KOMPUTASI: JURNAL ILMIAH ILMU KOMPUTER DAN MATEMATIKA
VOL. 23 (1) (2026), 100-107 p-ISSN: 693-7554, e-ISSN:2654-3990

A Metaheuristic Hybrid Approach for University Timetabling: Genetic Algorithm and Simulated Annealing

Septian Cahyadi[1], **Thesya Marcella**[2]

[1,2] Information Technology, Informatics and Tourism, Institute Business and Informatic Kesatuan Bogor, West Java, 16123, Indonesia

Abstract

This study addresses the recurrent course scheduling problem in universities. The problem involves constructing an optimal timetable by allocating courses, lecturers, and student groups to rooms and time slots while satisfying mandatory hard constraints and improving quality through soft constraints. Given the scale nine study programs, 148 courses, 123 classrooms, 82 class groups, and 147 active lecturers the problem exhibits combinatorial complexity. We propose a hybrid metaheuristic that integrates Genetic Algorithm (GA) and Simulated Annealing (SA) to balance global exploration and local exploitation. GA is selected for its robust exploration of large solution spaces and its proven applicability to university timetabling, while SA offers principled local refinement guided by an annealing schedule to reduce constraint violations. Prior work indicates that GA–SA hybrids can improve convergence and reduce computation time relative to standalone GA. We formalize the constraints, define a fitness function that prioritizes feasibility, and design neighbourhood operators tailored to timetabling moves. The proposed approach aims to deliver a robust timetable that satisfies institutional requirements and enhances operational efficiency.

Keywords: *Timetabling ,Course Schedulling, Hybrid Algorithm, Genetic Algorithm, Simulated Annealing*

1. Introduction

This research focuses on the issue of Course Scheduling, using IBI Kesatuan as the study object, where problems frequently arise at the beginning of each semester in a new academic year. The scheduling problem involves creating an optimal timetable by allocating courses, lecturers, and students into available rooms and time slots while adhering to various constraints. These constraints are divided into two categories: hard constraints and soft constraints. Hard constraints are rules that must be satisfied under any circumstances to produce a feasible schedule. Examples of hard constraints include ensuring that no student is scheduled for two courses at the same time, that no two courses are assigned to the same room at the same time, and that room capacity is sufficient to accommodate the number of students enrolled in the course. Meanwhile, soft constraints are rules expected to be fulfilled to improve schedule quality, such as meeting diverse lecturer time preferences.

At IBI Kesatuan, the challenges include scheduling for 9 study programs, 148 courses, 123 classrooms, 82 class groups, and 147 active lecturers. To handle this complexity, this research employs an optimization approach by implementing a hybrid algorithm that combines the Genetic Algorithm (GA) and Simulated Annealing (SA) techniques. The Genetic Algorithm is a popular solution for university scheduling problems because it can search broadly. This method is designed to produce schedules that not only meet all hard constraints but also comply with as many soft constraints as possible, resulting in an optimal timetable [1][2]. On the other hand, Simulated Annealing (SA) can be used as an approach to solve combinatorial optimization problems. By using a higher initial temperature and performing more iterations, SA reduces constraint violations. The higher the initial temperature, the fewer violations occur. Similarly, more iterations reduce the number of violations [3][11][12]. The combination of these two methods has also been applied in other studies and found that GA-SA can perform better than regular GA in optimization problems, showing the

**Corresponding author*. *E-mail address:* scahyadi@ibik.ac.id

Received: 30 December 2025, Accepted: 30 January 2026 and available online 31 January 2026
DOI: https://doi.org/10.33751/komputasi.v19i2.5260

ability of hybridization mechanisms to improve convergence. GA-SA reduces the number of iterations required for convergence from 15 seconds to 11 seconds [4]. Based on the challenges faced in the case study at the Institute of Business and Informatics Kesatuan, this research aims to contribute to solving these problems by implementing a combined approach of Genetic Algorithm and Simulated Annealing to optimize the efficiency of course scheduling at the Institute of Business and Informatics Kesatuan.

2. Methods

This study adopts a **research and development (R&D)** methodology combined with an **experimental quantitative approach** to optimize course scheduling at the Institute of Business and Informatics Kesatuan (IBIK). The research stages include **data collection, system design, hybrid algorithm implementation, testing, and performance evaluation** (Figure 1).

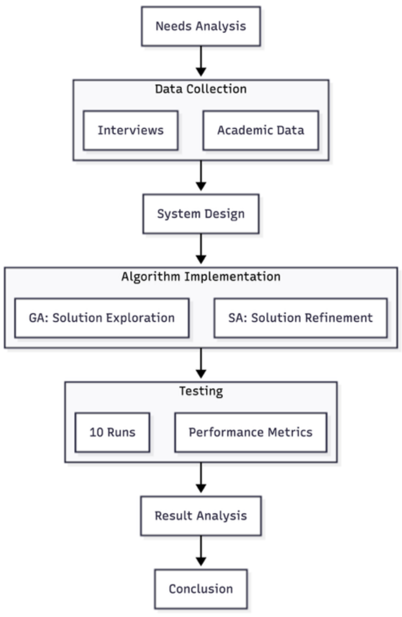


Figure 1. Research Diagram

2.1. Data Types and Sources

A dual-data approach was employed, integrating **primary and secondary sources** to comprehensively map system requirements.

- **Primary data** were obtained through in-depth interviews with IBI Kesatuan academic staff to identify constraints, operational challenges, and expectations for the new system, complemented by direct observation of the manual scheduling process.
- **Secondary data** were collected through document analysis and extraction from the IBIK Academic System database, encompassing historical records and active semester datasets. These datasets include **148 courses, 123 classrooms, 147 lecturers, and 82 class groups**.

All data were consolidated during the system design phase by identifying relevant entities and attributes—such as time slots, rooms, and lecturer availability—to construct an accurate scheduling model.

Tabel 1. Entity List

Entity	Attributes
Dosen	ID Dosen, Nama, Kualifikasi, Preferensi Jadwal
Mata Kuliah	Kode Mata Kuliah, Nama, SKS
Ruangan	Kode Ruangan, Nama, Kapasitas
Kelas	Kode Kelas, Mata Kuliah, Dosen, Ruangan, Jadwal
Preferensi Dosen	ID Dosen, Waktu Tersedia, Prioritas
Mengajar	ID, ID kelas, ID Dosen, ID Mata Kuliah, ID Semester
Slot	ID, start_time, end_time, is_lab_slot, day
Semester	ID, jenis, tahun_ajaran, tanggal_mulai, tanggal_berakhir

2.2. Hybrid Algorithm

The **hybrid GA–SA algorithm** developed in this study is designed to address the course scheduling optimization problem. It combines the **exploratory power of Genetic Algorithm (GA)** with the **exploitative capability of Simulated Annealing (SA)**. Figure 2 illustrates the workflow of the hybrid process.

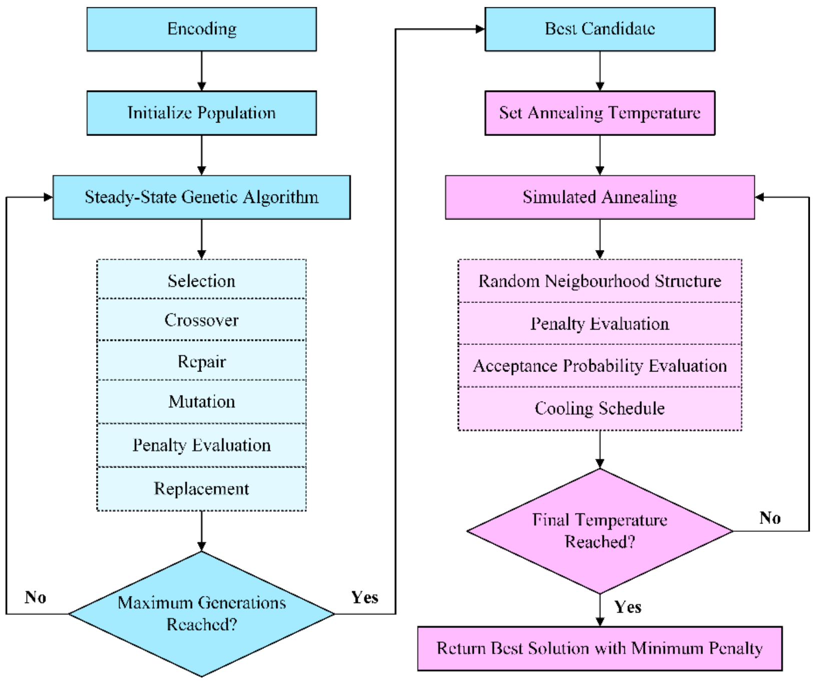

Encoding
Initialize Population
Steady-State Genetic Algorithm
Selection
Crossover
Repair
Mutation
Penalty Evaluation
Replacement
Maximum Generations Reached?
No
Yes
Best Candidate
Set Annealing Temperature
Simulated Annealing
Random Neigbourhood Structure
Penalty Evaluation
Acceptance Probability Evaluation
Cooling Schedule
Final Temperature Reached?
No
Yes
Return Best Solution with Minimum Penalty

Figure 2. Workflow of the hybrid process

The algorithm's performance is highly influenced by parameter selection, which governs the search behavior. **Table 2** details the parameters used for GA and SA in this research.

Table 2. Algorithma parameter

Parameter	GA Value	SA Value
Population Size	50	-
Generations	100	-
Crossover Rate	0.7	-
Mutation Rate	0.01	-
Initial Temperature	-	1000
Suhu Akhir	-	100
Final Temperature	-	0.95

These parameters require careful tuning to achieve an optimal balance. For instance, a population size that is too small limits solution diversity, while an excessively large size significantly increases computation time. Similarly, SA parameters such as initial temperature and cooling rate must be calibrated to avoid premature convergence to local optima. Parameter adjustment is essential for improving both solution quality and computational efficiency, and parameter evaluation helps determine the best configuration for the algorithm.

2.3. Constraints in Course Scheduling

Beyond parameter tuning, the algorithm must account for various constraints during the scheduling process. These constraints fall into two categories: **hard constraints** and **soft constraints**.

Hard constraints are mandatory conditions that must be satisfied for the schedule to be considered valid. Violation of any hard constraint renders the schedule infeasible and unacceptable. Therefore, the scheduling system must ensure that all hard constraints are met before generating a complete timetable. The hard constraints in course scheduling include:

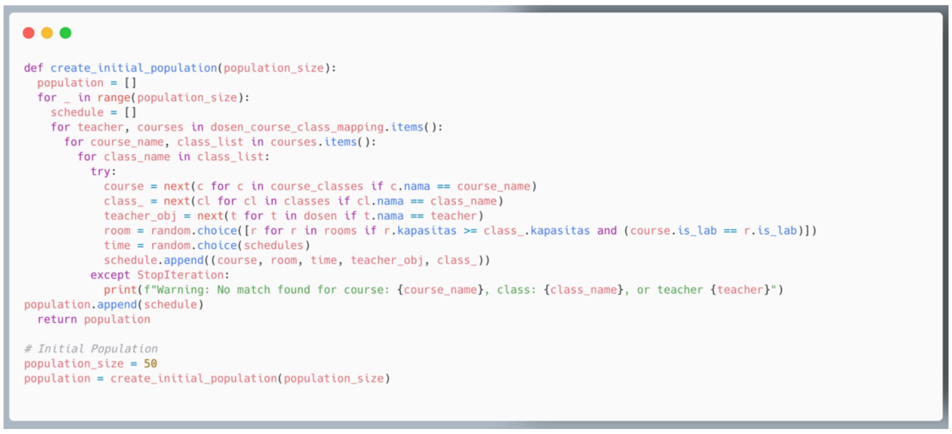
1. **Room Capacity:** The number of students in a class must not exceed the available room capacity.
2. **Laboratory Classes:** Courses requiring laboratory facilities must be scheduled in appropriate lab rooms.
3. **Class Shifts:** The same class cannot be scheduled in different shifts simultaneously.
4. **Time Conflicts (General):** Lecturers and students must not be scheduled to be in two different places at the same time.
5. **Room Schedule Conflicts:** Two or more classes cannot be scheduled in the same room at the same time.
6. **Class Schedule Conflicts:** Each class must have a unique schedule that does not overlap with other classes.
7. **Duplicate Scheduling:** A lecturer or student must not have overlapping schedules in different locations at the same time.

In contrast, **soft constraints** are desirable conditions that improve schedule quality but are not mandatory. Violating soft constraints does not invalidate the schedule; however, it may reduce its quality or fail to meet user preferences. Consequently, the scheduling system aims to maximize the fulfillment of soft constraints to enhance stakeholder satisfaction.

In this system, soft constraints primarily involve **lecturer time preferences**. Lecturers may prefer specific teaching times (e.g., morning sessions or avoiding certain hours). While these preferences are not compulsory, the more preferences are satisfied, the higher the overall quality of the schedule.

2.4. Chromosome Representation

In the Genetic Algorithm framework, the schedule is represented as a **chromosome** [5], which is implemented as a flexible list structure. Each chromosome corresponds to a single scheduling solution, where each element in the list is a **tuple** representing a specific assignment.

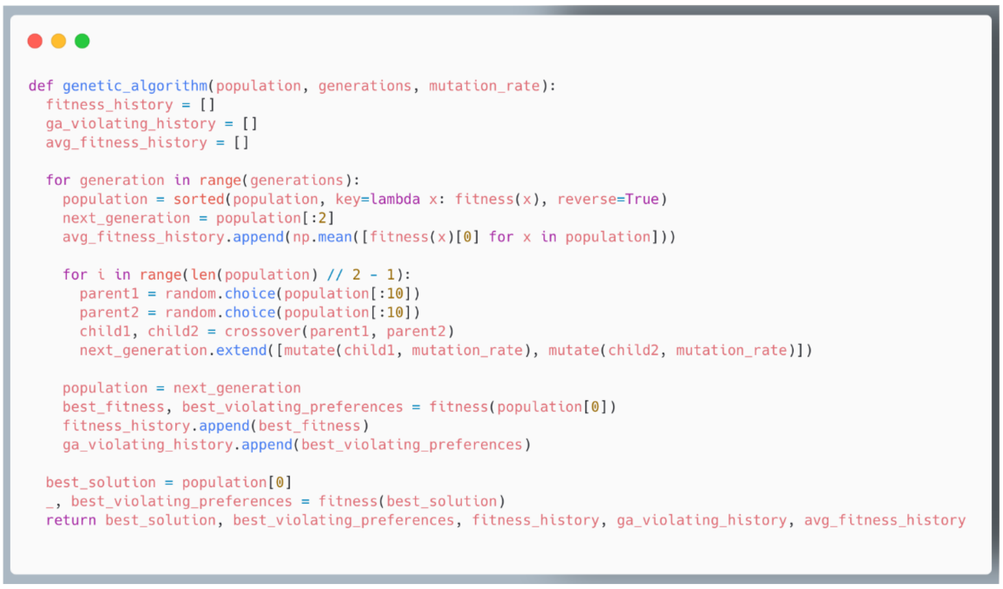


```python
def create_initial_population(population_size):
  population = []
  for _ in range(population_size):
    schedule = []
    for teacher, courses in dosen_course_class_mapping.items():
      for course_name, class_list in courses.items():
        for class_name in class_list:
          try:
            course = next(c for c in course_classes if c.nama == course_name)
            class_ = next(cl for cl in classes if cl.nama == class_name)
            teacher_obj = next(t for t in dosen if t.nama == teacher)
            room = random.choice([r for r in rooms if r.kapasitas >= class_.kapasitas and (course.is_lab == r.is_lab)])
            time = random.choice(schedules)
            schedule.append((course, room, time, teacher_obj, class_))
          except StopIteration:
            print(f"Warning: No match found for course: {course_name}, class: {class_name}, or teacher {teacher}")
population.append(schedule)
  return population

# Initial Population
population_size = 50
population = create_initial_population(population_size)
```

Figure 3. Population initiation code

For example, a tuple might take the following form: (Course1, Room101, 08:00–10:00, LecturerA, ClassX). This indicates that *Course1* is scheduled in *Room101* from 08:00 to 10:00, taught by *LecturerA* for *ClassX*. The flexibility of this representation allows the chromosome to store and manage complex scheduling information efficiently, while also facilitating genetic operations such as **crossover** and **mutation**. This design ensures that the algorithm can manipulate solutions effectively during the optimization process.



```
def genetic_algorithm(population, generations, mutation_rate):
  fitness_history = []
  ga_violating_history = []
  avg_fitness_history = []

  for generation in range(generations):
    population = sorted(population, key=lambda x: fitness(x), reverse=True)
    next_generation = population[:2]
    avg_fitness_history.append(np.mean([fitness(x)[0] for x in population]))

    for i in range(len(population) // 2 - 1):
      parent1 = random.choice(population[:10])
      parent2 = random.choice(population[:10])
      child1, child2 = crossover(parent1, parent2)
      next_generation.extend([mutate(child1, mutation_rate), mutate(child2, mutation_rate)])

    population = next_generation
    best_fitness, best_violating_preferences = fitness(population[0])
    fitness_history.append(best_fitness)
    ga_violating_history.append(best_violating_preferences)

  best_solution = population[0]
  _, best_violating_preferences = fitness(best_solution)
  return best_solution, best_violating_preferences, fitness_history, ga_violating_history, avg_fitness_history
```

Figure 4. *Genetic algorithm* code

The **mutation operator** is applied by randomly altering one element within a chromosome to ensure that diversity is maintained in the population. The process mutate(child, mutation_rate) introduces variation by modifying certain genes in the offspring based on the specified mutation rate.

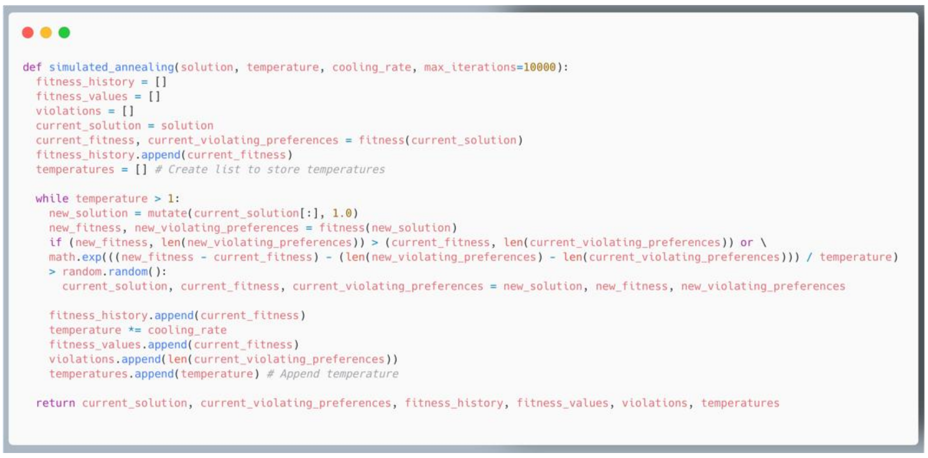
$$New\ Gen = Old\ Gen + Random\ Mutation \quad (3)$$

Mutation can randomly change the time or room assigned to a particular lecture session, thereby preventing the algorithm from becoming trapped in suboptimal solutions. This mechanism introduces small but significant differences within the population, which can pave the way toward better solutions.

Subsequently, a new generation is formed from the best individuals of the previous generation (through **elitism**) and offspring produced via **crossover** and **mutation**. Elitism ensures that the best solutions from the previous generation are always preserved.

2.5. Simulated Annealing

After the Genetic Algorithm (GA) explores the solution space, **Simulated Annealing (SA)** is employed to further refine the existing solution [8][9]. SA is inspired by the metal cooling process, where material is cooled gradually to achieve a more stable structure. Figure 4 illustrates the code implementation for simulated annealing.



```
def simulated_annealing(solution, temperature, cooling_rate, max_iterations=10000):
  fitness_history = []
  fitness_values = []
  violations = []
  current_solution = solution
  current_fitness, current_violating_preferences = fitness(current_solution)
  fitness_history.append(current_fitness)
  temperatures = [] # Create list to store temperatures

  while temperature > 1:
    new_solution = mutate(current_solution[:], 1.0)
    new_fitness, new_violating_preferences = fitness(new_solution)
    if (new_fitness, len(new_violating_preferences)) > (current_fitness, len(current_violating_preferences)) or \
    math.exp(((new_fitness - current_fitness) - (len(new_violating_preferences) - len(current_violating_preferences))) / temperature)
    > random.random():
      current_solution, current_fitness, current_violating_preferences = new_solution, new_fitness, new_violating_preferences

    fitness_history.append(current_fitness)
    temperature *= cooling_rate
    fitness_values.append(current_fitness)
    violations.append(len(current_violating_preferences))
    temperatures.append(temperature) # Append temperature

  return current_solution, current_violating_preferences, fitness_history, fitness_values, violations, temperatures
```

Figure 4. Simulated Annealing Code

At the start of the SA process, the temperature is set high to allow acceptance of solutions that may be worse than the current one. This mechanism enables the algorithm to explore the solution space more broadly and avoid becoming trapped in local optima. If the new solution has a better fitness value (lower in minimization problems) than the current solution, it is immediately accepted. However, if the new solution is worse, the **Metropolis Criterion** [10] is applied, allowing acceptance with a certain probability:

$$P = \exp\left(\frac{\Delta E}{T}\right) \quad (4)$$

where **ΔE** represents the change in fitness value and the number of violated preferences, and **T** denotes the current temperature. If Pexceeds a randomly generated number, the new solution is accepted.

The cooling rate determines how quickly the system temperature decreases exponentially, following:

$$T_{\text{new}} = T_{\text{old}} \times cooling\ rate \quad (5)$$

A cooling rate that is too fast may cause the algorithm to become stuck in local optima due to insufficient exploration, while a rate that is too slow can make the optimization process excessively long. Therefore, the cooling rate is calibrated to balance exploration and exploitation. At each iteration, the temperature decreases slightly, and a neighboring solution is generated through minor mutations of the current solution. Acceptance of this new solution depends on the temperature-based probability, enabling the algorithm to "jump" out of suboptimal solutions toward better ones.

In the hybrid approach, the first step is to execute GA to produce a strong initial solution. The best solution from GA, denoted as S_{GA}, is obtained using:

$$S_{GA} = Genetic\ Algorithm(Populasi, Generasi, Crossover\ Rate, Mutation\ Rate)$$

GA performs broad exploration to identify a feasible starting point, which is then optimized further by SA:

$$S_{SA} = Simulated\ Annealing(S_{GA}, T_0, Cooling\ Rate)$$

This process ensures that the final solution not only satisfies all constraints but also approaches global optimality. The hybrid algorithm terminates when predefined stopping conditions—such as a maximum number of iterations or allocated runtime—are met.

3. Result and Discussion

To evaluate the effectiveness of the hybrid algorithm proposed in this study, ten experimental runs were conducted using the same dataset. Each run produced a different schedule, but all were assessed using consistent criteria, including the number of conflicts, compliance with lecturer preferences, and room utilization. These tests generated several key metrics, such as the time required to produce a schedule and the fitness value of each solution.

Table 3. The experimental results

Run	**GA Best *Fitness***	**GA *Time* (s)**	**SA *Best Fitness***	**SA *Time* (s)**
1	-2009	25.08	-2041	1.03
2	-2048	24.23	-2022	1.02
3	-2029	24.28	-2027	1.02
4	-2029	24.29	-2063	1.03
5	-1989	24.19	-2059	1.03
6	-2029	24.33	-2069	1.03
7	-2041	23.99	-2040	1.02
8	-1981	24.06	-2029	1.02
9	-2045	24.08	-2023	1.02
10	-2073	23.53	-2027	1.67

Based on the experimental results presented in **Table 3**, the hybrid GA–SA method demonstrates significant advantages in course scheduling. Although the Genetic Algorithm (GA) alone produced varying fitness scores—with the best value reaching **81.19** and the worst at **79.27**—the average GA fitness score was **79.73**, indicating that GA can generate high-quality solutions. With an average execution time of **24.21 seconds**, GA delivers robust solutions, albeit with longer computation times compared to other methods.

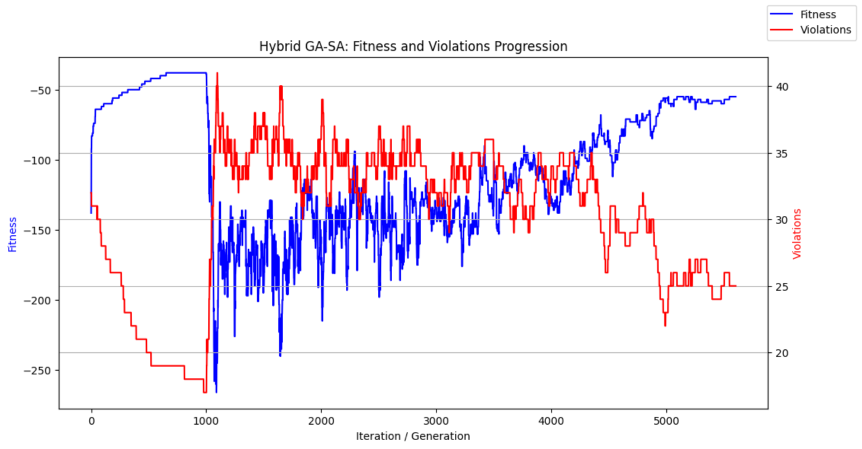


Figure 6. Grafik fitness dan violasi *hybrid* GA-SA

The following graph (**Figure 6**) illustrates the progression of fitness values for the hybrid GA–SA algorithm throughout the iterations. At the beginning of the process, GA significantly improves the fitness score from **–250** to **–50**. After the 1000th iteration, SA takes over and continues refining the solution, reaching approximately **–40** in the final iteration. This combination of GA and SA proves effective in producing high-quality schedules, where GA identifies a strong initial solution and SA enhances it further.

Regarding constraint violations, GA initially exhibits a high number of violations, which decreases substantially to **18** by the 1000th iteration. SA maintains this low violation count, despite minor fluctuations. Overall, the hybrid GA–SA algorithm effectively reduces constraint violations while preserving schedule quality.

The algorithm successfully satisfies all **hard constraints**, such as room capacity, laboratory class allocation, and time conflict resolution, resulting in a total penalty score of zero.

Table 4. Constraint Summary

Constraint	Weight	Violations	Total Penalty
Room Capacity	-20	0	0
Laboratory Class in Lab	-20	0	0
Class Shift	-20	0	0
Time Conflict (General)	-20	0	0
Room Schedule Conflict	-20	0	0
Class Schedule Conflict	-20	0	0
Duplicate Scheduling	-20	0	0
Lecturer Time Preferences	-5	3	-15

As shown in Table 4, some lecturer time preferences were not fully satisfied, primarily due to constraints requiring evening classes to be scheduled after 18:00. Nevertheless, the algorithm minimizes these penalties, ensuring that the resulting schedule remains high-quality and accommodates several lecturer preferences.

4. Conclusion

Based on the findings and implementation of the hybrid Genetic Algorithm–Simulated Annealing approach, the following conclusions can be drawn:

1. The developed hybrid algorithm is highly effective in generating optimal course schedules. The combination of GA and SA enables broad solution exploration while avoiding local optima, resulting in high-quality timetables.
2. Experimental results confirm that the algorithm can produce feasible and efficient schedules within a relatively short time. Compared to conventional scheduling methods, the system achieves minimal conflicts (only **1 conflict across 97 schedules**) and satisfies most lecturer preferences, indicating a significant improvement in schedule quality.
3. The system can generate **97 schedules in just one minute**, demonstrating superior speed compared to traditional methods.

For future research, several aspects warrant consideration. Although the hybrid algorithm has shown promising results, further optimization is possible. Future studies may focus on refining the algorithm to achieve even greater efficiency and integrating **deep learning techniques**, such as neural networks, to predict scheduling patterns. This integration could enhance accuracy and handle complex scenarios. Additionally, scalability testing is essential to ensure the system performs effectively with larger and more complex datasets.

References

[1] Assi, M., Halawi, B., & Haraty, R. A. (2018). Genetic Algorithm Analysis using the Graph Coloring Method for Solving the University Timetable Problem. Procedia Computer Science, 126, 899–906. https://doi.org/10.1016/j.procS.2018.08.024

[2] D.M. Premasiril. (2018). University Timetable Scheduling Using Genetic Algorithm Approach Case Study: Rajarata University OF Sri Lanka. Journal of Engineering Research and Application Www.Ijera.Com, 8(12), 30–35. https://doi.org/10.9790/9622-0812023035

[3] J. Bendi, R. K., & Junaidi, H. (2019). Simulated Annealing Approach for University Timetable Problem. Jurnal Ilmiah Matrik, 21(3), 204–213. https://doi.org/10.33557/jurnalmatrik.v21i3.723

[4] Suanpang, P., Jamjuntr, P., Jermsittiparsert, K., & Kaewyong, P. (2022). Tourism Service Scheduling in Smart City Based on Hybrid Genetic Algorithm Simulated Annealing Algorithm. Sustainability (Switzerland), 14(23). https://doi.org/10.3390/su142316293

[5] Matias, J. B., Fajardo, A. C., & Medina, R. M. (2018). Examining Genetic Algorithm with Guided Search and Self-Adaptive Neighborhood Strategies for Curriculum-Based Course Timetable Problem. Proceedings - 2018 4th International Conference on Advances in Computing, Communication and Automation, ICACCA 2018, 1–6. https://doi.org/10.1109/ICACCAF.2018.8776728

[6] Oswald, C., & Anand Deva Durai, C. (2014). Novel hybrid PSO algorithms with search optimization strategies for a University Course Timetabling Problem. 2013 5th International Conference on Advanced Computing, ICoAC 2013, 77–85. https://doi.org/10.1109/ICoAC.2013.6921931

[7] Swari, M. H. P., Putra, C. A., & Handika, I. P. S. (2022). Analisis Perbandingan Algoritma Genetika dan Modified Improved Particle Swarm Optimization dalam Penjadwalan Mata Kuliah. Jurnal Nasional Pendidikan Teknik Informatika (JANAPATI), 11(2), 92–101. https://doi.org/10.23887/janapati.v11i2.49061

[8] Mauritsius, T., Legowo, N., & Gunawan, F. E. (2018). Reducing the Timeslot Used in Examination Timetable Problem. Proceedings of 2018 International Conference on Information Management and Technology, ICIMTech 2018, September, 211–216. https://doi.org/10.1109/ICIMTech.2018.8528111

[9] Sylejmani, K., Gashi, E., & Ymeri, A. (2022). Simulated annealing with penalization for university course timetabling. Journal of Scheduling, 1–4. https://doi.org/10.1007/s10951-022-00747-5

[10] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equiation of State Calculations by Fast Computing Machines. Journal of Chemical Physics, 21, 1087–1092.

[11] Faroqi, A., & Suryanto, T. L. M. (2020). Pemanfaatan Google Calendar Untuk Pembuatan Kalender Akademik Di Smp Miftahul Ulum Surabaya. Jurnal Layanan Masyarakat (Journal of Public Services), 4(1), 13. https://doi.org/10.20473/jlm.v4i1.2020.13-16

[12] Yekti Narika. (2016). Implementasi Hybrid Algoritma Genetika dan Simulated Annealing untuk Penjadwalan Mata Pelajaran.