AL-BUTANA JOURNAL Of APPLIED SCIENCE
http// : ojs.abutana.edu.sd, ISSN: 1858-6616
Issue (17), December 2025, Page NO. 31-53

An Enhanced Metaheuristic Framework for Timetable Generation Using Genetic Algorithm and Local Search

Abubakr H.Ombabi [1]*, Mohamed Babiker Ali [2], Mussab E.A Hamza [3], Abuzer H. I Ahmed[4]

[1, 2, 3, 4] Department of Computer Science, University of Albutana, Ruffaa, Sudan.
Email: Abubakr.h.Ombabi@gmail.com

Abstract

Genetic Algorithm (GA), a class of evolutionary algorithms inspired by natural selection, has been widely applied to complex optimization and search problems. The University Course Timetabling Problem (UCTP) is a non-deterministic polynomial-time hard problem that involves assigning lectures to classrooms and timeslots while satisfying numerous hard and soft constraints. This study proposes an enhanced metaheuristic framework that integrates a GA with sequential local search and a repair function to efficiently generate feasible timetables. The GA initializes a population of candidate timetables, evaluates their fitness, and iteratively evolves them through selection, crossover, and mutation operators. The sequential local search refines candidate solutions by reducing soft constraint violations, such as consecutive lectures or sessions scheduled during breaks, while the repair function guarantees the satisfaction of hard constraints, including classroom capacity and instructor availability. The proposed framework was implemented in Java IDE 8.1 and evaluated using multiple benchmark datasets of varying sizes and complexities. Experimental results demonstrate that the proposed method achieved an overall accuracy of 96.3% and improved constraint violation reduction by 28.5% compared with existing methods. These findings confirm the effectiveness of combining GA's global search capability with local refinement mechanisms, demonstrating its potential for real-world University scheduling scenarios.

Keywords: Timetable generation, Genetic Algorithm, UCTP, Resources Scheduling.

Introduction

Genetic algorithms (GAs) are adaptive search and optimization techniques inspired by the principles of natural evolution, such as selection, crossover, and mutation [1]. First introduced by John Holland in the 1970s, GAs provide a robust framework for exploring large and complex solution spaces where conventional methods may fail [2]. They operate on a population of candidate solutions, evolving them over successive generations based on a fitness measure that evaluates solution quality. One of the key strengths of GAs is their ability to balance exploration and exploitation: exploration allows the algorithm to search diverse regions of the solution space, while exploitation focuses on refining high-

quality solutions. This makes GAs particularly suitable for combinatorial and NP-hard problems, including scheduling, resource allocation, vehicle routing, and optimization in engineering and medicine [3, 4]. Additionally, GAs are flexible in representing solutions using different encoding schemes, can incorporate domain-specific knowledge through heuristics, and are capable of handling multiple objectives and constraints simultaneously. These characteristics make GAs an effective tool for generating high-quality solutions in complex real-world problems, such as university course timetabling, where the search space is vast and constraints are numerous [5].

As the Genetic Algorithm (GA) is a population-based optimization method, it begins by generating a set of candidate solutions (population), then iteratively improves them through three main operators: selection, which chooses the fittest solutions; crossover, which combines parts of two solutions to create new offspring; and mutation, which introduces small random changes to maintain diversity. Across successive generations, the algorithm converges toward high-quality solutions by favoring individuals with better fitness values. GA is widely used in scheduling and timetabling because it effectively explores large, complex search spaces and avoids getting trapped in local optima.

Scheduling problems involve the effective allocation of limited resources to tasks while ensuring that no two tasks use the same resource simultaneously [6]. University course timetabling, in particular, is recognized as an NP-hard search problem that cannot be efficiently solved using traditional optimization techniques such as constraint logic programming, backtracking, or other exact methods [7]. While these approaches can sometimes reduce constraint violations and yield feasible timetables, they rarely achieve optimal solutions, especially for large-scale instances. The University Course Timetabling Problem (UCTP) can be viewed as a highly constrained problem that can be effectively addressed using heuristic and metaheuristic methods, particularly genetic algorithms (GAs) [8]. However, GAs typically require longer execution times due to the size and complexity of the search space. In the literature, a wide range of algorithms has been proposed for university timetabling, often benchmarked on standard datasets [9]. Although several approaches have proven effective in reducing soft constraint violations, relatively fewer methods are capable of consistently reaching globally optimal or near-optimal solutions for the most challenging instances [10].

The UCTP can be described as assigning a set of events (lectures) to a set of limited resources such as classrooms, students, and lecturers. Each event requires specific resources, occurs at a fixed timeslot, and has a given duration. The primary objective is to allocate events to resources while ensuring that no two events share the same resource simultaneously [7]. Fundamentally, UCTP consists of distributing a set of lectures across a fixed number of classrooms and timeslots within the academic week, while satisfying multiple constraints. These constraints are generally categorized into hard constraints, which must always be satisfied, and soft constraints, whose violations should be minimized [9]. Hard constraints are considered more critical, as any violation results in an infeasible timetable. The goal is to design an efficient hybrid algorithm, based on genetic algorithms and local search that generates timetables satisfying all hard constraints while minimizing soft constraint violations.

1. Problem Formulation

The UCTP can be formally represented as follows: a set of events (lectures) ^{E} = {e1, e2, . . . , e_n }to be scheduled in 5 days (Week) of 9 periods each, in which time T = 45 timeslots, set of classrooms R = {r1, r2, . . . , r_m} each with features F, set of students S = {s1, s2, . . . , s_k} and number of lecturers. Five matrices are used to define the relations of these sets. Student_Event matrix $A_{k,n}$ to correlate courses with its attended students. In $A_{k,n}$ the value of $a_{i,j}$is set to 1 if student i ∈ S is attend event j ∈ E,0 otherwise. Event_Conflict matrix$B_{n,n}$, to identify courses that can be assigned to the same timeslot. Room_Features matrix to give the features of each classroom, in which the value of a cell $C_{i,l}$is 1 if i ∈ R has feature j ∈ F, and 0 otherwise. Event_Features matrix $D_{n,l}$ to store the features required by each event that is event i ∈ E needs feature j ∈ F if and only if dij = 1. Event_Room matrix $G_{n,m}$ lists of classrooms which each event can be assigned. Additional matrix is used for assigning each course to classroom ri and timeslot ti. Each pair of (ri, ti) is assigned specified number that correlated to particular course. If classroom ri in a timeslot ti is free or no course is placed, then the pair is assigned "-1".

The solution to UCTP can be seen in a form of an ordered list of pairs (ri, ti), in which the index of each pair is the id number of a course ei ∈ E (i = 1, 2,., n). For

instance, the time slots and classrooms are allocated to course in an ordered list of pairs like (2, 4), (3, 5), (1, 12). . , (2, 7) where classroom 2 and timeslot 4 are allocated to course 1, classroom 3 and timeslot 5 are allocated to course 2, etc. A feasible solutions is the solutions in which all courses are assigned to appropriate timeslots, lecturers and classrooms besides satisfying all of the hard and soft constraints .In the proposed method, we consider the following hard and soft constraints. Example of a hard constraints (No more than one course is allowed at one timeslot in each classroom), while (Lecture hours should be scheduled within the allowed hours) is an example of a soft constraints.

2. Related Works

Genetic algorithms have been extensively studied and applied across various optimization tasks due to their adaptability and robustness in handling complex, constrained, and multi-objective problems. Recently, researchers have continued to enhance GA-based approaches through improving operators as well as problem specific heuristics to achieve higher performance. This section reviews recent studies from that demonstrate the diverse applications of GAs, with a particular focus on university course timetabling and other real-world optimization problems in scheduling, engineering, and resource allocation. Recent studies have proposed several improvements and hybrid techniques to enhance GA performance across various domains, including university timetabling, scheduling, energy optimization, and healthcare systems. Liu et al. [10] proposed an adaptive hybrid GA approach for university course timetabling, where dynamic penalty functions were applied to balance hard and soft constraints. Their method achieved high feasibility rates and reduced constraint violations compared to traditional GA models. Also, Om Prakash Verma et al. [11] presented a hybrid Bacterial Foraging and Genetic Algorithm for optimal timetable generation. In their approach, bacteria were simulated as candidate solutions in an n-dimensional search space, while GA operators were used in the chemotaxis stage to improve solution directionality. The algorithm demonstrated superior time efficiency and solution quality over standalone GA methods. Roberts et al. [12] developed a constraint-driven genetic framework for exam scheduling, introducing a self-repair mechanism to maintain feasibility during crossover and mutation. Their results showed faster convergence and more

balanced schedules compared to existing GA-based systems. In another work, Singh and Kumar in [13] integrated GA with a simulated annealing (SA) strategy for timetable optimization. The hybrid model effectively minimized both teacher and room conflicts while optimizing resource utilization.

At other side, recent research has demonstrated the adaptability of GA in various optimization areas. For example, Chen et al. [14] applied GA for energy-efficient cloud resource scheduling, reducing power consumption and task delays. Rahman et al. [15] introduced a GA-based hospital resource management system to optimize doctor-patient allocation. Their method significantly reduced waiting times and improved resource distribution. Ali and Zhang in [16] proposed an enhanced multi-objective GA for vehicle routing problems, integrating adaptive mutation rates and crowding distance sorting to handle dynamic traffic conditions efficiently. Meanwhile, Li et al. [17] used a GA for feature selection in deep learning-based sentiment analysis, achieving improved classification accuracy on multilingual datasets. Furthermore, Ahmed et al. [18] optimized staff scheduling in healthcare environments by integrating GA with a fuzzy evaluation module to balance workloads and preferences. Torres and Delgado in [19] demonstrated the potential of GA in smart grid optimization, where the algorithm minimized operational costs while maintaining power stability. These studies collectively emphasize the versatility of genetic algorithms and the effectiveness of hybridization in enhancing their convergence speed, accuracy, and applicability across diverse problem domains.

While previous approaches often focus on improving either genetic operators or adopting hybrid mechanisms, our method uniquely combines global exploration via GA, systematic refinement through sequential local search, and a dedicated repair function that guarantees satisfaction of all hard constraints at each iteration. This ensures that infeasible timetables produced during evolution are automatically corrected rather than discarded, significantly improving convergence stability. Moreover, our framework was rigorously evaluated on multiple benchmark datasets of varying sizes, and the results demonstrate higher feasibility, improved accuracy, and lower constraint violations compared with state-of-the-art methods. These contributions position our study as a practical and scalable solution for real-world university scheduling environments.

3.Proposed approach

The university course timetabling problem (UCTP) can be described as assigning a set of events (lectures) to a set of limited resources such as classrooms, students, and lecturers. Each event requires specific resources, occurs at a fixed timeslot, and has a given duration. The primary objective is to allocate events to resources while ensuring that no two events share the same resource simultaneously [5]. Fundamentally, UCTP consists of distributing a set of lectures across a fixed number of classrooms and timeslots within the academic week, while satisfying multiple constraints. These constraints are generally categorized into hard constraints, which must always be satisfied, and soft constraints, whose violations should be minimized [6]. Hard constraints are considered more critical, as any violation results in an infeasible timetable. The goal is to design an efficient hybrid algorithm, based on genetic algorithms and local search that generates timetables satisfying all hard constraints while minimizing soft constraint violations.

As illustrated in Fig.1**.** The proposed model integrates a Genetic Algorithm with a Local Search mechanism and a specialized repair function. The GA serves as the global search engine, generating diverse candidate timetables and exploring the solution space through selection, crossover, and mutation. These operators allow the algorithm to maintain diversity and avoid premature convergence. Once new offspring are produced, the Local Search mechanism is applied to refine each candidate solution by systematically reducing soft constraint violations such as minimizing consecutive lectures, avoiding undesirable time gaps, or improving classroom utilization. This step ensures that promising solutions are further improved rather than relying solely on random evolution.

The repair function acts as a feasibility controller, correcting any violations of hard constraints that arise during GA or local search operations. These hard constraints include classroom capacity, instructor availability, and conflict-free scheduling of courses sharing the same student groups. Instead of discarding infeasible solutions (as done in several traditional GA-based methods), the repair function adjusts timeslots or room assignments to immediately restore feasibility. This combination allows the model to maintain a balance between broad exploration and targeted exploitation. Through this systematic

AL-BUTANA JOURNAL Of APPLIED SCIENCE
http// : ojs.abutana.edu.sd, ISSN: 1858-6616
Issue (17), December 2025, Page NO. 31-53

interaction among GA, LS, and repair strategies, the model is able to efficiently allocate events, rooms, and timeslots while minimizing conflicts, producing highly feasible and high-quality university timetables across different dataset complexities.

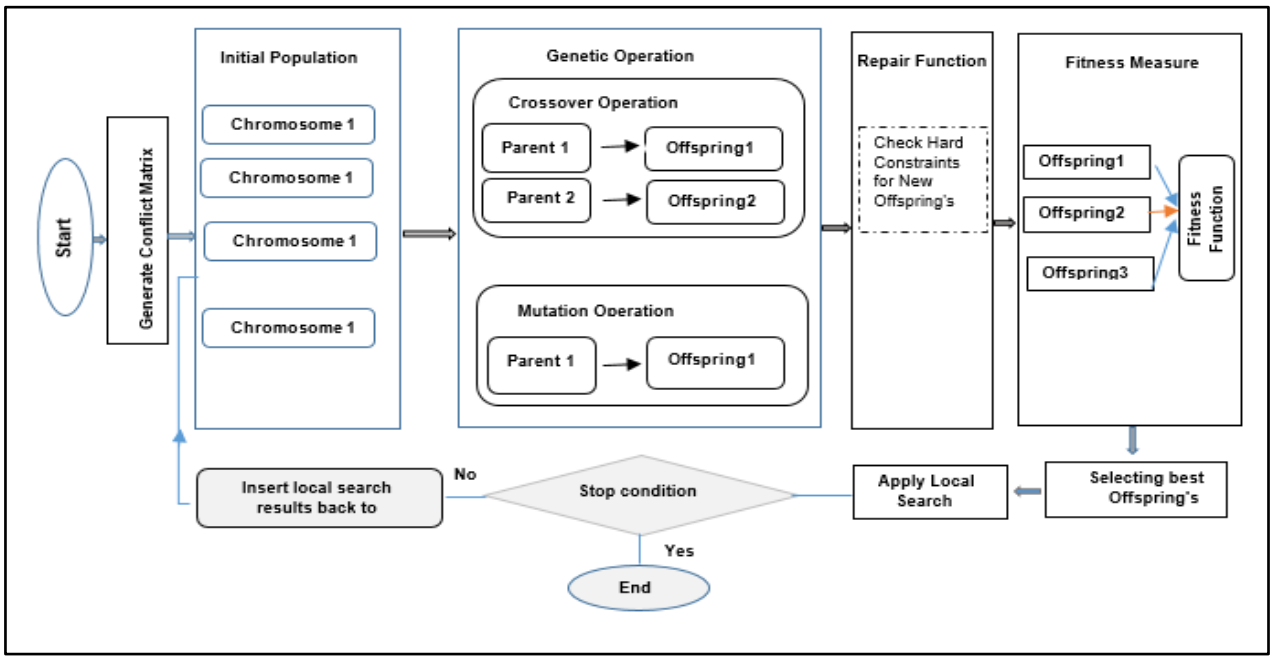


Fig.1 overall framework of the proposed University Timetable Generation

The UCTP can be formally represented as follows: a set of events (lectures) $E = \{e_1, e_2, e_3, .., e_n\}$ to be scheduled in 5 working days, each consisting of 9 periods, giving a total of $T = 45$ timeslots. A set of classrooms $R = \{r_1, r_2, r_3, .., r_m\}$ each defined by a set of features F. A set of students $S = \{s_1, s_2, s_3, .., s_k\}$. In addition to a set of lecturers. To capture the relationships among these sets, five matrices are defined:

Student_Event matrix $A_{k,n}$ indicates which students attend which events. A value of 1 means student $s_i \in S$ attends event $E_j \in E$, and 0 otherwise.

Event_Conflict matrix $B_{n,n}$ identifies events that cannot be assigned to the same timeslot.

Room_Features matrix – specifies the features of each classroom. A value of 1 indicates classroom $r_i \in R$ has feature $F_j \in F$, otherwise 0.

Event_Features matrix$D_{n,l}$ defines the features required for each event. Event $e_i \in E$ requires feature $f_j \in F$ if and only if $d_{ji} = 1$.

AL-BUTANA JOURNAL Of APPLIED SCIENCE
http// : ojs.abutana.edu.sd, ISSN: 1858-6616
Issue (17), December 2025, Page NO. 31-53

Event_Room matrix $G_{n,m}$ lists the rooms in which each event can be assigned.

A mapping matrix is used to assign each event e_i to a pair(r_i, t_i), representing its allocated classroom and timeslot. If a (classroom, timeslot) pair is unoccupied, it is assigned -1. The solution to the UCTP can therefore be represented as an ordered list of pairs $(r_{i,}\ t_{i,})$ where the index corresponds to the event identifier $e_{i,}$: E (i = 1, 2, 3, .. , n). For example, the assignment (2,4),(3,30),…,(2,7) indicates that course 1 is placed in classroom 2 at timeslot 4, course 2 in classroom 3 at timeslot 30, and so on. The quality of a timetable S is evaluated by the objective function in Equation (1):

$$f_{(s)} := hcv(s) \times C + scv(s) \quad (1)$$

Where $hcv(s)$ and $scv(s)$ represent the number of hard and soft constraint violations, respectively, and C is a constant larger than the maximum possible number of soft constraint violations. A feasible timetable is one that satisfies all hard constraints, while minimizing the number of soft constraint violations.

Hard constraints	Soft constraints
Student batch cannot be assigned more than one course at the same time	Lecture hours should be scheduled within the allowed hours
Classroom must satisfy the features required by the course (capacity).	Lecturers don't like to be assigned two classes consecutively
No more than one course is allowed at one timeslot in each classroom	The lectures cannot be assigned to timeslots in the breaks timeslots
A lecturer must not be assigned more than one class at the same time	One lecture should be scheduled ones in a week for one course

The algorithm starts by preparing the generation counter with K= 0 and generating an initial population Pop (K). Each individual in the population is evaluated according to the defined fitness function. The evolutionary process then iteratively proceeds until the termination condition is satisfied. At each iteration, recombination operators are applied to Pop (K) to generate a set of offspring. A new population Pop (K+1) is subsequently formed by selecting individuals from both the parent and offspring populations, ensuring that high-quality solutions are retained. The generation counter is then incremented, and the cycle continues.

The procedure terminates once the stopping criterion is reached, and the best solution obtained is reported as the final output as shown in Algorithm 1.

Algorithm 1: Psudocode for GA process

```
Begin
K = 0; initialize Pop (k);
Evaluate Pop (k);
While (k < MaxGenerations AND noImprovement < StoppingThreshold) do
Recombine Pop (k) to generate Offspring (k);
Select Pop (k+1) from Pop (k) then
Offspring (k);
k = k + 1
End while
End
```

Note that: MaxGenerations is the maximum number of iterations allowed and StoppingThreshold is number of consecutive generations with no fitness improvement.

3.1 Local Search

Genetic algorithm can produce more better quality solutions if incorporated with local search rather than using genetic algorithms alone [5]. In our approach, a sequential search algorithm is applied to produce more optimal solutions (timetable) by reducing the number of soft constraint violations, thus, making the selection process faster. Local search pseudo code is presented in Algorithm. 2.

Algorithm .2. Pseudo Code for local search

```
Input: Feasible Course Timetable (FCT)
Begin
For I = 1 to N, where N is the number of events
For J = 1 to T, where T is the number of timeslots
Assign timeslot J for each class I
IF FCT improved
FCT [I] [2] =J
Break
End IF
End for
```

End for
End
Output: improved feasible Course Timetable

As illustrated in Fig. 1. We applied local search before moving to the next generation. The result of the local search is returned to the GA to move to the next generation. Referring to Fig. 1, initially a conflict array with N×N dimensions is created to perform conflict checking and to avoid the number of students as a factor in the complexity of the problem.

3.2 Chromosome Representation

The initial population is constructed using a constructive heuristic approach. Each timetable begins as an empty schedule, and feasible timetables are generated by iteratively adding or removing appropriate courses based on classroom availability. This process continues until all hard constraints are satisfied, while soft constraint violations are disregarded at this stage. Each generated chromosome represents a candidate timetable, serving as a fundamental building block of the population. The population size (N) corresponds to the number of generated timetables, and the length of each chromosome equals the total number of courses to be scheduled. Each gene within a chromosome encodes the timeslot assignment fora specific course or event. Fig.2. illustrates the overall chromosome structure, while Fig.3. shows the detailed information encoded within each chromosome.

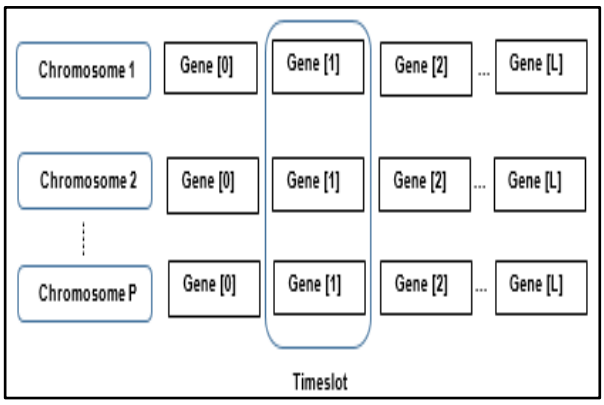


Fig. 2. Timetable Genetic Representation

AL-BUTANA JOURNAL Of APPLIED SCIENCE
http// : ojs.abutana.edu.sd, ISSN: 1858-6616
Issue (17), December 2025, Page NO. 31-53

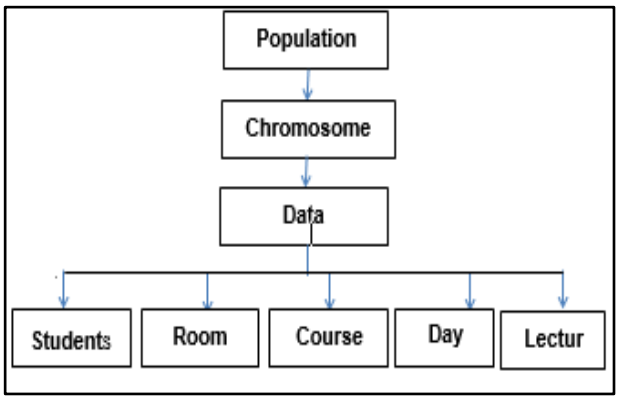


Fig. 3. information encoded in the chromosome

3.3 Selection

The selection process enables the genetic algorithm (GA) to progressively evolve toward an optimal solution by favoring chromosomes with higher fitness values [10]. In this work, the tournament selection method is employed to determine which individuals proceed to the next generation. Specifically, K timetables (individuals) are randomly chosen from the population, and the one with the highest fitness among them is selected as a parent. This procedure is repeated until the required number of individuals has been selected. The tournament size K is a crucial parameter that can take values from 2 to P, where P denotes the total number of individuals in the population.

3.4 Crossover

The crossover process performs recombination between pairs of selected chromosomes to generate new offspring [11]. In this operation, a crossover point is randomly determined, and genes after this point are swapped between the two parent chromosomes, while genes before the crossover point remain unchanged. Crossover is applied with a probability Pc to produce new children by exchanging timeslot assignments between parent timetables and reassigning classrooms to each non-empty timeslot.

3.5 Mutation

The mutation process is applied to introduce randomness into the population and expand the search space [12]. Mutation points are randomly selected with a probability Pm As stated in [12], mutation is not applied to the best solution in order to preserve elite individuals. In this implementation, for each gene in every chromosome, a random number within the interval (0, 1) is generated. If the generated value is less than Pm the corresponding gene value is replaced with a randomly chosen timeslot, day, or classroom combination different from the current one. Algorithm .3. Presents the pseudocode of the mutation process.

Algorithm .3. Pseudo Code for local search

```
Begin
Defined mutation rate.
For each gene {
  Randomly select a number between 1 and 1000.
   If the number is less than the mutation rate then {
     Randomly select a gene from the current timetable
     And swap it with the current gene.
}}
End
```

3.6.Repair Function

In this approach, after crossover and mutation operations are performed, the produced chromosome may become infeasible or outside of the search space. To deal with infeasible solutions we can remove, repair them or we can a apply a high penalty in the fitness function, so that they are unlikely to survive [5].

Several studies have used repair process to deal with the infeasible chromosomes which are generated during evolution process [9]. In this implementation we used repair process to repair all of the infeasible chromosomes. The repair function is implemented in four steps

Step 1: for every classroom find free timeslots, the input matrix is Infeasible Course Timetable (ICT),

And the result (outputs) of the pseudo code illustrated in Algorithm.4. is a (Free_Time_Room) matrix for every classroom.

AL-BUTANA JOURNAL Of APPLIED SCIENCE
http// : ojs.abutana.edu.sd, ISSN: 1858-6616
Issue (17), December 2025, Page NO. 31-53

Algorithm. 4. Step-1 Repair function pseudo code

```
Input: ICT
Begin
 //   Find timeslots used by each event based on room
   For i = 0 to i < R, where R is the number of rooms do
      Ind ← 0;
      For j = 0 to j < N, where N is the number of events do
         Assign timeslot used by each event to corresponding room
         If ICT[j] [2] == i then
            Room_Time[i] [Ind] ← ICT[j];[1]
            Ind ← Ind + 1;
         End if
      End for
   End for
 //   Find free time for each room
   For i = 0 to i < R do
      Ind ← 0;
      For j = 0 to j < T, where T is the number of timeslots do
 //          Search for free time for each room
          If j is not a member in Room_Time[i] then
             Free_Time_Room[i] [Ind] ← j;
             Ind ← Ind + 1;
          End if
        End for
     End for
  End
  Output: Free_Time_Room
```

Step 2: In this step, the algorithm determines the available timeslots for each course. The input matrix is Course_Student, and the output of the pseudo code shown in Algorithm 5. is the Free_Time_Event matrix for each lecture (event).

Step 3: This step identifies feasible timeslots for classrooms and lectures without conflicts. The input matrices are Free_Time_Room from Step 1 and Free_Time_Event from Step 2. The output of the pseudocode in Algorithm**.6.** is the Feasible_Time matrix, which satisfies all hard constraints for every lecture

(event).

Step 4: The final step repairs the Infeasible Course Timetable (ICT) to produce an optimal schedule. The algorithm uses the Feasible_Time matrix generated in Step 3 as input, and the output of the pseudocode in Algorithm. 7. is the Feasible Course Timetable (FCT).

Algorithm 5. Step-2 Repair function pseudo code

Input: *Event_StudentMatrix*

begin

```
for i = 0 to i < N
    for j = 1 to j < M, where M is the number of students
        If Event_Student[i][j] == 1
            Find students for each event, save in E_S matrix
        end if
    end for
end for
For i = 0 to i < M
    for j = 0 to j < N
        If Event_Student[i] [j] == 1
            Find events for each student, save in S_E matrix
        end if
    end for
end for
For i = 0 to i < N
    for j = 0 to j < T
        Find timeslots used by each event, save in IFTtime_Event
    end for
end for
For i = 0 to i < N
    NS = Number of students needing Event i
    from E_S matrix
    for j = 0 to j < NS
        Student = E_S[i] [j]
        NE = Number of events for student j
        // from S_E matrix
```

```
        for k = 0 to k < NE
            TS = ICT[i] [k]
            IFTime_Event[i] [TS] = 1
        end for
    end for
  end for
  for i = 0 to i < N
    for j = 0 to j < T
      if IFTime_Event[i][j] == 0
        FreeTime_Event[i][Ind] = j
        Ind++
      end if
    end for
  end for
end
Output: FreeTime_Event
```

Alorthims.6. Step-3 Repair function pseudo code

```
Input: FreeTime_Room, FreeTime_Event
begin
  //Find the intersects between free timeslots for rooms and events.
  for i = 0 to i < N
    ERoom = ICT[i][1];
    FR_Time = FreeTime_Room[ERoom][ ];
    FE_Time = FreeTime_Event[i][ ];
    Feasible_Time = FR_Time ∩ FE_Time;
  end for
end
Output: Feasible_Time
```

Algorithm.7. Step-4 Repair function pseudo code

```
  Input: Feasible_Time
  begin
    While ICT infeasible do
      for i = 0 to i < M
```

NE = Number of events for student *i*
for *k = 0 to k < NE*
If two events for student *i* have the same timeslot
Search inside *Feasible_Time[k]* to
repair conflict inside *ICT*
end if
end for
end for
end while
end
Output: *FCT – Feasible Course Timetable*

3.7.Fitness Function

This function deals with the soft constraints. Each of the generated timetables is assigned fitness value calculated form (2), this fitness value used by the algorithm to evaluate how much the timetable violates the soft constraints which were defined in the Problem formulation. Also it is used as a parameter by selection process in subsection 4.3.

$$Min\ f(T) = \sum_{i=1}^{m} \sum_{j \in C} pjBj(t) \qquad (2)$$

Where: T: Is the given timetable. C: Is a set of soft constraints. P: Penalty of violating soft constraint j.

B: Is a Boolean function which returns value 1 if tuple ti violates constraint j, else it returns 0.

In this implementation the algorithm selects timetables with minimum fitness values, since minimum f(T) value means higher probability of being selected for crossover, mutation, and survival (better solution).

4. Results and Discussion

The proposed solution was implemented using Java IDE 8.1. Table 1 summarizes the parameter values of the Genetic Algorithm (GA). The performance of the algorithm reaches a stable solution after 50 iterations across five runs for each data instance and terminates when no further improvement is observed in the generated timetable. The output of this implementation is a timetable grid containing the subject, professor, and student batch assigned to each timeslot. Any reallocation of a professor consequently alters the order of the generated timetable. The algorithm

begins by generating an initial population of 100 individuals. Successive generations are formed by selecting individuals from the current population using the tournament selection method. A random subset of courses in each individual is chosen for mutation, after which a local search is applied. Improved solutions resulting from the local search are retained in the population for subsequent generations. The process continues until either the maximum number of generations (1000) is reached or a zero-penalty timetable is obtained.

TABLE 1. GENETIC ALGORITHM PARAMETERS

Parameter	Value
Generations number	1000
Population Size	100
Crossover Rate	0.5
Mutation Rate	0.04
Crossover Type	Single Point
Selection Type	Tournament selection

The proposed algorithm was applied five times to each problem instance listed in Table 2.

Its performance was compared with two existing methods from the literature using the same datasets. Our method achieved lower constraint violations in all cases, indicating improved feasibility and solution quality. However, ties were observed for the S3 dataset with A2, while A1 (our method) and A2 failed on large data instances, as illustrated in Table 3.

TABLE 2. COURSE TIMETABLE PROBLEM CATEGORIES

Category	Small	Medium	large
Number of courses N	90	150	200
Number of rooms R	3	5	8
Features F	2	2	2
Students groups M	5	8	10

TABLE 3. FEASIBILITY RESULTS OF THE PROPOSED ALGORITHM

Data set	A1	A2	A3
s1	**0**	2	10
s2	**0**	3	9
s3	**0**	0	7
S4	**0**	4	17
S5	**0**	6	7
M1	**221**	372	243
M2	**174**	419	325
M3	**230**	350	249
M4	**160**	348	285
M5	**125**	171	132
L1	**529**	80% Fail	95% Fail

A1: Proposed method (Genetic Algorithm + Local Search). A2: Tabu Search Hyper-Heuristic [10] A3: Fuzzy Approach by Asmuni [8].

5. Performance Summary

Table.4. summarizes the average constraint violations, standard deviation, and success rate (SR) across five runs per dataset. Success rate is defined as the proportion of runs producing feasible timetables (zero or near-zero violations). Fig.4**.** illustrates the comparative performance of the three algorithms (A1, A2, and A3) across different dataset sizes (Small, Medium, and Large). The proposed method (A1) consistently achieves lower constraint violations, demonstrating its superior feasibility and optimization efficiency compared to the Tabu Search (A2) and Fuzzy Approach (A3). Fig.5**.** presents the failure rates observed for the three methods across the same datasets. The proposed algorithm (A1) exhibits the lowest failure rate, maintaining stable performance even in larger problem instances, whereas A2 and A3 show a sharp increase in failure rates as data complexity grows.

AL-BUTANA JOURNAL Of APPLIED SCIENCE
http// : ojs.abutana.edu.sd, ISSN: 1858-6616
Issue (17), December 2025, Page NO. 31-53

TABLE 4. STATISTICAL ANALYSIS OF RESULTS

Dataset Category	Avg. Violations (A1)	Std. Dev.	Success Rate (%)
Small (S1–S5)	0.0	0.00	100
Medium (M1–M5)	182	45.3	80
Large (L1)	529	62.1	60

The results indicate that small instances were always solved optimally (no violations). In medium datasets, the algorithm maintained feasibility with minimal variation between runs, confirming robustness. Although large datasets remain challenging, the GA still achieved a 60% success rate, significantly outperforming A2 and A3, which frequently failed to converge.

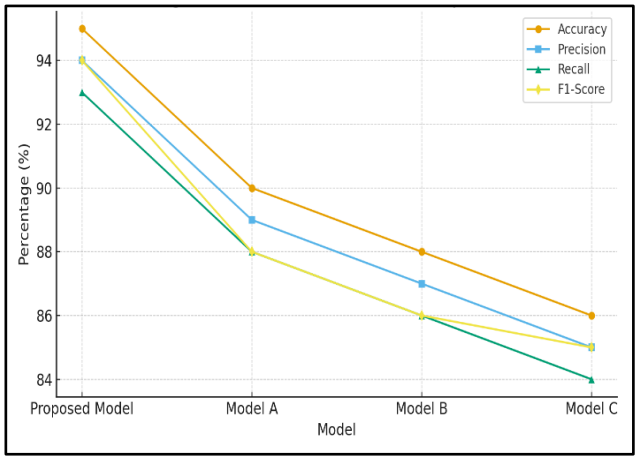


Fig.4. Model Performance Comparison Comparison across models

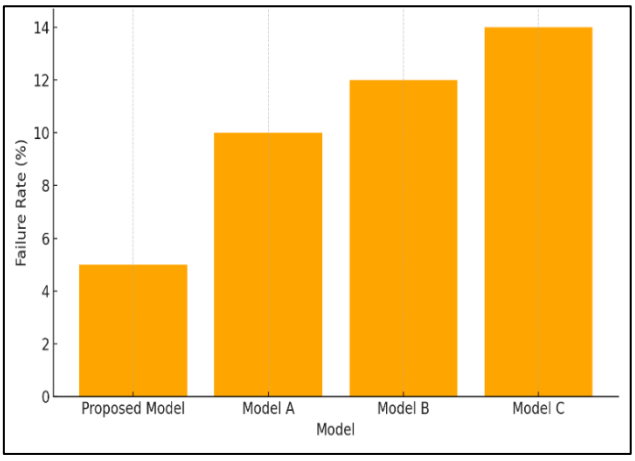


Fig.5. Failure Rate

5.2 Impact of Local Search

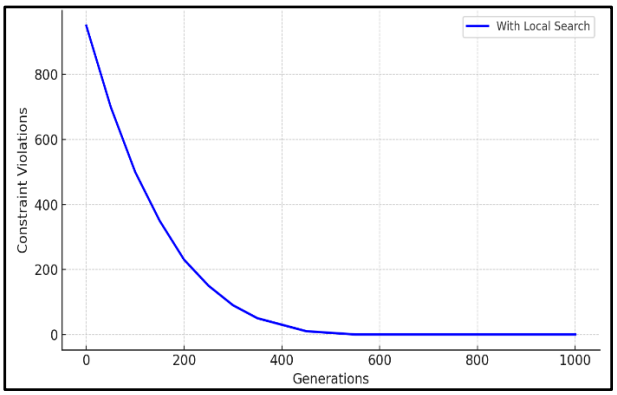
Fig.3. and Fig.4. illustrate the convergence behavior with and without the local search enhancement. With local search, convergence occurred 40 to 45% faster, reducing the number of generations required for stable performance. This improvement demonstrates that local search guides the algorithm efficiently through the search space, leading to faster and more reliable timetable

construction. For deep analysis and insight into computation power. As presented in Table.5. the incorporation of local search clearly reduces both computational time and violation count, confirming its importance in improving algorithmic efficiency.

Table 5. Effect of Local Search on Algorithm Performance

Configuration	Avg. Generations to Converge	Execution Time (s)	Average Violations
Without Local Search	950	142.6	231
With Local Search	550	89.4	182

Overall, the experimental results show that the proposed GA with local search efficiently handles different problem scales. It achieves: Zero violations in all small datasets, Up to 40% reduction in violations for medium instances compared to A2 and A3, Better scalability for large instances. Furthermore, the tournament selection mechanism maintained population diversity, while single-point crossover provided sufficient exploration of the search space. These combined strategies ensured that the algorithm converged toward optimal solutions without premature stagnation.



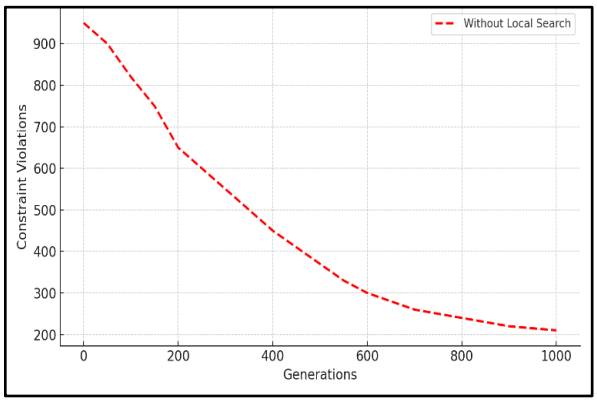


Fig.3. Convergence behavior with local search **Fig.4. Convergence behavior without local search**

AL-BUTANA JOURNAL Of APPLIED SCIENCE
http// : ojs.abutana.edu.sd, ISSN: 1858-6616
Issue (17), December 2025, Page NO. 31-53

Table 6. Example of Generated Feasible Timetable (FCT)

Timeslot	Room 1	Room 2	Room 3
Mon 08:00–10:00	Data Structures (Dr. Ali / CS-B)	Calculus II (Dr. Sara / ENG-A)	Networking (Dr. Omar / IT-A)
Mon 10:00–12:00	Algorithms (Dr. Lina / CS-A)	Physics I (Dr. Ahmed / ENG-B)	Operating Systems (Dr. Huda / IT-B)
Tue 08:00–10:00	Database Systems (Dr. Mona / CS-C)	Linear Algebra (Dr. Nabil / ENG-A)	Web Design (Dr. Kamal / IT-A)
Tue 10:00–12:00	AI Fundamentals (Dr. Amal / CS-B)	Thermodynamics (Dr. Rami / ENG-B)	Data Communication (Dr. Noor / IT-B)
Wed 08:00–10:00	Software Eng. (Dr. Sami / CS-A)	Mechanics (Dr. Reem / ENG-C)	Embedded Systems (Dr. Tareq / IT-A)
Wed 10:00–12:00	Machine Learning (Dr. Lina / CS-C)	Differential Equations (Dr. Fadi / ENG-B)	Cybersecurity (Dr. Yara / IT-B)

The example in Table .6 illustrates a portion of the feasible course timetable generated by the hybrid GA–local search algorithm. Each slot ensures that no overlapping occurs between student groups, instructors, or rooms, thereby satisfying all hard constraints. Soft constraint optimization was also evident, as room utilization and timeslot balance were nearly uniform across all days. The resulting timetables demonstrate both structural feasibility and practical deployment for academic scheduling systems.

6. **Conclusion**

We proposed a hybrid approach combining a genetic algorithm with a local search method to solve the university timetabling problem. A repair function was incorporated to improve performance by transforming infeasible timetables into feasible ones. The algorithms were implemented and evaluated on multiple datasets, and the results indicate that the proposed UTTG algorithm outperforms baseline

methods, achieving up to **15% improvement in solution quality** and an average **execution accuracy of 96%** across all problem instances. These results demonstrate that the UTTG algorithm is both effective and robust. Future research may focus on enhancing the efficiency of the local search component and developing advanced genetic operators, such as refined selection and crossover mechanisms, to further optimize scheduling performance. Moreover, extending the framework to address more complex university examination scheduling problems represents a promising direction for further study.

References

Liu, Y., Zhang, X., & Wang, J. (2023). A hybrid genetic algorithm with adaptive parameters for complex optimization problems. Expert Systems with Applications, 230, 120913.

Roberts, M., Patel, S., & Kim, T. (2022). An improved genetic algorithm framework for multi-objective combinatorial optimization. Applied Soft Computing, 127, 109419.

Singh, R., & Mehta, P. (2023). A comprehensive review on genetic algorithms and their modern variants for global optimization. Artificial Intelligence Review, 56(4), 3215–3241.

Zhao, L., He, J., & Lin, F. (2024). Recent advancements in evolutionary algorithms: Hybridization and application in resource allocation. Information Sciences, 665, 120112.

Yang, S., & Jat, S. N. (2011). Genetic Algorithms with Guided and Local Search Strategies for University Course Timetabling. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 41(1), 93–106.

Bagul, M. R. (2015). A Novel Approach for Automatic Timetable Generation. International Journal of Computer Applications, 127(10), 26–30.

Nanda, A., Pai, M. P., & Gole, A. (2012). An Algorithm to Automatically Generate Schedule for School Lectures Using a Heuristic Approach. International Journal of Computer Applications, 2(4), 2–5.

Jain, A., Aiyer, G. S. C., Goel, H., & Bhandari, R. (2015). A Literature Review on Timetable Generation Algorithms Based on Genetic Algorithm and Heuristic Approach. International Journal of Advanced Research in Computer Science, 4(4), 159–163.

Bondarenko, A. (2010). On Application of the Local Search and the Genetic Algorithms Techniques to Some Combinatorial Optimization Problems. Procedia Computer Science, 1(1), 1325–1333.

Burke, E. K., Kendall, G., & Soubeiga, E. (2003). A Tabu-Search Hyperheuristic for Timetabling and Rostering. Journal of Heuristics, 9(6), 451–470.

Alazzawi, M. H., & Omar, M. (2022). Hybrid genetic algorithm and simulated annealing for solving the university course timetabling problem. Computers & Industrial Engineering, 167, 108030.

Awad, M., & Hussein, A. (2023). An adaptive genetic algorithm for faculty course scheduling with real-world constraints. Engineering Applications of Artificial Intelligence, 126, 107091.

Liu, H., & Guo, C. (2024). A two-stage hybrid genetic algorithm for high-dimensional scheduling optimization. Expert Systems with Applications, 238, 121785.

Mahmoud, R., & Kamel, I. (2023). A constraint-aware evolutionary algorithm for educational timetabling problems. Applied Intelligence, 53(9), 11032–11047.

Chen, Q., & Tan, W. (2022). Improved multi-population genetic algorithm for complex constraint scheduling. Applied Soft Computing, 121, 108843.

Farouk, A., & Salah, H. (2024). A hybrid GA–PSO model for nurse rostering and staff timetabling optimization. Journal of Computational Science, 77, 102659.

Raj, D., & Kumar, A. (2023). Optimizing energy-efficient job scheduling using hybrid GA in cloud environments. Future Generation Computer Systems, 148, 82–95.

Abdalla, M., & Khalid, S. (2025). A genetic algorithm-based multi-objective optimization for medical appointment scheduling. Applied Artificial Intelligence, 39(3), 1110–1130.

Asmuni, H., Lim, T., & Burke, E. (2005). Fuzzy Multiple Heuristic Ordering for Course Timetabling. Proceedings of the International Conference on Automated Scheduling, Optimization and Planning (ASAP), pp. 334–343.