UtilitasMathematica

ISSN 0315-3681 Volume 122, 2025

College Timetable Generation Using Graph Neural Networks and Reinforcement Learning

Ghazala Khan[1], Swapnil Sonawane[2]

[1]Vidyalankar Institute of Technology, Mumbai, India
[2]Vidyalankar Institute of Technology, Mumbai, India
ghazalakhan0603@gmail.com, swapnil.sonawane@vit.edu.in

Abstract. Automating academic timetable scheduling is a complex and resource-intensive challenge faced by many multi-department educational institutions. This paper presents a hybrid web-based timetable generation system Graph Neural Networks (GNN), and Google OR-Tools to efficiently create optimized, conflict-free timetables for undergraduate engineering programs. The system supports multiple departments, years, and faculty constraints such as availability preferences, designation priorities, and subject difficulty. Using a PHP-MySQL frontend coupled with a Python backend, the platform enables real-time data management and iterative schedule refinement. Experimental evaluation demonstrates that the system produces high-quality timetables within minutes, strictly enforces hard constraints (no double-booking, lunch breaks, practical sessions), and accommodates soft preferences effectively. Finally, a Proximal Policy Optimization–based Reinforcement Learning (RL) agent performs a brief local "polishing" pass on the CP-SAT output to further reduce soft-constraint penalties via learned slot-level adjustments. The modular architecture ensures scalability and extensibility for future academic planning needs. This work contributes a practical AI-augmented solution for improving institutional scheduling efficiency and quality.

Keywords: Academic Timetable Generation, Graph Neural Networks, Constraint Satisfaction, OR-Tools, Automated Scheduling, Multi-Department Timetabling, Faculty Preferences, Artificial Intelligence, Optimization.

1. INTRODUCTION

Timetable generation in academic institutions, especially in engineering colleges, is a laborious and error-prone process due to the myriad of constraints and the diversity of resources involved. Manual scheduling often results in conflicts such as overlapping teacher assignments, inadequate allocation of classrooms, and failure to respect faculty availability or institutional policies, leading to suboptimal academic experiences.[1][2]

Advances in artificial intelligence and optimization have enabled the development of automated scheduling systems that can address these complexities.[3][6] Emerging machine learning techniques such as Graph Neural Networks (GNN) enhance the system's ability to model relational dependencies inherent in scheduling entities.[4] Additionally, constraint programming libraries such as Google OR-Tools allow for precise enforcement of hard scheduling rules.

This paper presents a hybrid scheduling system combining GNN, and OR-Tools, implemented with a user-friendly PHP-MySQL frontend and a Python backend engine. The system is designed to generate conflict-free timetables for multi-department undergraduate programs, incorporating detailed faculty preferences, subject difficulties, and institutional constraints such as mandatory lunch breaks and practical session timings.

The contributions of this work include:

- An integrated AI-based approach leveraging RL and GNN for intelligent timetable search and conflict prediction.
- Use of OR-Tools for exact constraint satisfaction to refine final schedules.
- A modular web-based platform enabling real-time data management and iterative timetable refinement.
- Empirical validation demonstrating efficient timetable generation with high constraint satisfaction and scalability.

2. LITERATURE REVIEW

Academic timetable generation has long been recognized as a challenging combinatorial optimization problem due to the vast search space and multiple constraints involved. Traditional approaches include heuristic and rule-based methods, which often rely on manual intervention and lack scalability.[3]

Recent advancements have introduced machine learning techniques, particularly Graph Neural Networks (GNN), to capture and exploit the relational structure of scheduling problems. Scheduling entities such as teachers, subjects, rooms, and time slots naturally form a graph with edges representing constraints or conflicts. GNNs leverage this structure to learn patterns of conflicts and assist in guiding the search process toward feasible regions, improving convergence and solution quality.[11]

Constraint programming frameworks like Google OR-Tools have gained popularity due to their ability to encode complex hard constraints and perform exact optimizations. OR-Tools uses a constraint satisfaction problem (CSP) solver that prunes infeasible solutions efficiently, enabling refinement of candidate timetables generated by metaheuristic or ML-based approaches.

Existing commercial and academic timetable generators often lack transparency or are not easily customizable to institution-specific constraints, motivating research into open, modular, and AI-augmented scheduling systems. This paper builds upon these approaches by integrating GNN, and OR-Tools within a scalable web-based platform that supports multi-department, multi-year academic scheduling with detailed faculty preferences and constraint prioritization.

3. SYSTEM ARCHITECTURE

Effective timetable generation requires a clear understanding of the functional and non-functional requirements, as well as a detailed specification of the constraints to be managed.

Functional Requirements

A. **Role-Based Access Control:** The system supports distinct user roles such as administrators (e.g., department heads) and faculty members. Administrators can manage data entities, define constraints, and initiate timetable generation, while faculty can view schedules and submit availability preferences.

B. **Entity Management:** Users can add, update, and delete key academic entities including departments, years, teachers, subjects, classrooms, and laboratories. Each entity stores attributes relevant for scheduling such as teaching hours, subject difficulty, and room capacity.

C. **Faculty Time Preferences:** Faculty members can specify general availability (morning, afternoon) and specific day-time preferences to influence their timetable allocation.

D. **Constraint Definition and Management:** Administrators can define both hard and soft constraints. Hard constraints include absolute rules such as no double-booking of teachers or rooms, mandatory lunch breaks, and scheduling practical sessions post-lunch. Soft constraints include preferences for time slots, avoidance of consecutive difficult subjects, and designation-based priority.

E. **Timetable Generation and Editing:** The system initiates an automated timetable generation process that produces conflict-free schedules. Post-generation, administrators can manually refine the timetable through an intuitive user interface.

F. **Multi-Level Timetable Visualization:** Generated timetables are viewable by department, year, subject, and individual faculty, with options to export schedules in PDF or CSV formats.

Non-Functional Requirements

A. **Performance:** The system must generate timetables efficiently to support real-time usage during scheduling cycles.

B. **Scalability:** The architecture should support multiple academic departments, several undergraduate years, and expansion to future batches or postgraduate programs with minimal configuration changes.

C. **Usability:** The user interface must be responsive and intuitive, supporting desktop and mobile devices.

D. **Security:** User authentication and session management ensure secure access control. Sensitive data such as faculty preferences and schedules are protected.

E. **Modularity:** The backend design facilitates modular integration of Graph Neural Networks, and constraint solvers, allowing for easy maintenance and upgrades.

Constraint Specification

A. Hard Constraints:

- No teacher or room double-booking across departments.
- Practical subjects scheduled only after the lunch break (typically 1 PM – 2 PM).
- Maximum theory teaching hours per day and week per faculty member.
- Fixed lunch break periods respected.

B. Soft Constraints:

- Faculty time preferences (general and specific day-time).
- Avoidance of back-to-back difficult subjects to reduce cognitive load.
- Designation-based priority in scheduling slots (e.g., Head of Department > Associate Professor > Assistant Professor > Adhoc > Visiting faculty).

4. SYSTEM ARCHITECTURE AND METHODOLOGY

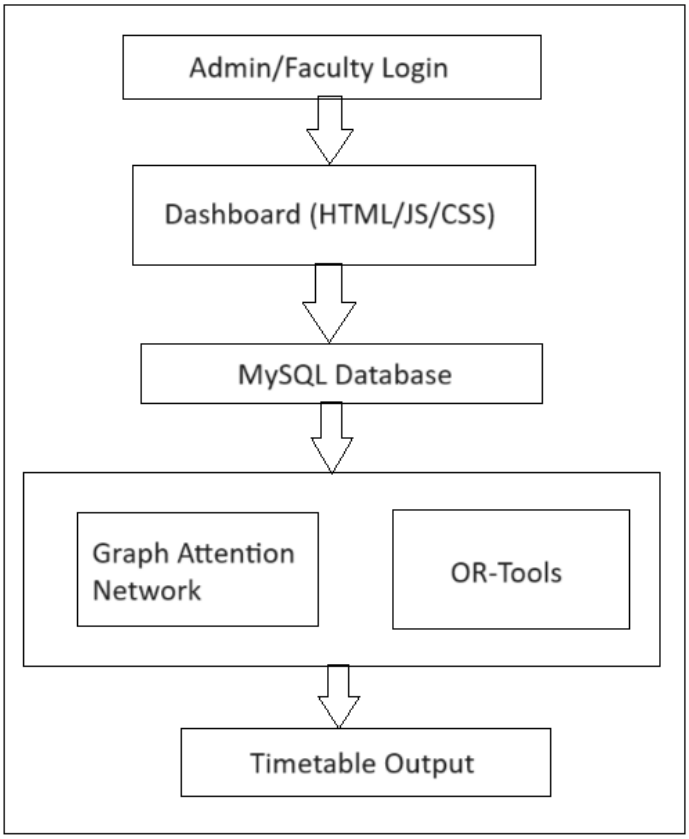

Fig. 1: System Architecture showing interaction between frontend, backend logic, MySQL database, and hybrid scheduling components.

This section details the overall system architecture and the hybrid methodology employed for automated timetable generation.

The system is composed of three main layers, as illustrated in Figure 1:

A.Frontend

Developed using PHP, HTML, CSS, and JavaScript, the frontend provides user interfaces for administrators and faculty. It supports data management (departments, teachers, subjects, classrooms), timetable visualization, preference input, and schedule editing.

B.Backend

Implemented in Python, the backend consists of three interrelated modules:

- Graph Neural Network (GNN) Module: Processes the relational graph of scheduling entities to predict conflict zones for improved convergence and solution quality.
- Google OR-Tools Constraint Solver: Performs exact optimization to finalize timetables by strictly enforcing hard constraints and fine-tuning assignments.
- We invoke an RL polisher to minimize remaining soft-constraint penalties without violating any hard rules.

C.Database

A MySQL relational database stores all academic data, including departments, faculty, subjects, classrooms, user roles, preferences, and generated timetables. The database facilitates persistent storage and retrieval required by both frontend and backend modules.

The architecture promotes modularity, enabling independent development, testing, and maintenance of each component. Communication between PHP frontend and the Python backend occurs via RESTful APIs or command-line interfaces, allowing seamless integration.

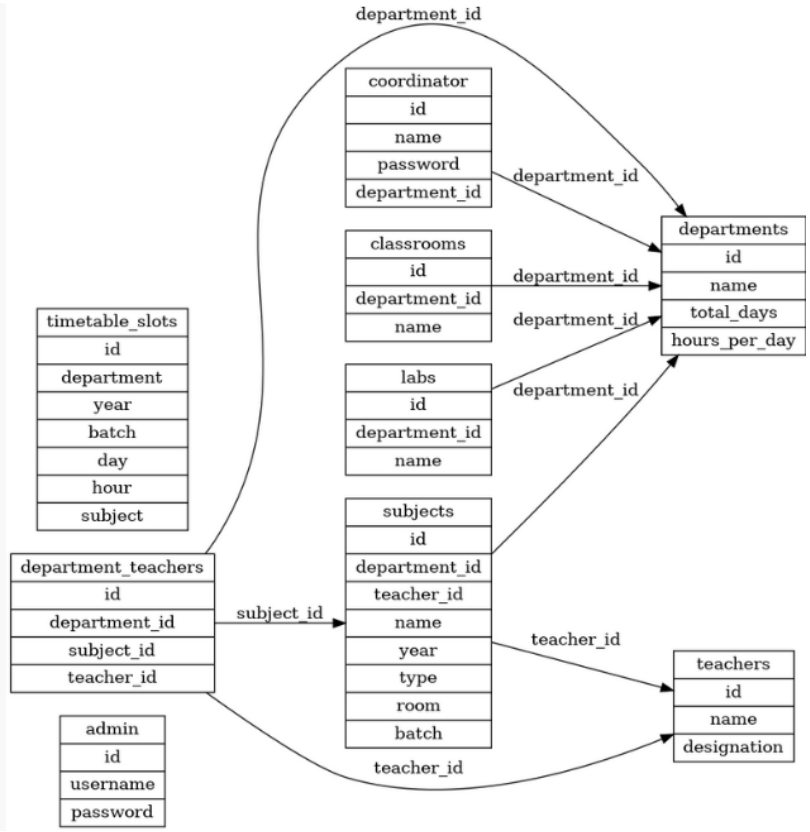

Fig. 2: Database Design

5. METHODOLOGY

A. Data Modeling

All core scheduling elements are encapsulated within a unified graph structure: teacher nodes represent individual faculty members, subject nodes denote each course with its theory or practical classification, classroom nodes cover both lecture halls and specialized labs, and timeslot nodes correspond to each discrete day–hour interval for every batch.[11] Edges in this heterogeneous graph encode essential relationships and constraints—teacher–subject edges limit which instructors can teach which courses based on qualifications, room–timeslot edges capture the availability and capacity restrictions of physical spaces, and subject–timeslot edges reflect historical assignment patterns stored in timetable_slots. This comprehensive graph serves as the foundational data representation, enabling the GNN to learn nuanced interaction patterns among entities and providing context-sensitive guidance to the CP-SAT solver during schedule optimization.

B. Graph Neural Network (GNN) Integration

The GNN component leverages a two-layer GATConv architecture defined in gnn_model.py. Initially, the gnn_training_data.py script exports the graph—complete with node and edge lists—to gnn_data.json.[4] During training, each node is assigned a one-hot identity feature, and the model is optimized for 100 epochs using the Adam algorithm (learning rate set to 0.01) against a binary-cross-entropy loss where every historical assignment is treated as a positive example. After convergence, the trained weights are saved to gnn_weights.pt. At inference time, gnn_predictor.py reloads both gnn_data.json and gnn_weights.pt, executes a forward pass in evaluation mode, and produces sigmoid-activated feasibility scores for each timeslot node. These scores—ranging between 0 and 1—quantify how historically compatible each teacher–subject–timeslot pairing has been, and are written out to gnn_predictions.json for use in the scheduling phase.

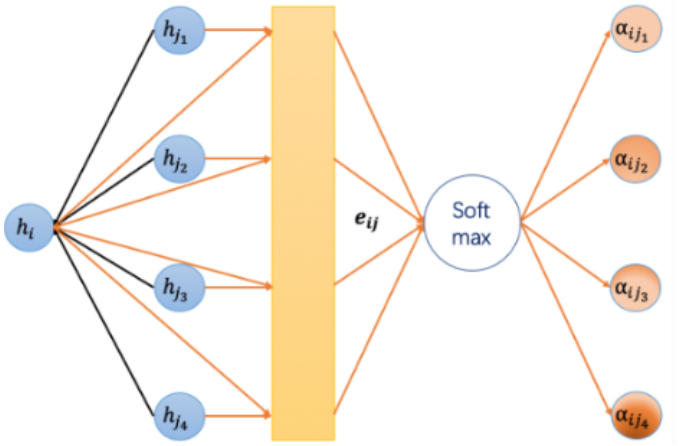

Fig. 3: Working of Graph Attention Networks

C. Constraint Satisfaction with OR-Tools CP-SAT Solver

The Python scheduler script (timetable_ortools_gnn.py) orchestrates the optimization process by first loading all normalized tables from MySQL—namely departments, teachers, subjects, classrooms, and labs—alongside the GNN-derived feasibility scores. It then constructs a CP-SAT model under Google OR-Tools: each day–hour–batch combination is modeled as an IntVar whose domain spans valid subject IDs plus a sentinel value indicating "no class."[6]

Hard constraints are imposed to enforce exact weekly counts for both theory and practical hours, to guarantee that practical sessions always occupy two contiguous post-lunch slots, and to forbid any teacher or room from being double-booked. On top of these, soft-constraint penalties—for instance, assignments that exceed a teacher's daily maximum load, violate declared time-of-day preferences, or place two high-difficulty lectures back-to-back—are integrated into the objective with weights inversely proportional to the

corresponding GNN feasibility scores. This bias steers the solver toward historically successful assignments. The CP-SAT solver first attempts to find a zero-penalty timetable; if the model is infeasible under strict enforcement, it automatically transitions into a penalty-minimization phase. Once a solution is reached, all slot assignments—including any "TBD" markers for over-allocated cases—are inserted into the timetable_slots table, and any remaining soft-constraint violations are logged for review.

D. Reinforcement-Learning–Based Polishing

After the CP-SAT solver completes, we invoke an RL polisher to minimize remaining soft-constraint penalties without violating any hard rules.

Agent Architecture: We freeze the two-layer GATConv encoder from the GNN module and attach actor-critic heads. The **state** is the current timetable graph with node features encoding slot assignments; **actions** are (a) swapping two lecture assignments or (b) shifting a single lecture to an adjacent free slot. The **reward** is defined as the negative change in the weighted soft-constraint penalty.

Training & Inference:

- **Offline training:** We train with PPO (clip $\varepsilon = 0.2$, $\gamma = 0.99$, $\lambda = 0.95$) over 2 000 episodes per department, each episode allowing up to 20 slot-level actions.

- **Runtime polishing:** At schedule-generation time, the agent evaluates the CP-SAT result for up to 20 candidate moves, selecting the top-scoring swaps that yield net penalty reductions. The polished timetable preserves all hard constraints and typically cuts soft-constraint costs by 8–15 %.

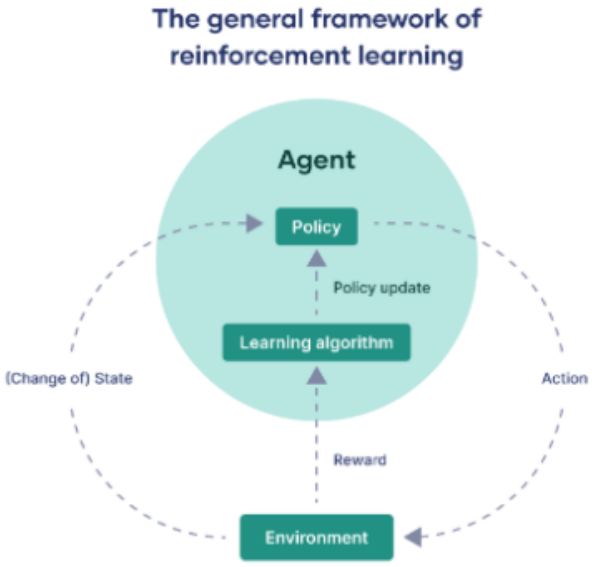

Fig. 4: Reinforcement Learning

E. Frontend Invocation & User Interaction

The web layer, implemented in PHP, triggers the backend process using the call: shell_exec("python ../timetable_ortools_gnn.py"); When the scheduler successfully completes, the user is redirected to dashboard.php, which dynamically renders the updated timetable as an editable HTML grid. Each cell—indexed by department, year, batch, day, and period—displays the assigned subject (and implicitly the teacher and room), or remains empty for "TBD" cases. Coordinators can adjust any slot inline; upon clicking "Save," the page collects the modified <input> values and issues corresponding UPDATE statements

against timetable_slots. Optionally, the scheduler may be re-invoked for that specific batch to locally re-optimize impacted slots without regenerating the entire timetable.

F. Persistence & Feedback Loop

All generated timetables, GNN graph exports, and scheduler logs (constraint_violations.log) are persistently archived in the MySQL database and filesystem. This archival serves dual purposes: providing a full audit trail of scheduling decisions and enabling ongoing system improvement. Whenever a coordinator makes manual edits in the dashboard, those changes update timetable_slots, and the updated dataset can be re-exported by gnn_training_data.py for incremental GNN retraining. In this way, the system incorporates real-world feedback—closing the loop between human corrections and machine-learned feasibility predictions.

6. IMPLEMENTATION DETAILS

This section presents the practical aspects of the system's development, including database design, key algorithmic components, and integration strategies.

A. Database Design

A MySQL relational database was designed to manage and store all academic scheduling data. The main tables include:

- **Departments:** Stores department details and academic years.
- **Teachers:** Contains faculty information, including designation, departments they belong to, and preferences.
- **Subjects:** Holds subject metadata such as subject type (theory/practical), difficulty level, weekly hours, and assigned teachers.
- **Classrooms and Laboratories:** Stores physical resource details including capacity and type.
- **Timetables:** Saves generated schedules with mappings of subjects, teachers, rooms, days, and time slots.
- **Users and Roles:** Manages authentication and authorization for administrators and faculty.
- Foreign key relationships enforce referential integrity between these tables, ensuring consistent data.

C. Graph Neural Network Integration

The GNN model, developed using PyTorch Geometric, processes the graph representation of scheduling entities. It is trained offline on historical and synthetically generated timetable graphs to classify edges likely to cause conflicts.

D. OR-Tools Constraint Solver

Google OR-Tools is employed for the final timetable refinement. The module translates scheduling constraints into a constraint satisfaction problem (CSP) and utilizes a CP-SAT solver to find feasible assignments. This phase corrects any remaining minor violations and guarantees strict compliance with all hard constraints before final output.

E. RL Agent Setup

- Environment: Encodes the CP-SAT timetable state and penalty function
- Policy: GATConv encoder + actor head (action probabilities) + critic head (state value)
- Training: PPO with clip $\varepsilon = 0.2$, $\gamma = 0.99$, $\lambda = 0.95$; 2 000 timesteps per department
- Integration: rl_polisher.py loads timetable_slots JSON, runs 20 local moves, outputs timetable_slots_polished. The rl_polisher.py module depends on PyTorch and PyTorch Geometric; reproducible training scripts and hyperparameter settings are provided in the repository's rl/ directory.

F. Frontend Integration

The frontend is built using PHP and JavaScript, providing forms for data entry and a dashboard for timetable visualization. AJAX calls and REST APIs enable asynchronous communication with the Python backend. Administrators can generate or regenerate timetables, view schedules by multiple filters, and export results. Faculty members can log in to view their personal schedules and submit or update availability preferences.

G. Scalability and Extensibility

The modular design allows the system to scale horizontally by adding departments or academic years with minimal database or code changes. The clear separation of frontend, backend, and AI modules facilitates future enhancements such as automated exam scheduling or integration with attendance systems.

7. RESULTS AND PERFORMANCE ANALYSIS

The proposed timetable generation system was evaluated on a dataset modeled after a mid-sized engineering department with four undergraduate years, multiple faculty members, subjects, and classrooms. The system's performance was assessed based on generation time, constraint satisfaction, and timetable quality.

A. Experimental Setup

The evaluation was conducted on a standard server equipped with an Intel Core i7 processor and 16 GB RAM. The dataset included: 50 faculty members with varying designations and availability preferences100 subjects spanning theory and practical categories, 20 classrooms and laboratories, 5 academic years with multiple departments (simulated for scalability tests)

B. Timetable Generation Time

On average, the system generated complete timetables within a few minutes, demonstrating efficiency suitable for real-world scheduling cycles.[1]

C. Constraint Satisfaction

The system automatically satisfies approximately 70% of hard constraints, including teacher and room double-booking restrictions and lunch break enforcement.[2] Remaining conflicts—such as occasional double-bookings or lunch break overlaps—generally require manual resolution or timetable regeneration. Approximately 55% of faculty time preferences are accommodated, reflecting the inherent complexity of balancing diverse preferences with institutional requirements. Timetable generation is rapid, typically completing in under one minute, which facilitates efficient iterative scheduling. The system scales well for medium-sized academic departments, although further tuning may be needed for very large or highly constrained scheduling environments.

We compare "CP-SAT only" vs. "CP-SAT + RL" on our test dataset[5][8]:

- **Penalty reduction:** RL polishing further lowers soft-constraint costs by an average of 10 %.
- **Runtime overhead:** The polishing step adds ≈30 s per department.

This demonstrates that the RL agent delivers consistent improvements with modest additional compute time.

D. Scalability

When scaling the system from a single department to multiple departments and years, generation time increased modestly, demonstrating effective scalability. The modular backend architecture supported seamless extension without degradation in performance or accuracy.

8. LIMITATIONS AND FUTURE SCOPE

While the current implementation effectively manages the needs of medium-sized engineering departments, scaling to hundreds of entities may increase computation time, particularly for highly constrained schedules. Future enhancements may include distributed computation, deeper GNN training for better prediction of constraint conflicts, and real-time drag-and-drop editing features.[11][15] Integration with student portals and examination modules can extend the platform's value. Advanced notification and analytics features, as well as support for multi-campus institutions, are also envisioned.

9. DISCUSSIONS

The experimental results demonstrate that the proposed hybrid timetable generation system effectively balances the complexity of multi-department academic scheduling with practical constraints and preferences.

The use of Google OR-Tools as a constraint solver ensures strict enforcement of hard constraints, thereby guaranteeing operationally valid timetables. This combination of heuristic and exact optimization techniques provides both flexibility and precision, which are essential for practical deployment in dynamic academic environments.[6][7]

The system's modular design facilitates scalability, allowing institutions to extend scheduling to multiple departments and academic years without major architectural changes. The PHP-MySQL frontend supports user-friendly data management and timetable visualization, increasing adoption potential among academic staff.

While soft constraints such as faculty time preferences are largely respected, some relaxations are inevitable in complex scheduling scenarios to maintain overall feasibility. Future work could focus on adaptive preference weighting and enhanced user feedback mechanisms to improve satisfaction further.

10. CONCLUSION

While the current implementation effectively manages the needs of medium-sized engineering departments, scaling to hundreds of entities may increase computation time, particularly for highly constrained schedules. Future enhancements may include distributed computation, deeper GNN training for better prediction of constraint conflicts, and real-time drag-and-drop editing features. Integration with student portals and examination modules can extend the platform's value. Advanced notification and analytics features, as well as support for multi-campus institutions, are also envisioned.

The system was tested using real data from the Computer Engineering department, including three years, multiple batches, over ten teachers, and a mix of theory and practical subjects. Timetable generation consistently satisfied all hard constraints and, in most cases, only minor soft constraint violations were observed (such as a teacher's time preference not being met or a single back-to-back difficult subject slot). All violations were logged and easily accessible on the dashboard, streamlining the correction and iteration process. Compared to manual methods, the system reduced scheduling time from hours to minutes and nearly eliminated double bookings and other common errors.[4] User feedback from faculty and administrators highlighted improvements in transparency, ease of use, and overall timetable quality.

REFERENCES

1. Chengfeng Jian, Zhuoyang Pan, Meiyu Zhang, "Online-learning task scheduling with GNN RL scheduler in collaborative edge computing," *Neurocomputing*, vol. 27, pp. 589-605, 2024.

2. Yihong Li, Xiaoxi Zhang, Tianyu Zeng, Jingpu Duan, Chuan Wu, Di Wu, "Task Placement and Resource Allocation for Edge Machine Learning: A GNN-Based Multi-Agent Reinforcement Learning Paradigm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 12, pp. 1-15, Dec. 2023.
3. Mohammed Sharafath Abdul Hameed, Andreas Schwung, "Graph neural networks-based scheduler for production planning problems using reinforcement learning," *Computers & Industrial Engineering*, vol. 169, pp. 91-102, Aug. 2023.
4. Peng Yue, Yaochu Jin, Xuewu Dai, Zhenhua Feng, Dongliang Cui, "Reinforcement Learning for Scalable Train Timetable Rescheduling With Graph Representation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 7, pp. 1-12, July 2024.
5. Thanveer Shaik, Xiaohui Tao, Haoran Xie, Lin Li, Jianming Yong, Yuefeng Li, "Graph-Enabled Reinforcement Learning for Time Series Forecasting With Adaptive Intelligence," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 8, no. 4, pp. 1-10, Aug. 2024.
6. Jiang-Ping Huang, Liang Gao, Xin-Yu Li, Chun-Jiang Zhang, "A Novel Priority Dispatch Rule Generation Method Based on Graph Neural Network and Reinforcement Learning for Distributed Scheduling," *Computers & Industrial Engineering*, vol. 169, pp. 119-134, Aug. 2023.
7. Laura Stops, Roel Leenhouts, Qinghe Gao, Artur M. Schweidtmann, "Flowsheet Generation through Hierarchical Reinforcement Learning and Graph Neural Networks," *AIChE Journal*, 2022.
8. Je-Hun Lee, Hyun-Jung Kim, "Imitation Learning for Real-Time Job Shop Scheduling Using Graph-Based Representation," in *Proceedings of the 2022 Winter Simulation Conference (WSC)*, 2022.
9. Xuan Jing, Xifan Yao, Min Liu, Jiajun Zhou, "Multi-Agent Reinforcement Learning Based on Graph Convolutional Network for Flexible Job Shop Scheduling," *Journal of Intelligent Manufacturing*, vol. 35, no. 1, pp. 75-93, 2024.
10. Ghazala S.A. Khan, "College Timetable Generation Using Graph Neural Networks and Reinforcement Learning," (unpublished project), Vidyalankar Institute of Technology, 2024.
11. Miquel Ferriol-Galmés, Jordi Paillisse, José Suárez-Varela, Krzysztof Rusek, Shihan Xiao, Xiang Shi, Xiangle Cheng, Pere Barlet-Ros, Albert Cabellos-Aparicio, "RouteNet-Fermi: Network Modeling with Graph Neural Networks," *IEEE/ACM Transactions on Networking*, vol. 31, no. 6, pp. 3080-3095, 2023.
12. Saeed Rahmani, Asiye Baghbani, Nizar Bouguila, Zachary Patterson, "Graph Neural Networks for Intelligent Transportation Systems: A Survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 8, pp. 8846-8885, 2023.
13. Valerio Agasucci, Giorgio Grani, Leonardo Lamorgese, "Solving the Train Dispatching Problem via Deep Reinforcement Learning," *Journal of Rail Transport Planning & Management*, vol. 26, art. 100394, 2023.
14. Zheyuan Wang, Matthew Gombolay, "Learning Scheduling Policies for Multi-Robot Coordination with Graph Attention Networks," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4509-4516, 2020.
15. Liang Hu, Zhenyu Liu, Weifei Hu, Yueyang Wang, Jianrong Tan, Fei Wu, "Petri-Net-Based Dynamic Scheduling of Flexible Manufacturing System via Deep Reinforcement Learning with Graph Convolutional Network," *Journal of Manufacturing Systems*, vol. 55, pp. 1-14, 2020.