JOURNAL OF DATA SCIENCE | Vol.2024:57
eISSN:2805-5160

Optimization Algorithms: A Comparison Study for Scheduling Problem at UIN Raden Fatah's Sharia and Law Faculty

Mustakim[1], Tri Basuki Kurniawan[1*], Misinem[2], Edi Surya Negara[1], Izman Herdiansyah[1]

[1]Magister of Information Technology, Universitas Bina Darma, Palembang, Indonesia
[2]Faculty of Vocation, Universitas Bina Darma, Palembang, Indonesia

[*]**Email:** tribasukikurniawan@binadarma.ac.id

Abstract

The rapid advancement of information and communication technology significantly impacts various sectors, including education, by enhancing administrative and academic processes through sophisticated algorithms and systems. At Raden Fatah State Islamic University Palembang, specifically within the Faculty of Sharia and Law, technology is pivotal in managing complex course scheduling challenges due to increasing student numbers and curriculum intricacies. This study examines the effectiveness of optimization algorithms in improving the efficiency and quality of academic scheduling. We focus on two prominent optimization techniques, Genetic Algorithms (GA) and Ant Colony Optimization (ACO), chosen for their capability to address the complex optimization problems typical in academic settings. The research encompasses a systematic approach, beginning with a clear definition of constraints and objectives, followed by designing and implementing both algorithms to address the scheduling issues at the Faculty of Sharia and Law. Our experimental evaluation compares the performance of GA and ACO across multiple metrics, including execution time, memory usage, fitness, and adaptability to dynamic conditions. Results indicate that while GA generally offers faster solutions, it requires more memory and shows variability in achieving optimal fitness levels. Conversely, ACO, though occasionally slower, consistently produces higher quality solutions with greater memory efficiency, making it more suitable for resource-constrained environments. The best results from the experiments highlight that ACO outperformed GA in terms of overall solution quality and resource efficiency, with an execution time of 19.27 seconds and 14,218.14 KB. Specifically, ACO consistently achieved near-optimal fitness scores with significantly lower memory usage compared to GA. This demonstrates ACO's robustness and suitability for handling complex scheduling problems where resource conservation is crucial. The choice between GA and ACO should be influenced by specific situational requirements—GA is recommended where speed is critical, while ACO is preferable in settings requiring high-quality, resource-efficient solutions. Future research should explore refining these algorithms, possibly through hybrid approaches that leverage the strengths of both to enhance their effectiveness and adaptability in complex scheduling scenarios. This study not only informs the academic community about effective scheduling practices but also sets a benchmark for future technological implementations in educational institutions.

Submission: 23 October 2024; **Acceptance:** 25 November 2024

Keywords

Genetic Algorithm, Ant Colony Optimization, Course Scheduling

Introduction

The development of information and communication technology has brought significant changes in various aspects of life, including in the education sector. Universities, as centers of learning and research, have utilized technology to improve and speed up administrative and academic processes. From digitalized learning management systems to the use of advanced algorithms for scheduling and resource management, technology has become an important catalyst in improving operational efficiency and educational quality. Specifically, at Raden Fatah State Islamic University Palembang, the use of technology not only facilitates better management of data and resources but also supports innovation in teaching and research methodology. Optimization algorithms, for example, have been used to solve complex scheduling problems, which previously required large amounts of time and human resources to solve.

The Faculty of Sharia and Law is one of the faculties at UIN Raden Fatah Palembang, which has an important role in the development of Sharia and legal science in Indonesia. This faculty has 5 study programs, of which there are four undergraduate study programs (S1) and one master or second-degree study program (S2), which are designed to prepare students to become legal experts who not only understand law in general but also Islamic law following Indonesian social and cultural context. As the number of students increases and the complexity of the curriculum offered, the Faculty of Sharia and Law faces challenges in managing course schedules effectively and efficiently, a critical aspect that influences academic performance and student and lecturer satisfaction.

Scheduling courses at the Faculty of Sharia and Law is a vital process that requires special attention to various important aspects to support the effectiveness of learning and meet the needs of lecturers and students. This process must be designed with a complex curriculum in mind, ensuring that all prerequisites and continuation schedules are arranged in the correct order. In addition, lecturers' availability and teaching time preferences should be integrated into the schedule while considering their other responsibilities, such as research and community service. Efficiency in the use of classroom space is also crucial, especially considering space limitations and the special needs of some schedules. It is also important to reduce clashes between courses so that students can take the desired schedule without obstacles.

Optimization algorithms in course scheduling play an important role in increasing the efficiency and effectiveness of managing academic schedules in educational institutions. Advances in optimization algorithms also allow universities to adapt quickly to sudden changes or needs, such as the shift to hybrid or online learning models that require more flexible schedule adjustments. This algorithm is specifically designed to overcome the challenges of creating a schedule that not only meets various constraints such as lecturer availability, classroom requirements, and course prerequisites but also maximizes student and lecturer satisfaction.

Implementation of this algorithm not only helps in reducing the time required to prepare a schedule but also offers the ability to simulate and predict various scheduling scenarios.

This is very helpful in making strategic decisions regarding managing the number of students per class and the distribution of lecturer teaching loads. Some researchers have been using optimization in their research, like Oktaviani and Riti (2023), comparing two algorithms, the Welch Powell algorithm and the Greedy algorithm, to schedule the lecturer room at the Faculty of Technique on their campus. Next, Ardiyani (2022) also uses a Genetic Algorithm and Steepest Ascent Hill Claiming to schedule the lecturer of the subjects. Before that, Laswi (2020) also used the Fitness of Spring algorithm and Tabu search algorithm to schedule at Universitas Andi Djemma Palopo. In our opinion, optimization course scheduling is an interesting area of research, and many researchers are trying to find the best solution to their specific problem.

There are many kinds of algorithms already used in course scheduling, such as genetic algorithm, greedy algorithm, tabu search, and so on. In this research, we used and compared two algorithms, namely genetic algorithm and Ant Colony Optimization. Comparing Genetic Algorithm and Ant Colony Optimization in course scheduling is essential to determine which approach better handles constraints, adapts to changes, and scales with problem complexity. This comparison provides insights into each algorithm's strengths and weaknesses, guiding the selection of the most effective optimization strategy for producing high-quality, adaptable, and scalable schedules.

Methodology Research

To solve the course scheduling optimization problem, we can follow a detailed step-by-step methodology that leverages both Genetic Algorithm (GA) and Ant Colony Optimization (ACO) techniques. This approach involves problem definition, model formulation, algorithm design, implementation, evaluation, and result comparison. Here's a detailed methodology with each step explained and references included.

Genetic Algorithm

Genetic Algorithm (GA) is an optimization technique inspired by the principles of natural selection and evolutionary biology, widely used to solve complex problems by evolving a population of candidate solutions over several generations (Gen & Lin, 2023). GA starts with an initial population of random solutions, often called "chromosomes," which represent potential answers to the problem. Each chromosome is encoded in a format suitable for manipulation, such as binary strings or real numbers (Alam et al., 2020).

The core process of GA involves three main operators: selection, crossover, and mutation. Selection involves choosing the fittest individuals from the population based on a fitness function that evaluates how well each solution meets the desired objectives (Almulla & Gay, 2022). Crossover combines two parent solutions to produce offspring, introducing diversity and new structures into the solution pool. Mutation introduces small random changes to individuals, maintaining genetic diversity and preventing the algorithm from getting stuck in local optima

(Zainuddin & Abd Samad, 2020). This iterative process continues until a termination condition is met, such as reaching a maximum number of generations or achieving a satisfactory fitness level.

GA is especially effective in complex scheduling, optimization, and multi-objective problems due to its robustness and ability to explore large search spaces (Maghawry et al., 2021). Its adaptability allows it to find near-optimal solutions where traditional deterministic methods struggle, making it a preferred approach in fields such as scheduling, machine learning, and artificial intelligence. The ongoing advancements in GA, such as hybrid approaches and adaptive mechanisms, further enhance its performance, making it a valuable tool for tackling real-world challenges in optimization (Azevedo et al., 2024).

Here is the pseudocode for the Genetic Algorithm, as shown in Figure 1.

```
// Genetic Algorithm Pseudocode
1.  Initialize a population of chromosomes (solutions) randomly.
2.  Evaluate the fitness of each chromosome using a fitness function.
3.  Repeat until the termination condition is met (e.g., maximum
    generations or satisfactory fitness level):
    a. Selection:
       -   Select pairs of chromosomes from the current population based
           on their fitness (e.g., tournament selection, roulette
           wheel).
    b. Crossover:
       -   Perform crossover on selected pairs to generate offspring
           (e.g., single-point or multi-point crossover).
    c. Mutation:
       -   Apply mutation to offspring with a given mutation probability
           (randomly alter genes to maintain diversity).
    d. Fitness Evaluation:
       -   Evaluate the fitness of the new offspring.
    e. Replacement:
       -   Replace the least fit individuals in the population with the
           new offspring.
4.  End Repeat.
5.  Return the best solution found (the chromosome with the highest.
```

Figure 1 Pseudo-code of Genetic Algorithm

Based on Figure 1. the algorithm starts by generating an initial population of solutions (chromosomes) randomly. Each chromosome represents a potential solution, such as a course schedule with assigned timeslots and rooms. The size of the population (number of chromosomes) is a critical parameter affecting the algorithm's performance.

Then, each chromosome is evaluated using a fitness function that measures how well the solution meets the problem's objectives and constraints. For course scheduling, the fitness function might evaluate criteria such as no conflicts between courses, balanced room usage, or minimized instructor scheduling conflicts. Higher fitness values represent better solutions.

Repeat until the termination condition is met (e.g., maximum generations or satisfactory fitness level). The algorithm iteratively improves the population until a predefined condition is met, such as reaching a maximum number of generations or finding a solution with an acceptable fitness score. Selection involves choosing pairs of parent chromosomes from the current population based on their fitness. Techniques like tournament selection, roulette wheel selection,

or rank selection are commonly used. Higher fitness chromosomes have a higher probability of being selected, ensuring that good solutions are propagated. Crossover is the process of combining two parent chromosomes to create offspring. This step introduces new combinations of genes (solution components) into the population. Types of crossover include single-point, two-point, and uniform crossover. The goal is to mix traits from parents to create potentially better offspring. Mutation introduces random changes to genes in the offspring, which helps maintain genetic diversity and avoid local optima. For example, in course scheduling, a mutation might swap time slots between two courses. Mutation occurs with a low probability to ensure the population remains diverse.

After crossover and mutation, the new offspring are evaluated using the fitness function. This evaluation determines how well the new solutions perform compared to the existing population. The offspring replace the least fit individuals in the population, ensuring that the population evolves over generations towards better solutions. The replacement strategy helps maintain a high-quality population.

The iterative process continues until the termination condition is met, ensuring that the algorithm stops at an optimal or near-optimal solution. The algorithm returns the best chromosome (solution) found during the iterations. This final solution is expected to be the most optimized in terms of the fitness function.

Ant Colony Optimization

Ant Colony Optimization (ACO) is a probabilistic optimization algorithm inspired by the foraging behavior of ants. ACO was introduced by Marco Dorigo in the early 1990s and is particularly effective in solving complex combinatorial problems such as routing, scheduling, and network optimization (Dorigo & Stützle, 2019). The core idea of ACO is based on how ants find the shortest path between their nest and a food source. Ants communicate indirectly through pheromone trails, which they deposit on paths they travel. The more pheromones on a path, the more attractive it becomes to other ants, leading them to follow and reinforce it. This positive feedback loop helps ants converge on the optimal path over time.

In ACO, artificial ants are deployed to construct solutions to the problem iteratively. The algorithm starts with ants randomly exploring paths and depositing pheromones based on the quality of the solution found. The quality is usually determined by a heuristic function that assesses the efficiency or cost of the path. As ants traverse different paths, they update the pheromone levels, influencing subsequent ants to favor paths with higher pheromone concentrations (Socha & Blum, 2024). Additionally, pheromone evaporation occurs to prevent premature convergence to suboptimal solutions, maintaining the algorithm's exploration capabilities. ACO balances exploration and exploitation effectively, as new paths can still emerge and be reinforced if they prove beneficial (Liu et al., 2023).

The iterative process continues until a termination criterion is met, such as reaching a maximum number of iterations or when the solutions converge satisfactorily. ACO’s strength lies in its ability to handle dynamic changes in the problem space, adapt to varying constraints, and find near-optimal solutions in complex environments (Zhou et al., 2022). This makes it particularly

suitable for scheduling problems, including course scheduling, where numerous variables and constraints are at (Savitri et al., 2020).

Here is the pseudocode for the Ant Colony Optimization, as shown in Figure 2.

```
    // Ant Colony Optimization Pseudocode
1.  Initialize pheromone trails and heuristic information.
2.  Repeat until the termination condition is met (e.g., maximum
    iterations or solution convergence):
    a.  Solution Construction: -
        For each ant in the colony:
        i.   Start from an initial state (e.g., first-course
             assignment).
        ii.  Iteratively build a solution by selecting the next
             component (e.g., time slot, room) based on pheromone and
             heuristic values.
        iii. Evaluate the quality of the constructed solution using a
             fitness function.
    b.  Pheromone Update: - Update pheromone trails based on the
        quality of the solutions. Increase pheromones on paths that
        form better solutions.
    c.  Pheromone Evaporation: - Reduce pheromone levels globally to
        avoid premature convergence (simulates evaporation).
3.  End Repeat.
4.  Return the best solution found (the solution with the highest
    quality).
```

Figure 2 Pseudo-code of Ant Colony Optimization

Based on Figure 2, ACO starts by initializing pheromone levels on all possible paths (e.g., timeslot assignments for courses) and defining heuristic information based on prior knowledge or problem-specific rules. Initial pheromone levels are usually set uniformly, and heuristic values guide ants toward promising paths. Next, the algorithm repeatedly constructs and refines solutions until a termination criterion, such as reaching a maximum number of iterations or when the solutions stabilize, is met.

Each ant in the colony builds a complete solution by selecting components (e.g., assigning a course to a timeslot and room) probabilistically. The probability of choosing a component depends on the pheromone levels and heuristic desirability. Ants prefer paths with higher pheromone concentrations and favorable heuristic values, balancing exploration and exploitation.

Comparing Genetic Algorithm (GA) and Ant Colony Optimization (ACO)

Comparing Genetic Algorithm (GA) and Ant Colony Optimization (ACO) in course scheduling problems is crucial because these algorithms employ different optimization strategies, which influence their performance in unique ways. GA is inspired by natural selection and evolves potential solutions through selection, crossover, and mutation, making it effective for problems where exploring various combinations is necessary (Albadr et al., 2022). In contrast, ACO mimics the foraging behavior of ants, constructing solutions incrementally based on pheromone trails and heuristic information, which helps in gradually building better solutions by reinforcing successful paths (Dorigo & Stützle, 2019). Understanding these distinct approaches helps in identifying which algorithm is more effective for specific scheduling challenges.

One key reason for comparison is performance evaluation, focusing on solution quality, convergence speed, and computational efficiency. Course scheduling is complex, involving numerous constraints such as avoiding conflicts, meeting room capacities, and accommodating instructor preferences. GA and ACO handle these constraints differently; ACO's incremental approach can better incorporate constraints during solution construction, whereas GA relies on population-based operations that may struggle with maintaining feasible solutions (Karimi et al., 2022). Performance evaluation of these algorithms helps to identify which is more suitable under varying constraints and requirements.

Adaptability and flexibility are also important factors, as course schedules often need to adjust to new constraints, courses, or dynamic changes. ACO's iterative nature typically offers better adaptability, responding well to changes by updating pheromone levels, whereas GA's adaptability depends heavily on the design of its operators and the balance of its exploration and exploitation mechanisms (Hussain & Muhammad, 2020). Scalability is another consideration; as scheduling problems grow in complexity, comparing how each algorithm handles increased size and constraint complexity helps determine the best approach for larger scenarios (Savitri et al., 2020).

Additionally, the balance between exploration and exploitation differs between the two algorithms, with GA favoring exploration due to its randomized nature, potentially offering a broader search of the solution space. In contrast, ACO focuses more on exploiting known good solutions, refining them through pheromone reinforcement. This balance significantly affects their performance in finding optimal schedules (Merdanoğlu et al., 2020). Practical factors, including ease of implementation, required computational resources, and robustness against poor initial conditions, also play a role in choosing the most suitable algorithm for course scheduling problems.

Both algorithms are powerful for solving course scheduling problems, and their performance varies based on problem complexity, constraints, and specific algorithm parameters, but each of them has its key points.

Genetic Algorithm: Emphasizes population-based search, with evolution driven by selection, crossover, and mutation. It is effective for exploring diverse solutions and is highly adaptable but relies heavily on the design of the fitness function and genetic operators.

Ant Colony Optimization: Utilizes pheromone-guided search with a focus on constructing solutions step-by-step. It is particularly good at exploiting known good paths while still exploring new possibilities through pheromone evaporation and probabilistic choices.

Constraints in Course Scheduling Optimization Problem

In the context of course schedule optimization, "hard constraints" and "soft constraints" play crucial roles in shaping the formulation and solution of the problem. Here's a detailed look at each:

Hard Constraints

Hard constraints are rules that must be strictly adhered to in the scheduling process. Violating these constraints renders a schedule invalid. These are typically non-negotiable requirements set by

institutional policies, physical realities, or legal requirements. Common examples of hard constraints in course scheduling include:

- **Room Capacity**: A class size cannot exceed the maximum seating capacity of the assigned room.
- **Instructor Availability**: Courses must be scheduled when the instructor is available to teach.
- **No Double Booking**: A room or instructor cannot be assigned to more than one class at the same time.
- **Course Prerequisites**: Some courses must be scheduled in such a way that students who need to take prerequisites can do so in the correct order.

Soft Constraints

Soft constraints, unlike hard constraints, are preferable conditions that should be met to the extent possible but can be compromised if necessary. These constraints are often related to convenience, preference, or optimization goals and are used to enhance the overall quality and functionality of the schedule. Examples of soft constraints include:

- **Minimizing Student Wait Time**: Scheduling classes in a way that minimizes idle time for students between courses.
- **Balancing Instructor Workload**: Distributing teaching hours evenly among faculty members to avoid overburdening any single instructor.
- **Preferred Timing**: Scheduling courses at times that are preferred by students or faculty, such as avoiding early morning or late evening slots if unpopular.
- **Maximizing Room Utilization**: Efficiently using available space to minimize the number of partially filled or unused classrooms.

Results and Discussion

Setting up the Constraints

In course schedule optimization, constraints are categorized into two main types: hard constraints and soft constraints. Hard constraints are essential rules that must be strictly followed to ensure the schedule is valid. These include ensuring that courses are scheduled only when rooms are available (room availability), aligning courses with the qualifications and availability of instructors to avoid conflicts (instructor assignment), scheduling courses in a sequence that respects prerequisite requirements (course prerequisites), and maintaining class sizes within the limits of room capacities (class size limit).

On the other hand, soft constraints are more flexible and are used to optimize the schedule for better functionality and user satisfaction. Examples of soft constraints include minimizing the idle time students spend between classes (minimizing student wait times), accommodating faculty preferences for teaching times and days wherever possible (faculty preferences), spreading courses evenly throughout the week to avoid daily workload imbalances (even distribution of courses), and efficiently using classrooms by matching class sizes to the capacities of available rooms (maximizing room utilization).

Together, these constraints ensure that the course schedule not only meets the basic functional requirements but also enhances the overall educational environment by addressing preferences and optimizing resource use. In this research, we proposed the constraints as follows in Table 1.

Table 1. The results and Comparison results.

No	Hard Constraints	Soft Constraints
1	Each lecturer may only teach one class in one subject in the room on the same day and at the same time.	Every extraordinary lecturer teaches maximum on Fridays and Saturdays.
2	Each class may not be scheduled for more than one course on the same day and time.	Each lecturer teaches a maximum of three courses in one day.
3	Each room may not have more than one course scheduled on the same day and time.	Each lecturer teaches a maximum of two consecutive courses in one day.
4	Each Homebase lecturer is required to teach in their respective study program.	Each class can study a maximum of three subjects in one day.
5	Each lecturer with the positions of vice chancellor, dean, and head of unit can only teach 6 credits or 3 courses.	Each class can study a maximum of two consecutive subjects in one day.
6	Each lecturer with the position of deputy dean, head of study program, and secretary of study can only teach 10 credits	Each lecturer with a position teaches a maximum of 2 days.
7	Each professor with a position can only teach 10 credits	Every male lecturer has a maximum free schedule on Friday.
8	Each professor without a position can only teach 12 credits.	Each lecturer can continue teaching until Friday
9	Each permanent lecturer without a position can only teach 16 credits	Every extraordinary lecturer teaches maximum on Fridays and Saturdays.
10	Each extraordinary lecturer can only teach 4 credits.	

Based on Table 1, the course scheduling constraints provided can be divided into hard constraints, which are mandatory and must be adhered to for a valid schedule, and soft constraints, which are more flexible and aim to optimize the scheduling experience. Hard Constraints involve strict rules around scheduling and teaching assignments. For instance, lecturers can only teach one class on a given subject in a specific room at the same time and date. Each class is similarly restricted from having more than one course scheduled concurrently. Rooms cannot be double-booked, and each must host only one course at any given time. Homebase lecturers are required to teach within their specific programs. Additionally, there are teaching load limitations for those in administrative positions, with varying credit caps depending on the role—ranging from 6 to 16 credits tailored to the lecturer's administrative responsibilities and rank.

Soft Constraints focus on optimizing the scheduling process according to preferences and practical logistics. These include limits on the number of courses a lecturer can teach per day, with a preference for limiting extraordinary lecturers to teaching primarily on Fridays and Saturdays. There are also guidelines suggesting that classes should not study more than three subjects per day and should avoid back-to-back classes in more than two subjects. For administrative lecturers, the teaching days are preferably capped at two, and specific provisions are made for male lecturers to ideally have free schedules on Fridays.

These constraints ensure a balanced, equitable, and efficient scheduling system that accommodates the needs and limits of both faculty and students while maintaining the integrity of the educational institution's operational standards.

Chromosome representation

In genetic algorithms used for scheduling, the representation of chromosomes and their components plays a crucial role in defining the solution space and the effectiveness of the algorithm. Each chromosome represents a potential solution to the scheduling problem and is composed of 25 genes, each gene corresponding to a specific scheduling attribute. The range for these genes is from 0 to 24, representing the real-number values that each gene can take within a chromosome.

The gene data in this scheduling scenario is composed of three main components: course data, time data, and space data, which together encompass all necessary details for constructing a viable schedule. Here's a more detailed breakdown of each component:

1. Course Data: This component of the gene includes sub-gen data related to lecturers, courses, and classes. Each gene in this segment specifies which lecturer is assigned to which course and which class is taking the course. This layer ensures that each course is properly linked to both its instructor and the appropriate student group.
2. Time Data: This component consists of sub-gen data for lecture days and hours. It encodes the specific days of the week and time slots during which the courses will be taught. Effective encoding of this data ensures that courses are scheduled at times when lecturers are available and that there are no conflicts between different courses requiring the same students or lecturers.
3. Room Data: The final component involves the sub-gen data for lecture halls. Each gene specifies which classroom or lecture hall will be used for each session. This part of the gene ensures that the physical resources are adequately allocated, avoiding room capacity issues and ensuring that each course is placed in a suitably equipped room.

The chromosome structure in this scheduling algorithm intricately combines lecturer, course, class, timing, and space allocation into a cohesive unit represented by a series of real numbers. Each gene within a chromosome comprehensively covers one aspect of the schedule, ensuring all factors, such as who is teaching, what is being taught, where it's taught, and when it's taught, are harmoniously aligned. This complex representation allows the genetic algorithm to effectively search through potential scheduling configurations to find the most optimal arrangement that meets all given constraints and requirements.

Pheromone representation

In ACO, the pheromone matrix is a vital component that simulates the pheromone trails left by ants. Each element in this matrix represents a potential decision in the scheduling process, such as assigning a particular time slot, room, or instructor to a specific course. The purpose of this matrix is to guide the artificial ants (simulated agents) in exploring and exploiting the solution space, thereby finding an optimal or near-optimal schedule for courses.

Initialization Process

1. Uniform Initial Values: Initially, all entries in the pheromone matrix are set to a uniform value. This initial value is typically small but positive, ensuring that no potential paths are excluded prematurely from exploration due to lack of attractiveness. The uniformity ensures that initially, no preference is given to any specific course assignment, promoting an unbiased exploration of all possible scheduling combinations.
2. Significance of the Initial Value: The exact value chosen for initialization often depends on specific aspects of the course scheduling problem and the parameters of the ACO algorithm, such as the evaporation rate of the pheromones. This rate affects how quickly pheromones decay, thus influencing decisions on replenishing or reinforcing paths with higher pheromone levels over time.
3. Normalization and Scaling: In some implementations, the initial pheromone values may be normalized to ensure that the sum of all pheromones stays within a defined range. This normalization helps maintain a balance between exploration (trying out new paths) and exploitation (deepening the search along promising paths), which is crucial for finding a balanced and efficient schedule.

Importance of Proper Initialization

The way the pheromone matrix is initialized plays a foundational role in the dynamics of the ACO algorithm applied to course scheduling. Proper initialization ensures that the algorithm begins with a state of exploration where no possibilities are overly favored, allowing the algorithm to adapt dynamically based on the experiences (simulated through pheromone trails) of the artificial ants. Over time, as ants traverse through different scheduling options and deposit pheromones based on the quality of the solutions (i.e., schedules with fewer conflicts and better resource utilization), the pheromone matrix evolves to reflect these learned preferences, guiding ants towards more optimal schedules.

This method of initialization and subsequent adaptive learning mimics the natural behavior of ants and is particularly effective in complex optimization problems like course scheduling, where multiple constraints and preferences must be balanced to achieve an optimal solution. The ACO's pheromone-based guidance helps explore diverse scheduling possibilities and refine them into practical, efficient schedules.

Fitness function

In the genetic algorithm designed for course scheduling, the evaluation of each potential solution, or chromosome, hinges on a well-defined fitness function. The fitness of a chromosome is a measure of how well it meets the requirements of the scheduling problem without violating constraints. The specific fitness function used in this scenario is represented by the equation:

$$Fitness = \frac{1}{(1 + RC + LC + CC)}$$

where RC represents a room clash, LC denotes a lecturer clash, and CC signifies a class clash. These components are critical in determining the effectiveness and viability of a given schedule.

1. **Room Clash (RC)**: This clash occurs when two or more courses are assigned to the same lecture hall at the same time. A high number of space clashes indicates a significant overlap

in room assignments, leading to a schedule that is not feasible in practice. Minimizing CRCRCR is essential for ensuring that physical resources are utilized without conflicts.

2. **Lecturer Clash (LC)**: A lecturer clash happens when a lecturer is scheduled to teach more than one course simultaneously. This type of clash directly impacts the practicality of the timetable, as a lecturer can only be in one place at one time. Reducing CDCDCD is crucial for maintaining the integrity and logistical feasibility of the course timetable.
3. **Class Clash (CC)**: This clash arises when a class is scheduled to attend more than one course at the same time. Like lecturer clashes, class clashes make a schedule impractical, as students cannot split their presence between two different courses simultaneously. Ensuring that CKCKCK is minimized is vital for a functional and student-friendly schedule.

The objective of the fitness function is to minimize the sum of these clashes, thus maximizing the overall fitness value of the chromosome. By inversely relating the fitness to the sum of these clashes (adding 1 to avoid division by zero and ensure stability), the fitness function effectively prioritizes schedules with fewer conflicts, guiding the genetic algorithm toward the most optimal and conflict-free solutions. The goal is to refine the search within the genetic algorithm to produce a course schedule that is as close to ideal as possible, minimizing disruption and maximizing resource utilization.

Results

Genetic Algorithm Experiment Results

To effectively analyze and present the results from executing a Genetic Algorithm (GA) 10 times for course scheduling optimization, including performance metrics such as execution time, memory usage, fitness, and the maximum number of iterations required to reach the best solutions, we can structure the information in a comprehensive and accessible manner. Here's a detailed breakdown, as shown in Table 2, and Figure 3 and Figure 4 show the execution time and memory usage bar chart.

Table 2. GA Experiment Results

No	Time (s)	Memory (Kbytes)	Output	
			Fitness	Generation
1	24.70	392,435.70	1	48
2	13.27	223,726.13	1	27
3	21.59	359,730.41	1	44
4	16.37	279,581.91	1	34
5	23.58	393,540.25	1	48
6	25.59	416,164.63	0.66	50
7	21.29	357,396.20	1	45
8	24.86	417,153.26	0.33	50
9	24.78	415,989.60	0.80	50
10	21.90	374,659.92	1	45

Figure 3 Execution time for ten running GA experiment results

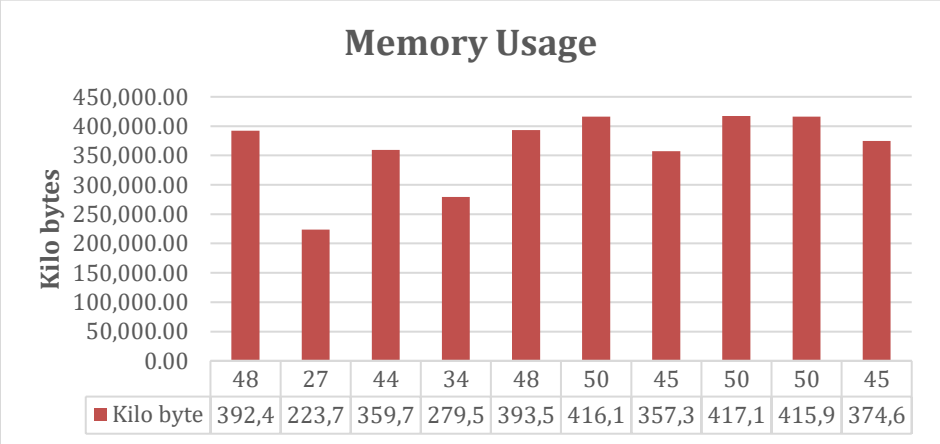

Figure 4 Memory usage for ten running GA experiment results

Based on Table 2, the data from 10 executions of a Genetic Algorithm (GA) used for course scheduling optimization reveals various insights into the algorithm's performance across metrics such as execution time, memory usage, fitness, and the number of generations needed to achieve optimal solutions.

In terms of execution time analysis, the execution times for the algorithm varied from 13.27 seconds to 25.59 seconds. The average time was around 21 seconds, indicating a moderate level of computational efficiency. Notably, the shortest execution time did not necessarily correlate with a compromised fitness outcome, as evidenced by the second run achieving the highest fitness in just 13.27 seconds. Next, the memory usage shows there was a significant fluctuation in memory usage across different runs, ranging from 223,726.13 KB to 417,153.26 KB. The runs with the highest memory usage, specifically run 6, 8, and 9, which all exceeded 415,000 KB, suggest that some instances of the algorithm may engage in more complex computations or retain more candidate solutions in memory.

Furthermore, in the fitness Analysis, fitness scores varied from 0.33 to 1.0, with most runs achieving a perfect score of 1.0, indicative of optimal solutions as per the fitness evaluation criteria. However, three runs did not reach a fitness score of 1, with the eighth run recording the lowest at 0.33, suggesting potential issues like premature convergence to local optima or insufficient exploration of the solution space. Lastly, the number of generations needed to reach the best solutions ranged from 27 to 50. Interestingly, runs requiring the maximum number of generations (50) corresponded to those with the lower fitness scores, implying that the algorithm struggled to find optimal solutions and required more iterations to attempt improvements.

While the GA generally performs well, achieving perfect fitness in most runs, the observed discrepancies in memory usage and the iterations required to optimize solutions provide critical areas for further research and algorithm refinement.

Ant Colony Optimization Experiment Results

We can organize the data thoroughly and understandably to efficiently analyze and present the findings from running an Ant Colony Optimization (ACO) ten (10) times for course scheduling optimization, including performance metrics like execution time, memory usage, fitness, and the maximum number of iterations needed to find the best solutions. Table 3 provides a full summary, and Figures 5 and 6 display a bar chart of memory utilization and execution time.

Table 3. Ant Colony Optimization Experiment Results

No	Time (s)	Memory (Kbytes)	Output	
			Fitness	Generation
1	152.05	14,225.30	1	15
2	486.51	13,943.97	0.992	50
3	158.16	14,223.44	1	16
4	497.32	13,943.97	0.994	50
5	325.55	14,230.98	1	34
6	37.54	14,221.10	1	4
7	19.27	14,218.14	1	2
8	180.41	14,227.08	1	19
9	37.83	14,218.88	1	4
10	483.85	13,943.97	0.991	50

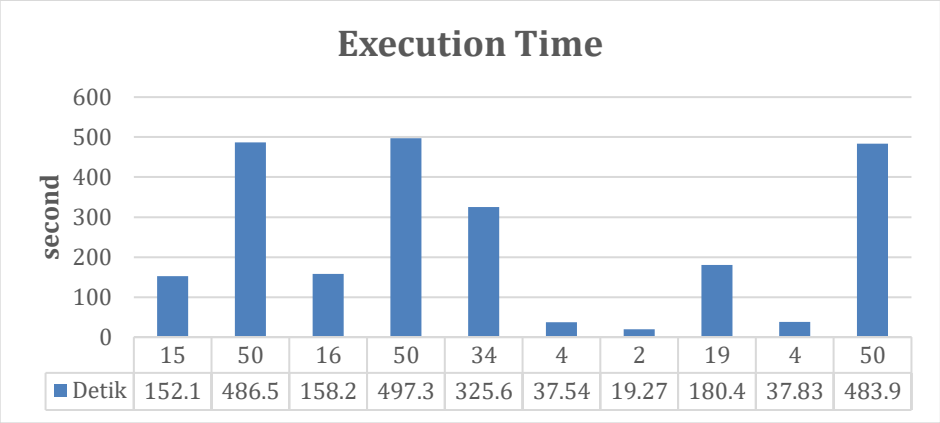

Figure 5 Execution time for ten running ACO experiment results.

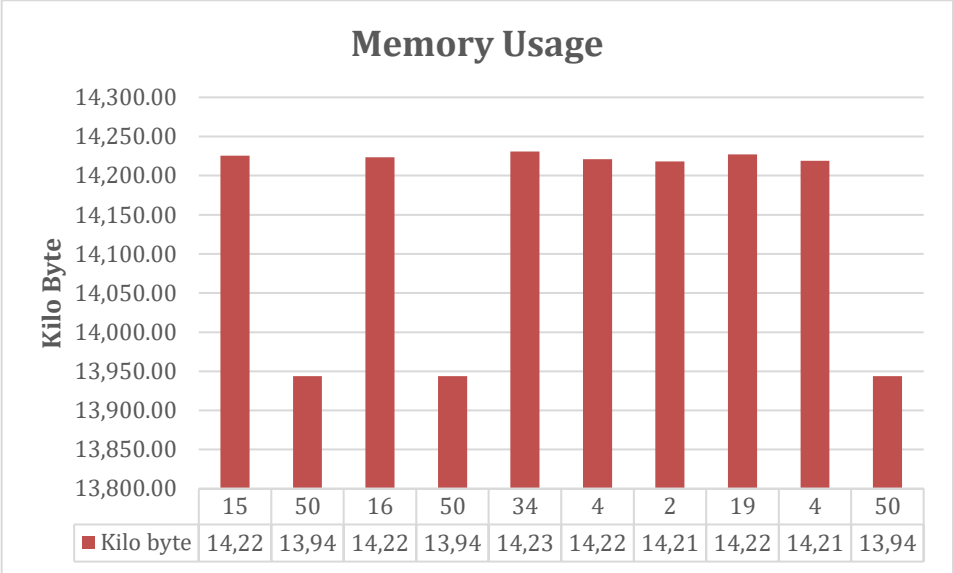

Figure 6 Memory usage for ten running ACO experiment results.

Analyzing the performance metrics from 10 runs of an Ant Colony Optimization (ACO) algorithm for a scheduling task, we observe significant variability in execution time, memory usage, fitness, and the number of generations needed for optimization.

The execution times ranged dramatically from a low of 19.27 seconds to a high of 497.32 seconds. Particularly notable are Runs 2, 4, and 10, each of which not only took significantly longer—over 480 seconds—but also scored the lowest fitness values. This suggests these runs may have faced complexities or inefficiencies, potentially struggling with the algorithm's convergence or exploration capabilities.

In terms of execution time, across all runs, memory usage remained relatively consistent, with most runs around 14,220 Kbytes and a slight dip to about 13,944 Kbytes in the longer, less successful runs. This stable memory usage across varied performance outcomes suggests that while the algorithm is generally efficient in its resource use, the longer execution times in some runs did not translate to increased memory demands.

Fitness scores were predominantly high, with most runs achieving a perfect score of 1.0, indicating successful optimization. However, the same runs that experienced extended execution times (Runs 2, 4, and 10) also reported fitness values just below 1.0, along with the maximum count of 50 generations. This indicates a struggle to efficiently find optimal solutions despite numerous iterations.

Analysis and Discussion

GA tends to complete tasks more quickly, with execution times consistently shorter than those of ACO, ranging from 13.27 to 25.59 seconds. However, it uses significantly more memory, with usage ranging from about 223,726 to 417,153 Kbytes. This suggests GA may require considerable resources, which could be a drawback in environments with limited computational capacity. While GA typically achieves high fitness scores, there are notable exceptions where it falls short, particularly in runs that also require the maximum number of generations. This suggests potential issues with consistency and efficiency in finding optimal solutions.

In contrast, ACO shows more variability in execution times but is considerably more efficient in terms of memory usage, maintaining around 14,220 Kbytes across all runs. ACO generally achieves near-optimal fitness scores, with most runs scoring perfect or near-perfect, even in those that take longer to execute. This indicates a robust ability to find high-quality solutions consistently, making it particularly suitable for complex problems where memory efficiency is critical.

When comparing the two, GA is faster but less consistent in achieving optimal fitness, and it requires more memory. ACO, while sometimes slower, provides a more consistent quality of solutions and uses resources more efficiently. This makes ACO a better choice for scenarios where resource constraints are significant, whereas GA might be preferred in situations where speed is critical, and resource availability is not a concern.

The choice between GA and ACO should be guided by the specific requirements of the task, particularly considering the trade-offs between speed, resource usage, and consistency in achieving high-quality solutions. ACO's resource efficiency and consistent performance make it advantageous for complex and memory-sensitive tasks, while GA's speed could be beneficial where quick solutions are prioritized.

Conclusion

The comparative analysis of Genetic Algorithms (GA) and Ant Colony Optimization (ACO) across multiple runs highlights distinct characteristics and suitability for optimization tasks within different constraints and requirements.

The GA is notably faster in achieving results, making it advantageous in scenarios where time efficiency is critical. However, this speed comes at the cost of higher memory usage, which might be a limiting factor in environments with constrained computational resources. Although GA generally achieves high fitness scores, its performance shows variability, and there are

instances where it struggles to find optimal solutions efficiently. This suggests a potential need for parameter tuning or enhancements in its approach to improve consistency and resource management.

ACO, on the other hand, demonstrates significant memory efficiency, maintaining low memory usage across all runs. It tends to be slower in some instances but consistently achieves high or near-optimal fitness scores, indicating robustness in solution quality. The efficiency and consistency of ACO make it particularly suitable for complex optimization problems where memory usage and solution quality are paramount, even if the computation takes slightly longer.

Choosing between GA and ACO should consider the specific demands of the problem at hand. For tasks where quick solutions are paramount and computational resources are abundant, GA could be the preferred choice. In contrast, ACO is better suited for scenarios requiring efficient memory usage and consistent high-quality solutions, particularly when dealing with complex and resource-sensitive environments.

Ultimately, both algorithms have their merits and can be effectively utilized in different contexts depending on the optimization needs. Future research and application should focus on leveraging the strengths of each algorithm, potentially integrating hybrid approaches or further refining their parameters to enhance their performance and adaptability to various optimization challenges.

References

Alam, T., Qamar, S., Dixit, A., & Benaida, M. (2020). *Genetic Algorithm: Reviews, Implementations, and Applications*. *12*, 57–77. https://doi.org/10.3991/ijep.v10i6.14567

Albadr, M., Tiun, S., Ayob, M., & Al-Dhief, F. (2020). Genetic Algorithm Based on Natural Selection Theory for Optimization Problems. *Symmetry*, *12*, 1–31. https://doi.org/10.3390/sym12111758

Almulla, H., & Gay, G. (2022). Learning how to search: generating effective test cases through adaptive fitness function selection. *Empirical Software Engineering*, *27*(2), 38. https://doi.org/10.1007/s10664-021-10048-8

Ardiyani, L. P. S. (2022). Perbandingan Algoritma Genetika dengan Algoritma Steepest Ascent Hill Climbing untuk Optimasi Penjadwalan Kuliah. *Jurnal Nasional Pendidikan Teknik Informatika (JANAPATI)*, *11*(1), 63. https://doi.org/10.23887/janapati.v11i1.43172

Azevedo, B. F., Rocha, A. M. A. C., & Pereira, A. I. (2024). Hybrid approaches to optimization and machine learning methods: a systematic literature review. *Machine Learning*, *113*(7), 4055–4097. https://doi.org/10.1007/s10994-023-06467-x

Dorigo, M., & Stützle, T. (2019). Ant Colony Optimization: Overview and Recent Advances. In M. Gendreau & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (pp. 311–351). Springer International Publishing. https://doi.org/10.1007/978-3-319-91086-4_10

Gen, M., & Lin, L. (2023). Genetic Algorithms and Their Applications. In H. Pham (Ed.), *Springer Handbook of Engineering Statistics* (pp. 635–674). Springer London. https://doi.org/10.1007/978-1-4471-7503-2_33

Han, S., & Xiao, L. (2022). An improved adaptive genetic algorithm. *SHS Web of Conferences, 140*, 01044. https://doi.org/10.1051/shsconf/202214001044

Hussain, A., & Muhammad, Y. S. (2020). Trade-off between exploration and exploitation with genetic algorithm using a novel selection operator. *Complex & Intelligent Systems, 6*(1), 1–14. https://doi.org/10.1007/s40747-019-0102-7

Karimi-Mamaghan, M., Mohammadi, M., Meyer, P., Karimi-Mamaghan, A. M., & Talbi, E.-G. (2022). Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *European Journal of Operational Research, 296*(2), 393–422. https://doi.org/https://doi.org/10.1016/j.ejor.2021.04.032

Laswi, A. (2020). Perbandingan Algoritma Fitness of Spring dan Algoritma Tabu Search pada Kasus Penjadwalan Perkuliahan. *ILKOM Jurnal Ilmiah*, *12*, 39–46. https://doi.org/10.33096/ilkom.v12i1.522.39-46

Liu, C., Wu, L., Xiao, W., Li, G., Xu, D., Guo, J., & Li, W. (2023). An improved heuristic mechanism ant colony optimization algorithm for solving path planning. *Knowledge-Based Systems, 271*, 110540. https://doi.org/https://doi.org/10.1016/j.knosys.2023.110540

Maghawry, A., Hodhod, R., Omar, Y., & Kholief, M. (2021). An approach for optimizing multi-objective problems using hybrid genetic algorithms. *Soft Computing, 25*(1), 389–405. https://doi.org/10.1007/s00500-020-05149-3

Merdanoğlu, H., Yakıcı, E., Doğan, O. T., Duran, S., & Karatas, M. (2020). Finding optimal schedules in a home energy management system. *Electric Power Systems Research, 182*, 106229. https://doi.org/https://doi.org/10.1016/j.epsr.2020.106229

Oktaviani, Y. C., & Riti, Y. F. (2023). Perbandingan Algoritma Welch Powell dan Algoritma Greedy dalam Optimasi Penjadwalan Ruang Kuliah Semester Genap Fakultas Teknik. *J I M P - Jurnal Informatika Merdeka Pasuruan*, *7*(3), 87. https://doi.org/10.51213/jimp.v7i3.560

Savitri, N. A., Pujawan, N., & Santosa, B. (2020). Resource-constrained project scheduling with ant colony optimization algorithm. In *JOURNAL OF CIVIL ENGINEERING* (Vol. 35, Issue 2). http://dx.doi.org/10.12962/j20861206.v35i2.8115

Zainuddin, F., & Abd Samad, M. F. (2020). *A Review of Crossover Methods and Problem Representation of Genetic Algorithm in Recent Engineering Applications*. *International Journal of Advanced Science and Technology*, *29*(6s),759–769. http://sersc.org/journals/index.php/IJAST/article/view/8903

Zhou, X., Ma, H., Gu, J., Chen, H., & Deng, W. (2022). Parameter adaptation-based ant colony optimization with dynamic hybrid mechanism. *Engineering Applications of Artificial Intelligence*, *114*, 105139. https://doi.org/https://doi.org/10.1016/j.engappai.2022.105139