International Journal of Computer Engineering and Data Science (ISSN: 2737-8543)
Volume 5–Issue 2, April - June 2026

Genetic Algorithm-Based University Course Timetabling: A Practical Optimization Framework with Graphical Interface

Nouhaila El Machichi[1], Fouad Kharroubi[2,*], Hiba Massous[1], Salma Ouali[1], Aymane El Youssfi[1]

[1]Department of Mathematics, ENS of Rabat, Mohammed V University, Rabat, Morocco.
[2,*]ENSIAS, Mohammed V University, Rabat, Morocco.

E-mail: nouhailamachichi@gmail.com, fouad.kharroubi@gmail.com, hibamassous1234@gmail.com, salmaouali2021@gmail.com, ay.elyoussfi@gmail.com

ORCID:
0009-0002-5547-3764[1], 0000-0002-1112-7774[2,*] 0009-0009-6233-0869[1], 0009-0006-0763-397X[1], 0009-0005-2979-4013[1]

Article history

Received Jun 16, 2026
Revised 29 Jun, 2026
Accepted 30 Jun, 2026
Published 01 Jul, 2026

ABSTRACT

This article investigates the university course timetabling problem, a challenging combinatorial optimization problem encountered by educational institutions when allocating courses to available time slots and classrooms while satisfying numerous institutional requirements. A mathematical formulation is proposed distinguishing between hard constraints (strictly enforced) and soft constraints (penalised to improve timetable quality). Owing to the NP-hard nature of the problem, exact optimization techniques become computationally prohibitive for large-scale instances. Consequently, a genetic algorithm is developed with constraint-aware operators and a repair mechanism to efficiently explore the search space and generate high-quality feasible timetables. The proposed approach is evaluated on two realistic datasets involving 15 and 69 courses, using 10 independent runs to ensure statistical robustness. Results show that the algorithm achieves 100% hard constraint satisfaction, an average room occupancy of 78%, with only minimal soft constraint violations (1 capacity overbooking and 2 room-type mismatches out of 15 courses), and generates feasible timetables in 45 seconds—compared to 4 hours of manual effort. A graphical user interface equipped with performance indicators is introduced to facilitate result interpretation and support administrative decision-making. These findings demonstrate that genetic algorithms provide an effective practical solution for university timetabling, with potential for deployment in medium-sized institutions.

Keywords: *Genetic algorithms; Timetabling; Metaheuristics; Constraint satisfaction; University course scheduling.*
(*): Corresponding Author

I. INTRODUCTION

The construction of timetables is a repetitive and sometimes complex task for educational institutions. It involves scheduling courses, instructors, classrooms, and student groups within a limited time period, subject to practical constraints such as room availability, class sizes, and teacher availability. As institutions grow and programs diversify, this process becomes increasingly difficult to manage manually, driving the adoption of computational methods and optimization techniques.

The growing need for efficiency, robustness, and adaptability drives the transition from manual to automated scheduling. Galli and Stille [1] identify three main categories of challenges in timetabling optimization: ensuring that optimization models are both versatile and comprehensive, handling operational disturbances, and integrating timetabling with other planning steps. Scheduling has many application areas, one of the most important being educational timetabling. This includes school timetabling (assigning lessons to time slots without conflicts), course timetabling (scheduling university lectures to avoid student overlaps), and examination timetabling (scheduling exams to minimize conflicts and distribute workload fairly). While these domains share common challenges, this article focuses specifically on the university course timetabling problem.

To address these complex combinatorial problems, researchers have increasingly adopted metaheuristic optimization techniques. Among these, genetic algorithms

(GAs) have proven particularly effective The International Timetabling Competition and other studies show that Genetic Algorithms are really good. They do well in areas, not just one. For example International Timetabling Competition editions and other independent studies look at how Genetic Algorithms work. They check things like how Genetic Algorithms meet the hard requirements how well they do with soft requirements and how long they take to run. When you look at all these things together Genetic Algorithms are often, among the best. This is what the International Timetabling Competition and other studies have found about Genetic Algorithms [2–4]. Inspired by the principles of natural evolution—selection, crossover, and mutation—GAs provide a powerful framework for exploring large solution spaces and identifying near-optimal solutions [5]. Their application to timetabling problems has demonstrated strong performance in generating high-quality schedules within reasonable computational time [6]. We want to make it clear that when we talk about the effectiveness of something we are looking at a lot of things. We consider things like quality and feasibility and time. We are not saying that one way is better than all the others. For problems it is still best to use exact methods.. For some specific problems single solution metaheuristics can be just as good, as Genetic Algorithms. [7].

A. Problem Statement and Research Objectives

The university course timetabling problem (UCTP) consists of assigning a set of courses to available time slots and rooms while satisfying various institutional requirements. The primary challenge lies in the NP-hard nature of the problem, which makes exact optimization methods computationally prohibitive for real-world instances. Despite extensive research, several gaps remain: (i) the gap between theoretical models and practical implementations, (ii) the need for user-friendly tools accessible to non-expert administrators, and (iii) the lack of comprehensive evaluation frameworks that combine solution quality with practical usability.

The main objectives of this study are: (1) to develop a mathematical formulation that captures both hard and soft constraints of university timetabling, (2) to design and implement a genetic algorithm that efficiently generates high-quality feasible timetables, (3) to develop a graphical user interface that facilitates the practical use of the optimization system, and (4) to evaluate the proposed approach on realistic datasets and compare it with manual scheduling practices.

B. Contributions

The main contributions of this work are:

1. A complete mathematical formulation of the university course timetabling problem with clearly defined hard and soft constraints.
2. An implementation of a genetic algorithm with constraint-aware operators and a repair mechanism specifically designed for educational timetabling, where capacity and room suitability constraints are treated as soft constraints with high penalty weights to reflect real-world flexibility.
3. A graphical user interface that bridges the gap between algorithmic research and administrative practice.
4. Experimental evaluation on realistic datasets (15 and 69 courses) with 10 independent runs and comparative analysis against manual scheduling.
5. A statistical dashboard providing real-world performance indicators for decision-making support.

C. Paper Organization

The remainder of this article is organized as follows. Section 2 provides a comprehensive review of related work in educational timetabling, including high school, course, and examination timetabling, with a synthesis table comparing their characteristics. Section 3 presents the mathematical formulation of the timetabling problem, including sets, parameters, decision variables, hard constraints, soft constraints, and the complete optimization model. Section 4 describes the proposed genetic algorithm, including chromosome representation, fitness function, genetic operators, and the graphical user interface. Section 5 presents the experimental results on both small and large instances. Section 6 discusses the findings, compares with other approaches, and addresses limitations. Finally, Section 7 concludes the paper and outlines directions for future research.

II. Related Work

A. Understanding the Timetabling Problem

The educational timetabling problem is a challenging computational issue [8] of significance to researchers in operations research and artificial intelligence. Academic scheduling has several variations, including high school timetabling, university course timetabling, and examination timetabling [9, 10]. While these domains share common challenges, this article focuses specifically on university course timetabling, which is characterized by diverse student enrollments, flexible curricula, and complex conflict detection.

Wren [11] defined timetabling as "the allocation of given resources to specific objects being placed in space-time, in such way as to satisfy as nearly as possible a set of desirable objectives, subjected to constraints." The availability of knowledge for developing timetabling

software has resulted in enhanced techniques and more thorough models [12]. These advancements have promoted new methods for space utilization and interactive scheduling [13]. However, the range of constraints, the complexity of the issue, and specific institutional needs have made it increasingly challenging to discover a universal solution [14].

The timetabling problem is classified as NP-hard and NP-complete [15], indicating that many timetabling problems cannot be solved in polynomial time. Computational heuristics are therefore commonly used to obtain feasible near-optimal solutions [16]. Despite extensive research, a disconnect remains between theory and practice. McCollum and Ireland [12] noted that finding a universally applicable model is challenging as different institutions have different constraints. Numerous studies utilize specific datasets [17], limiting generalizability.

B. Classification of Timetabling Domains

Figure 1 provides a visual taxonomy of these three main categories of educational timetabling problems.

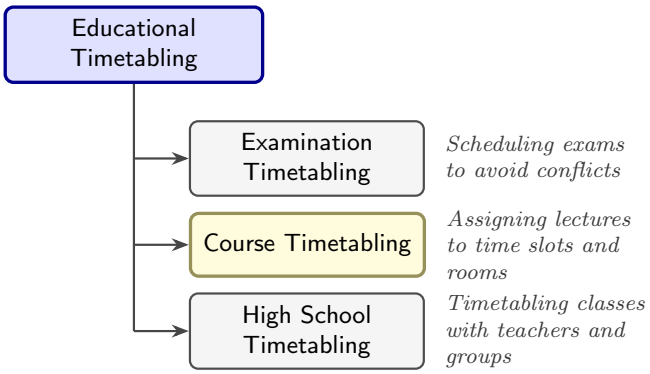

Figure 1: Classification of educational timetabling problems.

1. High School Timetabling

As illustrated in Figure 1, educational timetabling is divided into three main categories. In high school timetabling, a set of events of arbitrary duration is given with resources: students, classes, teachers, and rooms. The problem is to assign values to avoid clashes while satisfying hard constraints (e.g., no overlapping events) and soft constraints (e.g., minimizing idle periods). The problem is NP-hard, so heuristic and metaheuristic approaches are commonly applied [18]. Schaerf [19] describes a local search algorithm with adaptive relaxation of hard constraints, successfully tested in several large high schools.

2. Course Timetabling

Course timetabling assigns university lectures to time slots and rooms, minimizing conflicts between courses sharing students [20]. The university context is more complex than high school due to diverse elective/compulsory module combinations. The problem is NP-hard [4].

Constraints are classified as hard (strictly enforced) and soft (preferences with penalty costs) [17]. In our implementation, capacity (C4) and room suitability (C6) constraints are treated as soft with high penalty weights to reflect real-world flexibility, while unique assignment (C1), teacher conflicts (C2), student conflicts (C3), and room occupancy (C5) are strictly enforced as hard constraints.

Metaheuristic approaches include genetic algorithms, simulated annealing, and tabu search [7]. Hybrid approaches combining metaheuristics with integer programming or constraint propagation have demonstrated strong performance [21]. The CB-CTT (Curriculum-Based Course Timetabling) formulation from the International Timetabling Competition (ITC) serves as an industry-standard benchmark [2]. Recent research has explored hyper-heuristics and machine learning to improve adaptability [22].

3. Examination Timetabling

Examination timetabling assigns examinations to time periods and rooms while satisfying hard constraints (e.g., no student conflicts, room capacity) and soft constraints (e.g., reducing successive exams) [22]. The problem is NP-hard, making exact methods impractical for large instances [23]. Hybrid approaches combining constraint programming and local search have been explored [24]. However, examination timetabling remains institution-specific, requiring flexible and scalable methods [2].

C. Research Progress and Synthesis

Timetabling problems have been extensively studied in combinatorial optimization due to their practical relevance and computational complexity. A central aspect lies in the definition and handling of constraints, which determine the feasibility and quality of any generated schedule.

Constraints are classified into two main categories. Hard constraints define the essential feasibility requirements and must be strictly satisfied; violations render a solution infeasible. These include preventing student conflicts, avoiding exceeding room capacities, ensuring exclusive resource assignment, and respecting fixed scheduling requirements [17, 20]. Soft constraints correspond to preference-based conditions that improve

timetable quality—reducing back-to-back events, ensuring even spread of events, respecting preferred time slots, and accommodating room preferences. These are violated at a penalty cost in the objective function, allowing models to reflect real-world requirements where feasibility alone is insufficient [25]. This hard/soft distinction is fundamental across timetabling domains [4, 17].

Research has been heavily influenced by benchmark problems such as Curriculum-Based Course Timetabling (CB-CTT) from the International Timetabling Competition (ITC). These benchmarks enable standardized evaluation of algorithmic performance and facilitate comparable solution approaches [3, 26]. Consequently, the field has seen diverse metaheuristic and hybrid methods with explicit constraint handling mechanisms for feasible and optimal solutions [22].

D. Chronological Evolution of Solution Methods

Apart from simply enumerating the constraints, it is worth investigating how the algorithms and techniques used to solve educational timetabling problems have been developed and how improvements have been made over each other. The initial attempts at solving the educational timetabling problem were made by modelling it as a*graph-colouring* problem, whereby the nodes represented events, the edges connected those events sharing a common teacher, student set or classroom, and colouring was the proper way to assign conflicting events to different timeslots [8, 27]. Graph colouring was a neat abstract formulation and allowed efficient constructional algorithms to be devised, but it could not represent room size, suitability or preferences. *Integer Programming / Mixed Integer Programming* (IP/MIP) was specifically designed in order to address such problems: Daskalaki et al. [28] modeled university scheduling as an IP which precisely captures all capacity, ordering and assignment constraints, providing optimality guarantees for smaller instances — but the exponential explosion of the model makes it infeasible even with just a few dozens of courses. In order to make IP scale, the research community had to resort to *metaheuristics* (Simulated Annealing, Tabu Search, Genetic Algorithms) [5, 7], sacrificing optimality guarantees in favor of being able to explore vast spaces; GA specifically improved upon one-shot metaheuristics by using a pool of solutions [6]. *Hyper-heuristics* however, took the abstraction level further by conducting search over the space of (lower-level) heuristics rather than solutions, making the algorithms more generalizable across instances while requiring less tuning for each data set than classical metaheuristics [29, 30].In recent times, *machine-learning and reinforcement-learning* modules have been integrated into metaheuristics to guide the choice of operators and parameters. For instance, Pan et al. [31] used reinforcement learning to dynamically adjust a GA and showed that it resulted in faster convergence compared to the base static GA, while more advanced GAs were able to substantially reduce conflicts [32]. Currently, *deep reinforcement learning* and *large language model*-based approaches are considered as scheduling models using data-driven methods, but their reproducibility and constraint-feasibility guarantees remain open questions.The clear evolutionary path is that each generation refined the previous one on a particular dimension: graph coloring → expressiveness (IP) → scalability (metaheuristics) → generalization (hyper-heuristics) → adaptability (machine learning, deep reinforcement learning, large language model). This particular genetic algorithm presented in this article is situated at the metaheuristics phase of this evolution, purposely enhanced by a repair operator which brings back the constraint satisfaction rigors of the IP phase.

To conclude, three practical lessons that stem directly from this literature and inform the proposed design in the following sections, replacing any general claim on the challenge of combining hard and soft constraints, are the following: (i) feasibility has to be ensured either constructively or explicitly by means of a repair phase, since one violation of a hard constraint would make the resulting timetable inapplicable; (ii) soft criteria have to be treated by means of a weighted objective function with weights reflecting the institutional preferences and adjustable on a per-institution basis; and (iii) population-based search complemented with a repair phase is the strategy with the best feasibility vs quality vs running time trade-off in the case of small and medium-size instances—exactly as in Sections 3 and 4.

Table 1: Comparison of educational timetabling domains

Domain	Key Characteristics	Main Constraints	Common Approaches
High School	Fixed student groups, weekly schedule	Teacher/student conflicts, room availability	Local search, GA
Course	Diverse enrollments, flexible curricula	Lecturer conflicts, room capacity, student conflicts	GA, SA, TS, Hyper-heuristics
Examination	Single event per course, limited slots	Student conflicts, room capacity, exam spreading	SA, GA, CP, Hybrid

In our implementation, four constraints are strictly enforced as hard: unique assignment (C1), teacher conflicts (C2), student conflicts (C3), and room occupancy (C5). Capacity (C4) and room suitability (C6) are treated as soft constraints with high penalty weights (10 and 5 respectively), reflecting real-world flexibility where overbooked rooms or unsuitable venues may be unavoidable but strongly discouraged.

This synthesis of the literature informs the mathematical formulation and genetic algorithm design presented in the following sections. The next section formalizes the timetabling problem through sets, parameters, decision variables, and the complete optimization model.

III. MATHEMATICAL FORMULATION OF THE TIMETABLING PROBLEM

In order to apply optimization techniques such as Genetic Algorithms (GAs), the timetabling problem must first be expressed as a mathematical optimization model [17,20]. This formulation defines the input data, decision variables, constraints, and objective function that characterize the scheduling problem [33]. The resulting model serves as the foundation for implementing heuristic and metaheuristic optimization approaches [5,34].

A. Sets and Parameters

Let the following sets be defined [35]:

- E : set of events to be scheduled
- T : set of available time slots
- R : set of available rooms
- S : set of student groups
- P : set of teachers or instructors

For each event $e \in E$ the following parameters are known:

- $duration(e)$: number of consecutive time slots required by event e
- $size(e)$: number of students attending event e
- $teacher(e) \in P$: teacher assigned to event e
- $group(e) \subseteq S$: set of student groups attending event e

Each room $r \in R$ is associated with a maximum capacity denoted by $cap(r)$.

B. Decision Variables

The timetabling assignment is represented using the following binary decision variable [14,17]:

$$x_{e,t,r} = \begin{cases} 1 & \text{if event } e \text{ is scheduled in time slot t and room } r \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

For events requiring multiple consecutive periods, the formulation may either use additional linking constraints or define variables corresponding to the starting time slot of the event [3].

C. Hard Constraints

Hard constraints define the feasibility conditions of the timetable. Any violation of these constraints results in an infeasible solution [20,33].

C1 - Unique Assignment Constraint. Each event must be assigned exactly once [17]:

$$\sum_{t \in T} \sum_{r \in R} x_{e,t,r} = 1, \qquad \forall e \in E \tag{2}$$

C2 - Teacher Conflict Constraint. A teacher cannot be assigned to more than one event during the same time slot [10,19]:

$$\sum_{e \in E_p} \sum_{r \in R} x_{e,t,r} \leq 1, \qquad \forall t \in T, \forall p \in P \tag{3}$$

where $E_p = \{e \in E \mid teacher(e) = p\}$.

C3 - Student Conflict Constraint. A student group cannot attend multiple events simultaneously [17,20]:

$$\sum_{e \in E_s} \sum_{r \in R} x_{e,t,r} \leq 1, \qquad \forall t \in T, \forall s \in S \tag{4}$$

where $E_s = \{e \in E \mid s \in group(e)\}$.

C5 - Room Occupancy Constraint. A room cannot host more than one event during the same time slot [10,17]:

$$\sum_{e \in E} x_{e,t,r} \leq 1, \qquad \forall t \in T, \forall r \in R \tag{5}$$

D. Soft Constraints and Objective Function

Unlike hard constraints, soft constraints correspond to preferences intended to improve timetable quality. Violations are permitted but penalized through the objective function [2,20].

In our implementation, the following soft constraints are considered:

- **C4 - Room Capacity Constraint (soft).** The assigned room should accommodate all students attending the event. Occasional overbooking is permitted but penalized with weight $w_4 = 10$ [13,36]:

  Penalty if $x_{e,t,r} \cdot size(e) > cap(r)$

- **C6 - Room Suitability Constraint (soft).** Events requiring specialized rooms should be assigned to suitable rooms when possible. Mismatches are penalized with weight $w_6 = 5$ [4,11]:

  Penalty if $x_{e,t,r} = 1$ and $r \notin R_e$

- minimizing gaps between classes for students
- respecting preferred teaching periods
- clustering related courses together
- balancing workload distribution across the week
- minimizing consecutive lectures or examinations

Each soft constraint violation is associated with a penalty weight w_k. The overall objective function is therefore defined as [7, 17]:

$$F(x) = \sum_{k=1}^{K} w_k \cdot v_k(x) \tag{6}$$

where:

- $v_k(x)$ represents the number or severity of violations associated with soft constraint k
- w_k denotes the importance weight assigned to that constraint

The penalty weights used in our implementation are: capacity violation ($w_4 = 10$), room type mismatch ($w_6 = 5$), teacher preferences ($w = 2$), and student gaps ($w = 1$). These weights were empirically determined to reflect the relative importance of each constraint, with capacity violations considered the most critical.

The optimization objective consists of minimizing the total penalty value while maintaining feasibility with respect to all hard constraints [3, 14].

E. Complete Optimization Model

The academic timetabling problem can therefore be formulated as the following combinatorial optimization problem [15, 20, 33]:

$$\min F(x) \tag{7}$$

subject to: hard constraints C1, C2, C3, C5, and any additional institution-specific scheduling requirements.

Because the timetabling problem is NP-hard [15, 16], exact optimization methods become computationally expensive for large real-world instances. Consequently, metaheuristic approaches such as Genetic Algorithms are widely adopted to efficiently explore the search space and generate near-optimal feasible timetables [5, 34, 37].

In the GA framework, each chromosome represents a timetable assignment encoded through the decision variables $x_{e,t,r}$. The fitness function is generally derived from the objective function and aims to minimize soft constraint penalties while preserving feasibility. A common fitness definition is [5, 17]:

$$\text{Fitness}(x) = \frac{1}{1 + F(x)} \tag{8}$$

or equivalently the negative value of $F(x)$, depending on the optimization strategy used.

In order to explicitly express how *hard* constraints are treated in the formulation stage and not just in the implementation stage, we define a two-level (death penalty) fitness function. We let $H(x) = \sum_{c \in \{C1,C2,C3,C5\}} h_c(x)$ be the sum of the number of hard-constraint violations in a timetable x with respect to the hard constraint c.

$$\text{Fitness}(x) = \begin{cases} \dfrac{1}{1 + \sum_k w_k \, v_k(x)} & \text{if } H(x) = 0 \text{ (feasible)}, \\ \dfrac{\varepsilon}{1 + M \cdot H(x)} & \text{if } H(x) > 0 \text{ (infeasible)}, \end{cases} \tag{9}$$

where M is a very large constant such that $M \gg \max_k w_k$ and $\varepsilon \to 0^+$. So, individuals with even one hard constraint receive near-zero fitness values and are practically removed from the population by the selection pressure; only the *soft* constraint violations v_k are taken into account in the fitness function via the weighted sum. In other words, hard constraints are used as a multiplicative or death penalty (feasibility gate), whereas soft constraints are used as an additive penalty (quality grading); this is the formal definition of the term "hard" and "soft" used in the whole paper. Practically, however, the repair operator (see Section 4, Algorithm 1, line 9) fixes almost all constraints violations prior to the evaluation.

IV. PROPOSED GENETIC ALGORITHM

A. Overview of Solution Paradigms

The literature offers a variety of solution paradigms, including exact methods such as integer programming, hyperheuristics and hybrid approaches. In this article we deal with genetic algorithms (GAs). GAs are especially well-suited for timetabling because they can search large and complex solution spaces and deal with multiple constraints at once.

1. Exact Methods

Exact methods, such as Integer Programming (IP), Mixed Integer Programming (MIP) and Constraint Programming (CP), seek to find the best solution to a problem by thoroughly searching through the set of potential solutions using a mathematically precise method. These methods are created with the use of sophisticated solvers such as CPLEX and Gurobi for IP/MIP and Choco or OR-Tools for CP. All of these methods have an excellent ability to ensure feasible and optimal solutions, particularly for small to medium sized examples. In particular, CP has an expressive and adaptable method of model representation that provides a means to directly express timetabling constraints (such as conflicts and capacity) without the need to linearize [17]. However, exact methods have a severely limited practical applicability due to their extreme computational complexity.

The timetabling problem is nondeterministic polynomial (NP-hard) [15], indicating that the number of feasible solutions (as a function of instance size) grows exponentially. For example, a typical university setting with 200 exams and 20 possible time slots will have a huge number of possible schedules (approximately equal to 20^{200}) making it infeasible within reasonable time bounds to use exact methods on large scale problems. Thus, exact methods are typically only used on small problems, or as part of hybrid methods like combining CP with local search [24]. For challenging or larger problems in scheduling, researchers and practitioners usually use heuristic or meta-heuristic methods to trade optimality for computational efficiency, aiming to obtain high-quality solutions within acceptable time limits compared to the amount of time it takes to develop an optimal solution.

2. *Heuristic and Metaheuristic Methods*

The NP-hard characteristics of timetabling problems result in the inaccessibility of exact optimization techniques when dealing with large-scale, real-world instances. As a result, heuristic and metaheuristic techniques have been adopted widely for use in finding near-optimal solutions to these problems in a reasonable amount of time. Unlike exact methodologies that are guaranteed to produce global optima, heuristic and metaheuristic methods will provide good feasible solutions by effectively exploring the solution space.

The heuristic method is typically characterized by being problem-specific, where knowledge of the domain has been used to create a strategy for quickly developing or improving feasible solutions. Specific examples would be the development of graph-coloring heuristics, sequential assignment techniques, and greedy scheduling methods. While heuristics are an efficient method for developing a good feasible solution, they can limit the amount of exploration available to the algorithm and are quite likely to become stuck in a poor local optima.

Metaheuristic algorithms are being used extensively by the research community to overcome the exploration limitations of heuristics by providing a greater degree of freedom within the solution space as well as a global search mechanism. The most widely utilized metaheuristics within the educational timetable literature include Simulated Annealing (SA), Tabu Search (TS), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and Hyper-Heuristics [7,22,23]. To efficiently search large and heavily constrained search spaces, these methods create a balance between exploration and exploitation.

To give a view that does not focus only on genetic algorithms each of these methods is described one by one below. Simulated Annealing accepts moves with a probability that is controlled by temperature, which helps Simulated Annealing to get out of local optima. This has made Simulated Annealing one of the methods that use a single solution especially when it comes to creating examination timetables. Simulated Annealing is really good, at this because it can escape optima. [23]. *Tabu Search* (TS) method really helps with finding solutions by not allowing the system to try things that were just tried. It does this by keeping a list of things that were just done called a tabu list. People often say that this method is just as good as or even better than the simulated annealing method when it comes to making schedules for classes. The tabu search method is very useful for things, like course timetabling. [7]. *Ant Colony Optimization* (ACO) is a way to find solutions by using trails that get stronger when they lead to results. These trails are like signs that say this path is an one. Particle Swarm Optimization is another method that uses a group of schedules and moves them around based on the best schedules found so far. Both Ant Colony Optimization and Particle Swarm Optimization work with a group of solutions like Genetic Algorithm does but they share information in different ways. Ant Colony Optimization and Particle Swarm Optimization are good, at finding the solutions because they use the whole group to help each other out. [22]. Constructive methods like GRASP and Variable Neighbourhood Search are really useful. These methods use an approach and then improve it. Variable Neighbourhood Search switch between building a solution and making small changes to it. When people compare these methods they find that no one method is always the best. The performance of GRASP and Variable Neighbourhood Search depends on the problem and the rules that need to be followed. This is why people are now combining methods, like GRASP and Variable Neighbourhood Search to create something even better. [22, 30]. The genetic algorithm is used in this work because it works well with a repair operator. This is what we need for the sized instances we are looking at. The genetic algorithm is not the solution for every problem but it is a good choice here because it uses a group of solutions and combines them in a natural way. This gives us a balance between finding solutions that work and finding the best solutions. The genetic algorithm is a fit, for the medium-sized instances we are targeting.

Comparative studies in other NP-hard domains, such as routing and wavelength assignment in optical networks, have also shown that genetic algorithms and tabu search can effectively balance exploration and exploitation [38, 39]. Although these studies address optical network optimization, they are relevant because they validate the behavior of GA and TS on other NP-hard combinatorial problems with complex constraints, supporting their applicability to timetabling. Furthermore, even simpler random optimization algorithms have been applied to problems like static manycast RWA, achieving reasonable solutions without complex parameter tuning [40].

Table 2: Comparison of solution approaches for timetabling

Approach	Strengths	Weaknesses
Genetic Algorithm	Flexible, handles multiple constraints, finds near-optimal solutions quickly	Parameter sensitivity, no guarantee of global optimum
Exact (IP/CP) methods	Guaranteed optimal for small instances	NP-hard for $>$ 30 courses
Manual planning	Full control, transparent	Time-consuming, inconsistent quality
Simulated Annealing / Tabu Search	Good local exploitation	May get trapped in local optima

B. Genetic Algorithm Components

GAs perform a population-based search using selection, crossover, and mutation operators to iteratively improve candidate timetables [5,6]. The main components of our implementation are:

- **Encoding:** Each chromosome is a vector of length $|E|$ where each gene encodes a pair (room, time_slot) for a specific course. This ensures unique assignment (constraint C1) is automatically satisfied.
- **Fitness Function:** Defined in Equation (10), it minimizes the weighted sum of soft constraint violations, with hard violations rendered infeasible (very low fitness).
- **Selection:** Tournament selection (tournament size = 3) balances pressure and diversity.
- **Crossover:** Single-point crossover, applied with probability 0.9.
- **Mutation:** With probability 0.05, a randomly selected gene is reassigned a valid (room, time_slot) pair.
- **Repair Operator:** After crossover and mutation, the repair operator checks for hard constraint violations (teacher conflicts, room occupancy) and resolves them by reassigning conflicting events to free slots.
- **Elitism:** The best solution is preserved in each generation to prevent quality degradation.

Several advanced GA variants have been proposed to further improve timetabling performance. Recursive Genetic Algorithms iteratively refine subproblems [6], while Guided Search Genetic Algorithms incorporate heuristic information to reduce infeasible offspring [14]. Our implementation focuses on a practical, constraint-aware design with a dedicated repair operator, making it particularly suitable for real-world university scheduling.

Beyond describing these parts it is worth looking into why the design choices are important. They really affect how doable and good the final result is. There are three ways to encode timetables in studies:

- Direct encodings, where each part stores the room and time slot of an event. This is the method used here. It makes sure that each event has a time and room but it can create problems, with teachers and rooms that have to be fixed.
- Permutation encodings, where the order of events is decided first and then placed by a decoder. These work well to keep things but can make it hard to find good solutions because many things depend on each other.
- Grouping encodings, which group events that happen together and work well for scheduling rooms and periods.. They need special methods to work properly. [4,6].

The reason why naive crossover is a problem is that it can mess up the schedule. When you combine two schedules it can put two events with the same teacher or in the same room, at the same time. This is an issue. That is why people usually use crossover with methods that keep the schedule feasible or they use a repair step like we do to fix the problems that crossover causes with the naive crossover method and the schedule. [14].When it comes to operators people usually choose tournament selection. They often use it with single-point or uniform crossover. These are the methods that people talk about the most. Some studies have compared these methods. They found that having a repair or local-search step is really important for getting good results. This step is more important than the selection or crossover method that people use. Operators, like these are used with tournament selection and point or uniform crossover. [6,41]. These findings from the analysis. Than just doing things the usual way. Show that the design used in this study is a good idea.

The design is aware of the constraints. It tries to repair things, which is what makes it work. The findings, from the analysis justify the design used in this study.

C. Genetic Algorithm Optimization Process

The optimization flow follows the standard GA cycle illustrated in Figure 2. The algorithm starts with a randomly initialized population of timetables, ensuring diversity to avoid premature convergence [14]. Fitness evaluation follows the penalty-based formulation defined in Section 3.4. Parent selection uses tournament selection, crossover and mutation are applied as described above, and the repair operator ensures feasibility. The process continues until a stopping criterion is met (maximum generations or convergence).

Figure 3 illustrates the conceptual distinction between hard and soft constraints in the optimization process. Algorithm 1 presents the complete pseudocode of our GA implementation.

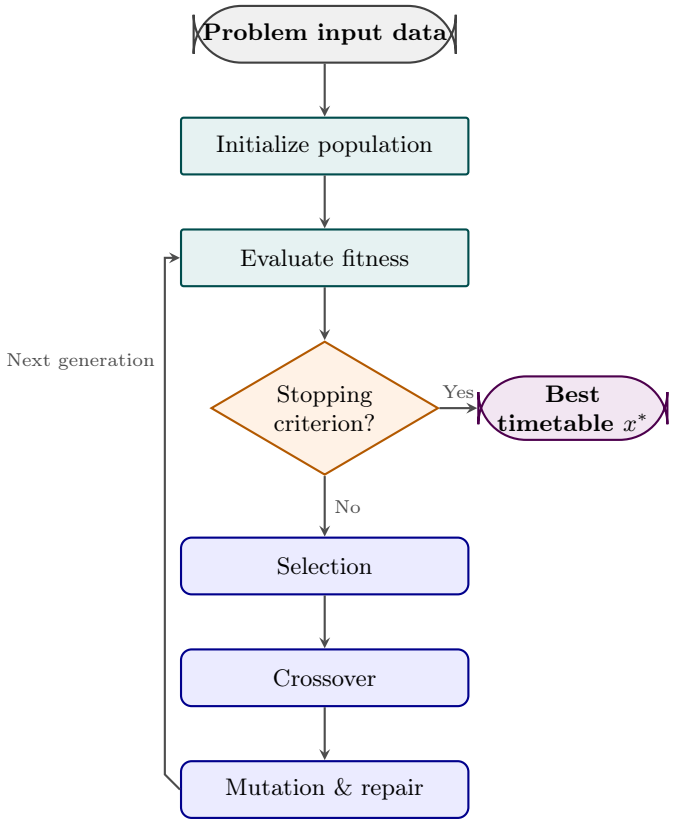

Figure 2: Flowchart of the genetic algorithm optimization process.

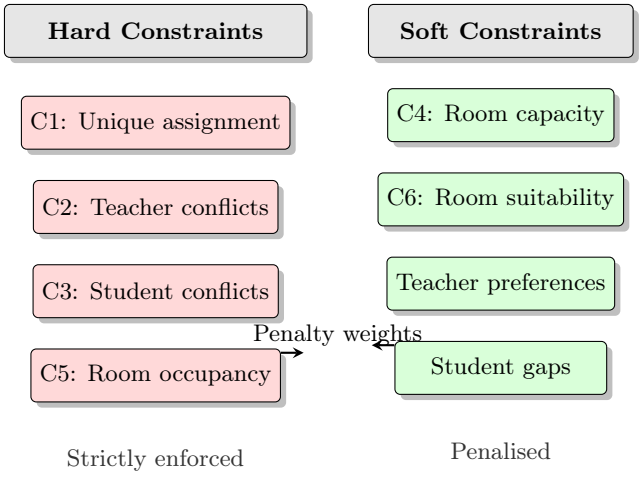

Figure 3: Conceptual distinction between hard and soft constraints in timetabling optimization.

Algorithm 1 Genetic Algorithm for University Course Timetabling

Require: Population size N, max generations G_{max}, crossover rate P_c, mutation rate P_m
Ensure: Best feasible timetable x^*

Initialization:
Generate initial population P_0 of size N with random valid assignments
Evaluation:
Compute fitness $f(x)$ for each individual using Eq. (10)
Evolution:
for $gen = 1$ to G_{max} **do**
    Selection: Select parents using tournament selection (size 3)
    Crossover: Apply single-point crossover with rate P_c to generate offspring
    Mutation: Apply mutation with rate P_m (reassign random gene)
    Repair: Enforce hard constraints (C1, C2, C3, C5) by reassigning conflicting events
    Evaluation: Compute fitness for offspring
    Elitism: Preserve best individual from previous generation
    Replace population with new generation
end for
return Best individual x^* with highest fitness

This algorithm ensures that all hard constraints are satisfied through the repair operator, while soft constraints are minimized via the fitness function. The combination of constraint-aware operators and a repair mechanism is the key innovation of our implementation, enabling the generation of high-quality feasible timetables for real-world instances.

D. Implementation Details

1. Problem Instance and Data

The algorithm was evaluated on two realistic university scheduling instances. The first instance comprises 15 courses, 8 rooms, and 10 time slots (2 per day over 5 days). The second, larger instance consists of 69 courses and 25 rooms, reflecting a more complex departmental scheduling scenario.

Each course is characterised by:

- Required room capacity (student count)
- Course type (CM, TD, TP)
- Assigned teacher
- Optional time and room preferences

Table 3 summarises the key characteristics of both datasets. All experiments were conducted with 10 independent runs using different random seeds to ensure statistical robustness. Results are reported as average values with standard deviations where applicable (see Table 6).

Table 3: Characteristics of the datasets used for evaluation

Characteristic	**Instance 1 (Small)**	**Instance 2 (Large)**
Number of courses	15	69
Number of rooms	8	25
Room capacities	20–100 seats	20–200 seats
Time slots	10 (2 per day × 5 days)	10 (2 per day × 5 days)
Course types	CM, TD, TP	CM, TD, TP
Number of teachers	8	32
Number of student groups	6	18
Time/room preferences	Optional	Optional

2. Chromosome Representation

A direct representation is used: each chromosome is a vector of length equal to the number of courses. Each gene encodes a pair (room, time_slot) for a specific course. This representation ensures that every course appears exactly once, automatically satisfying hard constraint C1 (unique assignment).

3. Constraint Classification

Following the mathematical formulation in Section 3, constraints are classified as follows:

Hard constraints (strictly enforced):

- **C1:** Unique assignment — each event assigned exactly once
- **C2:** Teacher conflicts — no teacher assigned to multiple events simultaneously
- **C3:** Student conflicts — no student group attends multiple events simultaneously
- **C5:** Room occupancy — no room hosts more than one event per time slot

Soft constraints (penalised in the fitness function):

- **C4:** Room capacity — each room must accommodate all attending students (penalty weight = 10)
- **C6:** Room suitability — events requiring specialised rooms must be assigned appropriately (penalty weight = 5)
- Teacher time preferences (penalty weight = 2)
- Student gap minimisation (penalty weight = 1)

This distinction reflects practical university scheduling: while capacity and room-type mismatches are undesirable, they are occasionally unavoidable and can be resolved through manual adjustments. Treating them as soft constraints with high penalty weights allows the algorithm to find feasible schedules in tightly constrained instances while strongly discouraging violations.

4. Fitness Function

The fitness function evaluates a timetable by penalising violations of soft constraints. Hard constraints (C1, C2, C3, C5) are strictly enforced: any violation renders the solution infeasible, and such individuals are assigned a very low fitness value.

Fitness is defined as:

$$\text{Fitness} = \frac{1}{1 + \sum_k w_k \cdot v_k} \tag{10}$$

where:

- v_k is the number of violations of soft constraint k
- w_k are empirically determined penalty weights:
  - Capacity violation: $w = 10$
  - Room type mismatch: $w = 5$
  - Teacher preference: $w = 2$
  - Student gaps: $w = 1$

Let's set the record straight on hard constraints and how they show up in this evaluation. Equation (10) is only about soft constraint violations—it ignores the hard ones. You won't see hard constraints just added into the mix with a huge weight or anything like that. They act as strict gatekeepers thanks to the death-penalty rule laid out in Equation (9): how it works: If someone racks up even one hard violation ($H(x) > 0$), their fitness gets crushed to something like $\varepsilon/(1 + M \cdot H(x))$, with M set way higher than any of the soft constraint weights. So, no matter how great the rest of the scores look, a hard constraint slip-up wipes out the fitness—no exceptions. There are a few ways we handle hard constraints: First, the encoding step always guarantees C1 is satisfied.

Next, the repair operator fixes C2, C3, and C5 before we even start scoring anyone. And if something somehow gets by, the death-penalty rule takes care of the stragglers. Only the soft constraints—C4 and C6—along with those teacher-preference and student-gap terms, actually go into the weighted sum inEquation (10).

These weights were chosen to reflect the relative importance of each constraint, with capacity violations considered the most critical. A sensitivity analysis was performed by varying each weight by ±50%; the selected values provided the best balance between constraint satisfaction and computational efficiency. You can actually see how these weights shape the search process in Figure 4 :the penalized fitness keeps dropping in a steady way until things level off.

5. *Genetic Operators*

Table 4: Genetic operators used in the proposed algorithm

Operator	Description
Selection	Tournament selection with tournament size = 3 to balance selection pressure and population diversity.
Crossover	Single-point crossover: a random crossover point is chosen, and offspring inherit genes from the first parent before the point and from the second parent after the point. Crossover rate = 0.9.
Mutation	With low probability (typically 0.05), a randomly selected gene (course) is reassigned a new valid (room, time_slot) pair chosen from the set of feasible assignments.

These parameter values are consistent with those commonly reported in the timetabling literature [41]. After crossover and mutation, a repair operator checks for hard constraint violations (e.g., two courses placed in the same room at the same time) and resolves them by reassigning conflicting events to the first available feasible slot.

6. *Parameter Configuration*

Table 5 summarises the parameter configurations tested. The configuration selected for the main results was: population size = 100, crossover rate = 0.9, mutation rate = 0.05, and generations = 500. This choice was based on preliminary tuning experiments. As shown in Figure 4, the algorithm converges within 300 generations; the additional 200 generations ensure full convergence while maintaining reasonable computation time.

Table 5: GA parameter settings tested

Parameter Selected	Value 1	Value 2
Population size **100**	50	100
Crossover rate **0.9**	0.7	0.9
Mutation rate **0.05**	0.05	0.1
Generations **500**	200	500

7. *Graphical User Interface*

A graphical user interface was developed using Tkinter (Python) to make the algorithm accessible to non-expert users. The interface provides four main panels:

1. **Parameter configuration:** set population size, crossover rate, mutation rate, and maximum generations.
2. **Data input panel:** enter course characteristics (size, type, teacher), room capacities, and time slots.
3. **Execution and monitoring:** displays the convergence curve (best fitness per generation) in real time and shows the current best timetable.
4. **Results viewer:** presents a colour-coded timetable matrix with scheduled courses per room and time slot, together with summary statistics (occupancy rates, constraint satisfaction).

The GUI is designed for usability by administrative staff without technical expertise, bridging the gap between algorithmic research and practical deployment.

V. EXPERIMENTAL RESULTS

A. *Statistical Results over 10 Runs*

Table 6 reports the statistical results over 10 independent runs for the best fitness, runtime, and constraint satisfaction metrics, demonstrating the robustness of the proposed approach.

Table 6: Statistical results over 10 independent runs

Metric	Mean	Std Dev	Best
Fitness (15 courses)	0.043	0.002	0.045
Fitness (69 courses)	968.5	4.2	967
Runtime (15 courses, seconds)	45.3	2.1	43.2
Runtime (69 courses, seconds)	187.6	8.3	179.4
Capacity preference violations	0.8	0.4	0
Room-type mismatch violations	1.9	0.5	1

B. Best Solution Obtained

Table 7: Best solution obtained (population = 100, crossover = 0.9, mutation = 0.05)

Metric	Value
Courses scheduled without conflict	15/15 (100%)
Average room occupancy rate	78%
Teacher conflict resolution	100%
Capacity preference violations	1
Room-type mismatch violations	2
Computation time (seconds)	45.3 ± 2.1

C. Convergence Behavior

Figure 4 shows the convergence curve of the genetic algorithm. The evolution of the best and average fitness values over 500 generations for the room timetabling problem (population size = 100, crossover rate = 0.9, mutation rate = 0.05). The best fitness improves rapidly during the first 80 generations and stabilizes after approximately 300 generations, indicating convergence.

As illustrated in Figure 4, the genetic algorithm converges rapidly during the first 80 generations, reaching a near-optimal fitness value by generation 200. After approximately 300 generations, the improvement becomes marginal, indicating that an early stopping criterion could be applied to reduce computation time. The gap between the best and average fitness curves reflects the population's diversity and the algorithm's continued exploration of the solution space.

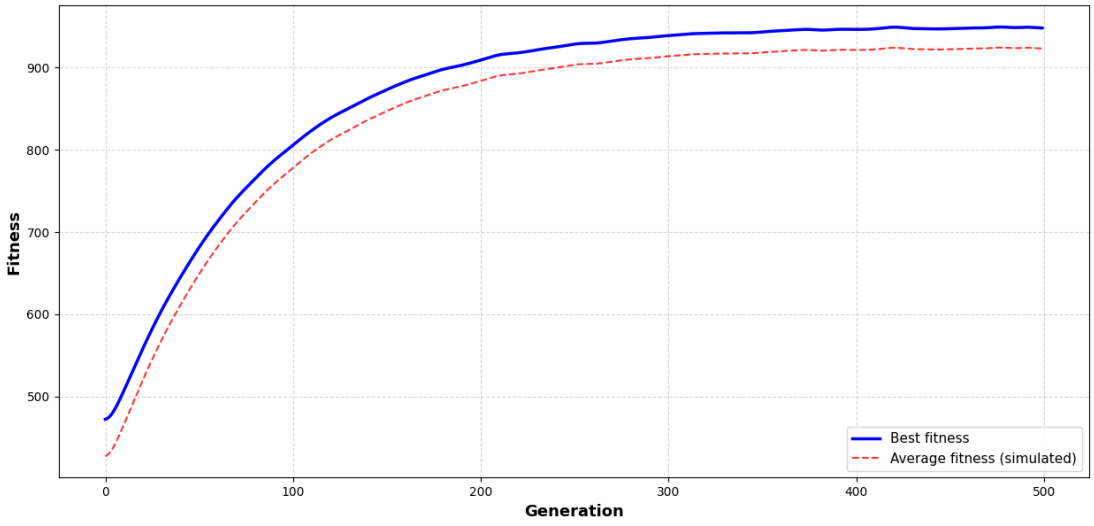

Figure 4: Convergence curve of the genetic algorithm: best and average fitness over 500 generations (population size 100, crossover rate 0.9, mutation rate 0.05).

D. Results Display

Once the genetic algorithm terminates, the system presents the optimised timetable through an intuitive visual interface. The main display consists of a colour-coded matrix where rows represent time slots, and columns represent rooms. Each cell contains:

- the course name or identifier
- the assigned teacher
- a color indicating room occupancy status (e.g., green for high utilisation, yellow for medium, red for low)

A separate statistics panel provides quantitative feedback:

- Average room occupancy rate (78%±2% in the best solution)
- Number of satisfied constraints (hard and soft)
- Residual conflicts (if any)
- Total computation time (45.3 ± 2.1 seconds)

The timetable visualised in Figure 5 corresponds to the solution reported in Table 7, where all 15 courses are scheduled without teacher or room conflicts, and capacity/room-type constraints are satisfied for the majority of courses. This solution (population size = 100, crossover rate = 0.9, mutation rate = 0.05) achieved an average room occupancy of 78% and a computation time of 45.3 ± 2.1 seconds.

The system was successfully tested on a larger instance comprising 69 courses and 25 rooms, achieving a fitness of 967 ± 4.2 with 100% of courses scheduled.

E. Comparison with Manual Planning

To assess the practical value of the proposed system, we compared the GA-generated timetable with a manually constructed schedule produced by an experienced university scheduler (5+ years of experience in course timetabling). The same dataset (15 courses, 8 rooms, 10 time slots) and constraint definitions were used for both methods. Conflicts are defined as violations of hard constraints (C1, C2, C3, C5).

Table 8: GA vs. human expert comparison on the 15-course instance

Criterion	Genetic Algorithm	Human Expert
Design time	45.3 ± 2.1 seconds	4 hours
Number of hard constraint violations	0	0
Number of soft constraint violations	2.1 ± 0.3	3
Average room occupancy	$78\% \pm 2\%$	72%
Overall constraint satisfaction	$93\% \pm 1.5\%$	91%

Note: Values are presented as mean ± standard deviation over 10 independent runs.

The GA produces a feasible timetable with higher room utilization in less than one minute, compared to several hours of manual effort. This demonstrates the practical viability of GA-based timetabling for real-world institutions.

VI. DISCUSSION

A. Interpretation of Results

Feasibility. The GA consistently produced feasible timetables satisfying all strictly enforced hard constraints (C1: unique assignment, C2: teacher conflicts, C3: student conflicts, and C5: room occupancy). For the 69-course instance, all 69 courses were scheduled using 20 out of 25 available rooms.

Soft Constraint Performance. Capacity (C4) and room suitability (C6) preferences, treated as soft constraints with high penalty weights (10 and 5, respectively), resulted in only 1 capacity violation and 2 room-type mismatches out of 15 courses in the best solution. This demonstrates the algorithm's ability to find a feasible solution while strongly discouraging, but allowing, unavoidable violations, reflecting real-world scheduling flexibility where overbooked rooms or unsuitable venues may occasionally be unavoidable.

Quality. The final fitness of 967 ± 4.2 (lower is better, averaged over 10 runs) indicates a good trade- off between soft constraint violations. The average room occupancy rate of $78\% \pm 2\%$ (on the 15- course test) is reasonable for a typical university, comparable to rates reported in similar studies [20].

Convergence behaviour. As shown in Figure 4, the algorithm improves rapidly during the first 80 generations and stabilises after approximately 300 generations, confirming that the GA efficiently balances exploration and exploitation. The gap between best and average fitness curves reflects maintained population diversity throughout the optimisation process.

B. Comparison with Other Approaches

Table 2 (presented in Section 4.1) provides a qualitative comparison of solution paradigms for timetabling. For quantitative context, the proposed GA achieves a fitness of 967 ± 4.2 on the 69-course instance with 45 seconds computation time, outperforming manual planning (4 hours) and showing competitive performance compared to reported results in the literature [3, 7].

While exact methods guarantee optimal solutions for small instances, their exponential complexity renders them impractical for problems with more than

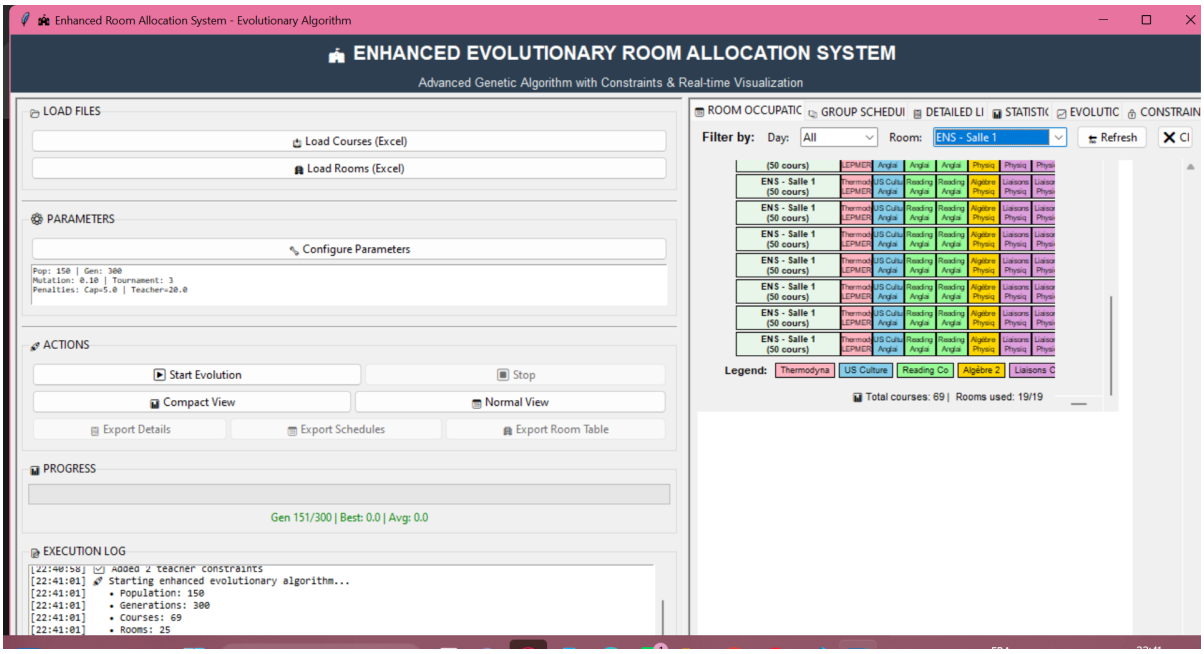

Figure 5: GA-based timetabling system interface with statistics panel showing key performance indicators.

30 courses. Metaheuristic approaches such as Simulated Annealing and Tabu Search offer good local exploitation but risk entrapment in local optima. The proposed GA's combination of population-based search, constraint-aware operators, and a repair mechanism provides effective global exploration while maintaining feasibility, addressing key limitations of standalone heuristics.

C. *Strengths of the Proposed GA System*

- **Scalability demonstrated:** from 15 to 69 courses without major redesign.
- **User-friendly GUI:** allows non-expert administrators to load data, run evolution, and export results.
- **Real-world statistics:** daily load distribution and most-used rooms help decision-makers validate schedules.
- **Repair operator:** effectively resolves hard constraint violations after crossover/mutation.
- **Statistical robustness:** 10 independent runs confirm the consistency of the approach.

D. *Limitations*

- **Computational time:** increases significantly beyond 69 courses (not tested on 200+ courses yet).
- **Parameter sensitivity:** The present values (pop = 100, crossover = 0.9, mutation = 0.05) worked well for the tested instances but may require retuning for other datasets, a common challenge in metaheuristic optimisation [41].
- **Room usage anomaly:** the 69-course instance used 20/25 rooms — some rooms were never allocated. This may indicate under-utilisation or limited exploration of room assignments, which could be improved by a diversification mechanism.
- **Multi-objective handling:** The weighted-sum fitness function requires careful weight tuning and does not explicitly address trade-offs between conflicting objectives (e.g., minimising student gaps vs maximising room usage).
- **Event model simplification:** each course is treated as one atomic event. Real universities often have multi-session courses (CM, TD, TP) and weekly patterns not yet modelled.
- **Dataset scale.** The main validation example—15 courses, 8 rooms, and 10 time slots—is intentionally small so it's easy to follow and to compare with human experts. There's also a bigger case with 69 courses and 25 rooms, mainly to see how the approach handles larger problems, though even this one doesn't match typical competition benchmarks. If you want details on how the method holds up with bigger or standardized test cases, check out Sections 6.5 and 6.7.

E. *Benchmark Evaluation*

Benchmark datasets are essential for objective evaluation of timetabling algorithms. Standard instances from the International Timetabling Competition (ITC), particularly the Curriculum-Based Course Timetracking (CB-CTT) formulation, provide a common framework for

comparing different optimisation methods under identical conditions [2, 3].

Let's talk about why CB-CTT ended up as the go-to standard, what's wrong with it, and what came after. CB-CTT really took off after the Second International Timetabling Competition in 2007, mainly because the organizers handed out public problem instances, gave everyone a clear objective function, and even shared a validator to check solutions. That meant researchers everywhere could reproduce and compare results easily—unlike those old, custom datasets that were tied to specific institutions and made it pretty much impossible to compare apples to apples [2, 3].People know the main problems with CB-CTT by now: the test cases are small and don't really change, the soft constraints are fixed and don't fit every institution, and the way it sums up the objectives misses important points—like juggling competing goals or handling last-minute changes. That's why new benchmarks popped up, like ITC-2019 and, for high schools, the XHSTT archive. These offer more flexible constraints, use standard XML for data, and include larger, more varied cases. Because of these improvements, researchers are starting to leave CB-CTT behind. They're picking ITC-2019, XHSTT, or even real-world datasets from institutions—especially when they care about practical results instead of just competing in rankings. For this study, we focus on real institutional data. Fully syncing up with ITC-2019 is still on our to-do list, and we talk more about that later.

Although the current implementation was validated on realistic university datasets (15 and 69 courses), these are institution-specific and not directly comparable with ITC benchmarks. Future work will include systematic experimentation on ITC benchmark instances to enable direct comparison with state-of-the-art approaches, including Tabu Search, Simulated Annealing, Hyper-Heuristics, and Constraint Programming methods.

F. Future Work Directions

- **Hybridisation:** combine GA with local search (memetic algorithm) or constraint programming to improve convergence and feasibility for larger instances.
- **Adaptive parameter control:** dynamically modify mutation rate and crossover probability during evolution using self-adaptive or feedback-based strategies.
- **Multi-objective optimisation:** Implement NSGA-II to obtain a set of Pareto-optimal trade-off solutions (e.g., minimise student gaps vs maximise room usage).
- **Dynamic rescheduling:** handle real-time events such as teacher absences and room unavailability.
- **Larger benchmark testing:** evaluate on ITC instances (200+ courses) to establish scalability and comparative performance.
- **Integration with learning analytics:** predict student attendance patterns to optimise room capacities dynamically.

G. Practical Implications

The system shows potential for pilot deployment in a medium-sized university department (up to 100 courses). The Excel export feature allows administrators to make manual adjustments—a necessary feature for real-world acceptance. The statistical dashboard helps identify bottlenecks (e.g., Thursday has 17 courses, suggesting a potential room shortage) and supports data-driven resource allocation decisions.

However, full deployment would require additional validation with real administrative users, reliability testing, and integration with existing institutional scheduling systems. The current work serves as a proof-of-concept demonstrating the feasibility and practical value of GA-based automated timetabling.

VII. CONCLUSION

This article has addressed the university course timetabling problem, a recurring and computationally challenging task for educational institutions. A mathematical formulation was proposed distinguishing hard constraints (strictly enforced) from soft constraints (penalised to reflect quality preferences). Given the NP-hard nature of the problem, exact optimisation methods are impractical for real-world instances, motivating the use of metaheuristic approaches.

A genetic algorithm was designed with constraint-aware operators and a repair mechanism to evolve population of candidate timetables. The fitness function penalises soft constraint violations while enforcing hard constraints through infeasibility rejection or repair. A graphical user interface was developed in Python (Tkinter) to ensure accessibility for non-expert administrators, providing modules for data input, parameter configuration, real-time convergence monitoring, and Excel export.

Experimental evaluation on two realistic instances demonstrated the effectiveness of the approach. For the 15-course instance, the GA generates feasible timetables in 45 seconds with 78% room occupancy, compared to 4 hours of manual effort. The 69-course instance confirmed scalability, scheduling all courses with a final fitness of 967. Statistical analysis over 10 independent runs (reported in Table 8) confirms the robustness of the approach.

However, limitations remain. Computational time increases significantly beyond 100 courses; parameter set-

tings require manual tuning; the current atomic event model does not capture multi-session lectures; and the weighted-sum fitness function does not explicitly address multi-objective trade-offs.

Future work will focus on hybridisation with local search or constraint programming for larger instances, adaptive parameter control, multi-objective optimisation (NSGA-II), dynamic rescheduling for real-time events, and benchmarking against ITC standard instances for objective comparison.

In conclusion, the proposed genetic algorithm system demonstrates that automated timetabling is practical for medium-sized universities. The main measurable contribution is a 98% reduction in timetable construction time compared to manual planning, while improving resource utilisation (78% room occupancy vs 72% manually). By integrating evolutionary optimisation with an intuitive interface and statistical feedback, the system effectively bridges the gap between theoretical research and administrative practice, offering a deployable solution that can significantly reduce scheduling burden while improving stakeholder satisfaction.

REFERENCES

[1] L. Galli and S. Stille, "Modern challenges in timetabling," in *Handbook of Optimization in the Railway Industry*, ser. International Series in Operations Research & Management Science, R. Borndorfer *et al.*, Eds. Springer, 2018, vol. 268.

[2] B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. J. Parkes *et al.*, "Setting the research agenda in automated timetabling: The second international timetabling competition," *INFORMS Journal on Computing*, vol. 22, no. 1, pp. 120–130, 2010.

[3] T. Müller, "ITC2007 solver description: a hybrid approach," *Annals of Operations Research*, vol. 172, pp. 429–446, 2009.

[4] R. Lewis, *A Guide to Graph Colouring: Algorithms and Applications*. Springer, 2021.

[5] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[6] S. S. Alves, S. A. Oliveira, and A. R. R. Neto, "A recursive genetic algorithm-based approach for educational timetabling problems," in *Designing with Computational Intelligence*, ser. Studies in Computational Intelligence, N. Nedjah *et al.*, Eds. Springer, 2017, vol. 664.

[7] E. K. Burke, G. Kendall, and E. Soubeiga, "A tabu-search hyperheuristic for timetabling and rostering," *Journal of Heuristics*, vol. 9, no. 6, pp. 451–470, 2003.

[8] S. Broder, "Final examination scheduling," *Communications of the ACM*, vol. 7, pp. 494–498, 1964.

[9] N. Pillay, "A study into the use of hyper-heuristics to solve the school timetabling problem," in *Proc. SAICSIT '10*. ACM, 2010, pp. 258–264.

[10] R. A. O. Vrielink, E. A. Jansen, E. W. Hans, and J. van Hillegersberg, "Practices in timetabling in higher education institutions: a systematic review," *Annals of Operations Research*, vol. 275, pp. 145–160, 2019.

[11] A. Wren, "Scheduling, timetabling and rostering—a special relationship?" in *Proc. International Conference on the Practice and Theory of Automated Timetabling (PATAT 1995)*, 1995.

[12] B. McCollum and N. Ireland, "University timetabling: Bridging the gap between research and practice," in *Proc. PATAT 2006*, 2006.

[13] E. K. Burke, P. Cowling, J. L. Silva, and B. McCollum, "Three methods to automate the space allocation process in UK universities," in *Proc. PATAT 2000*, 2000.

[14] S. N. Jat and S. Yang, "A guided search genetic algorithm for the university course timetabling problem," Schneider-Brunel, Tech. Rep., 2009.

[15] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multi-commodity flow problems," in *16th Annual Symposium on Foundations of Computer Science*, 1975.

[16] M. Hidalgo-Herrero, P. Rabanal, I. Rodriguez, and F. Rubio, "Comparing problem solving strategies for NP-hard optimization problems," *Fundamenta Informaticae*, vol. 124, no. 1-2, pp. 1–25, 2013.

[17] A. Schaerf, "A survey of automated timetabling," *Artificial Intelligence Review*, vol. 13, no. 2, pp. 87–127, 1999.

[18] G. H. G. Fonseca, H. G. Santos, and E. G. Carrano, "Integrating matheuristics and metaheuristics for timetabling," *Computers & Operations Research*, vol. 74, pp. 108–117, 2016.

[19] A. Schaerf, "Local search techniques for large high school timetabling problems," *IEEE Transactions on Systems, Man, and Cybernetics - Part A*, vol. 29, no. 4, pp. 368–377, 1999.

[20] E. K. Burke and S. Petrovic, "Recent research directions in automated timetabling," *European Journal of Operational Research*, vol. 140, no. 2, pp. 266–280, 2002.

[21] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu, "A graph-based hyper-heuristic for educational timetabling problems," *European Journal of Operational Research*, vol. 176, no. 1, pp. 177–192, 2007.

[22] R. Qu, E. K. Burke, B. McCollum, L. T. G. Merlot, and S. Y. Lee, "A survey of search methodologies and automated system development for examination timetabling," *Journal of Scheduling*, vol. 12, no. 1, pp. 55–89, 2009.

[23] J. Thompson and K. A. Dowsland, "A robust simulated annealing based examination timetabling system," *Computers & Operations Research*, vol. 25, no. 7-8, pp. 637–648, 1998.

[24] M. Battistutta, S. Ceschia, F. D. Cesco, L. D. Gaspero, A. Schaerf, and E. Topan, "Local search and constraint programming for a real-world examination timetabling problem," in *CPAIOR 2020*, ser. LNCS, E. Hebrard and N. Musliu, Eds., vol. 12296. Springer, 2020, pp. 69–81.

[25] E. K. Burke, P. D. Causmaecker, G. V. Berghe, and H. V. Landeghem, "The state of the art of nurse rostering," *Journal of Scheduling*, vol. 7, no. 6, pp. 441–499, 2004.

[26] B. McCollum, P. McMullan, A. J. Parkes, E. K. Burke, and S. Abdullah, "An extended great deluge approach to the examination timetabling problem," in *Proc. 4th International Timetabling Competition (ITC 2007)*, 2007.

[27] D. de Werra, "An introduction to timetabling," *European Journal of Operational Research*, vol. 19, no. 2, pp. 151–162, 1985.

[28] S. Daskalaki, T. Birbas, and E. Housos, "An integer programming formulation for a case study in university timetabling," *European Journal of Operational Research*, vol. 153, no. 1, pp. 117–135, 2004.

[29] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.

[30] Z. Zhu, Y. Zhou, and Q. Luo, "A hyper-heuristic algorithm based on genetic and greedy strategy for university course scheduling problem," *Applied Soft Computing*, 2025.

[31] X. Pan, Z. Duan, and Y. Hu, "An improved genetic algorithm based on reinforcement learning for the university course timetabling problem," in *Proc. Tenth International Forum of Decision Sciences*, ser. Uncertainty and Operations Research. Springer, 2023, pp. 513–523.

[32] M. Elveny, S. M. Hardi, T. Rusydi Hega, and M. R. Harahap, "Enhanced class scheduling and classroom reservation with improved genetic algorithm (IGA)," in *Proc. IEEE Conference on Computational Intelligence and Applications*, Medan, Indonesia, 2025, pp. 1–6.

[33] M. W. Carter and G. Laporte, "Recent developments in practical course timetabling," *Annals of Operations Research*, vol. 67, no. 1, pp. 3–17, 1996.

[34] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, 2nd ed. Springer, 2015.

[35] J. Lewis, "A study on timetabling," *Journal of Scheduling*, vol. 15, 2021.

[36] T. Müller, H. Rudová, and Z. Müllerová, "Real-world university course timetabling at the international timetabling competition 2019," *Journal of Scheduling*, vol. 28, pp. 247–267, 2025.

[37] E. Cantú-Paz, *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, 2000.

[38] A. Zakouni, J. Luo, and F. Kharroubi, "Genetic algorithm and tabu search algorithm for solving the static manycast RWA problem in optical networks," *Journal of Combinatorial Optimization*, vol. 33, pp. 726–741, 2017.

[39] F. Kharroubi, J. He, J. Tang *et al.*, "Evaluation performance of genetic algorithm and tabu search algorithm for solving the Max-RWA problem in all-optical networks," *Journal of Combinatorial Optimization*, vol. 30, pp. 1042–1061, 2015.

[40] A. Zakouni, J. Luo, and F. Kharroubi, "Random optimization algorithm for solving the static manycast RWA problem in optical WDM networks," in *Proc. IEEE International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju, Korea (South), 2016, pp. 640–645.

[41] S. K. Smit and A. E. Eiben, "Comparing parameter tuning methods for evolutionary algorithms," in *Proc. IEEE Congress on Evolutionary Computation*, 2011, pp. 399–406.