Classroom Allocation to sections (CSP)

Project submitted to the

SRM University – AP, Andhra Pradesh

for the Lab Evaluation of Course Artificial Intelligence (CSE455)

Under the Supervision of

Dr. Ashu Abdul

Assistant Professor

Department of Computer Science and Engineering

For

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

Chakilam Srija(AP22110010418)

Paladugu Vaishnovi(AP22110010420)

Sruthi Vihitha Potluri(AP22110010425)

Kolagotla Jahnavi(AP22110010434)

Lingam Chaitali(AP22110010437)

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

[April, 2025]

ACKNOWLEDGEMENT

We sincerely express our gratitude to our respected professor, Dr. Ashu Abdul, for his unwavering guidance, insightful feedback, and encouragement throughout the development of Classroom Allocation Using Constraint Satisfaction Problem (CSP). His expertise and patience were instrumental in helping us navigate the complexities of optimization techniques and algorithmic design.We are deeply thankful to our team members for their collaborative spirit, innovative problem-solving, and dedication to implementing a robust solution using Simulated Annealing and Greedy Algorithms. Each member’s contributions—whether in refining the CSP model, designing the PyQt5 GUI, or optimizing the scheduling logic—were vital to the project’s success.

This project has been an enriching journey, allowing us to explore the practical applications of constraint satisfaction, heuristic search, and combinatorial optimization. The challenges we overcame while balancing classroom utilization, conflict resolution, and fairness across sections have significantly deepened our understanding of algorithmic efficiency and real-world system design.We also extend our appreciation to our peers for their constructive feedback and to the institution for providing the resources that made this work possible. The knowledge gained through this experience will undoubtedly serve as a foundation for future endeavors in computational problem-solving.

Table of Contents

ABSTRACT

Classroom allocation is a complex scheduling problem that requires balancing multiple constraints, including room availability, time slots, and equitable resource distribution. In this project, we address the challenge of assigning classrooms to course sections across weekly schedules using a Constraint Satisfaction Problem (CSP) framework. Unlike traditional heuristic-based approaches, our solution leverages Simulated Annealing combined with a Greedy Algorithm to efficiently explore the solution space while minimizing conflicts and ensuring uniform classroom utilization.Key constraints include:No overlapping allocations (a classroom cannot be assigned to multiple sections in the same time slot).Fair distribution of classrooms among all sections.Balanced usage to prevent over- or under-utilization of any classroom.

Our implementation begins with a Greedy Algorithm to generate an initial feasible schedule, followed by Simulated Annealing to iteratively refine the solution by minimizing a cost function that penalizes constraint violations. The algorithm dynamically adjusts classroom assignments to improve fairness and efficiency while maintaining hard constraints.To enhance usability, we developed an interactive PyQt5-based GUI that visualizes the generated timetable, displays classroom utilization statistics, and allows for dynamic adjustments. Experimental results demonstrate that our approach produces conflict-free schedules within seconds, even for moderately sized academic setups, proving its practicality for real-world deployment.

This project highlights the effectiveness of metaheuristic optimization in solving CSPs, offering a scalable and flexible alternative to purely heuristic-driven methods. The system's ability to balance multiple objectives—fairness, efficiency, and constraint satisfaction—makes it a viable tool for academic scheduling in institutions with dynamic resource requirements.

INTRODUCTION

Classroom Allocation Problem:

The allocation of classrooms to course sections across multiple days and time slots presents a significant optimization challenge in academic institutions. Key constraints include ensuring no overlapping assignments, maintaining fair distribution of resources among sections, and achieving balanced utilization of all available classrooms. Traditional manual scheduling methods often struggle to meet these requirements efficiently, leading to suboptimal resource allocation and potential conflicts. This project addresses these challenges by formulating the problem as a Constraint Satisfaction Problem (CSP) and employing advanced optimization techniques to generate feasible and equitable schedules.

Why Use CSP with Simulated Annealing and Greedy Algorithms?

While CSP provides a robust framework for modeling scheduling constraints, solving it efficiently requires careful algorithmic design. This solution combines a Greedy Algorithm for initial feasible schedule generation with Simulated Annealing to refine the solution iteratively. This hybrid approach ensures that hard constraints are satisfied while optimizing for fairness and resource utilization. The Greedy Algorithm quickly constructs an initial assignment, and Simulated Annealing explores the solution space to minimize conflicts and improve balance, avoiding local optima through probabilistic acceptance of suboptimal moves.

Constraint Satisfaction Problem (CSP) in Classroom Allocation

In our CSP model, each section is treated as a variable, and its domain consists of all possible (classroom, day, time-slot) combinations. Constraints enforce no double-booking of classrooms, mandatory weekly sessions for each section, and uniform classroom usage. By framing the problem this way, we systematically eliminate invalid assignments and focus on feasible solutions.

Solution Strategy: Hybrid Optimization

Greedy Initialization: A Greedy Algorithm assigns classrooms to sections based on immediate availability, ensuring rapid construction of a conflict-free but potentially unbalanced schedule.

Simulated Annealing: This metaheuristic refines the initial schedule by iteratively perturbing assignments, accepting changes that reduce a cost function (penalizing uneven utilization and conflicts). The annealing process balances exploration and exploitation, converging to a high-quality solution.

Advantages Over Traditional Methods:

Scalability: Our approach handles larger problem sizes more effectively than brute-force or pure CSP solvers.

Fairness: Explicit optimization of classroom usage ensures equitable distribution.

Practicality: The PyQt5 GUI provides an intuitive interface for visualizing and adjusting schedules, making the system accessible to administrators.

By integrating CSP with Simulated Annealing and Greedy Algorithms, our solution achieves both efficiency and fairness, offering a practical tool for academic scheduling. The following sections detail our methodology, implementation, and results.

METHODOLOGY

Step 1: System Initialization

Objective: Establish the fundamental scheduling framework

Implementation:

Define temporal structure:

5 weekdays (Monday-Friday)

5 daily time slots (8:00-9:30 to 15:45-17:15)

Configure resources:

Classrooms: C101-C105

Academic sections: S1-S5

Courses: AI, JAVA, SE, DIP, SNA

Initialize tracking systems:

classroom_assignments: Tracks available classrooms per section

classroom_slot_tracker: Prevents double-booking conflicts using hash sets

section_slot_counts: Ensures balanced time slot distribution

Step 2: Greedy Schedule Construction

Objective: Generate a feasible initial solution

Algorithm:

Combinatorial Preparation:

Generate all possible (day, time-slot) pairs (25 combinations)

Randomly shuffle to avoid positional bias

Section-wise Allocation:

For each (day, slot) in shuffled combinations:

a. Identify sections needing more classrooms

b. Select section with fewest assigned classrooms

c. Assign unused classroom to this section

d. Handle conflicts via fallback mechanism:

- If preferred classroom booked, use any available

e. Distribute courses using:

- Minimum-count selection (balanced distribution)
- 30% random override (diversity)

f. Update all tracking dictionaries

Output: Raw schedule guaranteeing:

No classroom conflicts

All sections use all classrooms at least once

Step 3: Energy Calculation & Penalty System

Objective: Quantify schedule quality

Evaluation Metrics:

Hard Constraints (Binary Penalties):

+1 energy per section not using all classrooms

Soft Constraints (Continuous Penalties):

Absolute deviation from ideal classroom utilization:

avg_usage = total_assignments / classroom_count

penalty += |actual_usage - avg_usage| for each classroom

Energy Minimization Goal:

Perfect schedule: energy = 0

Typical optimized solution: energy ≤ 2

Step 4: Simulated Annealing Optimization

Objective: Refine schedule through metaheuristic search

Process Flow:

Parameter Initialization:

Temperature (T) = 1000

Cooling rate = 0.995

Max iterations = 1000

Core Loop:

While (T > 0.1) and (iterations < max_iterations):

a. Generate neighbor schedule via full reinitialization

b. Calculate ΔE = new_energy - current_energy

c. Acceptance Criteria:

- Always accept better solutions (ΔE < 0)
- Accept worse solutions with probability exp(-ΔE/T)

d. Update best solution if improved

e. Apply geometric cooling: T *= cooling_rate

Termination:

Early exit if energy=0 (perfect solution)

Final output: Best-found schedule

Step 5: PyQt5 GUI Implementation

Objective: Interactive schedule visualization

Component Breakdown:

Main Window Structure:

Central widget with vertical layout

Control panel (Generate button + status label)

Schedule table (5x5 grid)

Dual chart view (QtCharts integration)

Table Visualization:

For each cell (day, time-slot):

a. Retrieve classroom, section, course

b. Apply section-specific background color

c. Format text (bold, centered, multi-line)

d. Disable editing (read-only)

Color mapping: S1=Blue, S2=Orange, S3=Green, etc.

Analytics Charts:

Classroom Utilization:

Bar chart showing bookings per classroom

X-axis: Classroom IDs, **Y-axis:** Usage count

Section Allocation:

Bar chart comparing total assignments per section

Visual fairness indicator

Style Engineering:

CSS-like styling via Qt Style Sheets

Features:

Gradient button hover effects

Consistent color palette (#3a7ca5 primary)

Event Handling:

Generate button triggers:

Schedule regeneration

Real-time status updates

Chart refresh

Async UI updates during computation

Step 6: Constraint Validation

Objective: Ensure solution feasibility

Verification Mechanisms:

Hard Constraints:

Automatic conflict detection via classroom_slot_tracker

Full classroom coverage check in energy calculation

Soft Constraints:

Visual inspection through utilization charts

Energy value displayed in status bar

Error Handling:

Try-catch blocks for schedule generation

User-friendly error messages

Hybrid Optimization:

Combines greedy construction ($O(n)$) with simulated annealing ($O(kn^2)$)

Balances speed and solution quality

Energy-Based Evaluation:

Hierarchical penalty system prioritizes hard constraints

Continuous penalties drive balanced utilization

PyQt5 Advantages:

Hardware-accelerated rendering for smooth visuals

Native-looking widgets across platforms

MVC architecture separates logic and presentation

Adaptive Cooling:

Geometric temperature reduction balances exploration/exploitation

Dynamic acceptance probability avoids local optima

PERFORMANCE CHARACTERISTICS

Phase	Time Complexity	Key Operations
Greedy Initialization	O(n)	Slot assignment, conflict checks
Energy Calculation	O(n)	Dictionary traversals
Annealing Loop	$O(kn^2)$	Neighborhood generation
GUI Rendering	O(1)	Qt's native optimizations

This presents a breakdown of different phases in a simulated annealing-based scheduling system.
Each phase is listed with its time complexity and key operations involved.
Greedy Initialization and Energy Calculation run in linear time, while the Annealing Loop has a higher complexity due to iterative neighborhood exploration.
GUI Rendering is highly efficient, benefiting from Qt's native performance optimizations.

RESULTS

	8:00-9:30	9:45-11:15	11:30-13:00	14:00-15:30	15:45-17:15
Monday	S3 C104 JAVA	S2 C103 JAVA	S2 C101 AI	S3 C105 AI	S3 C101 SE
Tuesday	S1 C101 DIP	S5 C103 DIP	S4 C105 SNA	S2 C104 SNA	S1 C102 DIP
Wednesday	S5 C101 SNA	S1 C105 JAVA	S4 C103 JAVA	S5 C105 AI	S4 C101 DIP
Thursday	S3 C103 DIP	S2 C105 SE	S4 C104 SNA	S5 C104 SE	S4 C102 SE
Friday	S2 C102 SNA	S3 C102 AI	S1 C103 SE	S1 C104 AI	S5 C102 JAVA

	8:00-9:30	9:45-11:15	11:30-13:00	14:00-15:30	15:45-17:15
Monday	S4 C101 SNA	S3 C104 AI	S5 C101 SE	S1 C104 SE	S3 C101 SE
Tuesday	S5 C102 JAVA	S4 C102 SE	S1 C105 SNA	S3 C103 JAVA	S4 C105 DIP
Wednesday	S5 C105 DIP	S1 C101 DIP	S1 C102 JAVA	S3 C105 AI	S2 C103 JAVA
Thursday	S2 C102 SE	S2 C101 DIP	S2 C104 AI	S4 C104 JAVA	S5 C104 SNA
Friday	S2 C105 SNA	S3 C102 DIP	S1 C103 AI	S4 C103 SNA	S5 C103 AI

FUTURE SCOPE

The current hybrid optimization approach for classroom allocation provides a strong foundation, but there are several exciting directions for future enhancements and applications. Below are key areas for expansion:

Enhanced Optimization Techniques

a) Multi-Objective Optimization

Incorporate additional constraints:

Faculty preferences (teacher availability, subject expertise)

Room capacities (account for varying classroom sizes)

Student flow (minimize cross-campus movement)

Use Pareto optimization to balance competing objectives.

b) Machine Learning Integration

Predictive Cooling in Simulated Annealing:

Use reinforcement learning to dynamically adjust cooling rates based on optimization progress.

Initial Schedule Generation:

Train a model on historical data to produce better initial schedules.

c) Hybrid Metaheuristics

Combine simulated annealing with:

Genetic Algorithms (for population-based search)

Tabu Search (to avoid revisiting poor solutions)

Ant Colony Optimization (for dynamic resource allocation)

CONCLUSION

This classroom scheduling system effectively addresses academic timetabling challenges through a hybrid optimization approach. The greedy algorithm quickly generates an initial feasible schedule, while simulated annealing refines it for balanced resource distribution. The energy function ensures no conflicts exist and promotes fair classroom usage across sections.

The PyQt5 interface provides an intuitive visualization of the schedule, displaying color-coded assignments and utilization statistics. Users can regenerate schedules and verify constraints effortlessly. The system's efficiency allows it to produce high-quality solutions in seconds for typical academic scenarios.Future enhancements could integrate additional constraints like faculty availability and room capacities. Expanding to multi-campus coordination or incorporating machine learning for predictive scheduling would further improve its applicability. The modular design also allows adaptation to other domains like corporate meeting room bookings or hospital shift planning.

By combining algorithmic optimization with practical usability, this project demonstrates how automated systems can solve complex scheduling problems while maintaining flexibility for real-world requirements. Its balanced approach between speed, fairness, and user-friendliness makes it a reliable tool for academic institutions and beyond