2	A lecturer should teach no more than four consecutive hours.	For each lecturer timetable, locate every contiguous block of occupied hours; the length of every block must be no more than four.
3	A student’s consecutive classes should be held in the same building.	For each student, examine every pair of back-to-back classes; use the rooms timetable to confirm both classes share the same building.
4	A lecturer should receive at least one teaching hour.	This constraint is initially satisfied. Every lecturer should have at least one teaching hour.

Table 3.5.2 Tests for Soft Constraints

3.5.3 Resource Utilisation Tests

No.	Resource Utilisation Percentage	Verification
1	30%	Generate timetables with zero penalty cost.
2	60%	Generate timetables with zero penalty cost.
3	90%	Generate timetables with zero penalty cost.

Table 3.5.3 Tests for Resource Utilisation

Chapter 4
System Design

4.1 System Architecture Design

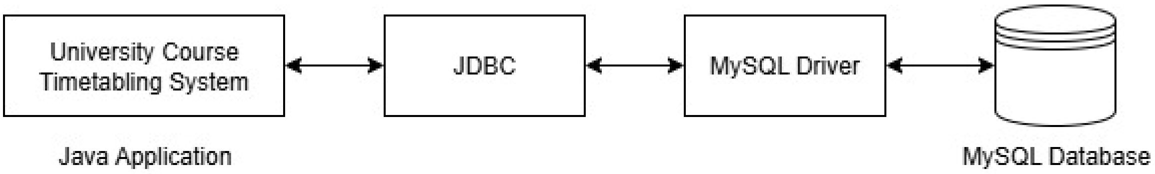
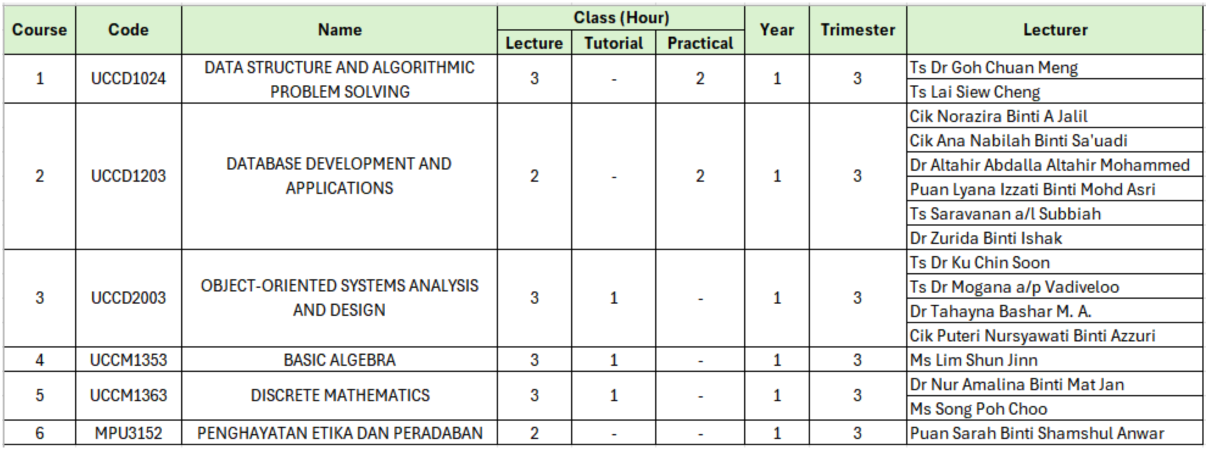


Figure 4.1.1 University Course Timetabling System Architecture

The system adopts a two-tier client-server model with a three-layer data-access design. The Java-based university course timetabling system (client) issues Structured Query Language (SQL) operations through the Java Database Connectivity (JDBC) application programming interface (API), which provides a standard interface for creating connections, preparing statements, and handling results. The MySQL JDBC driver implements that interface, translating JDBC calls into the MySQL wire protocol and managing details such as authentication, transaction control, and data type mapping. At the back end, the MySQL database (server) stores the schema and data, and enforces integrity via keys, constraints, and triggers. Communication is bidirectional along the chain between application, JDBC API, MySQL driver, and MySQL database. This ensures the results, errors, and metadata flow back to the application while keeping database specifics encapsulated behind JDBC.

4.2 Input Design

The collected data described in Section 3.2 provides the foundation for the partial mock dataset used in this project. To reduce complexity, only local students are considered. Consequently, courses intended exclusively for international students are omitted. For example, the course *Philosophy and Current Issues* is excluded, while *Penghayatan Etika dan Peradaban* remains. This yields a more uniform programme structure and avoids handling multiple parallel scenarios. The adjustment preserves essential relationships among courses, rooms and lecturers while keeping the dataset compact and easier to manipulate during development. Figure 4.2.1 shows the tailored subset of the collected data.

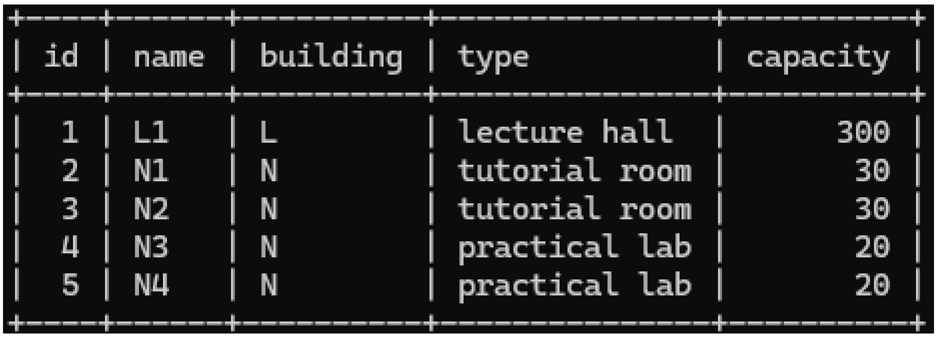


Course	Code	Name	Class (Hour)			Year	Trimester	Lecturer
			Lecture	Tutorial	Practical			
1	UCCD1024	DATA STRUCTURE AND ALGORITHMIC PROBLEM SOLVING	3	-	2	1	3	Ts Dr Goh Chuan Meng
								Ts Lai Siew Cheng
2	UCCD1203	DATABASE DEVELOPMENT AND APPLICATIONS	2	-	2	1	3	Cik Norazira Binti A Jalil
								Cik Ana Nabilah Binti Sa'uadi
								Dr Altahir Abdalla Altahir Mohammed
								Puan Lyana Izzati Binti Mohd Asri
								Ts Saravanan a/l Subbiah
								Dr Zurida Binti Ishak
3	UCCD2003	OBJECT-ORIENTED SYSTEMS ANALYSIS AND DESIGN	3	1	-	1	3	Ts Dr Ku Chin Soon
								Ts Dr Mogana a/p Vadiveloo
								Dr Tahayna Bashar M. A.
								Cik Puteri Nursyawati Binti Azzuri
4	UCCM1353	BASIC ALGEBRA	3	1	-	1	3	Ms Lim Shun Jinn
5	UCCM1363	DISCRETE MATHEMATICS	3	1	-	1	3	Dr Nur Amalina Binti Mat Jan
								Ms Song Poh Choo
6	MPU3152	PENGHAYATAN ETIKA DAN PERADABAN	2	-	-	1	3	Puan Sarah Binti Shamshul Anwar

Figure 4.2.1 Tailored Collected Data

This information is used to generate mocked rooms data. Other than that, 150 students and 15 student groups are generated programmatically. Figure 4.2.1 shows the courses, lecturers, course-to-lecturer assignments data while Figure 4.2.2 shows the mocked rooms data.

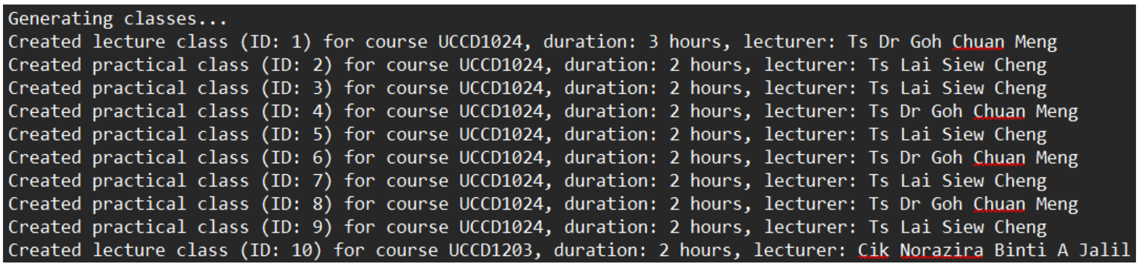
Based on this subset, the system initialises courses, lecturers, course-to-lecturer assignments, and per-course classes. Course, lecturer, and assignment data are generated from the available information. Besides, room conventions are defined, where building L hosts lectures, while building N hosts tutorials and practicals. These conventions inform the generation of the mock rooms dataset. In addition, 150 students and 15 student groups are created programmatically. Figure 4.2.1 shows the courses, lecturers, and course-to-lecturer assignments; Figure 4.2.2 shows the mock rooms data.



```
+----+------+----------+---------------+----------+
| id | name | building | type          | capacity |
+----+------+----------+---------------+----------+
|  1 | L1   | L        | lecture hall  |      300 |
|  2 | N1   | N        | tutorial room |       30 |
|  3 | N2   | N        | tutorial room |       30 |
|  4 | N3   | N        | practical lab |       20 |
|  5 | N4   | N        | practical lab |       20 |
+----+------+----------+---------------+----------+
```

Figure 4.2.2 Mock Rooms Data

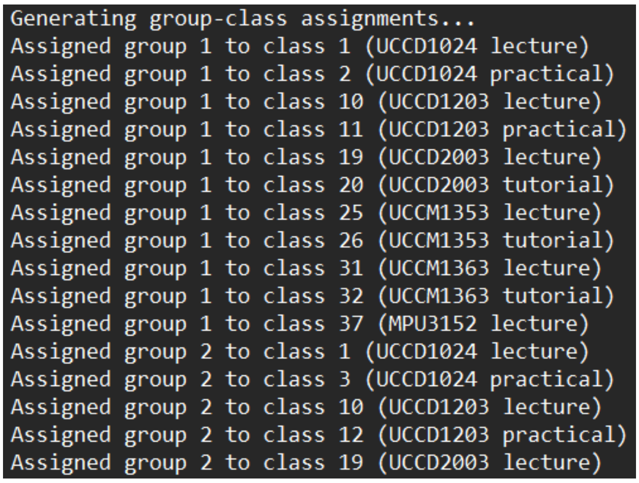
For each course, the system then generates the set of classes. It first estimates the number of parallel classes required by dividing the number of students by the capacity of the appropriate room type and taking the ceiling. For instance, 150 students enrol in course UCCD1024, which has 3 hours of lectures and 2 hours of practicals. Lectures are held in lecture halls (capacity 300), so only 1 lecture class is required. On the other hand, practicals are held in practical labs (capacity 20), so 8 practical classes are needed (⌈150/20⌉). In total, the course requires 9 classes (1 lecture plus 8 practicals). Each class is then assigned a lecturer chosen from those linked to the course, prioritising the lecturer with the fewest current teaching hours. Figure 4.2.3 shows an example of class generation.



```
Generating classes...
Created lecture class (ID: 1) for course UCCD1024, duration: 3 hours, lecturer: Ts Dr Goh Chuan Meng
Created practical class (ID: 2) for course UCCD1024, duration: 2 hours, lecturer: Ts Lai Siew Cheng
Created practical class (ID: 3) for course UCCD1024, duration: 2 hours, lecturer: Ts Lai Siew Cheng
Created practical class (ID: 4) for course UCCD1024, duration: 2 hours, lecturer: Ts Dr Goh Chuan Meng
Created practical class (ID: 5) for course UCCD1024, duration: 2 hours, lecturer: Ts Lai Siew Cheng
Created practical class (ID: 6) for course UCCD1024, duration: 2 hours, lecturer: Ts Dr Goh Chuan Meng
Created practical class (ID: 7) for course UCCD1024, duration: 2 hours, lecturer: Ts Lai Siew Cheng
Created practical class (ID: 8) for course UCCD1024, duration: 2 hours, lecturer: Ts Dr Goh Chuan Meng
Created practical class (ID: 9) for course UCCD1024, duration: 2 hours, lecturer: Ts Lai Siew Cheng
Created lecture class (ID: 10) for course UCCD1203, duration: 2 hours, lecturer: Cik Norazira Binti A Jalil
```

Figure 4.2.3 Output of Class Generation

After all classes are created, student groups are allocated to classes by academic year and trimester. Allocation is randomised but load-balancing, where each new group is placed into the currently least-loaded suitable class to maintain a balanced distribution. Room-capacity violations are prevented by a database trigger (see Section 4.5.3) that blocks any over-enrolment at insert time. Figure 4.2.4 shows an example of group-to-class assignment.



```
Generating group-class assignments...
Assigned group 1 to class 1 (UCCD1024 lecture)
Assigned group 1 to class 2 (UCCD1024 practical)
Assigned group 1 to class 10 (UCCD1203 lecture)
Assigned group 1 to class 11 (UCCD1203 practical)
Assigned group 1 to class 19 (UCCD2003 lecture)
Assigned group 1 to class 20 (UCCD2003 tutorial)
Assigned group 1 to class 25 (UCCM1353 lecture)
Assigned group 1 to class 26 (UCCM1353 tutorial)
Assigned group 1 to class 31 (UCCM1363 lecture)
Assigned group 1 to class 32 (UCCM1363 tutorial)
Assigned group 1 to class 37 (MPU3152 lecture)
Assigned group 2 to class 1 (UCCD1024 lecture)
Assigned group 2 to class 3 (UCCD1024 practical)
Assigned group 2 to class 10 (UCCD1203 lecture)
Assigned group 2 to class 12 (UCCD1203 practical)
Assigned group 2 to class 19 (UCCD2003 lecture)
```

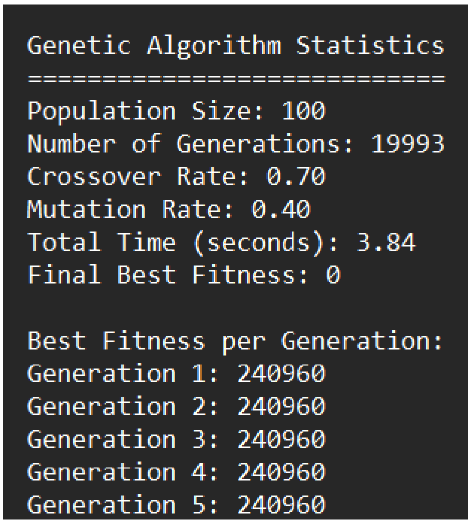
Figure 4.2.4 Output of Group-to-Class Assignment

Together, these inputs form a coherent, valid dataset for the university course timetabling system.

4.3 Output Design

The output of the university course timetabling system comprises several structured, well-organised components designed for analysis and usability.

For each run, genetic algorithm (GA) experiment statistics, such as the number of generations required to reach zero penalty cost, the time taken (seconds), and the fitness value per generation, are written to a text file. These metrics characterise the optimisation process and support subsequent analysis. Figure 4.3.1 shows an example of GA run statistics.

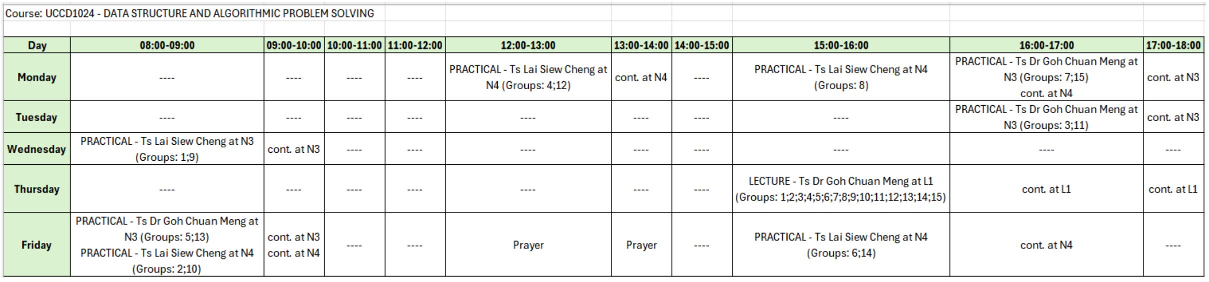


```
Genetic Algorithm Statistics
============================
Population Size: 100
Number of Generations: 19993
Crossover Rate: 0.70
Mutation Rate: 0.40
Total Time (seconds): 3.84
Final Best Fitness: 0

Best Fitness per Generation:
Generation 1: 240960
Generation 2: 240960
Generation 3: 240960
Generation 4: 240960
Generation 5: 240960
```

Figure 4.3.1 Statistics of GA Experiment

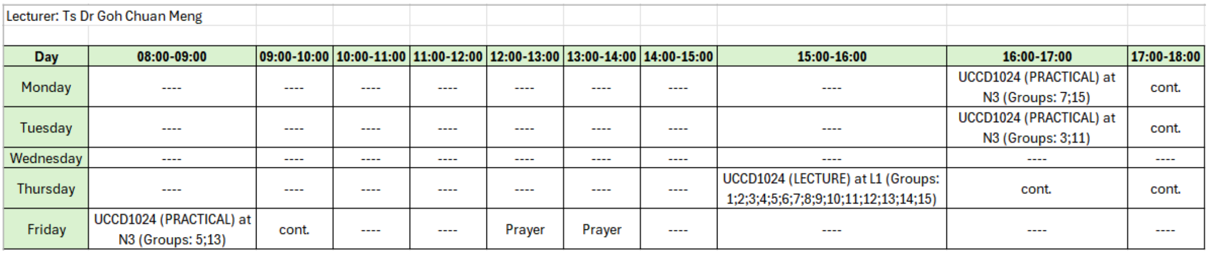
Other than that, multiple timetables are also generated as comma-separated values (CSV) files for each run due to the format's simplicity and broad tool support. Course, lecturer, student, and room timetables are produced. In course timetables (Figure 4.3.2), each class entry shows the class type, lecturer name, room, and enrolled student groups. In lecturer timetables (Figure 4.3.3), each class entry shows the course code, class type, room, and enrolled student groups. In student timetables (Figure 4.3.4), each class entry shows the course code, class type, lecturer name, and room. In room timetables (Figure 4.3.5), each class entry shows the course code, class type, and lecturer name. An overall timetable (Figure 4.3.6) that consolidates courses, lecturers, students, and rooms is also generated. There are some conventions adopted by the generated timetables: subsequent slots of a multi-hour class are marked "cont."; empty slots are marked "----"; on Fridays, slots 5 and 6 (12:00-14:00) are marked "Prayer".



Course: UCCD1024 - DATA STRUCTURE AND ALGORITHMIC PROBLEM SOLVING

Day	08:00-09:00	09:00-10:00	10:00-11:00	11:00-12:00	12:00-13:00	13:00-14:00	14:00-15:00	15:00-16:00	16:00-17:00	17:00-18:00
Monday	----	----	----	----	PRACTICAL - Ts Lai Siew Cheng at N4 (Groups: 4;12)	cont. at N4	----	PRACTICAL - Ts Lai Siew Cheng at N4 (Groups: 8)	PRACTICAL - Ts Dr Goh Chuan Meng at N3 (Groups: 7;15) cont. at N4	cont. at N3
Tuesday	----	----	----	----	----	----	----	----	PRACTICAL - Ts Dr Goh Chuan Meng at N3 (Groups: 3;11)	cont. at N3
Wednesday	PRACTICAL - Ts Lai Siew Cheng at N3 (Groups: 1;9)	cont. at N3	----	----	----	----	----	----	----	----
Thursday	----	----	----	----	----	----	----	LECTURE - Ts Dr Goh Chuan Meng at L1 (Groups: 1;2;3;4;5;6;7;8;9;10;11;12;13;14;15)	cont. at L1	cont. at L1
Friday	PRACTICAL - Ts Dr Goh Chuan Meng at N3 (Groups: 5;13) PRACTICAL - Ts Lai Siew Cheng at N4 (Groups: 2;10)	cont. at N3 cont. at N4	----	----	Prayer	Prayer	----	PRACTICAL - Ts Lai Siew Cheng at N4 (Groups: 6;14)	cont. at N4	----

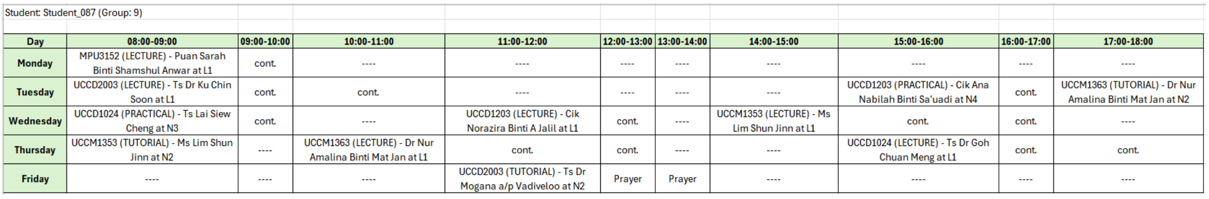
Figure 4.3.2 Timetable of Course UCCD1024



Lecturer: Ts Dr Goh Chuan Meng

Day	08:00-09:00	09:00-10:00	10:00-11:00	11:00-12:00	12:00-13:00	13:00-14:00	14:00-15:00	15:00-16:00	16:00-17:00	17:00-18:00
Monday	----	----	----	----	----	----	----	----	UCCD1024 (PRACTICAL) at N3 (Groups: 7;15)	cont.
Tuesday	----	----	----	----	----	----	----	----	UCCD1024 (PRACTICAL) at N3 (Groups: 3;11)	cont.
Wednesday	----	----	----	----	----	----	----	----	----	----
Thursday	----	----	----	----	----	----	----	UCCD1024 (LECTURE) at L1 (Groups: 1;2;3;4;5;6;7;8;9;10;11;12;13;14;15)	cont.	cont.
Friday	UCCD1024 (PRACTICAL) at N3 (Groups: 5;13)	cont.	----	----	Prayer	Prayer	----	----	----	----

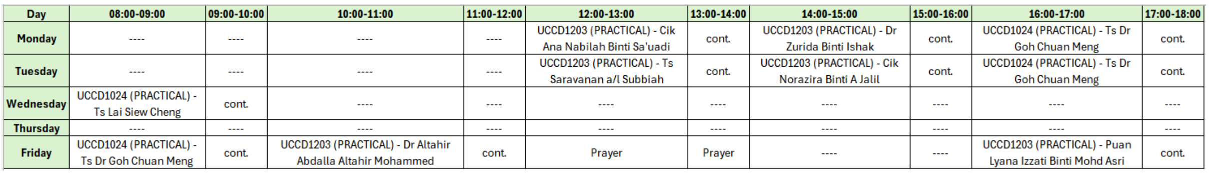
Figure 4.3.3 Timetable of Lecturer Ts Dr Goh Chuan Meng



Student: Student_087 (Group: 9)

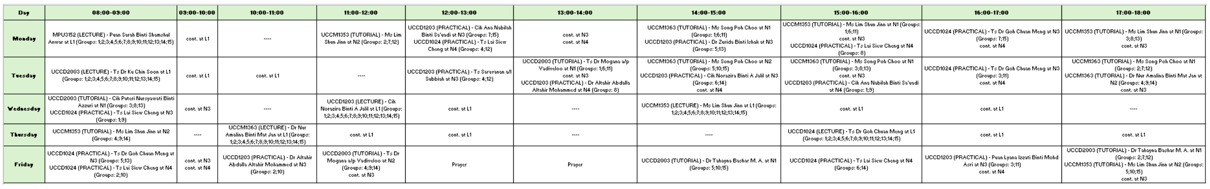
Day	08:00-09:00	09:00-10:00	10:00-11:00	11:00-12:00	12:00-13:00	13:00-14:00	14:00-15:00	15:00-16:00	16:00-17:00	17:00-18:00
Monday	MPU3152 (LECTURE) - Puan Sarah Binti Shamshul Anwar at L1	cont.	----	----	----	----	----	----	----	----
Tuesday	UCCD2003 (LECTURE) - Ts Dr Ku Chin Soon at L1	cont.	cont.	----	----	----	----	UCCD1203 (PRACTICAL) - Cik Ana Nabilah Binti Sa'uadi at N4	cont.	UCCM1363 (TUTORIAL) - Dr Nur Amalina Binti Mat Jan at N2
Wednesday	UCCD1024 (PRACTICAL) - Ts Lai Siew Cheng at N3	cont.	----	UCCD1203 (LECTURE) - Cik Norazira Binti A Jalil at L1	cont.	----	UCCM1353 (LECTURE) - Ms Lim Shun Jinn at L1	cont.	cont.	----
Thursday	UCCM1353 (TUTORIAL) - Ms Lim Shun Jinn at N2	----	UCCM1363 (LECTURE) - Dr Nur Amalina Binti Mat Jan at L1	cont.	cont.	----	----	UCCD1024 (LECTURE) - Ts Dr Goh Chuan Meng at L1	cont.	cont.
Friday	----	----	----	UCCD2003 (TUTORIAL) - Ts Dr Mogana a/p Vadiveloo at N2	Prayer	Prayer	----	----	----	----

Figure 4.3.4 Timetable of Student Student_087 from Group 9



Day	08:00-09:00	09:00-10:00	10:00-11:00	11:00-12:00	12:00-13:00	13:00-14:00	14:00-15:00	15:00-16:00	16:00-17:00	17:00-18:00
Monday	----	----	----	----	UCCD1203 (PRACTICAL) - Cik Ana Nabilah Binti Sa'uadi	cont.	UCCD1203 (PRACTICAL) - Dr Zurida Binti Ishak	cont.	UCCD1024 (PRACTICAL) - Ts Dr Goh Chuan Meng	cont.
Tuesday	----	----	----	----	UCCD1203 (PRACTICAL) - Ts Saravanan a/l Subbiah	cont.	UCCD1203 (PRACTICAL) - Cik Norazira Binti A Jalil	cont.	UCCD1024 (PRACTICAL) - Ts Dr Goh Chuan Meng	cont.
Wednesday	UCCD1024 (PRACTICAL) - Ts Lai Siew Cheng	cont.	----	----	----	----	----	----	----	----
Thursday	----	----	----	----	----	----	----	----	----	----
Friday	UCCD1024 (PRACTICAL) - Ts Dr Goh Chuan Meng	cont.	UCCD1203 (PRACTICAL) - Dr Altahir Abdalla Altahir Mohammed	cont.	Prayer	Prayer	----	----	UCCD1203 (PRACTICAL) - Puan Lyana Izzati Binti Mohd Asri	cont.

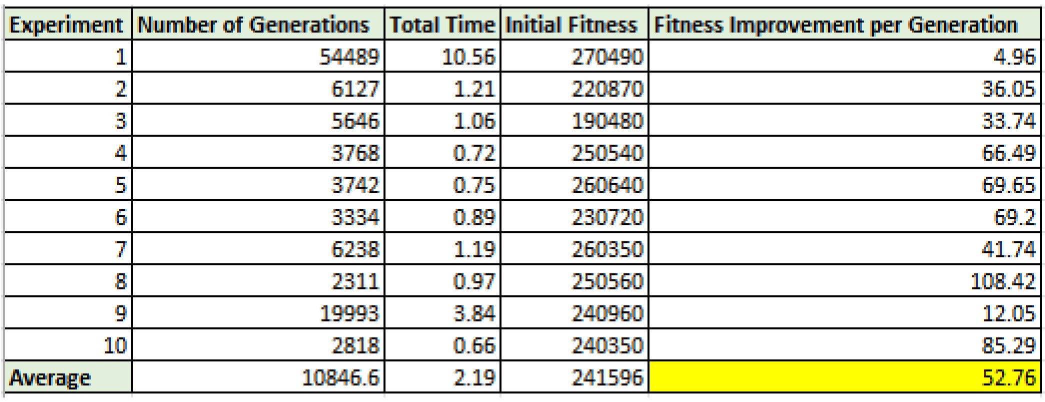
Figure 4.3.5 Timetable of Room N3



Day	08:00-09:00	09:00-10:00	10:00-11:00	11:00-12:00	12:00-13:00	13:00-14:00	14:00-15:00	15:00-16:00	16:00-17:00	17:00-18:00
Monday	MPU3152 (LECTURE) - Puan Sarah Binti Shamshul Anwar at L1 (Groups: 1;2;3;4;5;6;7;8;9;10;11;12;13;14;15)	cont. at L1	----	UCCM1353 (TUTORIAL) - Ms Lim Shun Jinn at N2 (Groups: 2;7;12)	UCCD1203 (PRACTICAL) - Cik Ana Nabilah Binti Sa'uadi at N3 (Groups: 7;15) UCCD1024 (PRACTICAL) - Ts Lai Siew Cheng at N4 (Groups: 4;12)	cont. at N3 cont. at N4	UCCM1363 (TUTORIAL) - Ms Song Poh Choo at N1 (Groups: 1;6;11) UCCD1203 (PRACTICAL) - Dr Zurida Binti Ishak at N3 (Groups: 5;13)	UCCM1353 (TUTORIAL) - Ms Lim Shun Jinn at N1 (Groups: 1;6;11) cont. at N3 UCCD1024 (PRACTICAL) - Ts Lai Siew Cheng at N4 (Groups: 8)	UCCD1024 (PRACTICAL) - Ts Dr Goh Chuan Meng at N3 (Groups: 7;15) cont. at N4	UCCM1353 (TUTORIAL) - Ms Lim Shun Jinn at N1 (Groups: 3;8;13) cont. at N3
Tuesday	UCCD2003 (LECTURE) - Ts Dr Ku Chin Soon at L1 (Groups: 1;2;3;4;5;6;7;8;9;10;11;12;13;14;15)	cont. at L1	cont. at L1	----	UCCD1203 (PRACTICAL) - Ts Saravanan a/l Subbiah at N3 (Groups: 4;12)	UCCD2003 (TUTORIAL) - Ts Dr Mogana a/p Vadiveloo at N1 (Groups: 1;6;11) cont. at N3 UCCD1203 (PRACTICAL) - Dr Altahir Abdalla Altahir Mohammed at N4 (Groups: 8)	UCCM1363 (TUTORIAL) - Ms Song Poh Choo at N2 (Groups: 5;10;15) UCCD1203 (PRACTICAL) - Cik Norazira Binti A Jalil at N3 (Groups: 6;14) cont. at N4	UCCM1363 (TUTORIAL) - Ms Song Poh Choo at N1 (Groups: 3;8;13) cont. at N3 UCCD1203 (PRACTICAL) - Cik Ana Nabilah Binti Sa'uadi at N4 (Groups: 1;9)	UCCD1024 (PRACTICAL) - Ts Dr Goh Chuan Meng at N3 (Groups: 3;11) cont. at N4	UCCM1363 (TUTORIAL) - Ms Song Poh Choo at N1 (Groups: 2;7;12) UCCM1363 (TUTORIAL) - Dr Nur Amalina Binti Mat Jan at N2 (Groups: 4;9;14) cont. at N3
Wednesday	UCCD2003 (TUTORIAL) - Cik Puteri Nursyawati Binti Azzuri at N1 (Groups: 3;8;13) UCCD1024 (PRACTICAL) - Ts Lai Siew Cheng at N3 (Groups: 1;9)	cont. at N3	----	UCCD1203 (LECTURE) - Cik Norazira Binti A Jalil at L1 (Groups: 1;2;3;4;5;6;7;8;9;10;11;12;13;14;15)	cont. at L1	----	UCCM1353 (LECTURE) - Ms Lim Shun Jinn at L1 (Groups: 1;2;3;4;5;6;7;8;9;10;11;12;13;14;15)	cont. at L1	cont. at L1	----
Thursday	UCCM1353 (TUTORIAL) - Ms Lim Shun Jinn at N2 (Groups: 4;9;14)	----	UCCM1363 (LECTURE) - Dr Nur Amalina Binti Mat Jan at L1 (Groups: 1;2;3;4;5;6;7;8;9;10;11;12;13;14;15)	cont. at L1	cont. at L1	----	----	UCCD1024 (LECTURE) - Ts Dr Goh Chuan Meng at L1 (Groups: 1;2;3;4;5;6;7;8;9;10;11;12;13;14;15)	cont. at L1	cont. at L1
Friday	UCCD1024 (PRACTICAL) - Ts Dr Goh Chuan Meng at N3 (Groups: 5;13) UCCD1024 (PRACTICAL) - Ts Lai Siew Cheng at N4 (Groups: 2;10)	cont. at N3 cont. at N4	UCCD1203 (PRACTICAL) - Dr Altahir Abdalla Altahir Mohammed at N3 (Groups: 2;10)	UCCD2003 (TUTORIAL) - Ts Dr Mogana a/p Vadiveloo at N2 (Groups: 4;9;14) cont. at N3	Prayer	Prayer	UCCD2003 (TUTORIAL) - Dr Tahayna Bashar M. A. at N1 (Groups: 5;10;15)	UCCD1024 (PRACTICAL) - Ts Lai Siew Cheng at N4 (Groups: 6;14)	UCCD1203 (PRACTICAL) - Puan Lyana Izzati Binti Mohd Asri at N3 (Groups: 3;11) cont. at N4	UCCD2003 (TUTORIAL) - Dr Tahayna Bashar M. A. at N1 (Groups: 2;7;12) UCCM1353 (TUTORIAL) - Ms Lim Shun Jinn at N2 (Groups: 5;10;15) cont. at N3

Figure 4.3.6 Overall Timetable

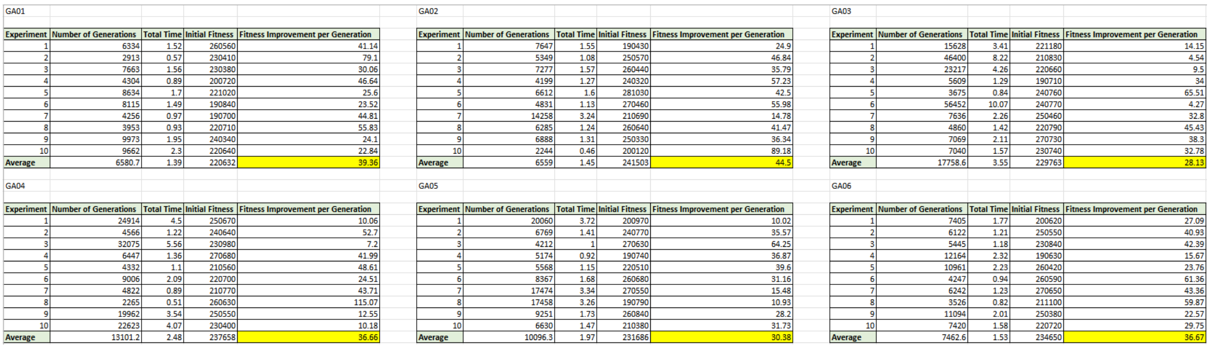
For each operator combination, after 10 experiments, a summary table is created showing the per-run statistics and their averages. A new metric proposed in this project, fitness improvement per generation, is also reported. It is calculated by dividing the initial penalty cost by the number of generations taken to achieve zero penalty cost. This provides a fairer basis for comparison because the GA initialises chromosomes randomly, leading to different initial fitness values. The metric mitigates this bias by emphasising per-generation progress rather than absolute fitness across an entire run. Figure 4.3.7 shows the statistics from 10 GA experiments for a single combination.



Experiment	Number of Generations	Total Time	Initial Fitness	Fitness Improvement per Generation
1	54489	10.56	270490	4.96
2	6127	1.21	220870	36.05
3	5646	1.06	190480	33.74
4	3768	0.72	250540	66.49
5	3742	0.75	260640	69.65
6	3334	0.89	230720	69.2
7	6238	1.19	260350	41.74
8	2311	0.97	250560	108.42
9	19993	3.84	240960	12.05
10	2818	0.66	240350	85.29
Average	10846.6	2.19	241596	52.76

Figure 4.3.7 Statistics of 10 GA Experiments Per Combination

After all experiments (64 × 10) are completed, the results are summarised for comparison across combinations. The combinations with the lowest and highest average fitness improvement per generation are identified as the worst and best, respectively. Figure 4.3.8 presents the detailed statistics for each combination; Figure 4.3.9 shows the overall summary; Figure 4.3.10 highlights the worst-performing combination.



GA01

Experiment	Number of Generations	Total Time	Initial Fitness	Fitness Improvement per Generation
1	6334	1.52	260560	41.14
2	2913	0.57	230410	79.1
3	7663	1.56	230380	30.06
4	4304	0.89	200720	46.64
5	8634	1.7	221020	25.6
6	8115	1.49	190840	23.52
7	4256	0.97	190700	44.81
8	3953	0.93	220710	55.83
9	9973	1.95	240340	24.1
10	9662	2.3	220640	22.84
Average	6580.7	1.39	220632	39.36

GA02

Experiment	Number of Generations	Total Time	Initial Fitness	Fitness Improvement per Generation
1	7647	1.55	190430	24.9
2	5349	1.08	250570	46.84
3	7277	1.57	260440	35.79
4	4199	1.27	240320	57.23
5	6612	1.6	281030	42.5
6	4831	1.13	270460	55.98
7	14258	3.24	210690	14.78
8	6285	1.24	260640	41.47
9	6888	1.31	250330	36.34
10	2244	0.46	200120	89.18
Average	6559	1.45	241503	44.5

GA03

Experiment	Number of Generations	Total Time	Initial Fitness	Fitness Improvement per Generation
1	15628	3.41	221180	14.15
2	46400	8.22	210830	4.54
3	23217	4.26	220660	9.5
4	5609	1.29	190710	34
5	3675	0.84	240760	65.51
6	56452	10.07	240770	4.27
7	7636	2.26	250460	32.8
8	4860	1.42	220790	45.43
9	7069	2.11	270730	38.3
10	7040	1.57	230740	32.78
Average	17758.6	3.55	229763	28.13

GA04

Experiment	Number of Generations	Total Time	Initial Fitness	Fitness Improvement per Generation
1	24914	4.5	250670	10.06
2	4566	1.22	240640	52.7
3	32075	5.56	230980	7.2
4	6447	1.36	270680	41.99
5	4332	1.1	210560	48.61
6	9006	2.09	220700	24.51
7	4822	0.89	210770	43.71
8	2265	0.51	260630	115.07
9	19962	3.54	250550	12.55
10	22623	4.07	230400	10.18
Average	13101.2	2.48	237658	36.66

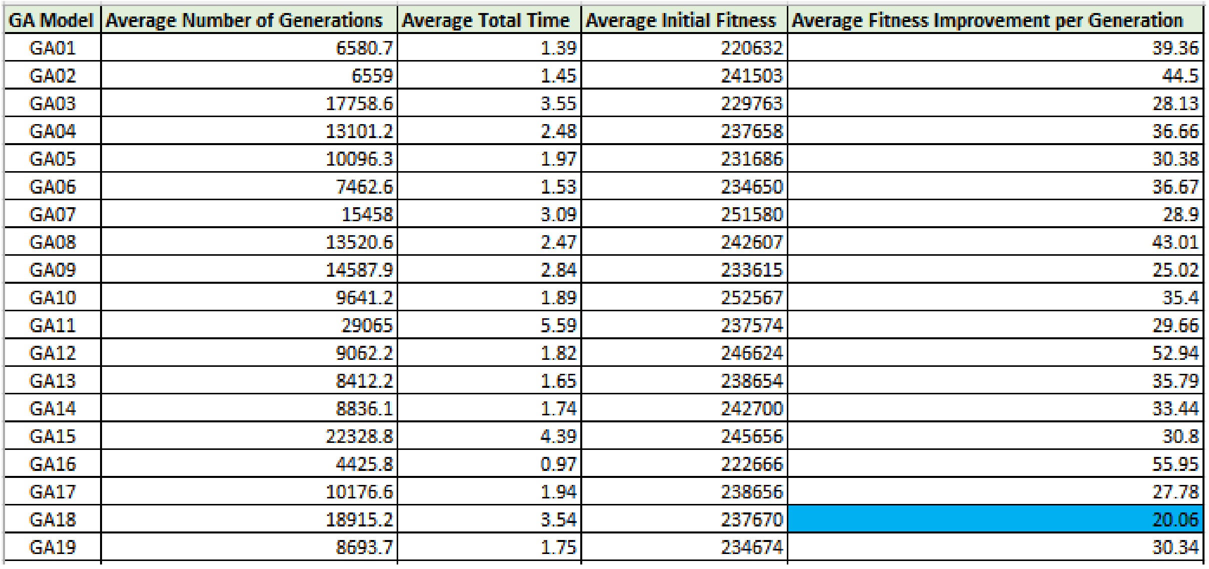
GA05

Experiment	Number of Generations	Total Time	Initial Fitness	Fitness Improvement per Generation
1	20060	3.72	200970	10.02
2	6769	1.41	240770	35.57
3	4212	1	270630	64.25
4	5174	0.92	190740	36.87
5	5568	1.15	220510	39.6
6	8367	1.68	260680	31.16
7	17474	3.34	270550	15.48
8	17458	3.26	190790	10.93
9	9251	1.73	260840	28.2
10	6630	1.47	210380	31.73
Average	10096.3	1.97	231686	30.38

GA06

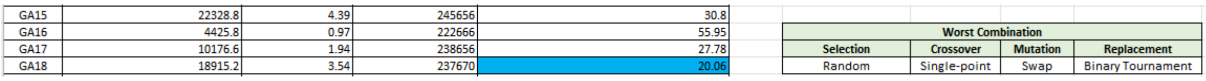
Experiment	Number of Generations	Total Time	Initial Fitness	Fitness Improvement per Generation
1	7405	1.77	200620	27.09
2	6122	1.21	250550	40.93
3	5445	1.18	230840	42.39
4	12164	2.32	190630	15.67
5	10961	2.23	260420	23.76
6	4247	0.94	260590	61.36
7	6242	1.23	270650	43.36
8	3526	0.82	211100	59.87
9	11094	2.01	250380	22.57
10	7420	1.58	220720	29.75
Average	7462.6	1.53	234650	36.67

Figure 4.3.8 Statistical Details of Each GA Combination



GA Model	Average Number of Generations	Average Total Time	Average Initial Fitness	Average Fitness Improvement per Generation
GA01	6580.7	1.39	220632	39.36
GA02	6559	1.45	241503	44.5
GA03	17758.6	3.55	229763	28.13
GA04	13101.2	2.48	237658	36.66
GA05	10096.3	1.97	231686	30.38
GA06	7462.6	1.53	234650	36.67
GA07	15458	3.09	251580	28.9
GA08	13520.6	2.47	242607	43.01
GA09	14587.9	2.84	233615	25.02
GA10	9641.2	1.89	252567	35.4
GA11	29065	5.59	237574	29.66
GA12	9062.2	1.82	246624	52.94
GA13	8412.2	1.65	238654	35.79
GA14	8836.1	1.74	242700	33.44
GA15	22328.8	4.39	245656	30.8
GA16	4425.8	0.97	222666	55.95
GA17	10176.6	1.94	238656	27.78
GA18	18915.2	3.54	237670	20.06
GA19	8693.7	1.75	234674	30.34

Figure 4.3.9 Statistical Summary of All GA Combinations

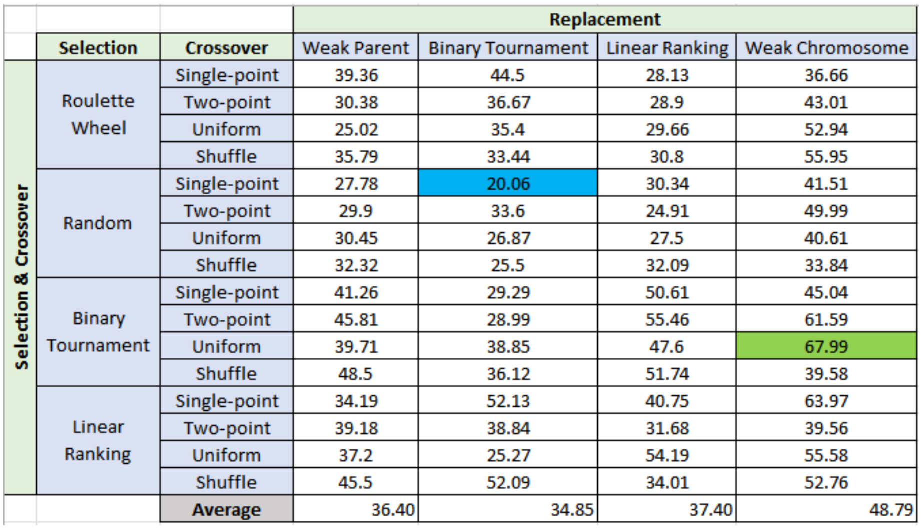


GA15	22328.8	4.39	245656	30.8
GA16	4425.8	0.97	222666	55.95
GA17	10176.6	1.94	238656	27.78
GA18	18915.2	3.54	237670	20.06

Worst Combination			
Selection	Crossover	Mutation	Replacement
Random	Single-point	Swap	Binary Tournament

Figure 4.3.10 Worst GA Combination

Operator-level comparison tables are also produced to enable like-for-like comparisons without confounding from other operators. For example, Figure 4.3.11 compares replacement operators.



			Replacement			
	Selection	Crossover	Weak Parent	Binary Tournament	Linear Ranking	Weak Chromosome
Selection & Crossover	Roulette Wheel	Single-point	39.36	44.5	28.13	36.66
		Two-point	30.38	36.67	28.9	43.01
		Uniform	25.02	35.4	29.66	52.94
		Shuffle	35.79	33.44	30.8	55.95
	Random	Single-point	27.78	20.06	30.34	41.51
		Two-point	29.9	33.6	24.91	49.99
		Uniform	30.45	26.87	27.5	40.61
		Shuffle	32.32	25.5	32.09	33.84
	Binary Tournament	Single-point	41.26	29.29	50.61	45.04
		Two-point	45.81	28.99	55.46	61.59
		Uniform	39.71	38.85	47.6	67.99
		Shuffle	48.5	36.12	51.74	39.58
	Linear Ranking	Single-point	34.19	52.13	40.75	63.97
		Two-point	39.18	38.84	31.68	39.56
		Uniform	37.2	25.27	54.19	55.58
		Shuffle	45.5	52.09	34.01	52.76
		Average	36.40	34.85	37.40	48.79

Figure 4.3.11 Comparison Table for Replacement Operator

4.4 Genetic Algorithm Design

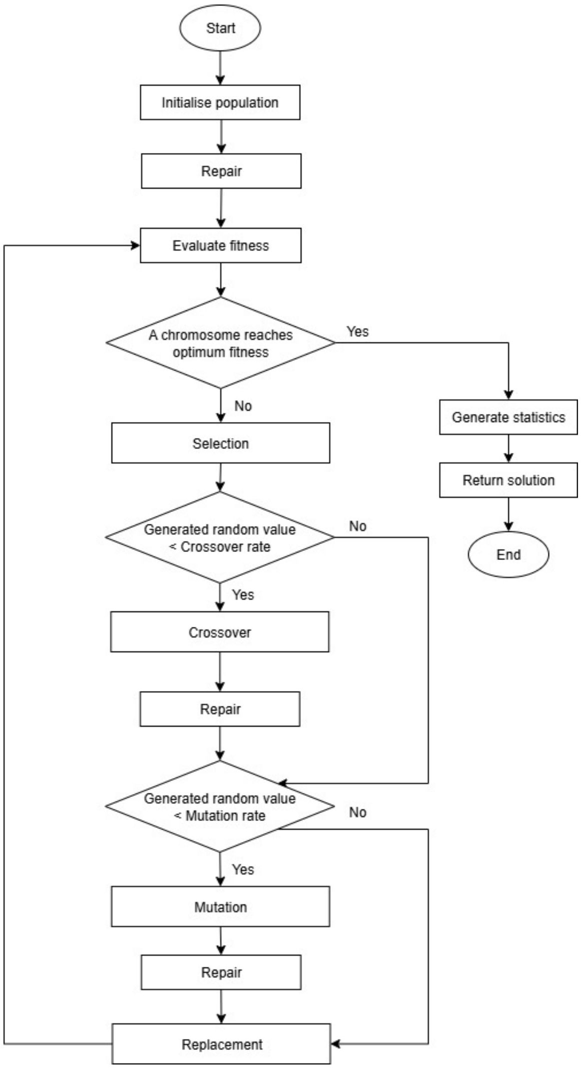
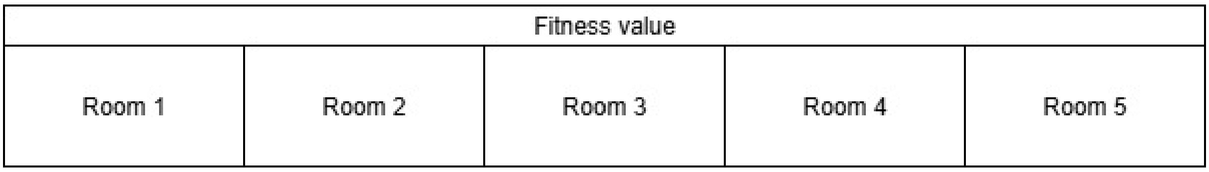


Figure 4.4.1 Flowchart of GA for University Course Timetabling System

Figure 4.4.1 outlines the genetic algorithm (GA) used by the university course timetabling system in this project. After setting key parameters (population size, crossover rate, and mutation rates), the algorithm creates a random population and immediately repairs each chromosome to remove duplicate or missing classes and illegal placements (cross-day segments and protected Friday prayer slots). A penalty-based fitness function then scores every timetable by counting hard and soft constraint violations; this includes the proposed soft constraint that penalises consecutive classes for a student that occur in different buildings to discourage inter-building moves. If a chromosome reaches the target fitness (zero cost), the run ends, experiment statistics are written, and the best timetable is returned. Otherwise, two parents are chosen, crossover and mutation are applied according to their probabilities, and each offspring is repaired before evaluation. A replacement policy then inserts the offspring into the population, typically displacing weaker chromosomes, and the loop repeats from fitness evaluation until the stopping condition is met.

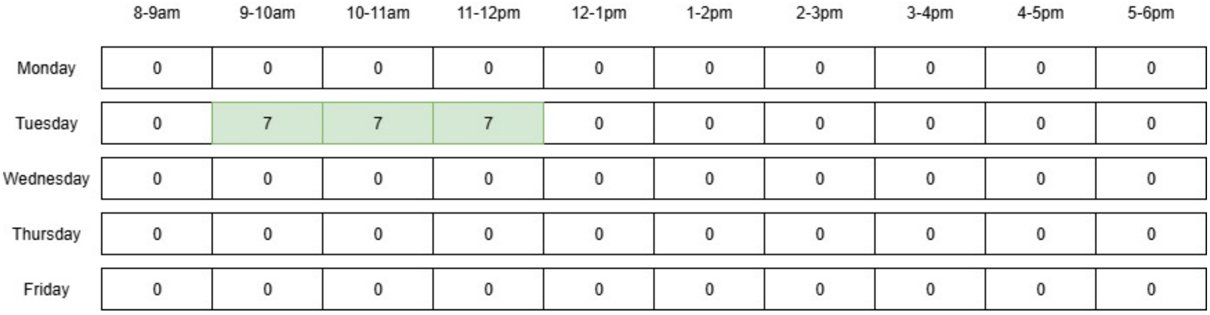
4.4.1 Chromosome Encoding

Each chromosome encodes a complete weekly timetable as a fixed-length integer array of 250 genes (5 rooms × 5 days × 10 slots), with a small header that stores the current penalty score. Gene values are class IDs (see Section 4.2): a value greater than 0 refer to specific lecture, tutorial, or practical instance, while 0 denotes an empty time slot. Multi-hour classes are represented by repeating the same class ID across consecutive slots equal to the class duration. For example, a three-hour class with ID 7 appears as [7, 7, 7]. Physically, the array is laid out in room-major order, then day-major within each room: the 50 daily slots of Room 1 (5 days × 10 slots) are followed by the 50 slots of Room 2, and so on to Room 5 (Figure 4.4.2); within each room, the 10 slots for Monday come before the 10 for Tuesday through to Friday. This layout supports constant-time edits per slot, fast checks for empty segments and contiguous blocks, and a clear, reproducible mapping between timetable semantics and array indices. Figure 4.4.3 illustrates the gene-level view of a room within the chromosome, including a three-hour class with ID 7 and empty slots.



Fitness value				
Room 1	Room 2	Room 3	Room 4	Room 5

Figure 4.4.2 Overview of A Chromosome



	8-9am	9-10am	10-11am	11-12pm	12-1pm	1-2pm	2-3pm	3-4pm	4-5pm	5-6pm
Monday	0	0	0	0	0	0	0	0	0	0
Tuesday	0	7	7	7	0	0	0	0	0	0
Wednesday	0	0	0	0	0	0	0	0	0	0
Thursday	0	0	0	0	0	0	0	0	0	0
Friday	0	0	0	0	0	0	0	0	0	0

Figure 4.4.3 Gene-Level View of A Room in A Chromosome

4.4.2 Population Initialisation

The initial population is generated with a greedy-random strategy to avoid early infeasibility and reduce timetable fragmentation. For each chromosome, an empty 250-gene array is created, and its header initialised. Classes are then ordered by its difficulty to place, with longer-duration classes treated as higher difficulty because they require more consecutive slots. The algorithm scans compatible rooms and days in index order to find a contiguous block long enough for the current class, then writes the class ID across those slots. This longest-first placement prevents classic fragmentation (for instance, two isolated one-slot gaps blocking a two-hour class) and leaves well-shaped space for later items. After longer classes are placed, shorter classes fill the remaining gaps. To maintain diversity, randomisation is injected at several points: when multiple feasible placements exist, one is chosen at random, and the room is randomly selected from the compatible set. Finally, each chromosome undergoes the standard repair process to address any residual overlaps or missing segments before fitness evaluation.

4.4.3 Fitness Evaluation

The fitness of a chromosome is indicated by its total penalty cost, where lower values signal better timetables. Penalties are accumulated per violation using fixed weights: hard-constraint breaches incur 10000 each, standard soft-constraint breaches incur 10, and the building-continuity constraint is weighted 20 to emphasise its importance in this project. Hard checks include clashes for students and lecturers, room-type compatibility, room capacity, protection of Friday prayer periods, missing or duplicated classes, and cross-day segment violations. Soft checks include excessive consecutive hours for students and lecturers, plus the building-continuity rule. Many of these errors are minimised by the chromosome encoding, greedy initialisation, and repair routines in the GA loop.

4.4.4 Repair

After each chromosome is built, crossed over, or mutated, a dedicated repair process restores feasibility and compacts the timetable so that later operators do not accumulate errors.

The repair process consists of three steps. First, it scans all 250 genes to count how many slots each class currently occupies. It then compares these counts with the required durations. If the count is not equals to the duration, this indicates that the class has missing, overfilled, or duplicated segments. All occurrences of the class are cleared, and the class is queued for correct placement. This guarantees that every multi-hour class appears as one contiguous block of its exact length and that no cross-day fragments slip in.

Second, for each queued class, the repair tries to place it using only compatible rooms and currently contiguous free slots. Friday's protected period is treated specially, where the slots 5 and 6 are unavailable. An effective checking on capacity per day caps Friday at six hours, which prevents pathological cases such as two 3-hour and one 2-hour classes appearing capacity-feasible but actually unplaceable across the two segments on Friday.

Third, if a class still unable to fit, the operator escalates through increasingly flexible, local repacking moves that behave like bounded backtracking. It starts with "same room, same day" relocation that clears that day in the room, then re-inserts all classes for that day tightly, block by block, to open a contiguous window for the target class. If unsuccessful, it practices "same room, across days" relocation that repeats the compact-and-reinsert process on another day of the same room until successful or there is no day left. If this approach failed as well, the process leverages "rooms of the same type, across days" strategy as a last resort, which clears and tightly repacks days across multiple rooms of the required type, subject to each day's capacity. Each repack writes classes back-to-back without gaps inside the allowed segments, ensuring the day layout is maximally compact before the next attempt.

Once all queued classes are placed, the chromosome contains no missing or duplicate segments, respects room-type compatibility and protected periods.

In short, the repair operator combines strict consistency checks with constrained backtracking and compact re-packing: if a class cannot fit as-is, it shifts earlier placements locally to make space, limits Friday's effective capacity to six hours to avoid impossible layouts, and packs contiguously so that future operations have the largest feasible blocks to work with.

4.4.5 Selection

4.4.5.1 Roulette Wheel Selection

In the roulette-wheel scheme, the implementation first finds the worst (maximum) penalty in the population and converts each chromosome's penalty into a non-negative weight by using the formula: worst fitness minus own fitness plus one, so that lower penalties yield larger weights. The weights are summed, a random draw is taken in between 0 and the total weight, and the operator scans cumulatively across the population to pick the first chromosome whose running total exceeds the random value. Repeating this procedure twice to return two parents. Because probabilities are proportional to the inverted penalty, every chromosome retains a chance while better timetables are favoured. Figure 4.4.4 shows that the lower the penalty cost, the higher the fitness, and thus there is higher probability of getting chosen as the parent because it spans across more space.

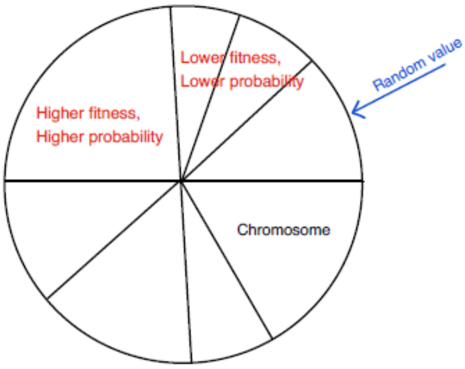


Figure 4.4.4 Roulette Wheel Selection in University Course Timetabling System

4.4.5.2 Random Selection

The random selector ignores fitness entirely. Each parent is drawn uniformly at random from the population, returning two independent picks per mating event. This maximises exploration and genetic diversity, providing a useful baseline and a way to inject variability without changing any other part of the pipeline.

4.4.5.3 Binary Tournament Selection

Binary tournament selection samples two distinct candidates uniformly at random and returns the one with the lower penalty; ties are effectively broken by the sampling order. This process is run twice to obtain two parents. The approach is simple, fast, and scale-free, where moderate selective pressure emerges naturally because mid-ranking individuals can still win when paired against weaker ones. Figure 4.4.5 shows the higher fitness chromosome wins the tournament and getting chosen as the parent.

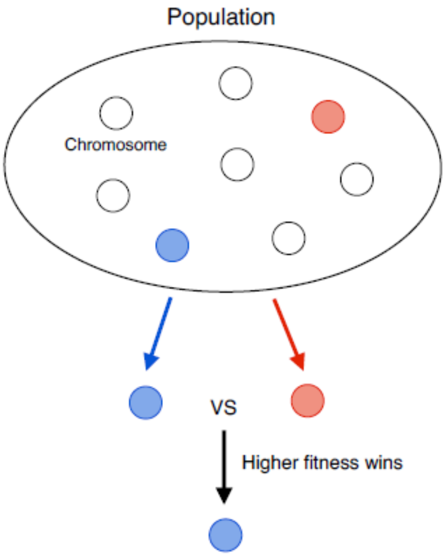


Figure 4.4.5 Binary Tournament Selection in University Course Timetabling System

4.4.5.4 Linear Ranking Selection

Linear ranking selection begins by sorting a copy of the population in ascending order of penalty (best first). The operator assigns a linear rank weight to index i as (N − i), so the best individual has weight N, the next N−1, and so on, with total weight equal to N(N+1)/2, where N represents the population size. To pick a parent, a random draw is taken over this total and a cumulative scan over the sorted list identifies the selected rank; repeating yields the second parent. Because probabilities depend on rank rather than raw penalty gaps like roulette wheel selection, selective pressure is controlled and premature takeover by a single outlier is reduced. Figure 4.4.6 shows the process of linear ranking selection which assigns weight according to rank rather than fitness.

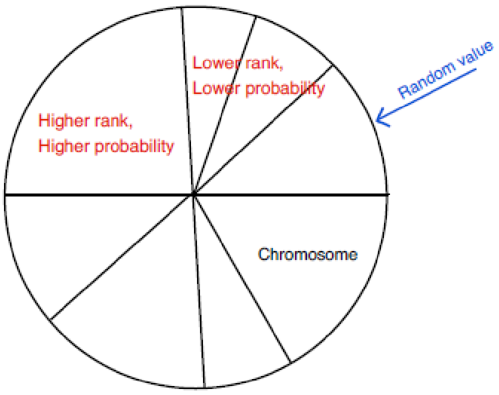


Figure 4.4.5 Linear Ranking Selection in University Course Timetabling System

4.4.6 Crossover

All crossover techniques in this system are room-based. Rooms are not interchangeable because capacity and room type (lecture hall, tutorial room, practical lab) are tied to the room itself, so cutting mid-room can easily produce invalid schedules (for example, a lecture class landing in a lab). Multi-hour classes also span consecutive slots, making it difficult to align identical cut points inside both parents without splitting a class. To avoid these issues, the algorithm performs crossover room by room: for each room, the offspring inherits the entire weekly schedule from one of the parents. This design preserves room-type and capacity semantics by construction and greatly reduces post-crossover repair.

4.4.6.1 Single-Point Crossover

A single classroom index is sampled as the crossover point in the room list (room-major layout). For each child, all classrooms before the point are copied wholesale from Parent 1, and all classrooms from the point onward are copied from Parent 2. The second child uses the complementary assignment. Because each classroom is transferred as a complete unit, no class is fragmented, and room constraints remain intact. Figure 4.4.6 shows the process of single-point crossover in this project.

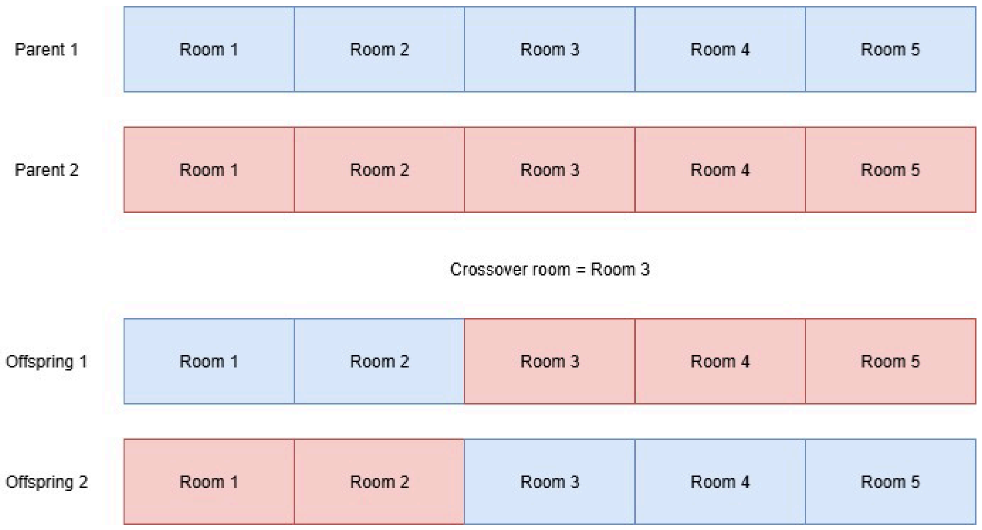


Figure 4.4.6 Single-Point Crossover in University Course Timetabling System

4.4.6.2 Two-Point Crossover

Two room indices are sampled. For each child, the middle block of rooms (from first room index (inclusive) to second room index (exclusive)) is copied from Parent 2, while the outer blocks come from Parent 1. The second child receives the complementary composition. This increases mixing compared with single-point crossover while still respecting classroom boundaries, ensuring that every inherited room timetable remains coherent. Figure 4.4.7 shows the process of two-point crossover in this project.

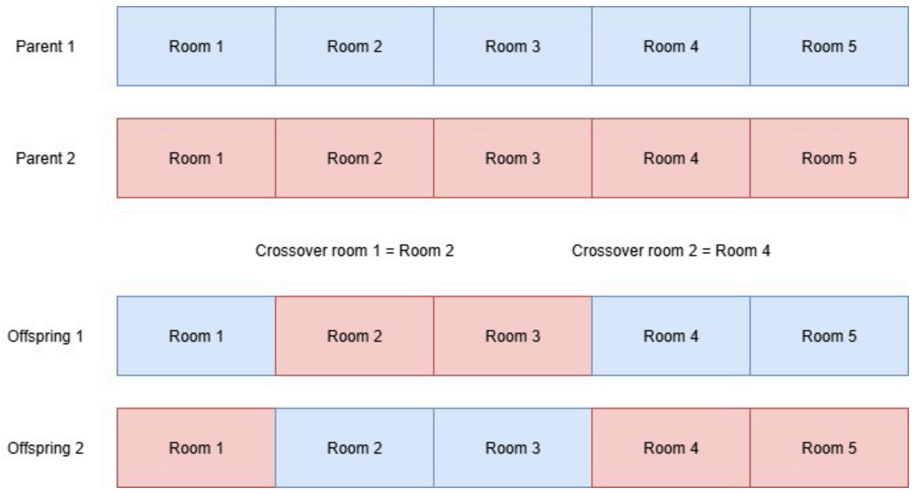


Figure 4.4.7 Two-Point Crossover in University Course Timetabling System

4.4.6.3 Uniform Crossover

Each classroom is treated independently with a Bernoulli draw, in which the possibility of each outcome is 0.5. If the generated random value is 0, the child takes the entire schedule of that classroom from Parent 1; otherwise from Parent 2. This produces fine-grained recombination at the level of rooms, encouraging diversity without risking mid-class splits or room-type mismatches. Figure 4.4.8 shows the process of uniform crossover in this project.

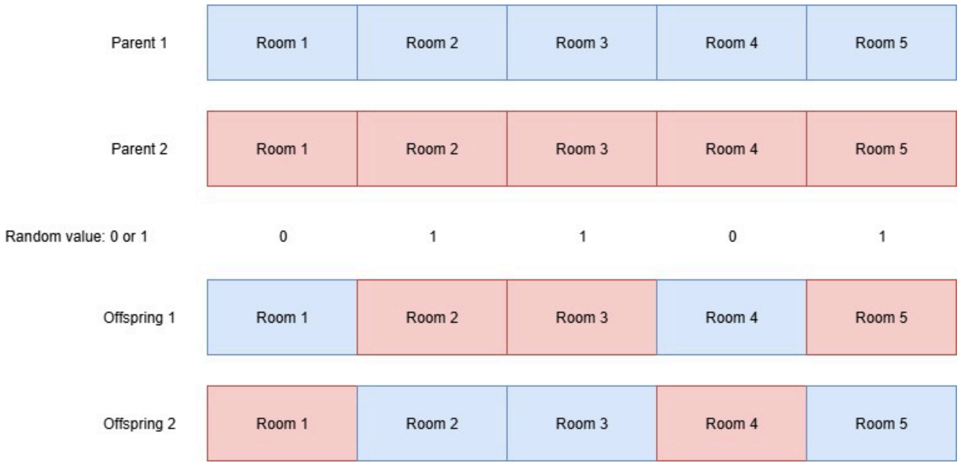


Figure 4.4.8 Uniform Crossover in University Course Timetabling System

4.4.6.4 Shuffle Crossover

Before applying room crossover, the algorithm generates a random permutation of room indices and applies single-point crossover to the list of shuffled indexes. The rooms to be crossed over is then determined from the room indices after the crossover point. Shuffling breaks positional bias among adjacent rooms and promotes better mixing of room clusters, while maintaining the room-by-room guarantee that preserves capacity and type semantics. Figure 4.4.9 shows the process of shuffle crossover in this project.

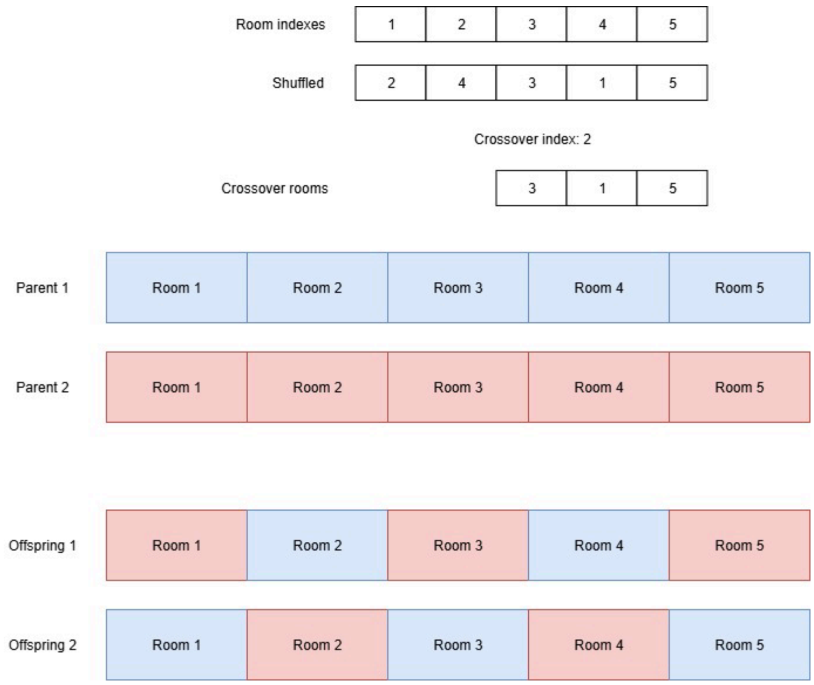


Figure 4.4.9 Shuffle Crossover in University Course Timetabling System

4.4.7 Mutation

After crossover, each offspring is subjected to mutation with a fixed probability. The implementation uses a contiguity-aware swap that moves classes without breaking multi-hour blocks or violating room-type semantics.

The operator first picks a random source index in the chromosome and maps it to (room, day, slot) using the room-major layout. It then restricts potential destinations to rooms of the same type as the source room, ensuring any move cannot place a lecture into a tutorial room or practical lab, or exceed capacity semantics tied to room type. For example, if the chosen gene is at lecture hall, the target gene must be selected from lecture halls only. A random destination index is drawn uniformly over all days and slots across these compatible rooms, which encourages broad exploration. As a result, the class may remain in place, move within the same room, or jump to a different but compatible room at the end of this operation.

If both positions are empty, a trivial swap has no effect. Otherwise, the operator performs a contiguous-block swap: it detects whether either index lies inside a multi-hour class, locates that class's full block, and swaps whole blocks rather than single genes. This preserves class contiguity by construction. The operator also guards against illegal shapes: swaps do not cross day boundaries, and Friday’s protected period (slots 5 and 6) is treated as a hard split, so no block is allowed to straddle it. If a clean block-for-block exchange is not possible (for instance, destination area is partially occupied or the blocks differ in length), the routine performs the safest available partial move by checking slots before and after both contiguous class blocks, hoping to find sufficient spaces to utilise those empty slots for block-to-block exchange. If unsuccessful, it finds the empty blocks in the room without limitation to before and after target class blocks. If no empty block is found, no swap is performed.

Two design choices make this mutation both safe and effective. First, room-type filtering means every mutated placement remains in a capacity- and usage-compatible room family, avoiding a large class of invalid schedules. Second, contiguity awareness means multi-hour classes are never torn apart; mutations explore alternative placements of entire classes instead of introducing fragmentation. After mutation, the repair operator runs to clear up any side effects (duplicates, missing segments, or cross-day violations), and the individual is re-evaluated for fitness.

4.4.8 Replacement

4.4.8.1 Weak Parent Replacement

After producing two offspring from a mating pair, the algorithm builds a candidate pool consisting of both parents and both offspring. It then sorts the candidates according to their fitness. The two best chromosomes out of the four candidates replace the two parents. This approach ensures the child replaces the parent only if the child’s penalty is strictly lower. If both children outperform both parents, both parents are replaced; if neither child improves on its mapped parent, the parents are retained, and the offspring are discarded. This keeps population size constant, preserves strong parental building blocks, and prevents regressions caused by inferior children. Figure 4.2.10 shows that Parent 1 and Offspring 2 (highlighted in green) is the two best chromosomes among the two parent and two offspring, therefore replacing Parent 1 and Parent 2 (highlighted in red) in the population.

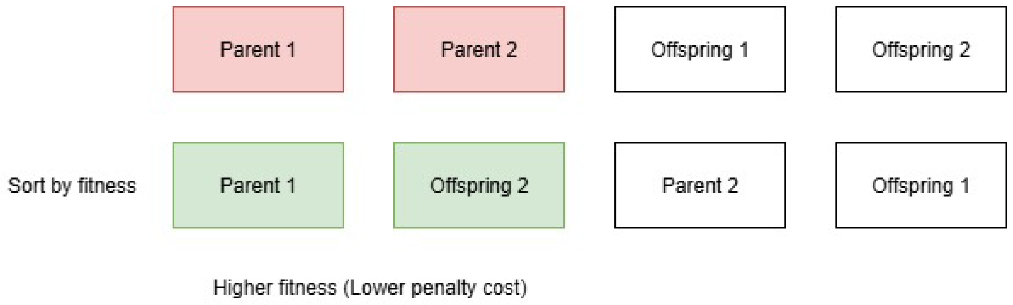


Figure 4.4.10 Weak Parent Replacement in University Course Timetabling System

4.4.8.1 Binary Tournament Replacement

For each offspring, two distinct candidates are sampled uniformly at random from the current population. The candidate with the higher penalty cost, that is the weaker one, is selected as the replacement target. This operation applies the opposite logic as the binary tournament selection, as selection aims to select good chromosomes while replacement aims to replace bad chromosomes. Figure 4.4.11 shows the process of choosing a replacement target from the population in binary tournament replacement.

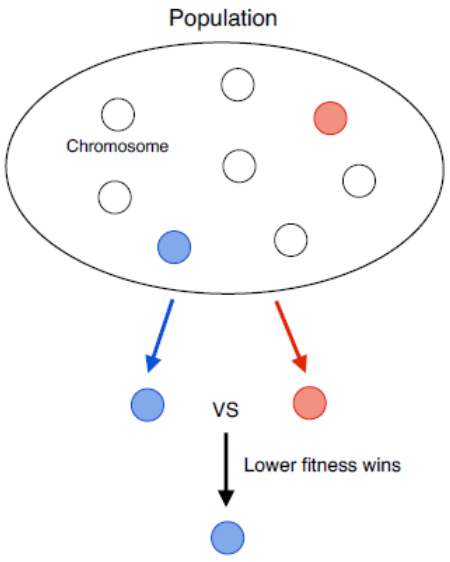


Figure 4.4.11 Replacement Target Selection Process in Binary Tournament Replacement

4.4.8.1 Linear Ranking Replacement

This is a proposed replacement technique inspired by the linear ranking selection strategy. The population is copied and sorted by penalty in descending order (worst first). A linear rank weight is assigned to index i as (N − i), so lower-ranked (worse) chromosomes are more likely to be chosen as replacement targets, while top individuals have the smallest removal probability. For each offspring, a target is drawn according to these rank weights; this decouples replacement from raw penalty gaps, protects elites probabilistically, and avoids premature loss of diversity. This operation applies the opposite logic as the linear ranking selection, as selection aims to select good chromosomes while replacement aims to replace bad chromosomes. Figure 4.4.12 shows the process of choosing a replacement target from the population in linear ranking replacement.

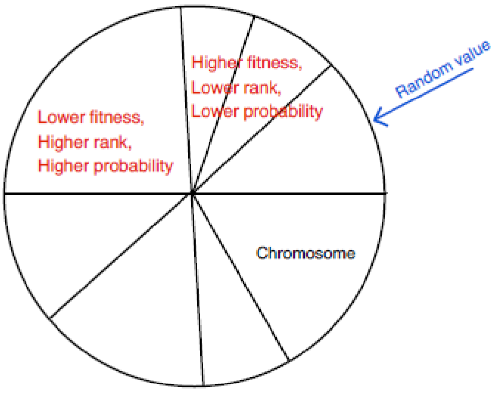


Figure 4.4.12 Replacement Target Selection Process in Liner Ranking Replacement

4.4.8.1 Weak Chromosome Replacement

This is a proposed replacement technique inspired by the weak parent replacement strategy. For each offspring, the algorithm identifies two global worst chromosomes in the population (maximum penalty) and adds it to a candidate pool consisting of both offspring. The candidates are then sorted according to their fitness. The best two chromosomes replace the positions of both parents in the population This policy injects a stronger selective pressure than the weak parent replacement toward continuous improvement while guaranteeing that the current best solutions are never overwritten by weaker individuals. Figure 4.3.13 shows that Worst 1 and Offspring 2 (highlighted in green) is the two best chromosomes among the two worst chromosomes and two offspring, therefore replacing Worst 1 and Worst 2 (highlighted in red) in the population.

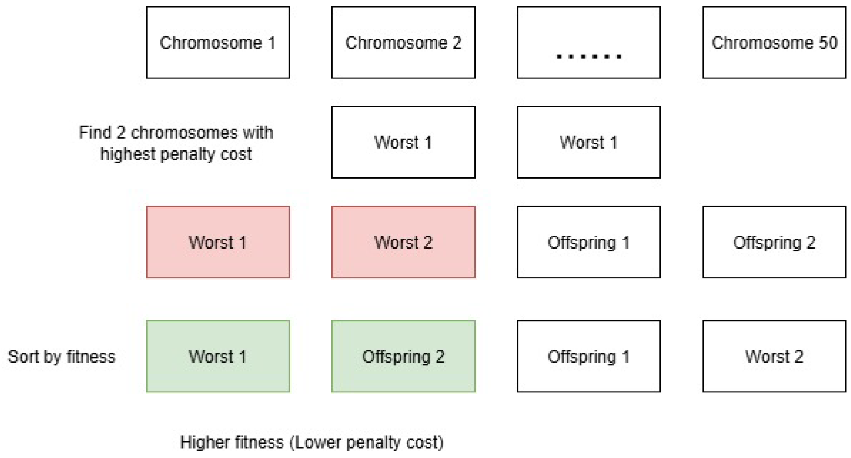


Figure 4.4.13 Weak Chromosome Replacement in University Course Timetabling System

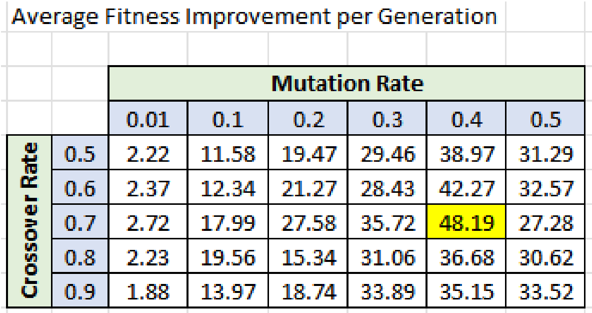
4.4.9 Parameter Settings

No.	Parameter	Value
1	Population Size	100
2	Crossover Probability	0.7
3	Mutation Probability	0.4

Table 4.4.1 Parameter Settings of GA in University Course Timetabling System

The crossover and mutation rate are determined by performing grid search across various parameter combinations. Experiments using combinations of 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 crossover rate and 0.01, 0.1, 0.2, 0.3, 0.4, 0.5 mutation rate are run to identify the best probabilities. The GA combination involved is using roulette wheel selection, single-point crossover, swap mutation, and weak parent replacement. Each parameter variation is run 10 times using the specified GA model. The results are then evaluated using the proposed metric (see Section 4.3).

Figure 4.4.14 shows the experiment results of the parameter testing. It is found that crossover rate at 0.7 and mutation rate at 0.4 performs the best. It can be clearly seen that the performance of GA increases as the mutation rate increases and reaches its peak at 0.4 and slowly falls at 0.5. On the other hand, the performance of GA model reaches its peak when crossover rate is set at 0.7.



Average Fitness Improvement per Generation

		Mutation Rate					
		0.01	0.1	0.2	0.3	0.4	0.5
Crossover Rate	0.5	2.22	11.58	19.47	29.46	38.97	31.29
	0.6	2.37	12.34	21.27	28.43	42.27	32.57
	0.7	2.72	17.99	27.58	35.72	48.19	27.28
	0.8	2.23	19.56	15.34	31.06	36.68	30.62
	0.9	1.88	13.97	18.74	33.89	35.15	33.52

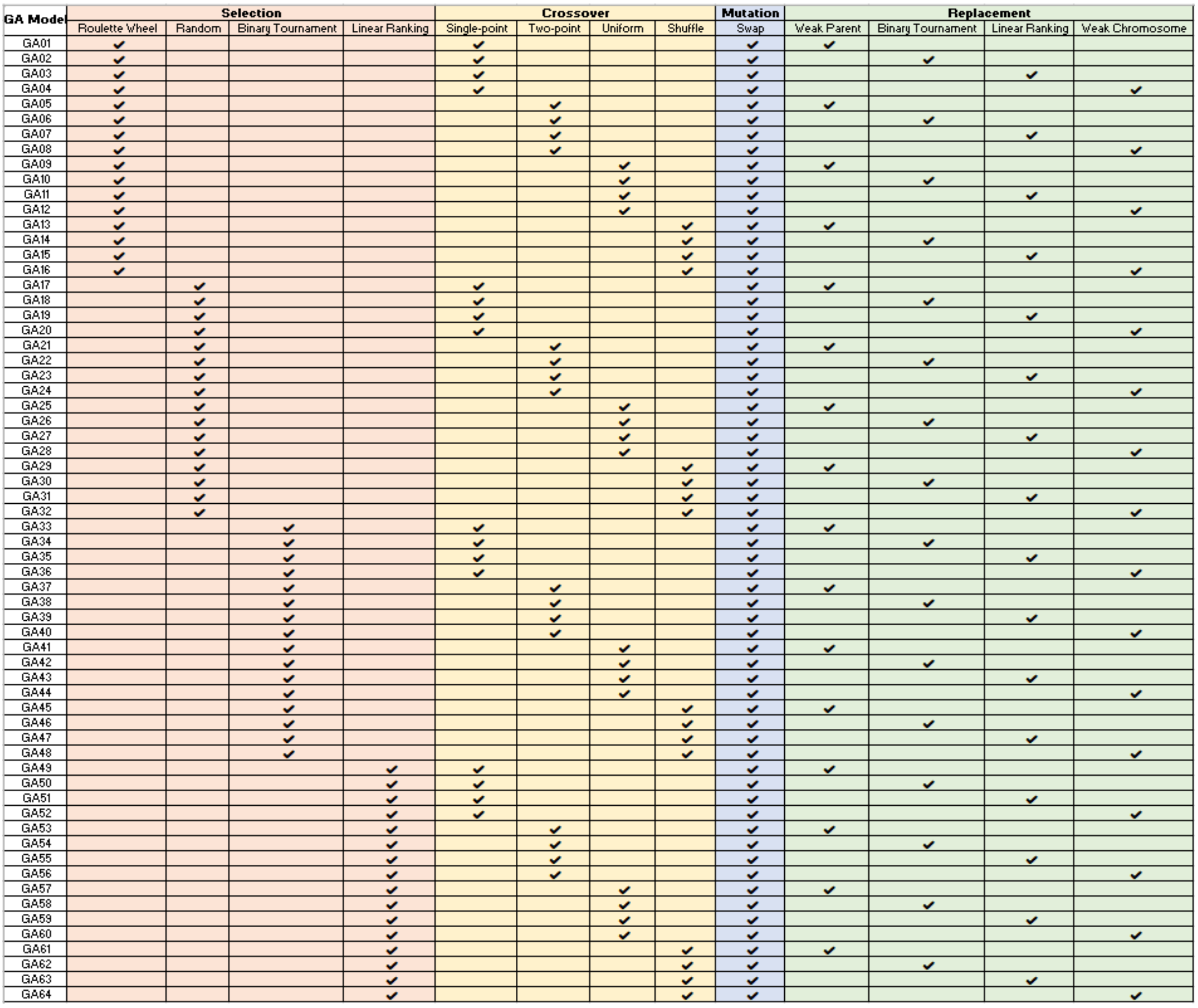
Figure 4.4.14 Experiment Results of Crossover and Mutation Variations

4.4.10 Operator Combinations

Selection	Crossover	Mutation	Replacement
Roulette Wheel	Single-Point	Swap	Weak Parent
Random	Two-Point	-	Binary Tournament
Binary Tournament	Uniform	-	Linear Ranking
Linear Ranking	Shuffle	-	Weak Chromosome

Table 4.4.2 GA Operator Techniques for Project

Table 4.4.2 shows the operator techniques applied in this project, which are 4 selection, 4 crossover, 1 mutation, and 4 replacement techniques. These components allow for the formation of 64 different genetic algorithm (GA) combinations as shown in Figure 4.4.15.



GA Model	Selection				Crossover				Mutation	Replacement			
	Roulette Wheel	Random	Binary Tournament	Linear Ranking	Single-point	Two-point	Uniform	Shuffle	Swap	Weak Parent	Binary Tournament	Linear Ranking	Weak Chromosome
GA01	✓				✓				✓	✓			
GA02	✓				✓				✓		✓		
GA03	✓				✓				✓			✓	
GA04	✓				✓				✓				✓
GA05	✓					✓			✓	✓			
GA06	✓					✓			✓		✓		
GA07	✓					✓			✓			✓	
GA08	✓					✓			✓				✓
GA09	✓						✓		✓	✓			
GA10	✓						✓		✓		✓		
GA11	✓						✓		✓			✓	
GA12	✓						✓		✓				✓
GA13	✓							✓	✓	✓			
GA14	✓							✓	✓		✓		
GA15	✓							✓	✓			✓	
GA16	✓							✓	✓				✓
GA17		✓			✓				✓	✓			
GA18		✓			✓				✓		✓		
GA19		✓			✓				✓			✓	
GA20		✓			✓				✓				✓
GA21		✓				✓			✓	✓			
GA22		✓				✓			✓		✓		
GA23		✓				✓			✓			✓	
GA24		✓				✓			✓				✓
GA25		✓					✓		✓	✓			
GA26		✓					✓		✓		✓		
GA27		✓					✓		✓			✓	
GA28		✓					✓		✓				✓
GA29		✓						✓	✓	✓			
GA30		✓						✓	✓		✓		
GA31		✓						✓	✓			✓	
GA32		✓						✓	✓				✓
GA33			✓		✓				✓	✓			
GA34			✓		✓				✓		✓		
GA35			✓		✓				✓			✓	
GA36			✓		✓				✓				✓
GA37			✓			✓			✓	✓			
GA38			✓			✓			✓		✓		
GA39			✓			✓			✓			✓	
GA40			✓			✓			✓				✓
GA41			✓				✓		✓	✓			
GA42			✓				✓		✓		✓		
GA43			✓				✓		✓			✓	
GA44			✓				✓		✓				✓
GA45			✓					✓	✓	✓			
GA46			✓					✓	✓		✓		
GA47			✓					✓	✓			✓	
GA48			✓					✓	✓				✓
GA49				✓	✓				✓	✓			
GA50				✓	✓				✓		✓		
GA51				✓	✓				✓			✓	
GA52				✓	✓				✓				✓
GA53				✓		✓			✓	✓			
GA54				✓		✓			✓		✓		
GA55				✓		✓			✓			✓	
GA56				✓		✓			✓				✓
GA57				✓			✓		✓	✓			
GA58				✓			✓		✓		✓		
GA59				✓			✓		✓			✓	
GA60				✓			✓		✓				✓
GA61				✓				✓	✓	✓			
GA62				✓				✓	✓		✓		
GA63				✓				✓	✓			✓	
GA64				✓				✓	✓				✓

Figure 4.4.15 GA Combinations

4.5 Data Storage Design

In this project, MySQL serves as the persistent data storage for the university course timetabling system. It is an open-source, production-grade, client-server relational database management system (RDBMS) that supports standard SQL and ACID (atomicity, consistency, isolation, durability) transactions. Its native features such as auto-increment columns, foreign keys, and triggers allow integrity rules to be enforced close to the data, reducing application-side complexity. Furthermore, the database integration with Java, the development programming language for this project, is straightforward using the JDBC API and the MySQL Connector/J database driver. In summary, MySQL combines relational rigour and transactional reliability with a simple Java integration path, making it a solid foundation for research-based projects.

4.5.1 Database Structure

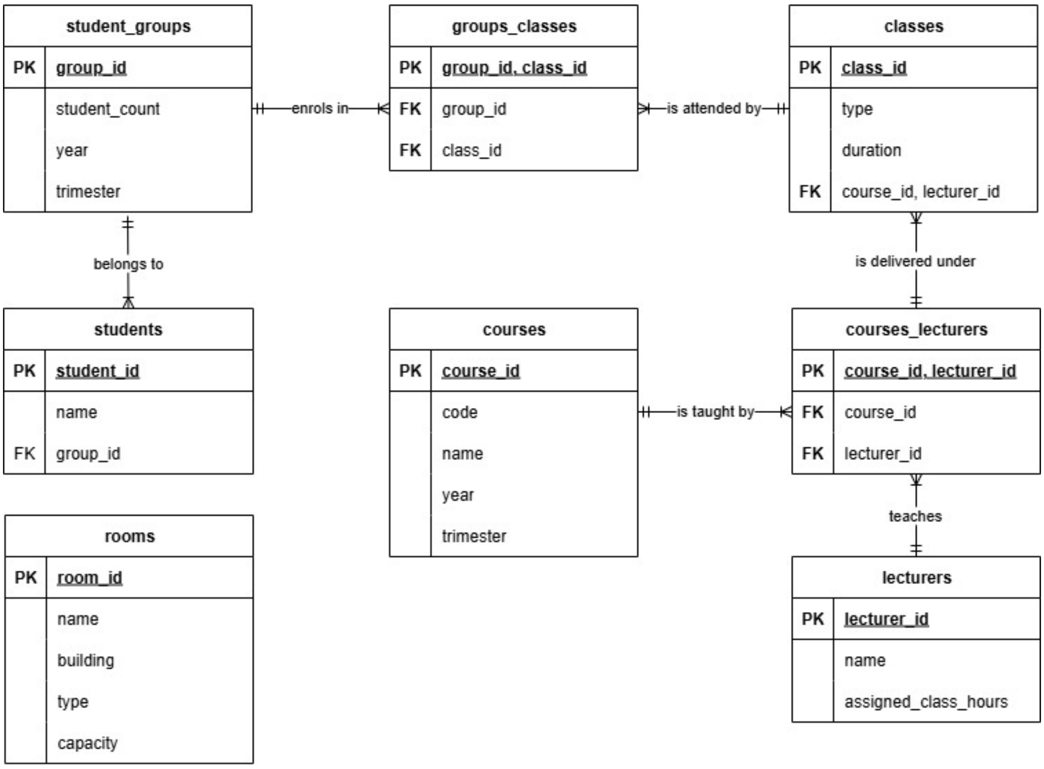


Figure 4.5.1 ERD for University Course Timetabling System

Figure 4.5.1 shows the entity relationship diagram (ERD) for the university timetabling system, comprising 6 core entities, which are Students, Student Groups, Courses, Lecturers, Classes, and Rooms, and the relationships that connect them. Each Student belongs to exactly one Student Group. Student Groups and Classes form a many-to-many relationship via an associative table, where students inherit class enrolments through their group rather than each student being linked to classes individually. Each Course generates one or more Classes, capturing the idea of parallel or repeat offerings. Every Class is taught by exactly one Lecturer, while a Lecturer may teach multiple Classes, creating a one-to-many association. Rooms are modelled independently. This is because class-to-room assignments are produced dynamically by the genetic algorithm (GA) during each run. Each chromosome represents a candidate timetable, therefore persisting these assignments in the database would create excessive, redundant writes and degrade performance. Keeping Rooms as a reference list ensures a consistent, controlled set of available spaces across experiments. Together, these relationships yield a normalised, searchable schema that supports conflict-free allocation of lecturers, students, courses, and rooms.

4.5.3 Table Structure

Students table

Attribute	Data Type	Constraint	Description
ID	INT	Primary key	Unique identifier of student
Name	VARCHAR(100)	Not NULL	Student name
Group ID	INT	Not NULL; Foreign key	Reference to student group

Table 4.5.1 Students Table Structure

Student Groups table

Attribute	Data Type	Constraint	Description
ID	INT	Primary key	Unique identifier of student group
Student count	TINYINT	Not NULL; Check if value is between 1 and 10	Number of students in group
Year	TINYINT	Not NULL; Check if value is more than or equals to 1	Current academic year
Trimester	TINYINT	Check if value is between 1 and 3	Current trimester

Table 4.5.2 Student Groups Table Structure

Courses table

Attribute	Data Type	Constraint	Description
ID	INT	Primary key	Unique identifier of course
Code	VARCHAR(20)	Not NULL; Unique	Course code

Name	VARCHAR(200)	Not NULL	Course name
Year	TINYINT	Not NULL; Check if value is more than or equals to 1	Academic year
Trimester	TINYINT	Check if value is between 1 and 3	Academic trimester

Table 4.5.3 Courses Table Structure

Lecturers table

Attribute	Data Type	Constraint	Description
ID	INT	Primary key	Unique identifier of lecturer
Name	VARCHAR(100)	Not NULL	Full name of lecturer
Assigned class hours	SMALLINT	Default is 0; Check if value is more than or equals to 0	Class hours assigned to lecturer

Table 4.5.4 Lecturers Table Structure

Rooms table

Attribute	Data Type	Constraint	Description
ID	INT	Primary key	Unique identifier of room
Name	VARCHAR(50)	Not NULL	Room name
Building	CHAR(1)	Not NULL; Check if value is between ‘A’ and ‘Z’	Building identifier
Type	ENUM(‘lecture hall’, ‘tutorial room’, ‘practical lab’)	Not NULL	Room type
Capacity	SMALLINT	Not NULL; Check if value is more than 0	Room capacity

Table 4.5.5 Rooms Table Structure

Classes table

Attribute	Data Type	Constraint	Description
ID	INT	Primary key	Unique identifier of class
Type	ENUM('lecture', 'tutorial', 'practical')	Not NULL	Class type
Duration	TINYINT	Not NULL; Check if value is between 1 and 10	Class duration (hours)
Course ID	INT	Not NULL; Foreign key	Reference to course-to-lecturer assignment
Lecturer ID	INT		

Table 4.5.6 Classes Table Structure

Courses-Lecturers table

Attribute	Data Type	Constraint	Description
Course ID	INT	Primary key; Foreign key	Reference to course
Lecturer ID	INT	Primary key; Foreign key	Reference to lecturer

Table 4.5.7 Courses-Lecturers Table Structure

Groups-Classes table

Attribute	Data Type	Constraint	Description
Group ID	INT	Primary key; Foreign key	Reference to student group
Class ID	INT	Primary key; Foreign key	Reference to class

Table 4.5.8 Groups-Classes Table Structure

4.5.3 Database Trigger

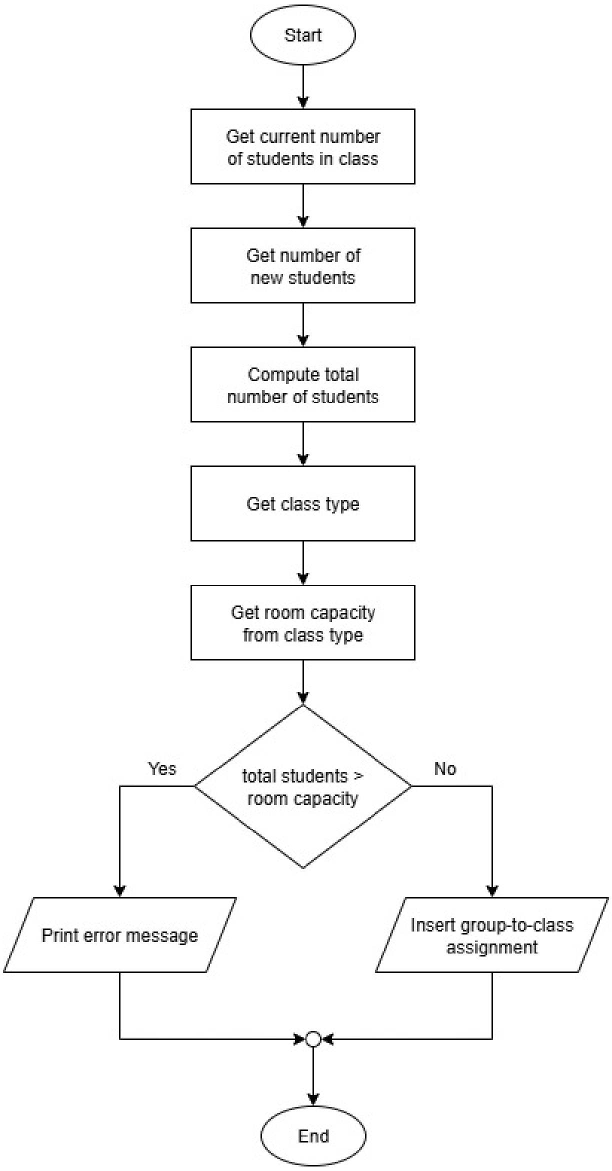

Start
Get current number of students in class
Get number of new students
Compute total number of students
Get class type
Get room capacity from class type
total students > room capacity
Yes
No
Print error message
Insert group-to-class assignment
End

Figure 4.5.2 Flowchart of Trigger

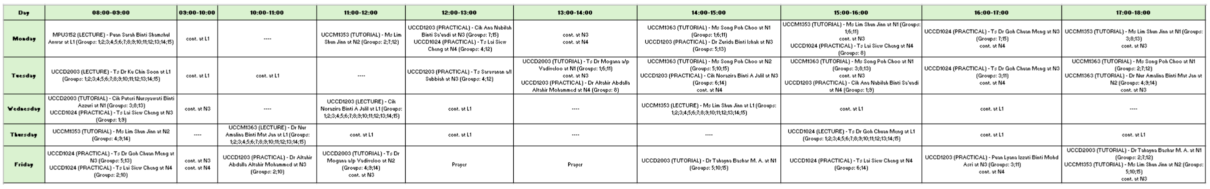
In this project, the university course timetabling system defines a trigger that runs before inserting every group-to-class assignment into database to enforce enrolment limits at the database layer. The workflow of this trigger is shown in Figure 4.5.2. When the database attempts to insert a new group-to-class link, the trigger first calculates the projected total number of students in that class by summing the number of students of all already-linked groups and adding the incoming group's size. Then, it looks up the class type and determines the applicable capacity threshold. For example, a lecture class is held in a lecture hall that has a capacity of 300 students. Lastly, it compares the projected total number of students against the limit. If the insert operation would exceed the allowed capacity, the database aborts the operation with a clear error message. By validating capacity close to the data, the trigger guarantees data consistency, prevents over-enrolment even under concurrent writes, and keeps constraint logic centralised and auditable.

Chapter 5

System Testing

5.1 Experiments on Constraints

The verification on constraints is conducted by observing the timetable in CSV format and making sure that all constraints are satisfied by the output. All output timetables are manually checked one-by-one to ensure the correctness since there is no one static, exact output due to the stochastic property of genetic algorithm (GA).



Day	08:00-09:00	09:00-10:00	10:00-11:00	11:00-12:00	12:00-13:00	13:00-14:00	14:00-15:00	15:00-16:00	16:00-17:00	17:00-18:00
Monday	MPU3152 (LECTURE) - Puan Sarah Binti Shamshul Anwar at L1 (Groups: 1;2;3;4;5;6;7;8;9;10;11;12;13;14;15)	cont. at L1	----	UCCM1353 (TUTORIAL) - Ms Lim Shun Jinn at N2 (Groups: 2;7;12)	UCCD1203 (PRACTICAL) - Cik Ana Nabilah Binti Sa'uadi at N3 (Groups: 7;15) UCCD1024 (PRACTICAL) - Ts Lai Siew Cheng at N4 (Groups: 4;12)	cont. at N3 cont. at N4	UCCM1363 (TUTORIAL) - Ms Song Poh Choo at N1 (Groups: 1;6;11) UCCD1203 (PRACTICAL) - Dr Zurida Binti Ishak at N3 (Groups: 5;13)	UCCM1353 (TUTORIAL) - Ms Lim Shun Jinn at N1 (Groups: 1;6;11) cont. at N3 UCCD1024 (PRACTICAL) - Ts Lai Siew Cheng at N4 (Groups: 8)	UCCD1024 (PRACTICAL) - Ts Dr Goh Chuan Meng at N3 (Groups: 7;15) cont. at N4	UCCM1353 (TUTORIAL) - Ms Lim Shun Jinn at N1 (Groups: 3;8;13) cont. at N3
Tuesday	UCCD2003 (LECTURE) - Ts Dr Ku Chin Soon at L1 (Groups: 1;2;3;4;5;6;7;8;9;10;11;12;13;14;15)	cont. at L1	cont. at L1	----	UCCD1203 (PRACTICAL) - Ts Saravanan a/l Subbiah at N3 (Groups: 4;12)	UCCD2003 (TUTORIAL) - Ts Dr Mogana a/p Vadiveloo at N1 (Groups: 1;6;11) cont. at N3 UCCD1203 (PRACTICAL) - Dr Altahir Abdalla Altahir Mohammed at N4 (Groups: 8)	UCCM1363 (TUTORIAL) - Ms Song Poh Choo at N2 (Groups: 5;10;15) UCCD1203 (PRACTICAL) - Cik Norazira Binti A Jalil at N3 (Groups: 6;14) cont. at N4	UCCM1363 (TUTORIAL) - Ms Song Poh Choo at N1 (Groups: 3;8;13) cont. at N3 UCCD1203 (PRACTICAL) - Cik Ana Nabilah Binti Sa'uadi at N4 (Groups: 1;9)	UCCD1024 (PRACTICAL) - Ts Dr Goh Chuan Meng at N3 (Groups: 3;11) cont. at N4	UCCM1363 (TUTORIAL) - Ms Song Poh Choo at N1 (Groups: 2;7;12) UCCM1363 (TUTORIAL) - Dr Nur Amalina Binti Mat Jan at N2 (Groups: 4;9;14) cont. at N3
Wednesday	UCCD2003 (TUTORIAL) - Cik Puteri Nursyawati Binti Azzuri at N1 (Groups: 3;8;13) UCCD1024 (PRACTICAL) - Ts Lai Siew Cheng at N3 (Groups: 1;9)	cont. at N3	----	UCCD1203 (LECTURE) - Cik Norazira Binti A Jalil at L1 (Groups: 1;2;3;4;5;6;7;8;9;10;11;12;13;14;15)	cont. at L1	----	UCCM1353 (LECTURE) - Ms Lim Shun Jinn at L1 (Groups: 1;2;3;4;5;6;7;8;9;10;11;12;13;14;15)	cont. at L1	cont. at L1	----
Thursday	UCCM1353 (TUTORIAL) - Ms Lim Shun Jinn at N2 (Groups: 4;9;14)	----	UCCM1363 (LECTURE) - Dr Nur Amalina Binti Mat Jan at L1 (Groups: 1;2;3;4;5;6;7;8;9;10;11;12;13;14;15)	cont. at L1	cont. at L1	----	----	UCCD1024 (LECTURE) - Ts Dr Goh Chuan Meng at L1 (Groups: 1;2;3;4;5;6;7;8;9;10;11;12;13;14;15)	cont. at L1	cont. at L1
Friday	UCCD1024 (PRACTICAL) - Ts Dr Goh Chuan Meng at N3 (Groups: 5;13) UCCD1024 (PRACTICAL) - Ts Lai Siew Cheng at N4 (Groups: 2;10)	cont. at N3 cont. at N4	UCCD1203 (PRACTICAL) - Dr Altahir Abdalla Altahir Mohammed at N3 (Groups: 2;10)	UCCD2003 (TUTORIAL) - Ts Dr Mogana a/p Vadiveloo at N2 (Groups: 4;9;14) cont. at N3	Prayer	Prayer	UCCD2003 (TUTORIAL) - Dr Tahayna Bashar M. A. at N1 (Groups: 5;10;15)	UCCD1024 (PRACTICAL) - Ts Lai Siew Cheng at N4 (Groups: 6;14)	UCCD1203 (PRACTICAL) - Puan Lyana Izzati Binti Mohd Asri at N3 (Groups: 3;11) cont. at N4	UCCD2003 (TUTORIAL) - Dr Tahayna Bashar M. A. at N1 (Groups: 2;7;12) UCCM1353 (TUTORIAL) - Ms Lim Shun Jinn at N2 (Groups: 5;10;15) cont. at N3

Figure 5.2.1 Overall Timetable for Result Verification

Figure 5.2.1 shows the overall timetable used for result verification for one experiment. This timetable consists of information from courses, lecturers, students, and rooms timetables, compiling all their data into one overview. Therefore, this summary timetable is used to verify the constraints in the timetable due to its simplicity without needing to check for other timetables.

No.	Hard Constraint	Observation	Status
1	A student must attend at most one class per time slot.	There is no repeating student group in each time slot.	Pass
2	A student must enrol in every course required for the semester.	Each course has the enrolment of each student group.	Pass
3	A lecturer must teach at most one class per time slot.	There is no repeating lecturer in each time slot.	Pass
4	A room must host at most one class per time slot.	There is no repeating room in each time slot.	Pass
5	A class must enrol at least one student.	There is no class with no student group enrolment.	Pass

6	A class must have exactly one lecturer.	There is only one lecturer for each class.	Pass
7	A class must be assigned a room once and only once.	There is only one lecturer for each class.	Pass
8	A lecture class must be held in a lecture hall.	All classes in room L1 (lecture hall) are lecture classes.	Pass
9	A tutorial class must be held in a tutorial room.	All classes in rooms N1 and N2 (tutorial rooms) are tutorial classes.	Pass
10	A practical class must be held in a computer lab.	All classes in rooms N3 and N4 (practical labs) are practical classes.	Pass
11	A class must not enrol more students than the room's capacity.	The number of students (number of student groups × 10) does not exceed room capacity (300 for L1, 30 for N1 and N2, 20 for N3 and N4) for each class.	Pass
12	A class must be scheduled on weekdays only.	Satisfied by system design.	Pass
13	A class must be scheduled between 08:00 and 18:00.	Satisfied by system design.	Pass
14	A class must not be scheduled on Friday between 12:00 and 14:00 (Muslim prayer time).	There is no class at the slot 5 and 6 of Friday.	Pass
15	A class must not span across multiple days.	Every class at the last slot of a day is different from the classes at the first slot of next day.	Pass

Table 5.1.1 Hard Constraint Verifications

No.	Soft Constraint	Observation	Status
1	A student should study no more than four consecutive hours.	There is no student group spanning across time slot more than four consecutive hours.	Pass

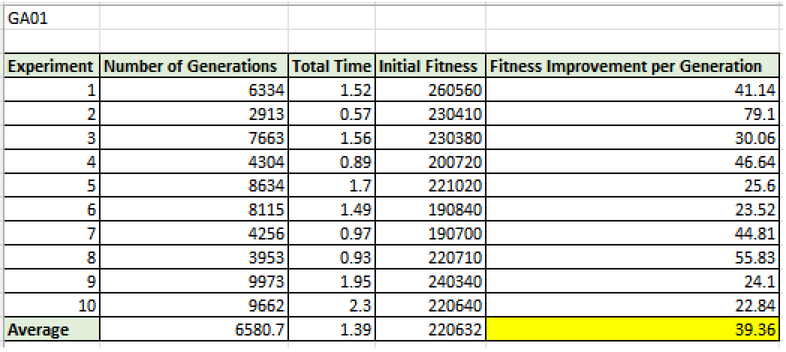
2	A lecturer should teach no more than four consecutive hours.	There is no lecturer spanning across time slot more than four consecutive hours.	Pass
3	A student's consecutive classes should be held in the same building.	Consecutive classes for each student group are in the same building.	Pass
4	A lecturer should receive at least one teaching hour.	Every lecturer has at least one class assignment.	Pass

Table 5.1.2 Soft Constraint Verifications

5.2 Experiments on GA Models

In these experiments, the input data design and output data format are specified in Section 4.2 and 4.3 respectively. The main performance comparison metric is the one proposed in this project, fitness improvement per generation.

The experiments on each GA combination (10 runs) generate a result as shown in Figure 5.2.1.

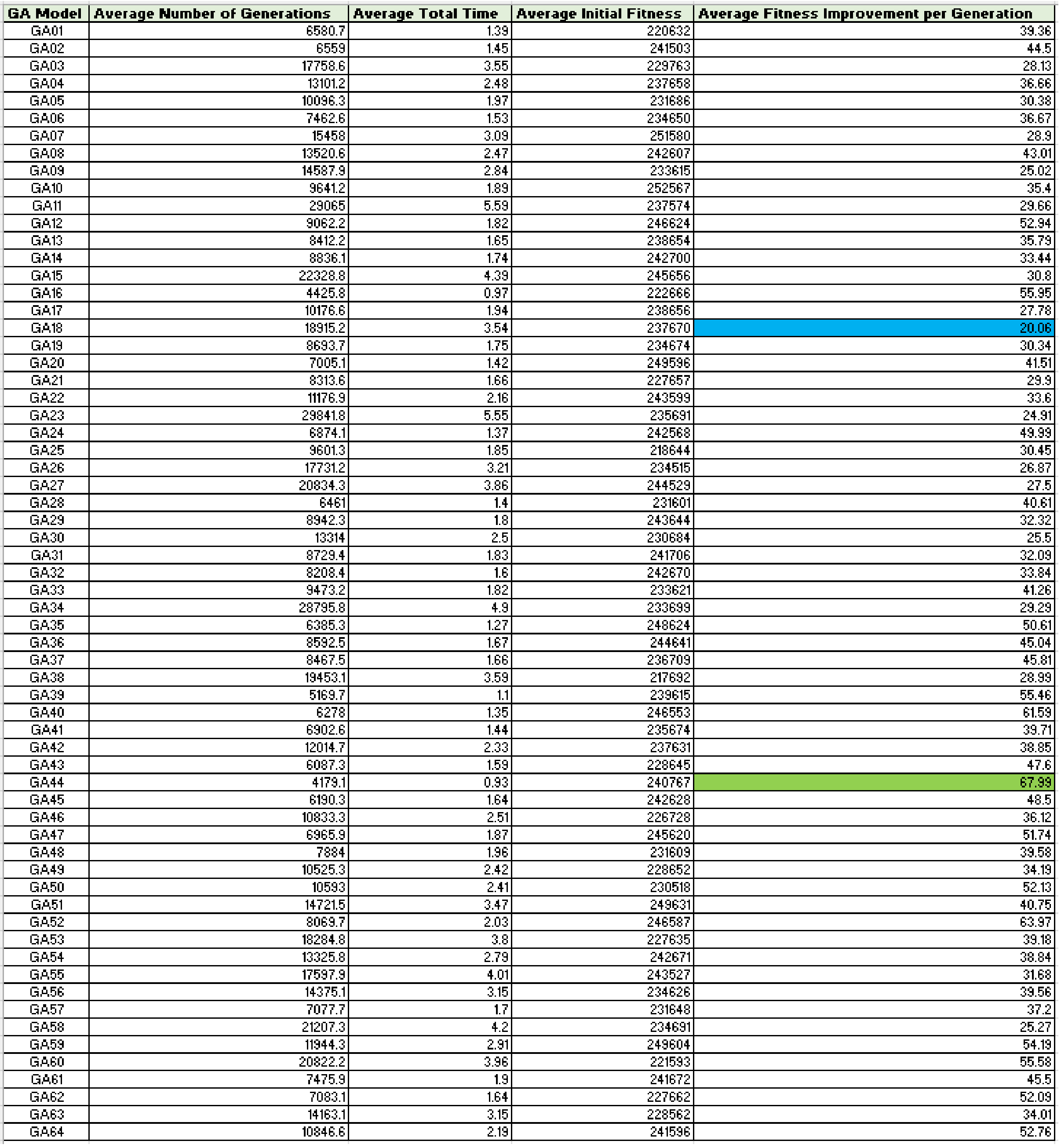


GA01

Experiment	Number of Generations	Total Time	Initial Fitness	Fitness Improvement per Generation
1	6334	1.52	260560	41.14
2	2913	0.57	230410	79.1
3	7663	1.56	230380	30.06
4	4304	0.89	200720	46.64
5	8634	1.7	221020	25.6
6	8115	1.49	190840	23.52
7	4256	0.97	190700	44.81
8	3953	0.93	220710	55.83
9	9973	1.95	240340	24.1
10	9662	2.3	220640	22.84
Average	6580.7	1.39	220632	39.36

Figure 5.2.1 Experiment Result of GA01 Model

The results of all the 64 GA combinations (see Section 4.4.10) are summarised into a table as shown in Figure 5.2.2.



GA Model	Average Number of Generations	Average Total Time	Average Initial Fitness	Average Fitness Improvement per Generation
GA01	6580.7	1.39	220632	39.36
GA02	6559	1.45	241503	44.5
GA03	17758.6	3.55	229763	28.13
GA04	13101.2	2.48	237658	36.66
GA05	10096.3	1.97	231686	30.38
GA06	7462.6	1.53	234650	36.67
GA07	15458	3.09	251580	28.9
GA08	13520.6	2.47	242607	43.01
GA09	14587.9	2.84	233615	25.02
GA10	9641.2	1.89	252567	35.4
GA11	29065	5.59	237574	29.66
GA12	9062.2	1.82	246624	52.94
GA13	8412.2	1.65	238654	35.79
GA14	8836.1	1.74	242700	33.44
GA15	22328.8	4.39	245656	30.8
GA16	4425.8	0.97	222666	55.95
GA17	10176.6	1.94	238656	27.78
GA18	18915.2	3.54	237670	20.06
GA19	8693.7	1.75	234674	30.34
GA20	7005.1	1.42	249596	41.51
GA21	8313.6	1.66	227657	29.9
GA22	11176.9	2.16	243599	33.6
GA23	29841.8	5.55	235691	24.91
GA24	6874.1	1.37	242568	49.99
GA25	9601.3	1.85	218644	30.45
GA26	17731.2	3.21	234515	26.87
GA27	20834.3	3.86	244529	27.5
GA28	6461	1.4	231601	40.61
GA29	8942.3	1.8	243644	32.32
GA30	13314	2.5	230684	25.5
GA31	8729.4	1.83	241706	32.09
GA32	8208.4	1.6	242670	33.84
GA33	9473.2	1.82	233621	41.26
GA34	28795.8	4.9	233699	29.29
GA35	6385.3	1.27	248624	50.61
GA36	8592.5	1.67	244641	45.04
GA37	8467.5	1.66	236709	45.81
GA38	19453.1	3.59	217692	28.99
GA39	5169.7	1.1	239615	55.46
GA40	6278	1.35	246553	61.59
GA41	6902.6	1.44	235674	39.71
GA42	12014.7	2.33	237631	38.85
GA43	6087.3	1.59	228645	47.6
GA44	4179.1	0.93	240767	67.99
GA45	6190.3	1.64	242628	48.5
GA46	10833.3	2.51	226728	36.12
GA47	6965.9	1.87	245620	51.74
GA48	7884	1.96	231609	39.58
GA49	10525.3	2.42	228652	34.19
GA50	10593	2.41	230518	52.13
GA51	14721.5	3.47	249631	40.75
GA52	8069.7	2.03	246587	63.97
GA53	18284.8	3.8	227635	39.18
GA54	13325.8	2.79	242671	38.84
GA55	17597.9	4.01	243527	31.68
GA56	14375.1	3.15	234626	39.56
GA57	7077.7	1.7	231648	37.2
GA58	21207.3	4.2	234691	25.27
GA59	11944.3	2.91	249604	54.19
GA60	20822.2	3.96	221593	55.58
GA61	7475.9	1.9	241672	45.5
GA62	7083.1	1.64	227662	52.09
GA63	14163.1	3.15	228562	34.01
GA64	10846.6	2.19	241596	52.76

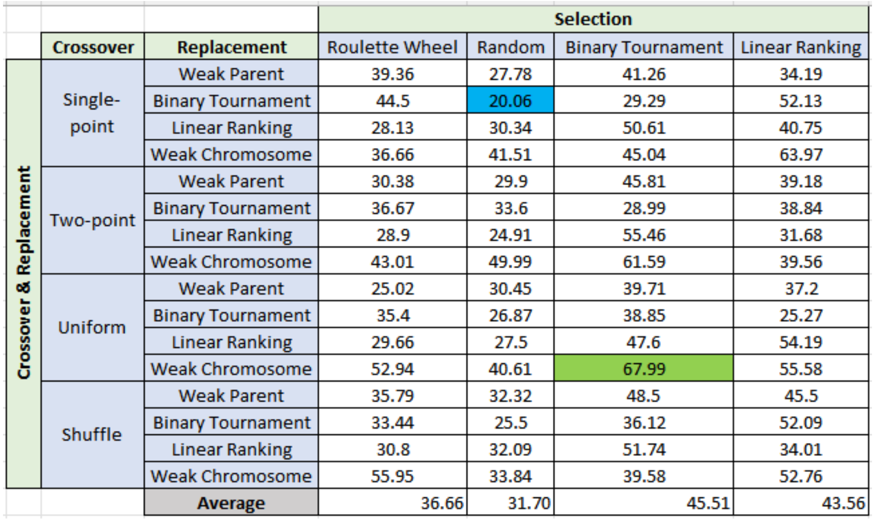
Figure 5.2.2 Summary of Experiment Results of All GA Combinations

From the summary table, it is clear that the GA18 is the worst model, which is formed by random selection, single-point crossover, swap mutation, and binary tournament replacement. On the other hand, the GA44 is the best model, which comprises of binary tournament selection, uniform crossover, swap mutation, and weak chromosome replacement.

5.2.1 Comparison among Operator Techniques

Several operator-level comparison tables are constructed to enable easier comparison with each other without interference of other operators.

5.2.1.1 Selection



			Selection			
	Crossover	Replacement	Roulette Wheel	Random	Binary Tournament	Linear Ranking
Crossover & Replacement	Single-point	Weak Parent	39.36	27.78	41.26	34.19
		Binary Tournament	44.5	20.06	29.29	52.13
		Linear Ranking	28.13	30.34	50.61	40.75
		Weak Chromosome	36.66	41.51	45.04	63.97
	Two-point	Weak Parent	30.38	29.9	45.81	39.18
		Binary Tournament	36.67	33.6	28.99	38.84
		Linear Ranking	28.9	24.91	55.46	31.68
		Weak Chromosome	43.01	49.99	61.59	39.56
	Uniform	Weak Parent	25.02	30.45	39.71	37.2
		Binary Tournament	35.4	26.87	38.85	25.27
		Linear Ranking	29.66	27.5	47.6	54.19
		Weak Chromosome	52.94	40.61	67.99	55.58
	Shuffle	Weak Parent	35.79	32.32	48.5	45.5
		Binary Tournament	33.44	25.5	36.12	52.09
		Linear Ranking	30.8	32.09	51.74	34.01
		Weak Chromosome	55.95	33.84	39.58	52.76
		Average	36.66	31.70	45.51	43.56

Figure 5.2.3 Selection Operator Comparison Table

Overall, the selection operator makes a clear difference. Averaged across all crossover-replacement pairings, Binary Tournament delivers the highest fitness improvement per generation (45.51), closely followed by Linear Ranking (43.56). Roulette Wheel lags behind (36.66), and Random is the weakest on average (31.70). In short, Binary Tournament is better than Linear Ranking, followed by Roulette Wheel and Random.

Binary Tournament also contains the single best cell in the grid (67.99 with Uniform crossover and Weak Chromosome replacement, highlighted in green). This is consistent with the theory, where small-k (k represents tournament size) tournaments impose steady selection pressure that quickly amplifies fitter individuals while still allowing diversity from occasional upsets. When coupled with an aggressive replacement like Weak Chromosome, exploitation is intensified, and the generation count to reach zero-cost drops, hence resulting in a larger improvement per generation value.

Linear Ranking is a close second on average and shows several strong combinations (for example, Single-point crossover with Weak Chromosome replacement at 63.97 and Single-Point crossover with Binary Tournament replacement at 52.13). Rank-based selection is insensitive to the absolute scaling of the penalty-based fitness used, so it avoids Roulette

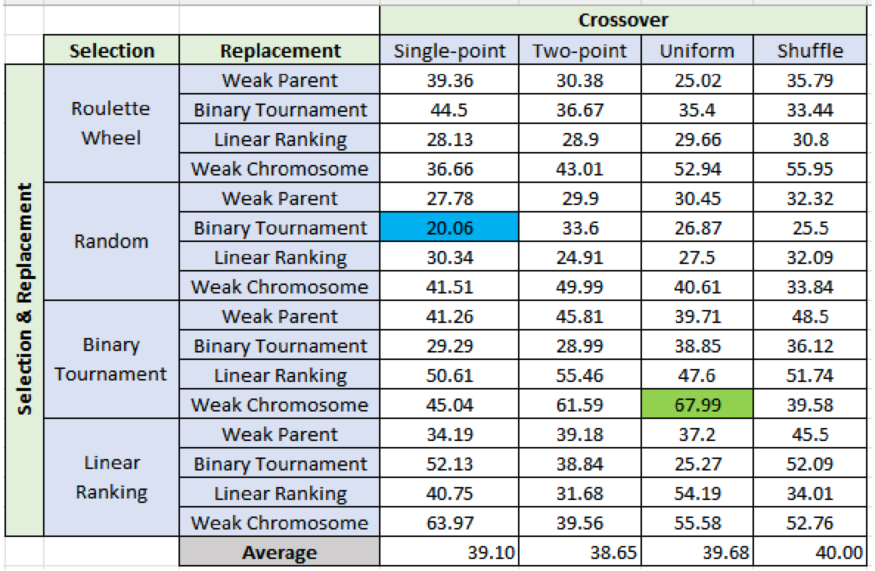
Wheel's tendency to over- or under-select when the population's costs are tightly clustered or highly skewed. The result is consistent progress across many crossover and replacement settings.

Roulette Wheel's middling average and lack of top-end cells reflect that sensitivity to fitness scaling. With penalty sums that shrink as the population improves, proportional selection can become noisy, where tiny absolute differences in cost translate to near-uniform sampling, diluting selection pressure. It does produce respectable outcomes in some rows (for instance, Two-point crossover with Weak Chromosome replacement at 43.01), but it is less robust overall than tournament or ranking.

Random selection performs worst and contains the global minimum (20.06 with Single-Point crossover and Binary Tournament replacement, highlighted in blue). With essentially no selection pressure, progress depends almost entirely on the crossover-replacement pair to stumble into improvements. It can look adequate only when paired with very strong replacement (for example, Two-point crossover with Weak Chromosome replacement reaches approximately 50), which underlines that the gains come from replacement rather than selection.

Taken together, the data supports using Binary Tournament as the main selector for the UCTP, with Linear Ranking as a solid alternative.

5.2.1.2 Crossover



			Crossover			
	Selection	Replacement	Single-point	Two-point	Uniform	Shuffle
Selection & Replacement	Roulette Wheel	Weak Parent	39.36	30.38	25.02	35.79
		Binary Tournament	44.5	36.67	35.4	33.44
		Linear Ranking	28.13	28.9	29.66	30.8
		Weak Chromosome	36.66	43.01	52.94	55.95
	Random	Weak Parent	27.78	29.9	30.45	32.32
		Binary Tournament	20.06	33.6	26.87	25.5
		Linear Ranking	30.34	24.91	27.5	32.09
		Weak Chromosome	41.51	49.99	40.61	33.84
	Binary Tournament	Weak Parent	41.26	45.81	39.71	48.5
		Binary Tournament	29.29	28.99	38.85	36.12
		Linear Ranking	50.61	55.46	47.6	51.74
		Weak Chromosome	45.04	61.59	67.99	39.58
	Linear Ranking	Weak Parent	34.19	39.18	37.2	45.5
		Binary Tournament	52.13	38.84	25.27	52.09
		Linear Ranking	40.75	31.68	54.19	34.01
		Weak Chromosome	63.97	39.56	55.58	52.76
		Average	39.10	38.65	39.68	40.00

Figure 5.2.4 Crossover Operator Comparison Table

Looking column-wise, the four crossover types are quite close on average, with Shuffle narrowly best (40.00), followed by Uniform (39.68), Single-point (39.10) and Two-point at last (38.65). The spread across column means is small (1.35). Therefore, it is concluded that, on this dataset and encoding, crossover choice affects progress per generation less than selection and replacement do. Still, there are meaningful differences in robustness and extremes.

Uniform crossover is consistently strong and delivers the overall best cell (67.99) with Binary Tournament selection and Weak Chromosome replacement. It also posts high values across other selections when paired with Weak Chromosome replacement (for example, 52.94 with Roulette Wheel selection, 55.58 with Linear Ranking selection). Room-wise mixing across the entire timetable injects diversity without relying on any particular cut position, so good room patterns discovered in one parent can permeate the other more reliably, especially under high selection pressure.

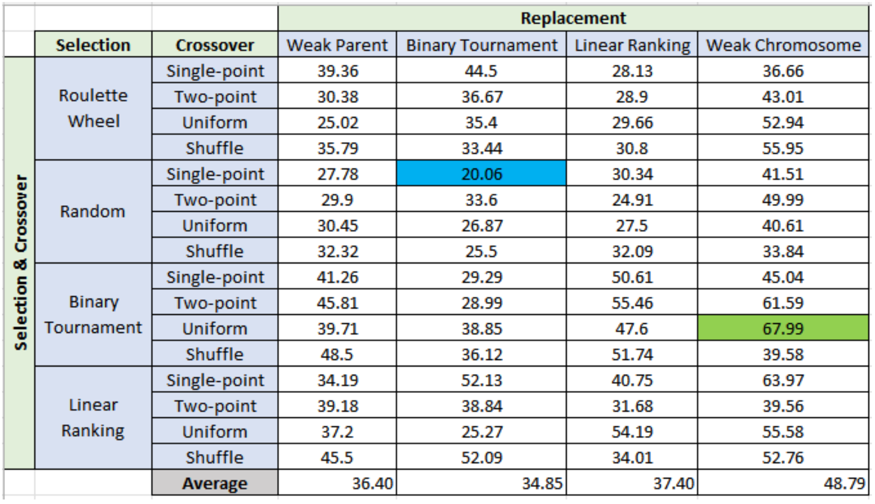
On average, Shuffle edges out the rest (40.00) and is notably robust: 55.95 with Roulette Wheel selection and Weak Chromosome replacement, 52.76 with Linear Ranking and Weak Chromosome replacement, and 51.74 with Binary Tournament selection and Linear Ranking replacement. By randomising the order of rooms before exchanging a segment (and unshuffling after), it reduces positional bias in the room-level representation. The effect is similar to uniform, broad recombination within the room, but with slightly steadier returns across the board because it does not depend on lucky crossover points.

The performance of Single-Point is highly pairing-sensitive. It reaches an excellent 63.97 with Linear Ranking selection and Weak Chromosome replacement, and 52.13 with Linear Ranking selection and Binary Tournament replacement but drops to the table’s worst (20.06) under Random selection and Binar Tournament replacement. Because only one contiguous segment of timetable is exchanged, the operator tends to be exploitative: when parents are already strong (from stronger selection), it propagates useful structures; when parents are mediocre (such as Random selection), it lacks diversity and stalls, hence the volatility.

Despite a few high spots (61.59 with Binary Tournament and Weak Chromosome; 55.46 with Binary Tournament and Linear Ranking replacement), its average is the lowest.

Swapping two segments room-by-room preserves a lot of parental layouts while adding just enough disruption to require more repairs, which often yields less net gain per generation than more diversified mixing. In other words, it can improve quickly when coupled with very strong selection and replacement but otherwise under-explores.

5.2.1.3 Replacement



			Replacement			
	Selection	Crossover	Weak Parent	Binary Tournament	Linear Ranking	Weak Chromosome
Selection & Crossover	Roulette Wheel	Single-point	39.36	44.5	28.13	36.66
		Two-point	30.38	36.67	28.9	43.01
		Uniform	25.02	35.4	29.66	52.94
		Shuffle	35.79	33.44	30.8	55.95
	Random	Single-point	27.78	20.06	30.34	41.51
		Two-point	29.9	33.6	24.91	49.99
		Uniform	30.45	26.87	27.5	40.61
		Shuffle	32.32	25.5	32.09	33.84
	Binary Tournament	Single-point	41.26	29.29	50.61	45.04
		Two-point	45.81	28.99	55.46	61.59
		Uniform	39.71	38.85	47.6	67.99
		Shuffle	48.5	36.12	51.74	39.58
	Linear Ranking	Single-point	34.19	52.13	40.75	63.97
		Two-point	39.18	38.84	31.68	39.56
		Uniform	37.2	25.27	54.19	55.58
		Shuffle	45.5	52.09	34.01	52.76
		Average	36.40	34.85	37.40	48.79

Figure 5.2.5 Replacement Operator Comparison Table

Reading by columns, Weak Chromosome clearly dominates. It posts the highest column-average (48.79) by a large margin over Linear Ranking (37.40), Weak Parent (36.40), and Binary Tournament replacement (34.85). It also contains the global best cell, 67.99 with Binary Tournament selection and Uniform crossover (highlighted in green). This pattern fits intuition: always ejecting the worst individual each iteration maximises exploitation pressure and guarantees the population floor rises, so the fitness improvement per generation metric benefits directly.

Linear Ranking is the next most reliable. Its column has few weak outliers and several strong pairings (e.g., Binary Tournament selection and Two-Point crossover at 55.46; Linear Ranking selection with Single-Point crossover at 40.75 and Uniform crossover at 54.19). Rank-based survivor choice is scale-invariant and tempers stochasticity, so it preserves steady progress even when fitness values get tightly clustered, hence the solid average.

Weak Parent sits close to Linear Ranking on average but is more sensitive to the quality of the two parents that produced the offspring. When one parent is already strong, replacing the weaker of the pair is sensible. However, if both parents are middling, the operator can recycle mediocrity, which caps the per-generation gains. Nonetheless, there are still good rows when upstream selection is strong (for example, Binary Tournament selection and Shuffle crossover at 48.5), but it is less consistently high than Weak Chromosome.

Binary Tournament replacement performs worst on average and includes the global minimum of 20.06 with Random selection and Single-point crossover (highlighted in blue). A small tournament to decide who leaves the population adds randomness at the survivor stage; without strong selection pressure, it can evict decent individuals and keep weaker ones, eroding building blocks assembled by crossover. That hurts measured improvement per generation unless counterbalanced by a very strong selector and an aggressive crossover.

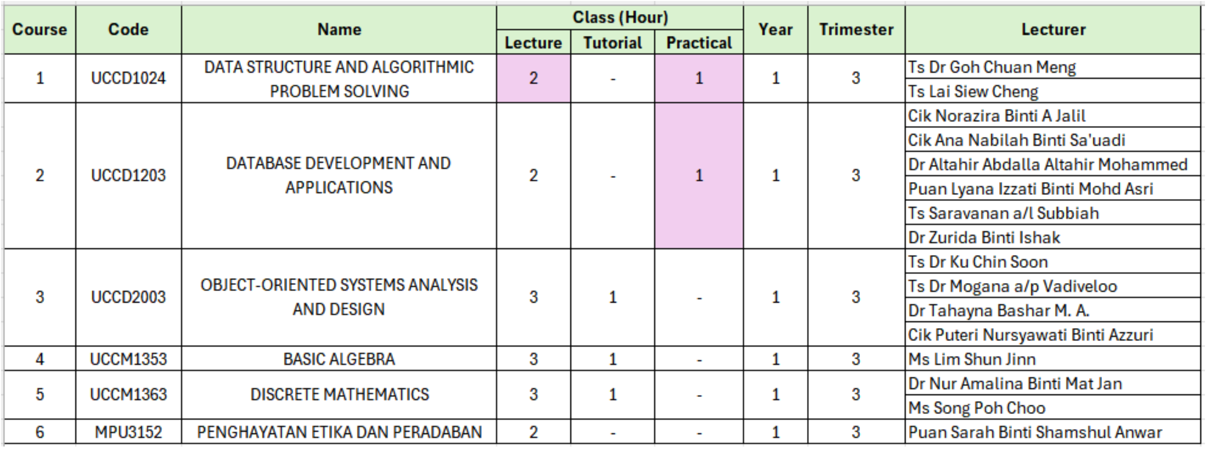
5.2.2 Comparison with Past Research

In this experiment, the result of the past research [8] is used for comparison. [8] adopted a GA combination that consists of 5-tournament selection, single-point crossover, and random mutation with Simple Search Neighbourhood (SSN) and Swap Search Neighbourhood (SWN) strategies. In its experiment, [8] achieves a fitness improvement for both datasets tested by it, which reduced the penalty cost from approximately 24000 to 1400 in 5 minutes. However, no optimum value was achieved.

With that said, the university course timetabling system developed in this project is theoretically better than the system of [8] in terms of performance since it is able to generate a perfect timetable. Nonetheless, the datasets applied in both research are different. Therefore, this conclusion still needs to be validated in the future study.

5.3 Experiments on Resource Utilisation

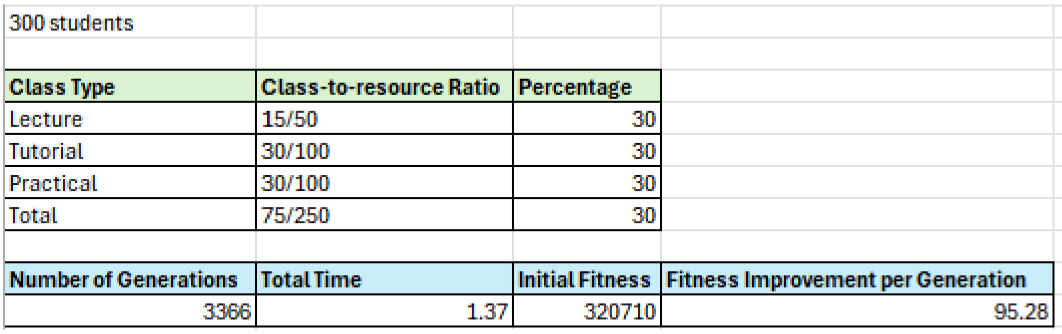
This section focuses on determining the maximum performance of the developed university timetabling system by testing it with different class-to-resource ratios. In these experiments, the inputs are the mostly the same as what described in section 4.2, except the number of students and the classes for each course. The number of students is manipulated in the experiments to control the resource utilisation percentage. On the other hand, the duration of some classes is adjusted, so that the class-to-resource ratio of all three types of classes (lecture, tutorial, practical) can achieve the same percentage, making the experiments easier and fairer. Figure 5.3.1 shows the adjusted classes data for each course. Furthermore, these experiments only run the GA once instead of 10 times like others because the primary objective of this experiment is to determine the feasibility of the system. The GA combination applied in these experiments is the best combination found from previous experiments, which is GA44 that comprises of binary tournament selection, uniform crossover, swap mutation, and weak chromosome replacement.



Course	Code	Name	Class (Hour)			Year	Trimester	Lecturer
			Lecture	Tutorial	Practical			
1	UCCD1024	DATA STRUCTURE AND ALGORITHMIC PROBLEM SOLVING	2	-	1	1	3	Ts Dr Goh Chuan Meng Ts Lai Siew Cheng
2	UCCD1203	DATABASE DEVELOPMENT AND APPLICATIONS	2	-	1	1	3	Cik Norazira Binti A Jalil Cik Ana Nabilah Binti Sa'uadi Dr Altahir Abdalla Altahir Mohammed Puan Lyana Izzati Binti Mohd Asri Ts Saravanan a/l Subbiah Dr Zurida Binti Ishak
3	UCCD2003	OBJECT-ORIENTED SYSTEMS ANALYSIS AND DESIGN	3	1	-	1	3	Ts Dr Ku Chin Soon Ts Dr Mogana a/p Vadiveloo Dr Tahayna Bashar M. A. Cik Puteri Nursyawati Binti Azzuri
4	UCCM1353	BASIC ALGEBRA	3	1	-	1	3	Ms Lim Shun Jinn
5	UCCM1363	DISCRETE MATHEMATICS	3	1	-	1	3	Dr Nur Amalina Binti Mat Jan Ms Song Poh Choo
6	MPU3152	PENGHAYATAN ETIKA DAN PERADABAN	2	-	-	1	3	Puan Sarah Binti Shamshul Anwar

Figure 5.3.1 Adjusted Class Information

When the number of students is 300, the resource utilisation is 30%. The system performs well by reaching optimum in 1.37 seconds while using only 3366 generations.



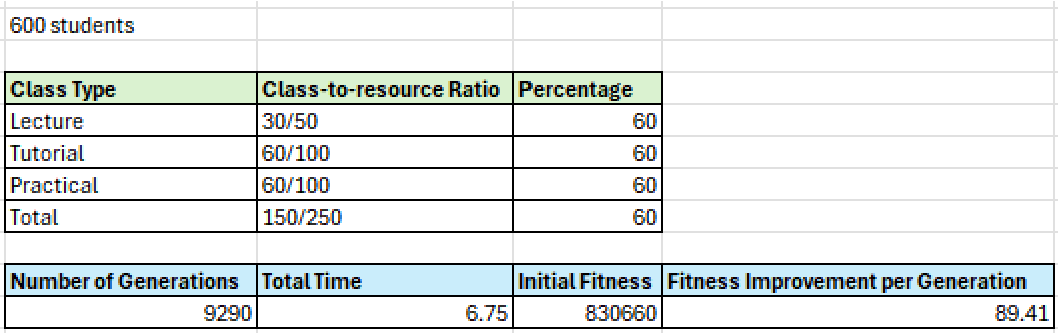
300 students

Class Type	Class-to-resource Ratio	Percentage
Lecture	15/50	30
Tutorial	30/100	30
Practical	30/100	30
Total	75/250	30

Number of Generations	Total Time	Initial Fitness	Fitness Improvement per Generation
3366	1.37	320710	95.28

Figure 5.4.2 Experiment Result of 30% Resource Utilisation

When the number of students is 600, the resource utilisation is 60%. The system successfully generates a feasible timetable in 6.75 seconds by using 9290 generations.



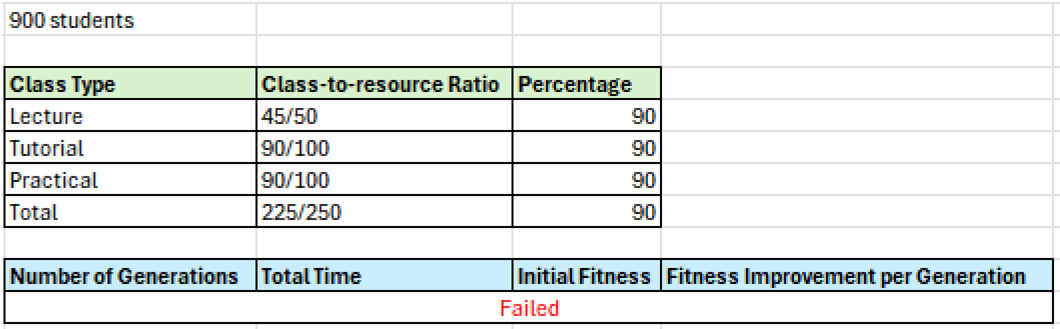
600 students

Class Type	Class-to-resource Ratio	Percentage
Lecture	30/50	60
Tutorial	60/100	60
Practical	60/100	60
Total	150/250	60

Number of Generations	Total Time	Initial Fitness	Fitness Improvement per Generation
9290	6.75	830660	89.41

Figure 5.4.3 Experiment Result of 60% Utilisation Testing

However, when the number of students is 900, and the resource utilisation is 90%, the system fails to generate a feasible solution. This might be because the experiments increase the number of students without adding lecturers, which raises the risk of breaching lecturer-related soft constraints, which is excessively long consecutive teaching hours.



900 students

Class Type	Class-to-resource Ratio	Percentage
Lecture	45/50	90
Tutorial	90/100	90
Practical	90/100	90
Total	225/250	90

Number of Generations	Total Time	Initial Fitness	Fitness Improvement per Generation
Failed			

Figure 5.4.4 Experiment Result of 90% Utilisation Testing

Chapter 6
Discussion

6.1 Objective Evaluation

Overall, this project meets all three stated objectives within the defined scope of a partial mock dataset and a GA-based solution pipeline.

First, the objective to formalise a comprehensive set of hard and soft constraints for the UCTP is achieved. Hard constraints such as capacity and resource non-overlap are explicitly modelled. On the other hand, soft constraints, including the new "same building for consecutive classes" rule, are incorporated into the fitness function as quantitative penalties, so their impact could be measured across experiments. As documented in this report, the same-building rule is implemented as a binary penalty rather than a distance-weighted cost. Nonetheless, this simplification still enables controlled, interpretable analysis of the constraint's effect and satisfies the objective's requirement to formalise and measure it.

Second, the project successfully designs and implements a flexible GA framework with exchangeable operators. Four selection methods, four crossover methods, one mutation, and four replacement methods (including two newly proposed replacement strategies) are composed into 64 distinct operator combinations without changes to the core engine. An immediate repair step after both crossover and mutation ensures chromosomes remains schedulable under the modelled constraints. This modularity and the use of room-based crossover are consistent with the design intent and demonstrate that the "plug-and-play" operator goal was achieved in practice.

Third, the evaluation objective is achieved: every GA combination is assessed on the partial mock dataset through multiple independent trials. For each run, the number of generations to reach feasibility (zero penalty), execution time, and initial penalty cost are recorded. The aggregated statistics are also computed to allow fair, comparative analysis across combinations. The resulting summaries are used to identify both the overall best combination and technique-specific strengths and weaknesses, fulfilling the objective's requirement for systematic, statistically informed evaluation.

6.2 System Novelties

This project introduces a student-centred soft constraint, building continuity, that encourages consecutive classes for the same student to be scheduled within the same building. By penalising inter-building moves between back-to-back sessions, the system explicitly targets reduced walking time and less lost learning, without over-constraining the search space.

A second novelty is a modular GA framework in which selection, crossover, mutation, and replacement operators are fully interchangeable. This design enables a like-for-like evaluation of 64 operator combinations on a controlled dataset, yielding the first systematic comparison of these techniques focused on the university course timetabling problem (UCTP). The resulting evidence helps researchers choose effective operator stacks and design more rigorous experiments.

The study also proposes two novel replacement techniques, which are linear ranking replacement and weak chromosome replacement, highlighting the often-overlooked impact of the replacement phase on GA performance. Notably, one of these techniques, weak chromosome replacement, forms part of the best-performing genetic algorithm (GA) model discovered in the experiments, underscoring that replacement can be as decisive as selection or crossover in guiding convergence.

Beyond operator design, the system conducts a targeted grid search over crossover and mutation probabilities and adopts the best-found settings in the final runs. This closes the loop between architecture and tuning, ensuring that reported gains come from principled parameter choices rather than ad-hoc defaults, and giving the community reproducible baselines for future comparisons.

Finally, the project introduces a fairness-oriented performance metric, which is fitness improvement per generation, computed as initial penalty cost divided by the number of generations. Because runs terminate upon reaching a perfect (zero-penalty) timetable, final fitness cannot discriminate performance; raw generation counts are also biased by differing initial costs. The new metric normalises progress across runs, offering a clearer view of how efficiently each operator combination reduces violations.

Operationally, every crossover and mutation are followed by an immediate repair step that restores feasibility before the next evaluation. In GA-based UCTP studies, positioning repair as an always-on, post-operator mechanism is novel. It preserves the freedom of aggressive search operators while maintaining valid timetables at each generation, improving stability and speeding convergence.

6.3 System Limitations

At high resource utilisation (approximately 90%), the system can fail to produce a feasible timetable. This pressure might be induced primarily due to the experiments increase the number of students without adding lecturers, which raises the risk of breaching lecturer-related soft constraints, which is excessively long consecutive teaching hours.

Besides, the evaluation is confined to a single, partial mock dataset. While this enables controlled comparisons, it limits external validity: real institutions vary in room typologies, building layouts, lecturer availability patterns, class durations, and group structures. In particular, the same-building soft constraint is operationalised as a binary penalty rather than a campus-graph distance; this simplification ignores heterogeneity in inter-building travel, for instance, adjacent buildings versus distant ones.

Algorithmically, this project benchmarks only genetic algorithms; no comparisons are made against other optimisation paradigms such as integer or constraint programming, large neighbourhood search, tabu search, simulated annealing, or hyper-heuristics. As a result, the work cannot claim algorithmic superiority since there is only relative performance within GA variants. Even within GA, tuning focuses on crossover and mutation probabilities; other influential knobs such as population size and alternative stopping criteria (fixed time or evaluation budgets) are not systematically explored.

Other than that, the always-on post-operator repair improves feasibility but adds computational overhead and may bias search dynamics by regularly pulling individuals towards the same feasible basins, potentially reducing population diversity. Likewise, the room-by-room crossover safeguards room-type and capacity validity but constrains recombination granularity; inter-room exchanges that could yield better global timetables are inhibited, which may slow exploration or entrench substructures.

6.4 Future Enhancement/Improvement

To strengthen external validity, the future research can evaluate the system on diverse, real-world datasets (different faculties, multiple campuses, variable slot lengths, mixed class durations). This should include public UCTP benchmarks and institution-specific corpora with richer heterogeneity (room typologies, lecturer availabilities, group structures). Alongside the current same-building constraint, researchers can model campus travel using a weighted graph (distances/elevators/stairs), so that penalties reflect true movement cost rather than a binary building match.

Besides, the future research can broaden the optimisation method beyond GA and compare against integer or constraint programming, large neighbourhood search (LNS), tabu search, simulated annealing, and hyper-heuristics. Such cross-paradigm baselines will provide invaluable insights into the UCTP domain.

Chapter 7
Conclusion

This project set out to address a practical and recurrent challenge in higher education: building conflict-free, student-friendly university course timetables under diverse institutional constraints. It contributes three things in tandem: a richer formulation of the UCTP that explicitly penalises inter-building moves between consecutive classes, a modular genetic-algorithm (GA) framework whose operators are fully interchangeable, and an empirical study that systematically benchmarks 64 operator combinations under a controlled, partially realistic dataset.

Methodologically, the work shows that a carefully engineered GA, implemented in Java, backed by MySQL, and reinforced by an immediate repair step, can accommodate an extensive constraint set (15 hard constraint and 4 soft constraint) while still exploring the search space effectively. Decoupling selection, crossover, mutation, and replacement allows like-for-like comparisons and clearer attribution of performance differences to operator design rather than to confounded implementation details. The grid-searched crossover and mutation probabilities, together with the proposed "fitness improvement per generation" metric aligned to a zero-penalty stopping rule, provide fairer, more interpretable comparisons when initial penalty costs vary across runs.

Empirically, the study demonstrates that operator choice matters materially. Different selection, crossover, and replacement techniques meaningfully shift convergence speed and computational effort, and the two proposed replacement strategies broaden the perspective on how survivor selection influences progress. Crucially, integrating the same-building soft constraint proves tractable: the framework can find feasible, penalty-free timetables for the mock setting without collapsing under the added spatial preference, indicating that student-centric travel considerations can be folded into automated timetabling at modest additional cost.

Practically, the system produces complete, auditable artefacts, which are course, lecturer, student, and room timetables, and enforces key data-side invariants (class capacity) close to the database. This strengthens reproducibility and operational trust: results, errors, and metadata flow cleanly through a standardised JDBC-MySQL stack, while the dataset mirrors key aspects of a real CS programme timetable to keep experiments realistic yet controllable.

Like any focused study, this work has boundaries. Feasibility can degrade at very high resource utilisation; the dataset is a single, partial mock that cannot capture the heterogeneity of real campuses; and the algorithmic scope is intentionally limited to GA variants. These are honest constraints, not flaws, and they point directly to next steps: scaling to multi-cohort and multi-campus settings, modelling travel on a campus graph rather than with a binary penalty, and comparing against other optimisation paradigms. Beyond that, richer objectives, fairness across student groups, lecturer workload smoothness, and resilience to late changes, invite multi-objective or rescheduling extensions.

In summary, the project closes two gaps at once: it operationalises a neglected but student-meaningful spatial preference, and it offers the first systematic comparison of 64 GA operator combinations for UCTP within a unified codebase. The resulting insights and artefacts, which are constraint set, dataset, framework, operators, and metrics, form a solid, reusable foundation. They help timetable practitioners pick effective operator configurations with greater confidence and give researchers a clear runway for deeper studies that push from feasibility toward equity, realism, and robustness in automated university course timetabling.

REFERENCES

[1] S. Abdipoor, Razali Yaakob, Say Leng Goh, and S. Abdullah, “Meta-heuristic approaches for the University Course Timetabling Problem,” *Intelligent Systems with Applications*, vol. 19, pp. 200253–200253, Sep. 2023, doi: https://doi.org/10.1016/j.iswa.2023.200253.

[2] H. Babaei, J. Karimpour, and A. Hadidi, “A survey of approaches for university course timetabling problem,” *Computers & Industrial Engineering*, vol. 86, pp. 43–59, Aug. 2015, doi: https://doi.org/10.1016/j.cie.2014.11.010.

[3] T. Guilmeau, E. Chouzenoux, and V. Elvira, “Simulated Annealing: a Review and a New Scheme,” *2021 IEEE Statistical Signal Processing Workshop (SSP)*, pp. 101–105, Jul. 2021, doi: https://doi.org/10.1109/SSP49050.2021.9513782.

[4] Hatice Erdoğan Akbulut, Feriştah Özçelik, and Tuğba Saraç, “A simulated annealing algorithm for the faculty-level university course timetabling problem,” *Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi*, vol. 30, no. 1, pp. 17–34, Feb. 2024.

[5] F. H. Awad, A. Al-kubaisi, and M. Mahmood, “Large-scale timetabling problems with adaptive tabu search,” *Journal of Intelligent Systems*, vol. 31, no. 1, pp. 168–176, Jan. 2022, doi: https://doi.org/10.1515/jisys-2022-0003.

[6] V. K. Prajapati, M. Jain, and L. Chouhan, “Tabu Search Algorithm (TSA): A Comprehensive Survey,” *2020 3rd International Conference on Emerging Technologies in Computer Engineering: Machine Learning and Internet of Things (ICETCE)*, pp. 1–8, Feb. 2020, doi: https://doi.org/10.1109/ICETCE48199.2020.9091743.

[7] C. H. Wong, S. L. Goh, and J. Likoh, “A Genetic Algorithm for the Real-world University Course Timetabling Problem,” *2022 IEEE 18th International Colloquium on Signal Processing & Applications (CSPA)*, pp. 46–50, May 2022, doi: https://doi.org/10.1109/cspa55076.2022.9781907.

[8] K. Y. Junn, J. H. Obit, and R. Alfred, “The Study of Genetic Algorithm Approach to Solving University Course Timetabling Problem,” *Computational Science and*

Technology. ICCST 2017., pp. 454–463, Feb. 2018, doi: https://doi.org/10.1007/978-981-10-8276-4_43.

[9] Sk. I. Hossain, M. A. H. Akhand, M. I. R. Shuvo, N. Siddique, and H. Adeli, "Optimization of University Course Scheduling Problem using Particle Swarm Optimization with Selective Search," *Expert Systems with Applications*, vol. 127, pp. 9–24, Aug. 2019, doi: https://doi.org/10.1016/j.eswa.2019.02.026.

[10] A. G. Gad, "Particle Swarm Optimization Algorithm and Its Applications: A Systematic Review," *Archives of Computational Methods in Engineering*, vol. 29, no. 5, pp. 2531–2561, Apr. 2022, doi: https://doi.org/10.1007/s11831-021-09694-4.

[11] A. Mahmud, "Highly Constrained University Class Scheduling using Ant Colony Optimization," *International Journal of Computer Science & Information Technology (IJCSIT)* , vol. 13, no. 1, Feb. 2021, Available: https://ssrn.com/abstract=3801441

[12] M. C. Chen, S. N. Sze, S. L. Goh, N. R. Sabar, and G. Kendall, "A Survey of University Course Timetabling Problem: Perspectives, Trends and Opportunities," *IEEE Access*, vol. 9, pp. 106515–106529, Jul. 2021, doi: https://doi.org/10.1109/access.2021.3100613.

[13] W. Deng, J. Xu, and H. Zhao, "An Improved Ant Colony Optimization Algorithm Based on Hybrid Strategies for Scheduling Problem," *IEEE Access*, vol. 7, pp. 20281–20292, Feb. 2019, doi: https://doi.org/10.1109/access.2019.2897580.

[14] H. Zheng, Y. Peng, J. Guo, and Y.-C. Chen, "Course scheduling algorithm based on improved binary cuckoo search," *The Journal of Supercomputing*, vol. 78, no. 9, pp. 11895–11920, Feb. 2022, doi: https://doi.org/10.1007/s11227-022-04341-6.

[15] M. A. Jebur and H. S. Abdullah, "Timetabling problem solving based on best-nests cuckoo search," *Bulletin of Electrical Engineering and Informatics*, vol. 10, no. 6, pp. 3333–3340, Dec. 2021, doi: https://doi.org/10.11591/eei.v10i6.3206.

[16] J. Wahid and M. H. Naimah, "Hybrid harmony search with great deluge for UUM CAS curriculum based course timetabling," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 9, no. 1–2, pp. 33–38, Apr. 2017.

[17] F. Qin, A. M. Zain, and K.-Q. Zhou, "Harmony search algorithm and related variants: A systematic review," *Swarm and Evolutionary Computation*, vol. 74, p. 101126, Oct. 2022, doi: https://doi.org/10.1016/j.swevo.2022.101126.

[18] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia Tools and Applications*, vol. 80, no. 5, Oct. 2020, doi: https://doi.org/10.1007/s11042-020-10139-6.

[19] Seng Poh Lim and H. Haron, "Performance of Different Techniques Applied in Genetic Algorithm towards Benchmark Functions," *Lecture notes in computer science*, pp. 255–264, Jan. 2013, doi: https://doi.org/10.1007/978-3-642-36546-1_27.

[20] S. P. Lim, H. Hoon, and C. H. Ong, "Performance Comparison of Different Operation Techniques in Genetic Algorithm towards Benchmark Functions," *2018 8th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, pp. 59–64, Nov. 2018, doi: https://doi.org/10.1109/iccsce.2018.8684990.

[21] A. Shukla, H. M. Pandey, and D. Mehrotra, "Comparative review of selection techniques in genetic algorithm," *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, pp. 515–519, Feb. 2015, doi: https://doi.org/10.1109/ablaze.2015.7154916.

[22] S. N. Sivanandam and S. N. Deepa, *Introduction to Genetic Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. doi: https://doi.org/10.1007/978-3-540-73190-0.

[23] S. Karakatič and V. Podgorelec, "A survey of genetic algorithms for solving multi depot vehicle routing problem," *Applied Soft Computing*, vol. 27, pp. 519–532, Feb. 2015, doi: https://doi.org/10.1016/j.asoc.2014.11.005.

[24] S. Mirjalili, "Genetic Algorithm," *Studies in Computational Intelligence*, vol. 780, pp. 43–55, Jun. 2018, doi: https://doi.org/10.1007/978-3-319-93025-1_4.

[25] H. M. Pandey, "Performance Evaluation of Selection Methods of Genetic Algorithm and Network Security Concerns," *Procedia Computer Science*, vol. 78, pp. 13–18, 2016, doi: https://doi.org/10.1016/j.procs.2016.02.004.

[26] M. Lozano, F. Herrera, and J. R. Cano, “Replacement strategies to preserve useful diversity in steady-state genetic algorithms,” *Information Sciences*, vol. 178, no. 23, pp. 4421–4433, Dec. 2008, doi: https://doi.org/10.1016/j.ins.2008.07.031.

[27] L. Manzoni, L. Mariot, and E. Tuba, “Balanced crossover operators in Genetic Algorithms,” *Swarm and Evolutionary Computation*, vol. 54, p. 100646, May 2020, doi: https://doi.org/10.1016/j.swevo.2020.100646.

[28] S. M. Lim, A. B. Md. Sultan, Md. N. Sulaiman, A. Mustapha, and K. Y. Leong, “Crossover and Mutation Operators of Genetic Algorithms,” *International Journal of Machine Learning and Computing*, vol. 7, no. 1, pp. 9–12, Feb. 2017, doi: https://doi.org/10.18178/ijmlc.2017.7.1.611.

[29] U. A.J. and S. P.D., “CROSSOVER OPERATORS IN GENETIC ALGORITHMS: A REVIEW,” *ICTACT Journal on Soft Computing*, vol. 06, no. 01, pp. 1083–1092, Oct. 2015, doi: https://doi.org/10.21917/ijsc.2015.0150.

[30] Y. Jaradat, M. Masoud, I. Jannoud, A. Manasrah, and A. Zerek, “Comparison of Genetic Algorithm Crossover Operators on WSN Lifetime,” *2022 IEEE 2nd International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA)*, pp. 356–360, May 2022, doi: https://doi.org/10.1109/MI-STA54861.2022.9837587.

[31] N. Indrianti, R. A. C. Leuveano, S. H. Abdul-Rashid, and M. I. Ridho, “Green Vehicle Routing Problem Optimization for LPG Distribution: Genetic Algorithms for Complex Constraints and Emission Reduction,” *Sustainability*, vol. 17, no. 3, pp. 1144–1144, Jan. 2025, doi: https://doi.org/10.3390/su17031144.

[32] R. L. Kadri and F. F. Boctor, “An efficient genetic algorithm to solve the resource-constrained project scheduling problem with transfer times: The single mode case,” *European Journal of Operational Research*, vol. 265, no. 2, pp. 454–462, Mar. 2018, doi: https://doi.org/10.1016/j.ejor.2017.07.027.

[33] N. Rikatsih and W. F. Mahmudy, “Adaptive Genetic Algorithm Based on Crossover and Mutation Method for Optimization of Poultry Feed Composition,” *2018 International Conference on Sustainable Information Engineering and Technology (SIET)*, pp. 110–114, Nov. 2018, doi: https://doi.org/10.1109/siet.2018.8693167.

[34] S. Mirjalili, J. Song Dong, A. S. Sadiq, and H. Faris, “Genetic Algorithm: Theory, Literature Review, and Application in Image Reconstruction,” *Nature-Inspired Optimizers*, pp. 69–85, Feb. 2019, doi: https://doi.org/10.1007/978-3-030-12127-3_5.

[35] M. Abdel-Basset, R. Mohamed, M. Abouhawwash, V. Chang, and S. Askar, “A Local Search-Based Generalized Normal Distribution Algorithm for Permutation Flow Shop Scheduling,” *Applied Sciences*, vol. 11, no. 11, p. 4837, May 2021, doi: https://doi.org/10.3390/app11114837.

[36] N. Soni and T. Kumar, “Study of Various Mutation Operators in Genetic Algorithms,” *International Journal of Computer Science and Information Technologies (IJCSIT)*, vol. 5, no. 3, pp. 4519–4521, 2014.

[37] S. P. Lim and H. Haron, “Performance comparison of Genetic Algorithm, Differential Evolution and Particle Swarm Optimization towards benchmark functions,” *2013 IEEE Conference on Open Systems (ICOS)*, Dec. 2013, doi: https://doi.org/10.1109/icos.2013.6735045.

[38] H. Alghamdi, T. Alsubait, H. Alhakami, and A. Baz, “A Review of Optimization Algorithms for University Timetable Scheduling,” *Engineering, Technology & Applied Science Research*, vol. 10, no. 6, pp. 6410–6417, Dec. 2020, doi: https://doi.org/10.48084/etasr.3832.

[39] P. Kora and P. Yadlapalli, "Crossover Operators in Genetic Algorithms: A Review," International Journal of Computer Applications, vol. 162, no. 10, pp. 34-36, Mar. 2017, doi: 10.5120/ijca2017913370.

[40] I. H. Khan, “Assessing Different Crossover Operators for Travelling Salesman Problem,” International Journal of Intelligent Systems and Applications, vol. 7, no. 11, pp. 19–25, Oct. 2015, doi: 10.5815/ijisa.2015.11.03.

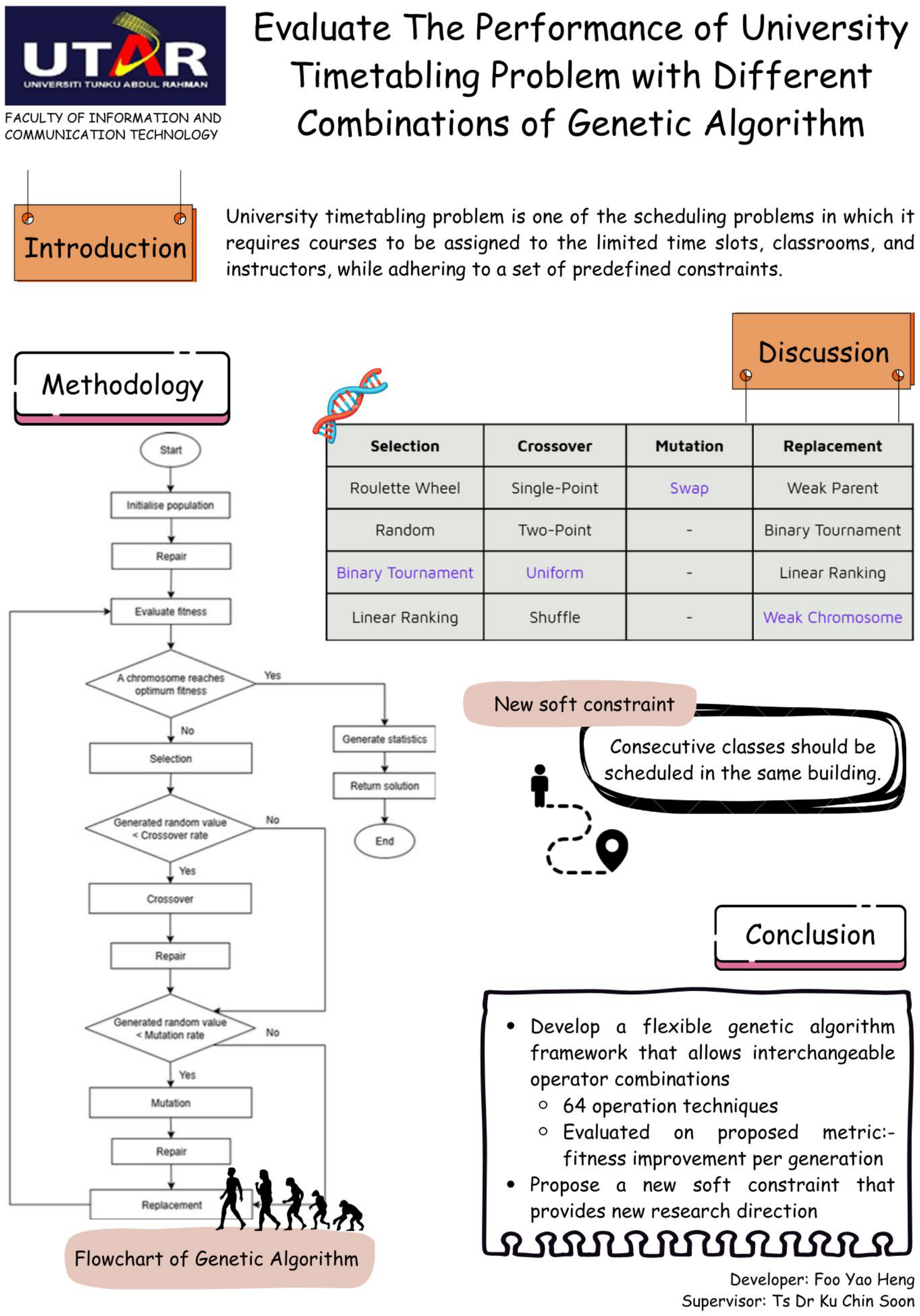
[41] S. Picek and M. Golub, “Comparison of a crossover operator in binary-coded genetic algorithms,” *WSEAS Transactions on Computers*, vol. 9, Sep. 2010, doi: https://doi.org/10.5555/1865335.1865350.

[42] F. J. Burkowski, "Shuffle crossover and mutual information," *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, Washington, DC, USA, 1999, pp. 1574-1580 Vol. 2, doi: 10.1109/CEC.1999.782671.

[43] P. Kora and P. Yadlapalli, "Crossover Operators in Genetic Algorithms: A Review," International Journal of Computer Applications, vol. 162, no. 10, pp. 1-6, Mar. 2017, doi: 10.5120/ijca2017913370.

APPENDIX

Poster



UNIVERSITI TUNKU ABDUL RAHMAN
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Evaluate The Performance of University Timetabling Problem with Different Combinations of Genetic Algorithm

Introduction

University timetabling problem is one of the scheduling problems in which it requires courses to be assigned to the limited time slots, classrooms, and instructors, while adhering to a set of predefined constraints.

Methodology

Flowchart of Genetic Algorithm

Discussion

Selection	Crossover	Mutation	Replacement
Roulette Wheel	Single-Point	Swap	Weak Parent
Random	Two-Point	-	Binary Tournament
Binary Tournament	Uniform	-	Linear Ranking
Linear Ranking	Shuffle	-	Weak Chromosome

New soft constraint

Consecutive classes should be scheduled in the same building.

Conclusion

- Develop a flexible genetic algorithm framework that allows interchangeable operator combinations
  - 64 operation techniques
  - Evaluated on proposed metric:- fitness improvement per generation
- Propose a new soft constraint that provides new research direction

Developer: Foo Yao Heng
Supervisor: Ts Dr Ku Chin Soon