MJI
Media Jurnal Informatika
Vol. 17, No. 2, December 2025
p-issn : 2088-2114, e-issn : 2477-2542

Course Schedule Optimization Using a Java-Based Ant Colony Optimization

Theo Buana Pongsumarre [a,1,*], Wahyuni [b,2], Muhammad Fahmi [b,3]

[a] Informatics Enginer Study Program, *STMIK Widya Cipta Dharma*, Indonesia
[b] Informatics Enginer Study Program, *STMIK Widya Cipta Dharma*, Indonesia
[c] Information System Study Program, *STMIK Widya Cipta Dharma*, Indonesia
[1] 2243906@wicida.ac.id*; [2] wahyuni@wicida.ac.id; [3] mfahmi@wicida.ac.id
* corresponding author

ARTICLE INFO

Article history
Received 2025/11/25
Revised 2025/12/06
Accepted 2025/12/20

Keywords
ACO algorithm
course scheduling
Java programming
metaheuristic
optimization

ABSTRACT

Course timetabling in higher education is a complex combinatorial problem due to constraints related to lecturer availability, limited classroom resources, and fixed weekly time-slot structures. As the number of courses and class sections increases, manual scheduling becomes increasingly inefficient and prone to conflicts, particularly room clashes and overlapping lecturer assignments. This study develops and evaluates an automatic course scheduling system based on the Ant Colony Optimization (ACO) algorithm and implements it as a Java-based desktop application to generate feasible timetables under real institutional conditions. An experimental computational approach is employed, in which artificial ants construct candidate schedules through probabilistic selection influenced by pheromone trails and heuristic information. Timetable quality is evaluated using a weighted cost function that prioritizes hard-constraint satisfaction, such as preventing lecturer and room clashes, while also incorporating soft-constraint penalties related to lecturer forbidden timeslots and schedule distribution balance. The system is tested using real academic data from an undergraduate study program, including courses, lecturers, classrooms, and predefined weekly timeslots. Experimental results show that the proposed system consistently generates conflict-free timetables, achieving a conflict value of zero across all repeated runs under the selected parameter configuration. Beyond feasibility, the optimization process continues to refine timetable quality by reducing soft-constraint penalties, as indicated by the convergence behavior observed across repeated executions. This repeated-run evaluation provides insight into the stochastic optimization characteristics of the ACO-based approach under fixed parameter settings. These findings indicate that the Java-based ACO approach effectively supports automated university course scheduling and provides a practical solution for producing feasible and well-structured timetables.

1. Introduction

University course timetabling is an academic planning activity conducted before each semester to allocate courses, lecturers, rooms, and timeslots in a coordinated manner. A timetable is considered feasible when fundamental constraints are satisfied so that teaching and learning can proceed without clashes. In practice, feasibility is primarily determined by mandatory constraints such as ensuring that a lecturer is not assigned to two classes at the same time and that a room is not used simultaneously, while timetable quality can be improved by considering preferences and balancing class distribution across days and sessions. The literature also confirms that timetabling remains a critical managerial issue in higher education because it affects service quality, workload, and operational stability [1], [2], [3]. Practical constraints include limited room availability, a relatively fixed weekly timeslot structure, and diverse lecturer teaching-availability patterns. Common conflicts observed in practice include room clashes and overlapping lecturer assignments, which

makes manual scheduling require repeated cross-checking. As data volume grows, manual procedures tend to become more time-consuming and more prone to errors. In addition, administrative changes—such as opening parallel class sections, replacing lecturers, or introducing additional time restrictions—often trigger timetable reconstruction, increase workload, and may disrupt teaching if conflicts remain in the published schedule [4], [5]**.** Operationally, manual timetabling generally involves iterative checking and revision to avoid conflicts; this complexity tends to increase when lecturer preferences are considered, as discussed in higher-education timetabling studies [4], [5]**.** Such preferences narrow the set of feasible timeslot combinations and often require adjustments to maintain overall timetable feasibility. From an optimization perspective, course timetabling can be formulated as assigning course–lecturer–room–timeslot combinations within a very large search space and is commonly classified as an NP-hard problem. Consequently, exact approaches often become less practical as problem size increases or constraints become more diverse, motivating the use of approximation strategies and metaheuristics. Prior studies show that metaheuristics can explore large solution spaces efficiently and iteratively improve candidate timetables within reasonable computation time, particularly when the problem contains many interacting constraints [1], [3], [6], [7]**.** Among metaheuristic families, Ant Colony Optimization (ACO) is frequently applied because it constructs solutions incrementally and "learns" from previous search experience through pheromone reinforcement. Several studies report that ACO variants can produce feasible timetables in highly constrained cases, and intelligent optimization approaches are often associated with conflict reduction and improved scheduling efficiency [5], [8], [9]**.** Meanwhile, other metaheuristics such as Genetic Algorithms, Simulated Annealing, and Particle Swarm Optimization have also been widely applied to balance exploration of new solutions and exploitation of promising regions in the search space [10], [11], [12]. Nevertheless, two practical gaps remain notable. First, many studies emphasize evaluation on simulated data or standardized benchmark datasets, which may not fully represent the policy rules and data characteristics of a specific study program; as a result, benchmark performance does not always translate directly to operational settings [3], [6]. Second, the development of desktop tools that are ready for use by academic administrators, as well as empirical analysis of how core ACO parameters—particularly the number of ants and iteration limits—affect performance on real institutional data, remains relatively limited [13], [14].Closing these gaps is important so that algorithmic contributions extend beyond conceptual advances and support adoption in program-level scheduling contexts. Therefore, this study develops an automatic timetabling system based on Ant Colony Optimization (ACO), implemented as a Java desktop application, and evaluated using real academic data from the Informatics Engineering Study Program of STMIK Widya Cipta Dharma. The system models feasibility primarily through hard constraints, such as preventing lecturer and room clashes, while soft constraints are incorporated to support schedule quality refinement, including lecturer forbidden timeslots and balanced distribution of classes across days and timeslots. The evaluation is conducted through repeated runs on the same dataset to assess the system's ability to consistently generate conflict-free timetables and to examine how the optimization process contributes to timetable refinement under realistic institutional conditions. The contributions of this study include applying ACO to program-level institutional data, embedding the optimization process into a practical Java desktop application usable by non-technical users, and providing empirical evidence of how parameter configuration influences convergence toward feasible and well-structured schedules [15].

2. Method

This study uses an experimental computational approach to develop and evaluate an automatic course scheduling system based on Ant Colony Optimization (ACO), implemented as a Java desktop application. The research data were obtained from the academic records of the Informatics Engineering Study Program at STMIK Widya Cipta Dharma for the 2024 academic year (odd and even semesters), including class entries, assigned lecturers, the list of available rooms, enrolment numbers, and the weekly timeslot structure used by the program. The scheduling process is modeled as assigning course–lecturer–room–timeslot combinations in a large search space; therefore, solution evaluation is performed using a cost function that combines hard-constraint and soft-constraint violations. Hard constraints are prioritized to ensure timetable feasibility (such as preventing lecturer clashes and room clashes), while soft constraints are used to improve timetable quality through lecturer forbidden timeslot preferences and schedule distribution balancing. Procedurally, the system is developed using the Waterfall model, progressing through requirement analysis, system design, implementation, testing, and deployment. During testing, the application is executed repeatedly on the same dataset to examine output consistency and to evaluate the ability of ACO to produce feasible timetables based on conflict/cost indicators computed from the defined constraints. Details of the research approach, object and scope, data sources, tools, development stages, and analysis techniques are described in Subsections 2.1 to 2.6.

2.1 Type and Approach of Research

This research is an experimental computational study. ACO is applied as the optimization technique to construct candidate timetables iteratively, and system performance is examined through repeated runs on the same institutional dataset. The evaluation follows common practice in university timetabling studies that assess feasibility and quality indicators under configurable parameter settings [5], [6]. Repeated execution is used to observe convergence behavior and solution stability, evaluated through the number of iterations required to reach feasible schedules and the final cost values obtained across runs.

2.2 Object and Scope of Research

The object of the study is an ACO-based course scheduling system implemented as a Java desktop application for program-level use. The scope covers timetable construction for the Informatics Engineering Study Program of STMIK Widya Cipta Dharma by assigning each class entry to one room and one weekly timeslot within the program's scheduling structure. The scheduling horizon is defined as five weekdays (Monday–Friday) with three timeslots per day.

2.3 Data Collection Techniques

The study uses institutional academic records from the Informatics Engineering Study Program of STMIK Widya Cipta Dharma for the 2024 academic year (odd and even semesters). The dataset comprises class entries (sections), lecturer assignments, available rooms, student enrolment figures, and a fixed weekly timeslot structure. Using official institutional data (rather than simulated or benchmark instances) ensures that the developed system reflects real operational conditions. A summary of the dataset is provided in Table 1.

Table 1. Research dataset overview for the 2024 (Odd–Even) course scheduling case.

Item	Value
Dataset label	Courses in the 2024 Academic Year (Odd and Even Semesters)
Class entries (sections)	37 (36 unique courses)
Resources	21 lecturers; 12 rooms
Time structure	5 days (Mon–Fri); 3 timeslots/day; 15 timeslots/week
Timeslot schedule	08:00–09:40; 10:00–11:40; 13:00–14:40 (100 minutes)
Academic/load info	2 SKS per class; enrolment 41–73 (avg 57.43); room capacity data not available

2.4 Tools and Materials Used

The scheduling system is implemented as a Java desktop application with a graphical user interface using Java Swing. The application integrates data input, ACO execution, and timetable visualization within one system to support direct use by academic administrators. Inputs (courses, lecturers, rooms, enrolment, and timeslots) are stored in internal data structures and processed by the ACO procedure to generate a feasible timetable.

2.5 Research Procedures or Stages

Constraint modeling and penalty weights.

Schedule feasibility and quality are evaluated using hard constraints and soft constraints embedded in the scheduling cost function. Hard constraints represent mandatory rules to prevent fundamental conflicts, namely room clashes and lecturer clashes. Soft constraints represent preference-based criteria, including lecturer forbidden timeslots and balancing of schedule distribution across days and sessions. The separation between hard and soft constraints follows the common formulation adopted in university course timetabling problems [1], [6]. Penalty weights are defined prior to execution based on operational scheduling considerations and the characteristics of the institutional data used. Preliminary testing is conducted to ensure that the selected weights allow the system to consistently achieve feasible schedules while preserving clear differentiation between hard-constraint and soft-constraint violations. Once defined, penalty weights are kept constant throughout all experiments to ensure evaluation consistency and maintain interpretability of the optimization results. The constraints and penalty weights implemented in the system are presented in Table 2.

Table 2. Hard and Soft Constraints Used in the Scheduling Evaluation.

Type	Constraint	Violation rule (based on implementation)	Penalty(added to cost)
Hard	Room clash	Two different courses are assigned to the same timeslot and same room	+1.00 per clash
Hard	Lecturer clash	Two different courses taught by the same lecturer are assigned to the same timeslot	+1.00 per clash
Soft	Lecturer forbidden timeslot	A course is placed in a timeslot that is forbidden for its lecturer	+0.30 per violation
Soft	Schedule balancing (day & timeslot distribution)	Penalizes overloaded timeslots, overloaded days, and empty timeslots	+0.15×(slot overload) +0.20×(day overload) +0.03×

Ant Colony Optimization (ACO) model and transition rule.

Ant Colony Optimization (ACO) is a population-based metaheuristic inspired by the collective behavior of ants in discovering efficient paths and has been widely applied to combinatorial optimization problems, including university course timetabling [6], [8], [16]. In this study, the timetabling problem is represented as a graph in which nodes correspond to feasible course–room–timeslot assignments. Ants incrementally construct complete timetables by selecting the next assignment according to a probabilistic transition rule.

$$P_{ij} = \frac{(\tau_{ij}^{\alpha})(\eta_{ij}^{\beta})}{\sum_{k}(\tau_{ik}^{\alpha})(\eta_{ik}^{\beta})} \quad (1)$$

where p_{ij} is the probability of moving from node i to node j, τ_{ij} denotes the pheromone value, η_{ij} represents the heuristic desirability, and α and β control the relative influence of pheromone and heuristic nformation, respectively [2], [16]. After all ants generate candidate timetables, each solution is evaluated and pheromone values are updated to reinforce higher-quality assignments [1], [6].The parameters $\alpha = 1$ and $\beta = 3$ are selected to emphasize heuristic information during solution construction while preserving pheromone influence for long-term learning. This configuration is commonly adopted in ACO-based timetabling and scheduling studies to balance exploration and exploitation. Alternative parameter settings were initially explored, but $\alpha = 1$ and $\beta = 3$ consistently produced more stable convergence and feasible solutions across multiple runs, making this configuration suitable for the problem characteristics considered in this study.

Schedule evaluation function.

Candidate timetables are evaluated using a weighted cost function that aggregates hard and soft constraint violations:

$$F(x) = w_h \cdot Viol_{hard}(x) + w_s \cdot Viol_{soft}(x) \quad (2)$$

where $Viol_{hard}(x)$ counts mandatory constraint violations and $Viol_{soft}(x)$ counts preference-based violations. Larger weights are assigned to hard violations so that feasibility is prioritized before quality refinement, consistent with prior timetabling models [5], [6], [17]

Pheromone update and reinforcement.

At the end of each iteration, pheromone values are updated through evaporation and deposition:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij} \quad (3)$$

where ρ is the evaporation rate that helps prevent premature convergence[2], [16], [18].Pheromone reinforcement is defined as:

$$\Delta\tau_{ij} = \frac{Q}{cost} \quad (4)$$

where Q is a pheromone constant and $cost$ is the timetable cost obtained from Eq. (2). Lower-cost solutions receive larger reinforcement values[18], [19].

Parameter settings.
The number of ants and maximum iterations are configurable via the application interface. The chosen α, β, and ρ values are adopted based on commonly used configurations in related timetabling studies and preliminary experimentation to ensure stable convergence behavior. In the implementation, pheromone influence is set to α = 1, heuristic influence to β = 3, and evaporation rate to ρ = 0.3, while other parameters such as the number of ants and iteration limits are defined by the user according to problem size. The ACO parameter settings used in this study are summarized in Table 3.

Table 3. ACO Algorithm Parameter Values.

Parameter	Symbol	Value	Short Description
Number of Ant	m	10	Number of ants constructed in each iteration
Max Iteration	$maxIter$	50	Upper limit of optimization iterations per run
Pheromone Influence	α	1	Pheromone weight
Heuristic Influence	β	3	Heuristic weight
Evaporation Rate	ρ	0.3	Pheromone loss rate
Reinforcement Constant	Q	500	Addition of best solution pheromones
Initial Pheromone	τ_0	0.1	Initial pheromone value

System development model (Waterfall) and implementation.
The system is developed using the Waterfall model, consisting of requirement analysis, design, implementation, testing, and deployment. Requirement analysis defines scheduling data, constraints, and parameter inputs. The design stage specifies representations for schedules, pheromone trails, and heuristic information. Implementation translates the ACO equations and scheduling logic into a Java application. The optimization workflow is organized into modular Java classes responsible for data representation, schedule construction, evaluation, and pheromone updating. Testing focuses on feasibility verification and repeated-run evaluation using the same dataset. Deployment ensures the application can be used in real academic scheduling work. The development flow is illustrated in Figure 1 [20].

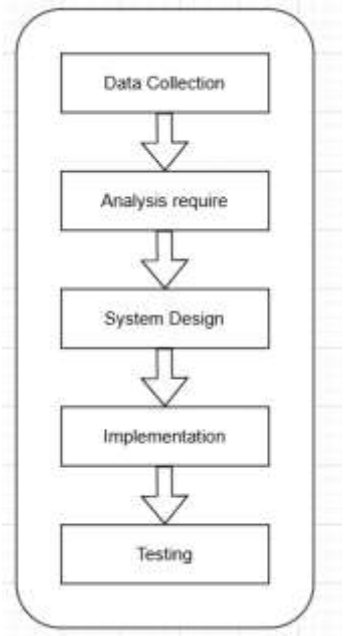

Figure 1. Waterfall-based stages of system development.

2.6 Data Analysis Techniques

Data analysis is conducted by evaluating the performance of the generated timetables using the cost function in Eq. (2), which aggregates hard- and soft-constraint violations. Feasibility is primarily determined by the absence of hard-constraint conflicts, namely room clashes and lecturer clashes, while soft-constraint penalties are used to reflect preference-oriented schedule quality. The system is executed repeatedly on the same dataset to examine consistency and convergence behavior, and the results are summarized based on feasibility status and the final conflict/cost value obtained in each run. This analysis supports observation of the stochastic behavior of the algorithm without claiming comparative optimality beyond the defined constraint model.

3. Results and Discussion

This section presents the results and discussion of the proposed Ant Colony Optimization (ACO)–based course scheduling system. The evaluation is conducted using real institutional academic data through repeated executions of the algorithm in order to observe solution consistency and convergence behavior. The analysis focuses on two main aspects: timetable feasibility, defined by the satisfaction of hard constraints without conflicts, and optimization behavior, reflected by changes in the scheduling cost over successive iterations. Across repeated runs, the first conflict-free solution was generally obtained within the early-to-middle iterations, with only minor variation between runs, reflecting the stochastic search behavior of the ACO algorithm under fixed parameter settings. To support this analysis, the optimization results are presented using convergence visualizations. The first curve illustrates the average cost trend across iterations, representing the optimization trajectory of the algorithm, while the second curve shows the feasibility rate per iteration, indicating when conflict-free schedules are achieved during the search process

3.1 Cost Convergence Behavior

The average cost convergence behavior of the ACO-based scheduling system across multiple runs shows a gradual reduction in cost values as the number of iterations increases. This trend indicates that the algorithm progressively improves timetable quality after the initial construction phase, rather than stopping once a feasible solution is found. Cost fluctuations observed in early iterations reflect the stochastic nature of ACO and its exploration of alternative scheduling combinations, while the stabilization of the cost trend in later iterations suggests increasing influence of pheromone reinforcement toward lower-cost assignments [5], [6]. The behavior of the cost convergence across different runs also suggests that ACO efficiently refines solutions by iterating toward a global optimum, reducing the impact of early randomness [17]. assignments Figure 2.

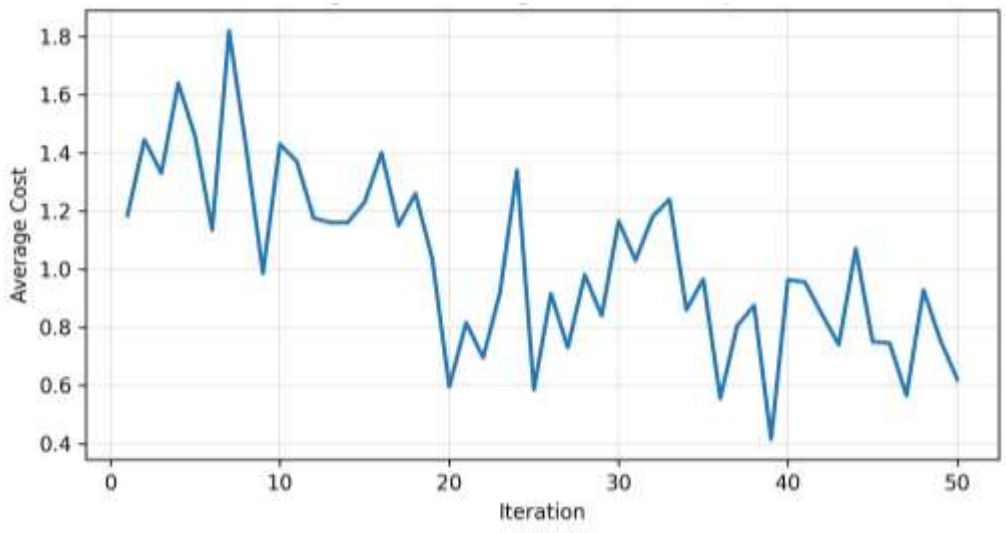

Figure 2. Average cost convergence across iterations over multiple runs.

3.2 Feasibility per Iteration

The feasibility rate per iteration aggregated over multiple runs increases as the optimization process progresses, indicating that conflict-free schedules are achieved more consistently after a certain number of iterations. In the early stages, the feasibility rate remains low due to the difficulty of satisfying all hard constraints during initial solution construction. As iterations advance, the feasibility rate stabilizes, demonstrating the algorithm's ability to reliably converge toward feasible schedules under stochastic search

behavior and to prioritize hard-constraint satisfaction before further quality refinement [1], [6], [17]. assignments.Figure 3.

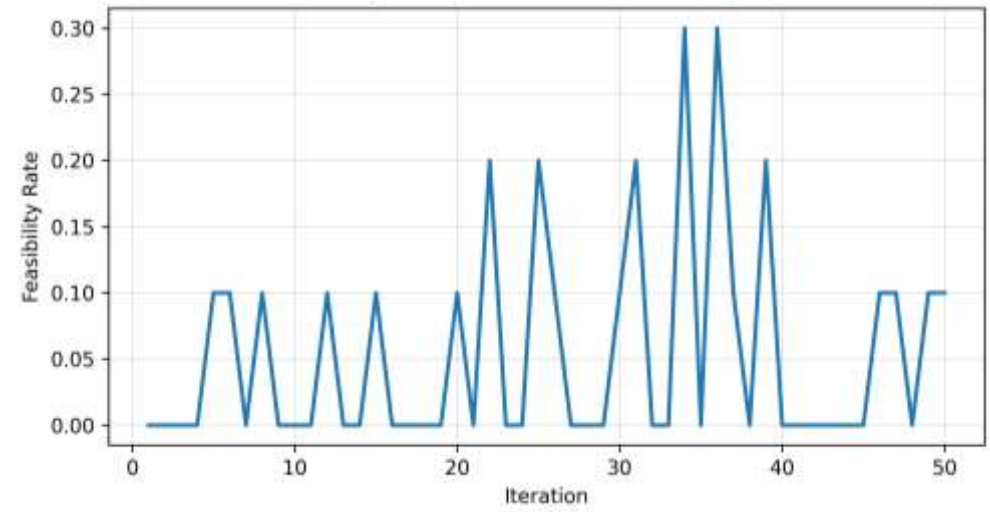

Figure 3. Feasibility rate per iteration aggregated over multiple runs.

3.3 Presentation of Research Results

The developed system is implemented as a Java desktop application with a graphical user interface that supports academic data management, ACO parameter configuration, optimization execution, and result presentation. Through the interface, users can manage course, lecturer, room, and timeslot data, specify ACO parameters (including ant count and iteration limits), and execute the scheduling procedure in a single workflow. The system displays the generated timetable along with detected conflicts to support feasibility validation, as illustrated in Figure 4.

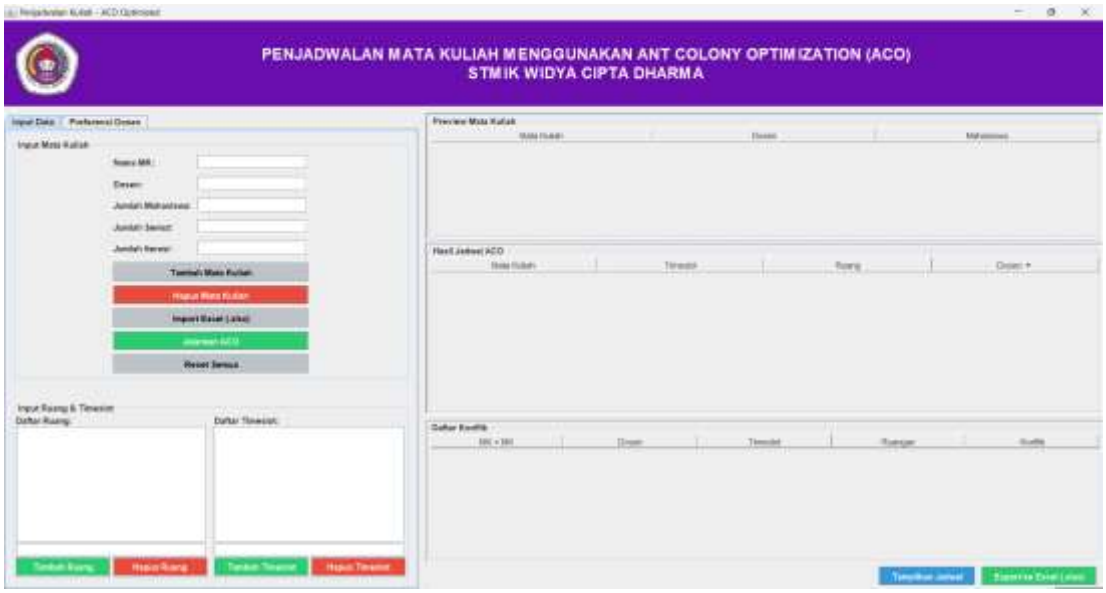

Figure 4. Primary interface of the ACO scheduling application.

To support operational use, the application provides an integrated screen for entering scheduling data and configuring key optimization settings. The output view presents the optimized timetable together with conflict information so that users can review feasibility and perform administrative follow-up actions when necessary. An example of this integrated interface is shown in Figure 5.

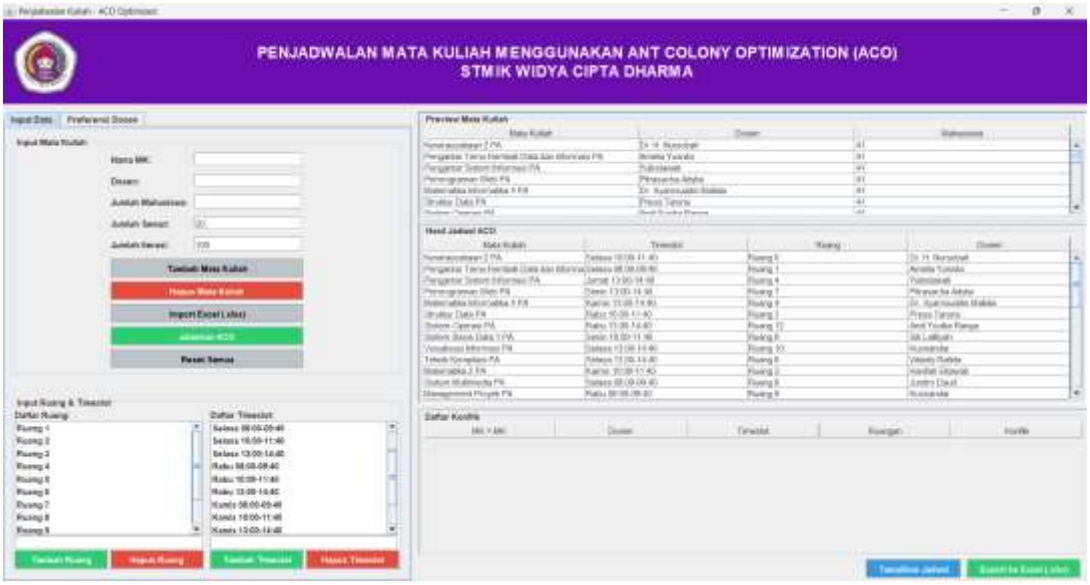

Figure 5. Integrated GUI for the ACO-based scheduling system.

To incorporate lecturer availability considerations, the system includes a dedicated interface for lecturer time preferences. Users can specify preferred and restricted timeslots for each lecturer, and these inputs are incorporated into the scheduling model as soft constraints. This design allows the system to generate schedules that better reflect actual teaching conditions while maintaining feasibility requirements. The lecturer preference input screen is presented in Figure 6.

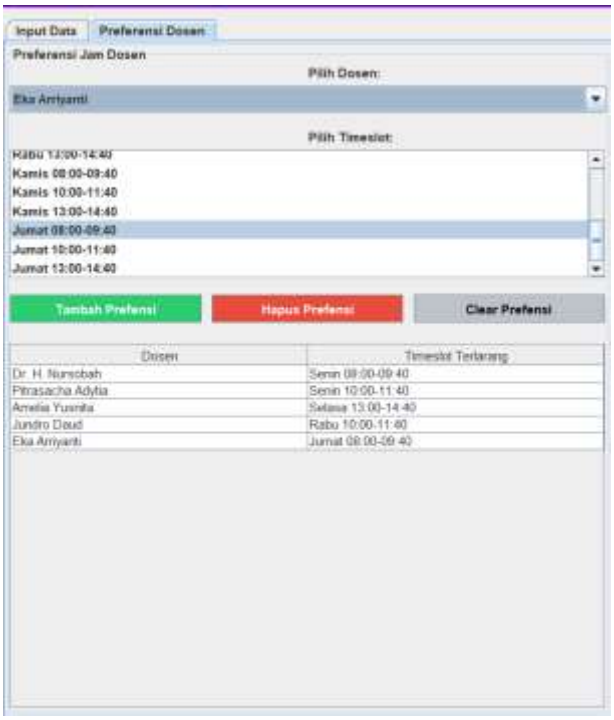

Figure 6. Interface for setting lecturer time preferences.

After optimization, the timetable output is displayed in a day-by-timeslot matrix format. This representation facilitates verification that lecturers and rooms are not assigned to multiple classes simultaneously and provides a concise overview of course allocations across the weekly structure. An example timetable generated from the institutional dataset is shown in Figure 7.

Figure 7. Timetable matrix produced by the system.

3.4 Analysis of Findings

The experimental results indicate that Ant Colony Optimization (ACO) can address the timetabling requirements of the Informatics Engineering Study Program of STMIK Widya Cipta Dharma when applied to real academic data. The observed reduction of conflicts across iterations suggests that pheromone reinforcement progressively biases the search toward more feasible assignment combinations, consistent with prior work showing that ACO is effective for highly constrained timetabling problems [1], [2], [3]. The results further suggest that system performance is sensitive to core parameter settings, particularly the number of ants and the maximum iterations. Increasing these values generally improves exploration and provides more opportunities for pheromone-based refinement, which can increase the consistency of achieving conflict-free solutions. However, excessively large settings may increase computation time without proportional improvements, implying that practical deployment benefits from balanced parameter selection [17].

Repeated-run outcomes also show that the configured setting is sufficient to reliably reach feasibility, while lecturer preferences primarily influence convergence speed rather than the final feasibility status. Because ACO is stochastic, the iteration at which the first feasible solution appears can vary even when the dataset and parameters are fixed. When lecturer forbidden-timeslot preferences are enabled, the search space becomes more constrained through additional penalties, which can shift the exploration–exploitation balance and delay feasibility in some repetitions [17], [18].

This behavior is reflected in the repeated-run comparison that summarizes final feasibility, first feasible iteration, and runtime under both preference settings. Using institutional records increases the practical relevance of the study because the generated timetables reflect the program's real constraints, including lecturer availability patterns, room lists, and timeslot structures. Embedding the optimization procedure into a Java desktop application also improves usability for academic administrators and supports direct operational use, aligning with applied scheduling studies that emphasize deployable tools rather than purely benchmark evaluations [16], [18]. Despite these positive outcomes, performance may vary with instance size and parameter configurations. Future work can investigate adaptive parameter control and scalability improvements. Another promising direction is hybridization, in which ACO is combined with local search or other optimization mechanisms to strengthen robustness and improve solution quality. For example, hybrid scheduling approaches that integrate ACO with tabu search have been reported to improve search effectiveness in minimizing tardiness and refining solution quality, suggesting a relevant direction for enhancing timetabling performance under complex constraints [5], [6].

3.5 Implications of the Results

The findings indicate that integrating Ant Colony Optimization (ACO) into a Java desktop application can provide a practical operational solution for course timetabling at the study-program level. In practice, the system supports academic data management (courses, lecturers, rooms, and timeslots), user-configurable parameter setting, and automated optimization, while presenting the resulting timetable together with conflict information to facilitate validation. The dedicated lecturer preference module further implies that teaching-availability considerations can be incorporated systematically through soft constraints, enabling schedules that better reflect real teaching conditions without compromising feasibility. From an academic perspective, this study strengthens empirical evidence that ACO is suitable for timetabling problems with many interacting constraints and large search spaces, particularly when evaluated on real institutional data rather than simulated instances. In addition, the study contributes a deployable tool that can be used by non-technical academic staff, thereby helping bridge optimization methods and administrative scheduling needs. Future practical implications include extending institutional policy rules, integrating the system with an academic information system, and applying the approach to larger-scale scheduling scenarios across multiple programs.

3.6 Limitations of the Study

Several limitations should be acknowledged to contextualize the results and guide future work. First, the dataset is derived from a single study program and one academic year, so constraint characteristics and data patterns may be specific to that institutional context. Second, room capacity information is not available in the dataset; therefore, capacity-based constraints (e.g., matching enrolment to room capacity) could not be modeled as hard or soft constraints in the current evaluation. Third, the system is implemented as a Java desktop application, which implies dependency on the user's local environment and does not yet support multiuser access or direct web-based integration. Fourth, although the application allows users to configure ACO parameters, the exploration of parameter-tuning strategies can be extended, including adaptive parameter control to improve stability on larger instances. Finally, timetable quality is influenced by the completeness and accuracy of input data and lecturer preference specifications; when preference data are incomplete or change over time, inputs must be updated before re-optimization. These limitations suggest directions for future research, including adding room-capacity constraints, improving scalability, adopting adaptive parameter tuning, and integrating the system with institutional academic information systems for more automated scheduling workflows.

4. Conclusion

This research developed a Java-based desktop application that automates university course timetabling using Ant Colony Optimization (ACO). The system integrates academic data input, configurable algorithm parameters, optimization execution, and timetable visualization within a single interface to support practical scheduling at the study-program level. Evaluation was conducted using real institutional data from the Informatics Engineering Study Program of STMIK Widya Cipta Dharma with a fixed configuration of 10 ants and 50 iterations per run. Experimental results demonstrate that the proposed system consistently produces feasible timetables without room or lecturer conflicts, achieving a minimum cost value of zero across all runs. As shown by the cost convergence curve, the optimization process continues beyond initial feasibility by gradually reducing soft-constraint penalties, while the feasibility-per-iteration curve indicates that conflict-free solutions are typically reached after a certain number of iterations. Variations in the convergence point across runs reflect the stochastic nature of ACO, particularly during early exploration phases.

The convergence patterns observed in the two curves also highlight opportunities for further improvement. Specifically, the cost convergence curve shows periods of stagnation after feasibility is reached, suggesting that additional local refinement mechanisms could accelerate cost reduction. Similarly, the feasibility rate curve indicates that convergence speed varies across runs, implying sensitivity to parameter settings during early search stages. Based on these observations, future work may explore hybrid optimization strategies that combine ACO with local search or tabu search to enhance post-feasibility solution refinement, as well as adaptive or reinforcement learning–based parameter control to regulate exploration–exploitation balance and reduce convergence variability. Such extensions are directly motivated by the algorithmic behavior observed in the convergence and feasibility curves and would strengthen the system's alignment with advanced AI-based optimization research while maintaining practical applicability.

Acknowledgment

The authors acknowledge the Informatics Engineering Study Program of STMIK Widya Cipta Dharma, Samarinda, for providing formal research permission, supplying official scheduling data, and supporting the evaluation of the proposed scheduling system..

Declarations

Author contribution. The author conceptualized the study, collected and prepared the institutional dataset, designed and implemented the ACO-based scheduling system in Java, conducted the experiments and analysis, and wrote and revised the manuscript.
Funding statement. This research received no external funding.
Conflict of interest. The author declares no conflict of interest.
Additional information. No additional information is available for this paper.

Data and Software Availability Statements

Data availability. The dataset used in this study is derived from institutional academic records of the Informatics Engineering Study Program of STMIK Widya Cipta Dharma (2024 academic year). Due to institutional privacy and administrative policies, the raw data are not publicly available. Aggregated and anonymized information supporting the findings is provided in the manuscript, and further details may be made available from the author upon reasonable request and with institutional permission.

Software availability. The Java netbens desktop application and the source code used to generate and evaluate the timetables are available from the author upon reasonable request.

References

[1] M.-C. Chen, S. N. Sze, S. L. Goh, N. R. Sabar, and G. Kendall, "A survey of university course timetabling problem: perspectives, trends and opportunities," *IEEE Access*, vol. 9, pp. 106515–106541, 2021, doi: 10.1109/ACCESS.2021.3100613.

[2] M. S. Shiri, S. M. Khorramizadeh, and V. Ahmadi, "New heuristic local search method for university course timetabling problem," *Innovation Management and Operational Strategies*, vol. 3, no. 4, pp. 452–464, 2022, doi: 10.22105/imos.2022.332030.1215.

[3] S. Ceschia, L. Di Gaspero, and A. Schaerf, "Educational timetabling: problems, benchmarks, and state-of-the-art," *European Journal of Operational Research*, vol. 308, no. 1, pp. 1–18, 2023, doi: 10.1016/j.ejor.2022.07.011.

[4] R. Ge and J. Chen, "Analysis of college course scheduling problem based on ant colony algorithm," *Computational Intelligence and Neuroscience*, vol. 2022, Article ID 7918323, 2022, doi: 10.1155/2022/7918323.

[5] C. B. C. Mallari, J. L. G. San Juan, and R. C. Li, "The university coursework timetabling problem: an optimization approach to synchronizing course calendars," *Computers & Industrial Engineering*, vol. 184, Article 109561, 2023, doi: 10.1016/j.cie.2023.109561.

[6] S. Abdipoor, R. Yaakob, S. L. Goh, and S. Abdullah, "Meta-heuristic approaches for the university course timetabling problem," *Intelligent Systems with Applications*, vol. 19, Article 200253, 2023, doi: 10.1016/j.iswa.2023.200253.

[7] S. Potluri and K. S. Rao, "Optimization model for QoS based task scheduling in cloud computing environment," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 18, no. 2, pp. 1081–1088, 2020, doi: 10.11591/ijeecs.v18.i2.pp1081-1088.

[8] Al-Mahmud, "Highly constrained university class scheduling using ant colony optimization," *International Journal of Computer Science and Information Technology*, vol. 13, no. 1, pp. 21–32, 2021, doi: 10.5121/ijcsit.2021.13102.

[9] Y. Ikhwani, K. Marzuki, and A. Ramadhan, "Automated university lecture schedule generator based on evolutionary algorithm," *Matrik: Jurnal Manajemen, Teknik Informatika dan Rekayasa Komputer*, vol. 22, no. 1, pp. 129–138, 2022, doi: 10.30812/matrik.v22i1.2215.

[10] Z. Gu, Z. Lü, and J.-K. Hao, "From integer programming to machine learning: a technical review on solving university timetabling problems," *Computation*, vol. 12, no. 1, Article 12, 2024, doi: 10.3390/computation12010012.

[11] R. Hidayat, H. Herlina, M. Abduh, and I. Y. Wulandari, "Optimasi berbasis swarm intelligence untuk penjadwalan mata kuliah di perguruan tinggi," *Jurnal Teknologi dan Sains Modern*, vol. 2, no. 3, pp. 114–123, 2025, doi: 10.69930/jtsm.v2i3.380.

[12] D. Romaguera, J. Plender-Nabasa, J. Matias, and L. Austero, "Development of a web-based course timetabling system based on an enhanced genetic algorithm," *Procedia Computer Science*, vol. 234, pp. 1714–1721, 2024, doi: 10.1016/j.procs.2024.03.177.

[13] A. Johan, R. Pratama, and D. Siregar, "Optimization of university course scheduling using metaheuristic approaches," *Journal of Informatics and Computer Science*, vol. 8, no. 1, pp. 45–54, 2024.

[14] M. Davison and A. Kheiri, "Modelling and solving the university course timetabling problem with hybrid teaching considerations," *Journal of Scheduling*, vol. 28, no. 2, pp. 195–215, 2025, doi: 10.1007/s10951-024-00817-w.

[15] T. Purniawan, M. Fahmi, and Salmon, "Penerapan information architecture untuk optimalisasi website sistem informasi akademik (SIAK) STMIK Widya Cipta Dharma," *Jurnal Nasional Komputasi dan Teknologi Informasi (JNKTI)*, vol. 8, no. 3, pp. 1174–1184, 2025.

[16] C. Blum, "Ant colony optimization: a bibliometric review," *Physics of Life Reviews*, vol. 51, pp. 87–95, 2024, doi: 10.1016/j.plrev.2024.09.014.

[17] M. Chen, F. Werner, and M. Shokouhifar, "Mathematical modeling and exact optimizing of university course scheduling considering preferences of professors," *Axioms*, vol. 12, no. 5, Article 498, 2023, doi: 10.3390/axioms12050498.

[18] H. J. Christanto and Y. A. Singgalen, "Analysis and design of student guidance information system through software development life cycle (SDLC) and waterfall model," *Journal of Information Systems and Informatics*, vol. 5, no. 1, pp. 259–270, 2023, doi: 10.51519/journalisi.v5i1.443.

[19] M. Neroni, "Ant colony optimization with warm-up," *Algorithms*, vol. 14, no. 10, Article 295, 2021, doi: 10.3390/a14100295.

[20] A. Perez-Napalit, "Ant-inspired scheduling: integrating constraint satisfaction problem with ant colony optimization," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 13, no. 3, pp. 112–118, 2024, doi: 10.30534/ijatcse/2024/031332024.

[21] H. M. H. Bestari, P. P. Suryadhini, and Nopendri, "Flow shop scheduling using a combination of ant colony optimization algorithm and tabu search algorithm to minimize total tardiness," *Jurnal Teknik Industri*, vol. 10, no. 2, pp. 383–392, 2024.

[22] M. S. Shiri, S. M. Khorramizadeh, and V. Ahmadi, "New heuristic local search method for university course timetabling problem," *Innovation Management and Operational Strategies*, vol. 3, no. 4, pp. 452–464, 2022, doi: 10.22105/imos.2022.332030.1215.