ATCSJ
Applied Technology and Computing Science Journal
Open Access **p-ISSN 2621-4458 | e-ISSN 2621-4474**

Available online to journal2.unusa.ac.id
UNUSA
S4-Accredited – SK No.79/M/KPT/2023
Journal Page is available to https://journal2.unusa.ac.id/index.php/ATCSJ

Optimizing Course Scheduling Efficiency through Genetic Algorithms

Nur Shabrina Meutia[1*], Rizqi Putri Nourma Budiarti [2]

[1,2] *Universitas Nahdlatul Ulama Surabaya, Indonesia*
Jl. Raya Jemursari No.57, Surabaya
[1*]shabrinameutia@unusa.ac.id, [2] rizqi.putri.nb@unusa.ac.id

Article history:

Received 20 July 2025
Revised 10 August 2025
Accepted 28 August 2025
Available online 9 Sept 2025

Keywords:

Course Schedule,
Genetic Algorithm,
Chromosome,
Fitness Function,
Optimation.

Abstract

Course scheduling is an important aspect of educational administration in academic institutions. An effective procedure enhances students' educational experience, maximizes resource utilization, and lowers operational expenses. However, course scheduling often faces various constraints and complexities, such as limited space, time and human resources. Therefore, an effective and efficient approach is needed to solve the course scheduling problem. This study implements the genetic Algorithm to solve the problem of optimization course scheduling. This study intends to develop a course scheduling application using genetic algorithm to enhance the effectiveness of course scheduling in educational institutions. There are 8 genetic algorithm procedures for solving problems in this research; Encoding techniques, initial population, fitness function, selection, crossover, mutation, elitism and the condition of iteration is complete when the maximum has been reached, and the fitness value is 1. The best result from 25 iteration and 15 population found at probability of crossover is 0,5 and mutation rate is 10%. The lowest fitness value is 0,09 with the fastest execution time, that is 395 seconds for subjects in odd semester and 563 seconds for subjects in even semester.

I. INTRODUCTION

Course scheduling is an important aspect of educational administration in academic institutions. An effective procedure enhances students' educational experience, maximizes resource utilization, and lowers operational expenses [1]. However, course scheduling often faces various constraints and complexities, such as limited space, time and human resources. Course scheduling at the university faces complex types of scheduling problems [2]. Currently, the scheduling process within the Informatics Engineering Study Program of University X is still carried out manually. Manual scheduling is prone to various challenges and inefficiencies [3]. Some of the problems associated with manual scheduling in the Informatics Engineering Study Program include overlapping course schedules due to a limited number of available classrooms [4]. The department must meet several conditions in carrying out the scheduling process. First, early-semester students attend morning classes, while senior students are allocated to afternoon sessions. The second

[1*] Corresponding author

condition is that Courses with a credit load of three semester credit units (3 SCU) are expected to be scheduled in the morning. The distribution of class schedules must also be balanced to prevent an excessive concentration of lectures on a single day [5]. Moreover, class schedules need to be synchronized with lecturers' availability to avoid conflicts, such as a lecturer being assigned to teach different courses simultaneously. As a result of these recurring issues, schedule revisions are frequently necessary [6, 7].

To address these issues, the author proposes a scheduling approach using the Genetic Algorithm method. The Genetic Algorithm is a heuristic technique developed based on the principles of genetics and the natural selection process from Darwin's theory of evolution. Genetic Algorithms process a group of potential solutions simultaneously, rather than focusing on just one [8]. Furthermore, the Genetic Algorithm offers significant flexibility, allowing it to be combined with domain-specific heuristic methods to achieve more efficient implementations in specialized problem contexts [9]. This algorithm is particularly well-suited for solving complex problems that are difficult to resolve using conventional methods. While the solutions produced by Genetic Algorithms are not guaranteed to be optimal, they are typically "good enough" and acceptable within practical constraints.

Based on the above explanation, this research aims to answer the following question: 1) How to implement genetic algorithms method for automated scheduling in the Informatics Engineering Study Program at the University X. 2) How capable is the Genetic Algorithm approach in optimizing solutions for scheduling. The scheduling process aims to produce optimal and efficient solutions that reduce conflicts between room usage and class times, thereby avoiding the need to completely reorganize the existing schedule.

II. METHODS

The process of course scheduling using a Genetic Algorithm involves several essential stages. These stages are systematically illustrated in the flowchart presented below.

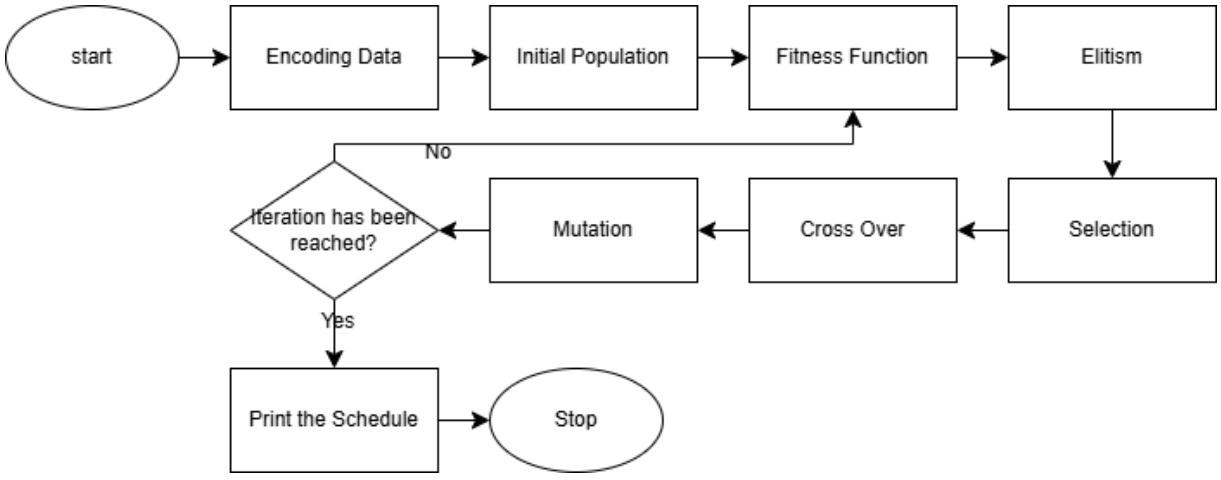

Figure 1. Essential Stages of GA

The description of Figure 1 is provided as follows:

A. Encoding Data

Encoding refers to the process of representing chromosomes as potential solutions to a given problem. In this study, the chromosomes are encoded using strings or varchars.

B. Initial Population

Population initialization involves defining how genes and chromosomes represent the population. The generation of the initial population entails creating a number of individuals or chromosomes, either randomly or through a predetermined method [10].

In this scheduling study utilizing a genetic algorithm, the solution generated corresponds to a class timetable. An example of population initialization is provided using data from Table I (course information), Table II (classroom data), Table III (schedule information) and Table IV (Group Class).

Table 1. course information

No	kode_mk	nama_mk	kode_dosen	sks	smt	kelas
1	M1	Kecerdasan Buatan	D1	3	5	A
2	M2	Metode Numerik	D2	3	3	A
3	M3	Sistem Digital	D2	3	1	A
4	M4	Algoritma dan Pemrograman	D3	3	1	A

Table 2. classroom data

No	id_jadwal	hari	waktu
1	J1	Senin	7.30 – 10.10
2	J2	Senin	10.20 – 12.50
3	J3	Selasa	7.30 – 10.10
4	J4	Selasa	10.20 – 12.50

Table 3. schedule information

No	Id_ruang	Nama_ruang	kapasitas
1	R1	Komputasi Dasar	80
2	R2	C2-04	40

Table 4. Group Class

No	Id_kelas	Nama_kelas
1	K1	Kelas A
2	K2	Kelas B

Based on this data, a population of four chromosomes, each containing four genes, is encoded to match the available courses and student groups. Each gene encodes a sequence representing the course, group, classroom, and time slot. The population is generated randomly, as illustrated in Table V.

Table 5. Random Population

	Gen 1	Gen 2	Gen 3	Gen 4
Kromosom1	M1K1R1J2	M3**K1R2J4**	M2K1R1J3	M4**K1R2J4**
Kromosom2	M2K1R1J3	M4**K1**R2**J2**	M3**K1**R2**J2**	M1K1R1J4
Kromosom3	**M3**K1R2**J3**	M4K1R2J2	M1K1R1J4	**M2**K1R1**J3**
Kromosom4	M2K1R2J4	M3K1**R2J2**	M4K1R1J3	M1K1**R2J2**

C. Fitness Function

The fitness value reflects the effectiveness or quality of a chromosome (individual) as a solution. It serves as a benchmark for identifying optimal solutions in the genetic algorithm [11]. Violations arise when conflicts occur between groups, lecturers, classrooms, or courses. In course scheduling, a solution is considered better when it results in fewer violations.

$$F = \frac{1}{1+(\sum C1 + \sum C2 + \sum C3 + \sum C4 + \sum C5)} \quad (1)$$

Where:
F = fitness value
C1 = Number of room conflicts
C2 = Number of class conflicts
C3 = Number of lecturers and course conflicts

C4 = 3-credit courses not scheduled in the morning
C5 = First-semester students not scheduled in the morning

The fitness values for the data from Table V are as follows:

- **F Chromosome 1** = $\frac{1}{1+(1+1+0+0+0)}$ = 0,33
- **F Chromosome 2** = $\frac{1}{1+(1+1+0+0+0)}$ = 0,33
- **F Chromosome 3** = $\frac{1}{1+(0+0+1+0+0)}$ = 0,5
- **F Chromosome 4** = $\frac{1}{1+(1+0+0+0+0)}$ = 0,5

D. Selection

Selection is the process of choosing which chromosomes will persist in the population. The formation of the next generation of chromosomes is carried out using the roulette-wheel selection method. In this approach, each chromosome occupies a segment of the wheel proportional to its fitness score. Chromosomes with higher fitness values have a greater likelihood of being selected, while those with lower values have a reduced chance. The selection procedure using the roulette-wheel method involves the following steps:

1. Calculating the fitness value for each chromosome (see Table VI: Chromosome Fitness Values).

Table 6. Chromosome Fitness

Kromosom	Nilai Fitness
3	0,5
4	0,5
1	0,3
2	0,3
Total	**1,66**

2. Computing the selection probability for each chromosome (see Table VII: Chromosome Probability Values).

Table 7. Chromosome Probability Values

Kromosom	Nilai Fitness
3	0,5/1,66 = 0,301
4	0,5/1,66 = 0,301
1	0,33/1,66 = 0,199
2	0,33/1,66 = 0,199
Total	**1**

3. Defining an interval within the range [0–1] for each chromosome based on its probability value (see Table VIII: Chromosome Probability Values).

Table 8. Chromosome Intervals

Kromosom	Nilai Fitness
3	0 – 0,301
4	0,302 – 0,602
1	0,602 – 0,801
2	0,802 - 1

Suppose the newly generated population consists of random values [0.4; 0.2; 0.75; and 0.85]. Then, the arrangement of chromosomes in the new population can be seen in Table VIIII.

Table 9. Random Population Resulting from Selection

	Gen 1	Gen 2	Gen 3	Gen 4
Kromosom1	M2K1R2J4	M3K1**R2J2**	M4K1R1J3	M1K1**R2J2**
Kromosom2	**M3**K1R2**J3**	M4K1R2J2	M1K1R1J4	**M2**K1R1**J3**
Kromosom3	M1K1R1J2	M3**K1R2J4**	M2K1R1J3	M4**K1R2J4**
Kromosom4	M2K1R1J3	M4**K1**R2**J2**	M3**K1**R2**J2**	M1K1R1J4

E. Cross Over

Crossover is an operator in genetic algorithms used to generate new chromosomes that inherit characteristics from their parents, similar to the natural process of reproduction [12]. The method used is the N-point crossover method. Crossover is performed only if the randomly generated chromosome has a probability value less than the predefined crossover probability (Pc). In this case, Pc is set to 0.5.

From the selection process described in section 3.6.4, it can be observed that Chromosome 3 and Chromosome 4 have probability values less than the crossover probability (rnd < Pc).

Chromosome 3:
M3K1R2J3 M4K1R2J2 | M1K1R1J4 **M2**K1R1**J3**
Chromosome 4:
M2K1R2J4 M3K1**R2J2** | M4K1R1J3 M1K1**R2J2**

The result of the crossover between the two chromosomes is:

Chromosome 3:
M3K1R2J3 M4K1**R2J2** **M4K1**R1J3 M1K1**R2J2**
Chromosome 4:
M2K1R2J4 M3K1R2J2 M1K1R1J4 M2K1R1J3

F. Mutation

Another way to generate new individuals is through mutation. The probability of gene mutation occurring in a chromosome is very small. Therefore, in the implementation of genetic algorithms, the mutation probability is usually set to a low value—typically less than ½ (mutation rate) The mutation probability is generally set in the range [0–1]. In this case, the mutation probability (Mr) is set to 0.1. Assume that a set of random numbers between [0–1] is generated. For example, the population is represented by the values [0.01; 0.25; 0.6; and 0.4]. Table X shows a comparison between the mutation values and the mutation rate [13].

Table 10. Comparison of Mutation Value

Mutation Value (Random)	***Mutation Rate***	**R < Mr**
0,01	0,1	Yes
0,25	0,1	No
0,6	0,1	No
0,4	0,1	No

From Table IX, only the population with a random value of 0.01 will undergo mutation. The next step is to randomly select the gene locations to be mutated. Assume that genes 2 and 3 are randomly selected and will be modified.

Chromosome before mutation:
M1K1R1J1 M3**K1**R2**J4** M2K1R2J3 **M4**K1R1**J4**
Chromosome after mutation:
M1K1R1J1 M3K1R2J2 M2K1R2J1 M4K1R1J4

G. Termination

Termination is a condition that must be met to end an iteration. In this study, the condition is considered fulfilled when the maximum number of iterations has been reached and the fitness value equals 1.

H. Elitism

Elitism is the process of preserving individuals with high fitness values to ensure they are not lost during the evolution process. The process is considered complete when either the optimal solution is found or the maximum number of iterations is reached. The resulting chromosome to be used is the one with the highest rank and the best fitness value.

III. RESULTS AND DISCUSSIONS

Testing was conducted using a case study method. Three test cases were used. The first case involves even semester courses for the academic year 2015/2016, with a total of 5 rooms. The second case involves odd semester courses for the academic year 2016/2017, with 4 rooms. The third case tests the impact of varying crossover and mutation probability values. The final test evaluates the application's scheduling computation time.

A. Case Study: Even Semester Courses of Academic Year 2015/2016

In the first case, there are 5 rooms, which means there are 80 available course slots. These slots are divided into two types: 50 slots for 3-credit courses and 30 slots for 2-credit courses. In the even semester, there are 22 courses with 3 credits, 23 courses with 2 credits, and 35 empty course slots. The following section presents the schedule report created manually, and the schedule report generated by the system. Figure 2 shows the schedule produced by the system.

Even Semester Course Schedule

hari	jam	R1	R2	R3	R4	R5
Senin	07.30 - 10.00	IB1209/B/DL	IC2227/B/AHJ	IC3238/B/AZ	IC1210/A/IBK	-/D/
Senin	10.10 - 12.40	IC3238/A/AZ	-/A/	IC2224/B/RA	IC1215/B/SEA	IC3236/B/WAA
Senin	13.30 - 15.10	-/A/	-/E/	-/J/	-/S/	-/P/
Selasa	07.30 - 10.00	IC2226/B/AYH	-/E/	IB1212/B/RA	IC1214/A/NAD	IC2223/A/SEA
Selasa	10.10 - 12.40	IC3237/A/DL	IC3132/A/IGP	IC2223/B/SEA	-/G/	IB1212/A/RA
Selasa	13.30 - 15.10	-/V/	IC3239/I/MAA	ID3067/D/IBK	-/I/	ID3063/F/FB
Rabu	07.30 - 10.00	IC2228/A/AHJ	-/O/	IA1208/A/DL	IE1213/B/AYH	IC3240/B/FB
Rabu	10.10 - 12.40	-/C/	-/B/	IE1213/A/AYH	IC2227/A/AHJ	IC2228/B/RA
Rabu	13.30 - 15.10	IC3239/A/MAA	-/F/	ID3061/H/FB	-/T/	IC3237/B/DL
Kamis	07.30 - 10.00	-/J/	-/H/	IC1215/A/SEA	IC2225/A/MAA	-/K/
Kamis	10.10 - 12.40	IB1209/A/DL	-/P/	IC3236/A/WAA	IC1210/B/IBK	-/L/
Kamis	13.30 - 15.10	-/U/	-/B/	ID5069/G/IGP	-/G/	-/W/
Jumat	07.30 - 09.10	-/X/	-/C/	-/R/	ID4072/C/WAA	-/Y/
Jumat	09.20 - 11.00	-/D/	ID4059/E/NAD	-/H/	-/Z/	-/Q/
Sabtu	07.30 - 10.00	IC2226/A/AYH	IC1214/B/NAD	IC3132 /B/IGP	-/F/	IC3240/A/FB
Sabtu	10.10 - 12.40	-/M/	IC2224/A/RA	-/I/	IC2225/B/MAA	IA1208/B/DL

Figure 2. Schedule Report for Even Semester Generated by The System

Figure 2 shows the scheduling result using the genetic algorithm with a population size of 15 and 25 iterations. The computation time required to produce this schedule was 9 minutes and 23 seconds. This schedule represents the best population with a fitness value of 1. The schedule generated by the system is considered good because there are no conflicts between lecturers, courses, or rooms. An advantage of using this scheduling application is time efficiency. A manually created schedule would take a long time, whereas the system-generated schedule took less than 15 minutes. This greatly facilitates the scheduling process, especially when revisions are needed. The system-generated schedule has also been adjusted to meet the

needs of the Informatics Engineering Study Program. The system ensures that early-semester courses are scheduled in the morning and prioritizes 3-credit courses in morning slots.

Even semester courses with 5 rooms result in a higher likelihood of conflicts. Since each row contains 5 genes, the chance of clashes is greater compared to having only 4 genes. This is due to the large number of course slots, which total 80. As a result, there are many empty course slots, considering there are only 45 courses in total, 22 with 3 credits and 23 with 2 credits.

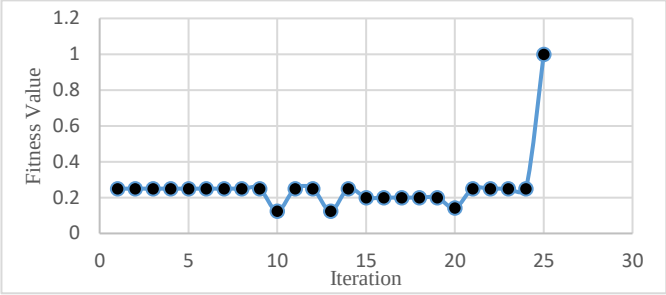

Figure 3. Fitness Value Graph in the Even Semester

Table 11. Violations in the Schedule Population for the Even Semester.

Iteration	Violation					Fitness Value
	Room	Group	Lecturer and Courses	3 SCU	Semester jadwal siang	
1	0	2	1	0	0	0,25
2	0	2	1	0	0	0,25
3	0	2	1	0	0	0,25
4	0	2	1	0	0	0,25
5	0	2	1	0	0	0,25
6	0	2	1	0	0	0,25
7	0	2	1	0	0	0,25
8	0	2	1	0	0	0,25
9	0	2	1	0	0	0,25
10	0	4	3	0	0	0,125
11	0	2	1	0	0	0,25
12	0	2	1	0	0	0,25
13	0	4	3	0	0	0,125
14	0	2	1	0	0	0,25
15	0	3	1	0	0	0,2
16	0	3	1	0	0	0,2

Figure 3 shows the fitness value graph of the highest fitness scores from 15 populations over 25 iterations. Table X displays the number of violations in the population with the highest fitness value in each iteration. From the first to the ninth iteration, the fitness value was 0.125, with a total of 3 violations. Then, in the tenth iteration, the fitness value dropped to 0.125 and increased to 0.25 in the eleventh iteration. In the thirteenth iteration, the fitness value dropped again to 0.125. In the fifteenth iteration, it increased to 0.2. By the twentieth iteration, the fitness value decreased to 0.142. In the twenty-first iteration, it rose again to 0.25. In the final iteration, the best population was found with a fitness value of 1.

Figure 3 illustrates the fluctuating fitness values in each iteration. Typically, the more iterations performed, the smaller the fitness value of the population becomes. However, it is also possible for the fitness value to drop further, as shown in the graph. This is due to the random nature of parent selection during recombination [14]. Nonetheless, populations with high fitness values are retained and not lost throughout the genetic algorithm process.

The fitness value represents the number of violations within a population. A low fitness value indicates that many violations occurred in that population. Conversely, the higher the fitness value, the fewer the violations. In the case of the even semester schedule, the lowest total number of violations was 7, with a

fitness value of 0.125. The highest fitness value was 1, indicating no violations in the population, making it the best solution.

B. Case Study: Odd Semester Courses of Academic Year 2016/2017

In the second case, there are only 4 rooms, which means there are 64 available course slots. Similar to the first case, these slots are divided into two types: 40 slots for 3-credit courses and 24 slots for 2-credit courses. In the odd semester, there are 21 courses with 3 credits, 21 courses with 2 credits, and 23 empty slots. The following section presents the schedule report created manually and the schedule report generated by the system. Figure 4 shows the schedule produced by the system.

Odd Semester Course Schedule

hari	jam	R1	R2	R3	R4
Senin	07.30 - 10.00	IC3129/A/IBK	IC2124/B/AZ	IC2123/A/AYH	ID3134/B/WAA
Senin	10.10 - 12.40	IB1108/B/IGP	ID3134/A/WAA	IC3129/B/IBK	-/F/
Senin	13.30 - 15.10	ID3073/F/IBK	-/N/	-/J/	-/O/
Selasa	07.30 - 10.00	IC4145/A/FB	IC2124/A/AZ	IC2119/B/RA	IC3130/B/AYH
Selasa	10.10 - 12.40	ID6070/C/AHJ	IC3135/A/AZ	-/E/	ID2120/A/NAD
Selasa	13.30 - 15.10	IC4143/A/MAA	-/P/	-/T/	-/U/
Rabu	07.30 - 10.00	IC2119/A/RA	-/A/	IC3130/A/AYH	ID4060/D/IBK
Rabu	10.10 - 12.40	IA4147/B/DL	IC3135/B/AZ	IC3131/A/DL	ID2120/B/NAD
Rabu	13.30 - 15.10	ID4051/H/RA	-/L/	ID4065/E/IGP	-/V/
Kamis	07.30 - 10.00	IC3133/A/BI	-/B/	IC4142/A/MAA	IB1108/A/IGP
Kamis	10.10 - 12.40	IC2122/A/SEA	IC2123/B/AYH	-/D/	IC4142/B/MAA
Kamis	13.30 - 15.10	ID3072/E/WAA	IC2121/B/AHJ	-/S/	IA4147/A/DL
Jumat	07.30 - 09.10	-/Q/	-/M/	ID4064/D/SEA	-/I/
Jumat	09.20 - 11.00	IC4143/B/MAA	-/R/	ID3055/C/FB	ID3071/G/NAD
Sabtu	07.30 - 10.00	-/C/	IC3131/B/DL	IC4145/B/FB	-/G/
Sabtu	10.10 - 12.40	IC4142/B/MAA	IC2122/B/SEA	IC3133/B/BI	IC2121/A/AHJ

Figure 4. Schedule Report for Odd Semester Generated by the System

Figure 4 shows the scheduling result using the genetic algorithm with a population size of 15 and 25 iterations. The computation time required to produce this schedule was 6 minutes and 35 seconds, which is shorter than the scheduling time for the even semester. This is due to the smaller number of rooms. In the odd semester, with only 4 rooms, fewer violations occurred compared to the even semester, which had 5 rooms. This is because each row contains only 4 genes, reducing the likelihood of conflicts. Additionally, the computation time for scheduling odd semester courses is faster than for the even semester.

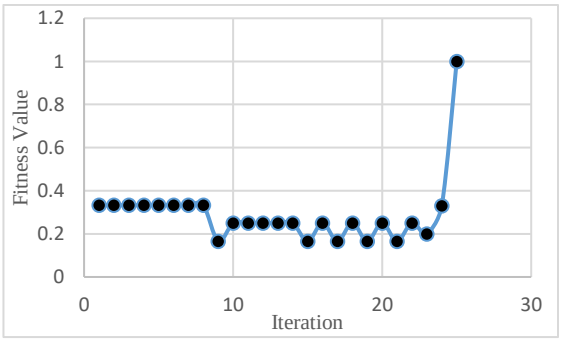

Figure 5. Fitness Value Graph in the Odd Semester

Table 12. Violations in the Schedule Population for the Odd Semester.

Iteration	Violation					Nilai fitness
	Room	Group	Lecturer and Courses	3 SCU	Semester jadwal siang	
1	0	1	1	0	0	0,333
2	0	2	0	0	0	0,333
3	0	2	0	0	0	0,333
4	0	2	0	0	0	0,333
5	0	2	0	0	0	0,333
6	0	2	0	0	0	0,333
7	0	2	0	0	0	0,333
8	0	2	0	0	0	0,333
9	0	4	1	0	0	0,166

Figure 5 shows the graph of the highest fitness values for 15 populations over 25 iterations. When comparing the fitness value graphs in Figure 3 and Figure 5, the odd semester has higher fitness values. This proves that the fewer the number of rooms, the fewer the violations. Table XII displays the number of violations from the population with the highest fitness value in each iteration.

From the first to the eighth iteration, the fitness value was 0.333. In the ninth iteration, it dropped to 0.166, then increased to 0.25 from the tenth to the fourteenth iteration. In the fifteenth iteration, the fitness value dropped again to 0.166, and then rose to 0.5 in subsequent iterations, continuing until the twenty-second iteration. In the twenty-third iteration, the fitness value dropped to 0.2. The fitness value then increased to 0.33 in the twenty-fourth iteration. Finally, in the twenty-fifth iteration, the best population with a fitness value of 1 was found.

Figure 5 displays inconsistent fitness values across iterations. Typically, the more iterations are performed, the lower the fitness value of the population. However, it is still possible for the fitness value to decrease again, as shown in the graph. This occurs due to the random parent selection process during recombination.

C. Test Case Using Crossover Probability and Mutation Rate

This test was conducted by inputting different values for crossover probability and mutation rate. The test used 15 populations with 25 iterations and a crossover probability of 0.5. Figure 6 shows the results of the test.

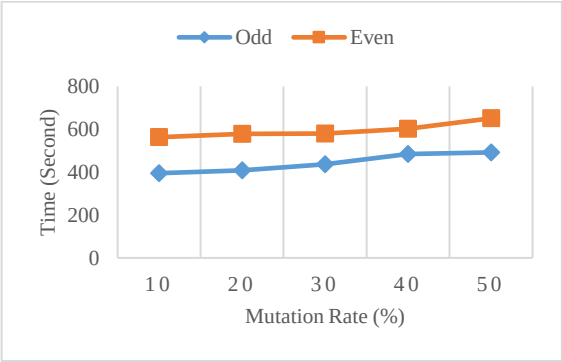

Figure 6. Computation Time Graph

From Figure 6, it can be concluded that the higher the mutation probability or mutation rate, the longer the computation time required.

D. Computation Time Testing

Computation time refers to the duration needed for a single execution of the program. Figure 7 presents a graph of the execution time of the course scheduling application with a population size of 15.

E-ISSN:*2621-4474, ISSN:2621-4458.*

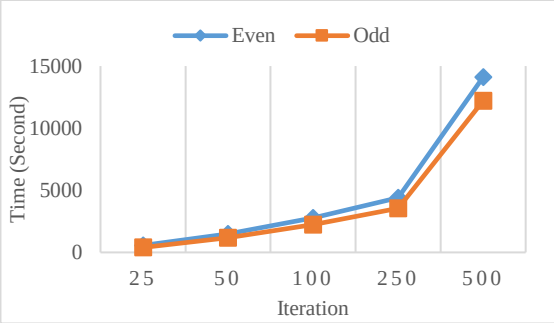

Figure 7. Computation Time Graph

From Figure 7, it can be concluded that the greater the number of iterations, the longer the execution time required. The number of rooms also affects the computation time. The even-semester schedule with 5 rooms takes longer to process compared to the odd-semester schedule with 4 rooms. This is because the more rooms there are, the more course slots exist, which can slow down execution time.

IV. CONCLUSIONS AND RECOMMENDATIONS

The course scheduling application based on the genetic algorithm serves to automatically generate class timetables and address scheduling-related challenges. The optimal solution is represented by a population with a fitness value of 1, indicating that all hard constraints have been satisfied and no violations remain [15]. The number of classrooms and courses significantly influences execution time; as this increase, so does the time required for computation. The genetic algorithm has proven to be relatively effective for scheduling tasks. In experiments involving 25 iterations and a population size of 15, the best results were obtained using a crossover probability of 0.5 and a mutation rate of 10%. The lowest recorded fitness value was 0.09, with the fastest execution times being 395 seconds for odd-semester scheduling and 563 seconds for even-semester scheduling. However, the application is still in need of refinement, as violations of hard constraints persist, indicating the necessity of incorporating a switching or repair mechanism to improve overall solution quality.

.

V. REFERENCES

[1] A. A. Gozali and S. Fujimura, "Solving University Course Timetabling Problem Using Multi-Depth Genetic Algorithm: Solving UCTP Using MDGA," in *SHS Web of Conferences*, 2020.

[2] I. A. Ashari, M. A. Muslim and Alamsyah, "Comparison Performance of Genetic Algorithm and Ant Colony Optimization in Course Scheduling Optimizing," *Scientific Journal of Informatics*, pp. 149 - 158, 2016.

[3] T. R. Ahyana and Y. Jumaryadi, "Perancangan Sistem Informasi Penjadwalan Mengajar Menggunakan Metode Algoritma Genetika (Studi Kasus: Smk Satria Jakarta)," *Ensiklopedia of Journal*, vol. 1, no. 2, 2019.

[4] Q. Zhang, "An optimized solution to the course scheduling problem in universities under an improved genetic algorithm," *Journal of Intelligent Systems*, vol. 31, no. 1, pp. 1065 - 1073, 2022.

[5] C. B. Mallari, J. L. S. Juan and R. Li, "The university coursework timetabling problem: An optimization approach to synchronizing course calendars," *Computers & Industrial Engineering*, vol. 184, 2023.

[6] A. Andi and D. Nasien, "Optimization of Genetic Algorithm in Courses Scheduling," *IT Journal Research and Development*, vol. 6, no. 2, pp. 151-161, 2022.

[7] A. Rezaeipanah, S. S. Matoori and G. Ahmadi, "A hybrid algorithm for the university course timetabling problem using the improved parallel genetic algorithm and local search," *Applied Intelligence*, vol. 51, no. 1, 2021.

[8] F. A. Omara and M. M. Arafa, "Genetic algorithms for task scheduling problem," *Journal of Parallel and Distributed Computing*, vol. 70, no. 1, pp. 13-22, 2010.

[9] E. A. Abdelhalim and G. A. E. Khayat, "A Utilization-based Genetic Algorithm for Solving the University Timetabling Problem (UGA)," *Alexandria Engineering Journal*, vol. 55, pp. 1395-1409, 2016.

[10] R. Prosad, M. A. R. Khan and I. Ahammad, "Design of Class Routine and Exam Hall Invigilation System based on Genetic Algorithm and Greedy Approach," *Asian Journal of Research in Computer Science*, vol. 13, no. 3, pp. 28-44, 2022.

[11] Z. Zhou, F. Li, H. Zhu, H. Xie, J. H. Abawajy and M. U. Chowdhury, "An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments," *Neural Computing and Applications*, vol. 32, pp. 1531-1541, 2020.

[12] M. Assi, B. Halawi and R. A. Haraty, "Genetic Algorithm Analysis using the Graph Coloring Method for Solving the University Timetable Problem," *Procedia Computer Science*, pp. 899-906, 2018.

[13] X. Chen, X.-G. Yue, R. Li, A. Zhumadillayeva and R. Liu, "Design and Application of an Improved Genetic Algorithm to a Class Scheduling System," *International Journal of Emerging Technology in Learning*, vol. 16, no. 1, pp. 44-59, 2021.

[14] j. Xu, "Improved Genetic Algorithm to Solve the Scheduling Problem of College English Courses," *Complexity*, 2021.

[15] J. Arias-Osorio and A. Mora-Esquivel, "A solution to the university course timetabling problem using a hybrid method based on genetic algorithms," *DYNA*, vol. 87, no. 215, pp. 47-56, 2020.