AI for Automated Scheduling (College Timetable Generator)

Brendan Badhe
Student, CSE(AI&ML)
Manipal Institute of Technology (MAHE)
Manipal, India
brendan.mitmpl2023@learner.manipal.edu

Vikasan S Nayak
Student, CSE(AI&ML)
Manipal Institute of Technology (MAHE)
Manipal, India
vikasan.mitmpl2023@learner.manipal.edu

Prajwal Manoj Dube
Student, CSE(AI&ML)
Manipal Institute of Technology (MAHE)
Manipal, India
prajwal3.mitmpl2023@learner.manipal.edu

Dr. Rajesh Mahadeva
Associate Professor, Department of Computer Science and Engineering
Manipal Institute of Technology (MAHE)
Manipal, India
rajesh.mahadeva@manipal.edu

***Abstract*—Timetable scheduling is a complex problem that involves allocating courses, instructors, classrooms, and more while satisfying multiple constraints. This project implements an AI-based timetable generator using Constraint Satisfaction Problem (CSP) and Genetic Algorithm (GA). The CSP approach ensures feasibility by enforcing hard constraints, while GA optimizes the solution by minimizing conflicts. Our system efficiently generates a weekly schedule while considering faculty preferences, room availability, credit hours, and more. The results demonstrate improved scheduling accuracy with reduced conflicts, making it a practical solution for academic institutions.**

Keywords— Timetable Scheduling, Constraint Satisfaction Problem, Genetic Algorithm, Optimization, AI-based Scheduling

I. Introduction

Timetable scheduling is a well-known combinatorial optimization problem that educational institutions face every semester. The implementation addresses the NP-hard timetabling problem by first verifying feasibility through CSP and then optimizing schedules using GA. The system effectively manages multiple constraints including course-instructor assignments, room layouts, and specialized time slots while balancing workload distribution and minimizing scheduling gaps. Manual scheduling is not only time-consuming but also error prone, often leading to conflicts in faculty allocation, room assignments, and course timings. These conflicts result in inefficient resource utilization and dissatisfaction among students and instructors.

The primary challenges in timetable scheduling include:

- Avoiding resource conflicts: Preventing double-booking of instructors or classrooms.
- Ensuring course requirements are met: Assigning the correct number of sessions per course while considering course type (lecture/lab).
- Respecting instructor availability: Ensuring instructors are assigned classes only during their available slots.
- Balancing period distribution: Avoiding excessive gaps between classes for students and distributing instructor workload evenly across the week.

AI-based approaches, particularly Constraint Satisfaction Problems (CSP) and Genetic Algorithms (GA), offer efficient and automated solutions to these problems. This two-phase approach balances the need for constraint satisfaction with optimization quality.

II. Literature Review

Timetable scheduling is still one of the most difficult combinatorial optimization problems in higher education, where limited resources like rooms, teachers, and time slots are to be allocated while satisfying multiple constraints. AI-based methods, including Constraint Satisfaction Programming (CSP) and Genetic Algorithms (GA), have been suggested to solve this problem. This review provides an overview of major studies that investigate these two methods, with an emphasis on their ability to generate optimized timetables.

Zhang and Lau (2005) were the first to apply CSP to university timetable scheduling. The authors formulated the problem as a set of variables (representing resources like rooms, instructors, and time slots), domains (potential values for each variable), and constraints (rules that have to be fulfilled, such as no instructor should be scheduled for more than one class at a time). They used a backtracking search algorithm to identify solutions that satisfy all hard constraints, and thus CSP was a viable method for generating conflict-free timetables. CSP ensures feasibility when generating timetables since it strictly enforces no conflicts to prevent double-booking of instructors or rooms. While CSP-based approaches are dependable, they are computationally intensive, particularly when applied to large-scale timetables with numerous variables and constraints.

The paper "An Automated Timetable Generator for College" (IJREAM) discusses the automation of scheduling through CSP with the aim of creating timetables without conflicts. The method proves that CSP can be used to successfully automate the scheduling process in schools, especially in generating timetables free of conflicts. While CSP guarantees timetables to be conflict-free, Genetic Algorithms (GA) present an option to fine-tune these solutions by enhancing the quality of schedules in general. Genetic Algorithms operate by evolving a population of potential solutions through selection, crossover, and mutation operations. The algorithms can iteratively optimize a list of candidate schedules to reduce soft constraint violations, e.g., instructor workload balancing and room usage maximization.

In a number of studies, GAs have been demonstrated to enhance timetable quality by minimizing soft constraint violations, distributing instructor workloads so that no single instructor is overloaded, and maximizing room and resource utilization so that available facilities are utilized to the best effect. For instance, Zhang and Lau (2005) explained that GAs may be used to improve the CSP solutions, particularly in cases where there are several feasible solutions but one wants to optimize specific areas, such as reducing idle times or room overflows. Although GAs can improve solution quality, they can still take a lot of computation time, especially in big problems, and are not always guaranteed to lead to the global optimum because they are stochastic in nature. The combination of CSP and GA offers an effective solution to the timetable scheduling issue. Although CSP guarantees conflict-free schedules with a sound method, GA further refines such solutions by optimizing soft constraints, which eventually lead to more balanced and efficient timetables. More research on hybrid strategies, taking advantage of the two methods' strengths, has enormous potential for creating sophisticated automated scheduling systems. Future research may also aim to further improve the scalability of CSP and GA methods to work with even bigger datasets and timetable constraints. As AI solutions keep improving, these approaches can become widespread practice in schools for automated timetable generation.

III. Problem Formulation

Given:

- A set of courses $C = \{c_1, c_2,., c_n\}$, each with a specific number of required sessions.
- A set of instructors $I = \{i_1, i_2,., i_m\}$, each assigned to teach one or more courses.
- A set of days $D = \{d_1, d_2, d_p\}$ in the academic week.
- For each day, a set of time periods $P = \{p_1, p_2,., p_k\}$, some designated as regular periods and others as laboratory periods.
- A mapping of instructors to courses.

The objective is to create a schedule that assigns each course session to a specific day and time such that:

A. Hard constraints:

1. Each course must be assigned exactly the required number of sessions (credit hours).

2. Course sessions cannot share the same time slot (no double booking).

3. The same course should not be scheduled on the same day multiple times.

4. Laboratory courses must be scheduled in laboratory periods, and non-laboratory courses should be scheduled in regular periods.

B. Soft constraints:

1. Minimize gaps between classes on any given day

2. Balance the teaching load across all days of the week

3. Minimize instructor scheduling inefficiencies

4. Even distribution of courses across available days

The problem can be mathematically modeled as a constraint satisfaction problem with an objective function that quantifies violations of soft constraints. The solution space is combinatorial and grows exponentially with the number of courses, instructors, and time slots.

IV. Methodology

Our solution consists of two main stages: a feasibility check using constraint satisfaction problem and an optimization phase using genetic algorithm.

A. Constraint Satisfaction

In the first phase, we use a constraint satisfaction problem (CSP) formulation to determine whether a feasible solution exists. This step is crucial because it quickly identifies instances where hard constraints cannot be simultaneously satisfied, saving computational resources that would otherwise be wasted on optimization.

The CSP model defines variables for each course session, with domains consisting of all day period combinations. Constraints are added to enforce the requirements:

1. The AllDifferentConstraint ensures no double booking occurs.
2. Course-specific constraints enforce that sessions of the same course are scheduled on different days.
3. Domain restrictions ensure that lab courses are scheduled during lab periods, and regular courses during regular periods.

The Python Constraint library is used to implement this phase. If a solution exists, the algorithm proceeds to the optimization phase; otherwise, it reports that the problem is infeasible.

B. Initial Schedule Generation

Before applying the genetic algorithm, we generate an initial schedule that satisfies all hard constraints while attempting to partially satisfy soft constraints. This initialization procedure uses a greedy approach:

1. Courses are sorted by type (regular courses first, then lab courses)
2. For each course, the algorithm searches for the best day and slot that minimizes a combined score based on gap creation and load balance.
3. The selection process considers period compatibility (lab courses to lab periods) while avoiding double bookings.

This procedure creates a feasible starting point that already has some desirable properties, which helps the genetic algorithm converge faster with high-quality solutions.

C. Genetic Algorithm Optimization

The genetic algorithm optimizes the schedule by evolving a population of feasible solutions over multiple generations. The python DEAP library is used to implement this phase. We define the following GA components:

1. Representation: Everyone in the population is represented as a list of tuples (course, day, period, instructor), encoding a complete timetable.
2. Fitness Function: The fitness of a schedule is evaluated based on:
    a. Penalty for missing or excessive course sessions.

   b. Penalty for scheduling the same course multiple times on the same day.
   c. Penalty for double-booked slots
   d. Penalty for period type incompatibility
   e. Penalty for unbalanced day loads
   f. Penalty for gaps between classes
3. Crossover Operator: A single-point crossover exchanges portions of two parent schedules to create offspring schedules.
4. Mutation Operator: Two types of mutations are implemented:
   a. Rebalancing: Moves a session from a heavily loaded day to a lightly loaded day
   b. Gap reduction: Within a day, reschedules a session to minimize gaps.
5. Selection: Tournament selection with a tournament size of 3 is used to select individuals for reproduction.

The algorithm runs for a maximum of 25 generations or until a time limit of 60 seconds is reached, maintaining a hall of fame to preserve the best solution found.

D. Fitness Evaluation

The fitness function assigns penalties for:

1. Missing or extra course sessions (100 and 50 points respectively)
2. Multiple sessions of the same course on the same day (30 points)
3. Double-booked slots (200 points)
4. Period type incompatibility (10-20 points)
5. Unbalanced day loads (10 points per unit deviation)
6. Scheduling gaps (20 points per gap)

E. Schedule Formatting and Analysis

The final schedule is converted into a structured format suitable for implementation and analysis. The system provides comprehensive statistics including:

1. Day load distribution
2. Total session count and average per day
3. Standard deviation of the day load, measuring balance quality
4. Gap analysis, identifying and quantifying inefficiencies.
5. Instructor workload analysis, showing session distribution across days.

This analysis helps administrators understand the quality of the generated schedule and identify potential areas for manual adjustment if necessary.

V. Experimental Results

We evaluated our approach using a realistic academic scheduling scenario with the following characteristics:

- 8 courses (5 regular courses, 3 laboratory courses)
- 5 instructors with assigned teaching responsibilities
- 6 days (Monday through Saturday)
- Two different daily layouts with varying period structures

A. Feasibility Analysis

The CSP phase successfully identified the feasibility of the test case, confirming that a valid schedule satisfying all hard constraints exists. This validation step took less than one second, demonstrating the efficiency of the constraint programming approach for feasibility checking.

B. Schedule Quality

The genetic algorithm produced a high-quality schedule with the following characteristics:

- Sessions were well-distributed across available days.
- All course session requirements were met.
- Laboratory courses were correctly assigned to laboratory periods.
- The number of gaps between classes was minimized.
- Instructor schedules showed reasonable teaching distribution.

C. Computational Performance

Our hybrid approach demonstrated good computational efficiency:

- CSP Feasibility check: <1 second
- Initial schedule generation: <1 generation
- A time limit of 60 seconds for the genetic algorithm
- Population size of 50 individuals
- Maximum of 25 generations
- Crossover probability of 0.5
- Mutation probability of 0.3

The total computation time remained under one minute, making the approach practical for real-world scheduling scenarios where schedules are typically generated once per term.

Algorithmic Complexity:

- CSP Feasibility Check: Exponential in the worst case but practical for the problem size
- Initial Solution Generation: O (|courses| × |days| × |periods|)
- Genetic Algorithm: O (population_size × generations × evaluation_complexity)

D. Comparison with Baseline Approaches

To establish the effectiveness of our hybrid approach, we compared it against two baseline methods:

1. A pure CSP approach that attempts to satisfy both hard and soft constraints.
2. A pure GA approach without the feasibility pre-check and specialized initialization.

The results showed that:

- The pure CSP approach struggled to find optimal solutions within reasonable time limits when considering all constraints.
- The pure GA approach sometimes produced infeasible schedules or required more generations to reach comparable quality.
- Our hybrid approach consistently produced feasible, high-quality schedules with better computational efficiency.

E. Modularity and Extensibility

The code is structured modularly with clear separation between:

- Core scheduling logic.
- Constraint formulation
- Genetic algorithm components

- Result formatting and statistics

F. Robustness and Features

The implementation includes several robust mechanisms:

- Exception handling throughout critical sections
- Fallback to initial schedules if genetic optimization fails.
- Domain filtering to ensure period compatibility.

VI. Results and Discussion

A. Output Format

The system provides comprehensive output including:

- Complete timetable for each day
- Period-by-period breakdown with course and instructor information
- Statistical analysis of the schedule quality
- Detailed instructor schedules with workload distribution

B. Visualization and Reporting

The output presents:

- Course distribution metrics
- Standard deviation of daily loads
- Gap analysis across the schedule
- Instructor teaching load and day distribution

C. Limitations and Future Work

1) Current Limitations

- Limited to a single classroom
- Fixed instructor-course assignments
- Deterministic layout assignments
- No student cohort or group constraints

2) Potential Enhancements

- Multi-room scheduling capabilities.
- Instructor's availability constraints
- Real-time rescheduling capabilities
- Improved parallelization of the genetic algorithm

VII. Conclusion

This paper presented a hybrid CSP-GA approach for the academic course timetabling problem. The two-phase method first establishes feasibility through constraint satisfaction techniques and then optimizes the schedule using a genetic algorithm with specialized operators.

Experimental results demonstrated that our approach effectively generates high-quality schedules that satisfy all hard constraints while optimizing soft constraints related to gap minimization, load balancing, and instructor satisfaction. The computational efficiency of the hybrid approach makes it practical for real-world scheduling scenarios.

The presented approach offers a valuable tool for academic administrators tasked with creating efficient and balanced course schedules. Its modular design allows for customization to address institution specific requirements and preferences.

References

1. Zhang, L., & Lau, S. K. (2005). Constructing university timetable using constraint satisfaction programming approach. In International Conference on Computational Intelligence for Modelling, Control, and Automation (CIMCA-IAWTIC'06).
2. IJREAM. (n.d.). An Automated Timetable Generator for College.
3. Russell, S., & Norvig, P. (2016). Artificial Intelligence: A Modern Approach (3rd ed.). Pearson.
4. Genetic Algorithms In Scheduling | Restackio, accessed on March 15, 2025, https://www.restack.io/p/evolutionary-algorithms-answer-genetic-algorithms-scheduling cat-ai