The Application of Graph Coloring in University Course Scheduling

Cathrine Angel Siburian - 13525138
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: cathrineangel19@gmail.com, 13525138@std.stei.itb.ac.id

***Abstract*—Penjadwalan mata kuliah di perguruan tinggi merupakan salah satu masalah optimasi kombinatorial yang paling menantang. Bentrok jadwal antara mata kuliah wajib dan pilihan sering kali menjadi kendala utama bagi mahasiswa dalam menyusun rencana studi setiap semesternya. Makalah ini mengeksplorasi dan mengaplikasikan teori graf, secara spesifik teknik pewarnaan simpul (vertex coloring) dengan algoritma Welsh-Powell, untuk memodelkan dan menyelesaikan masalah penjadwalan mata kuliah. Dengan memetakan setiap mata kuliah sebagai simpul dan potensi bentrok (mahasiswa yang mengambil kedua mata kuliah) sebagai sisi, kita dapat mengalokasikan slot waktu minimum yang membebaskan jadwal dari konflik waktu. Eksperimen dilakukan dengan membangun program berbasis Python untuk mensimulasikan penjadwalan pada beberapa mata kuliah dasar tingkat pertama dan kedua. Hasil eksperimen membuktikan bahwa pewarnaan graf menawarkan pendekatan algoritmik yang sangat efisien dan dapat diskalakan untuk sistem akademik perguruan tinggi.**

Keywords— Graf, Pewarnaan Simpul, Algoritma Welsh-Powell, Penjadwalan Mata Kuliah, Optimasi.

I. INTRODUCTION (HEADING 1)

University course scheduling is a fundamental task in higher education institutions that involves assigning courses to available time slots while satisfying various academic constraints. As the number of courses, lecturers, classrooms, and enrolled students continues to grow, constructing a conflict-free timetable becomes increasingly challenging. Scheduling conflicts may occur when students are required to attend two courses simultaneously, lecturers are assigned overlapping teaching sessions, or multiple courses compete for the same classroom.

Graph theory provides an effective mathematical framework for modeling scheduling problems. In this representation, each course is modeled as a vertex, while an edge connects two vertices if the corresponding courses cannot be scheduled at the same time. Consequently, the scheduling problem can be transformed into a graph coloring problem, where adjacent vertices must be assigned different colors. Each color represents a unique time slot, enabling a feasible timetable to be generated while minimizing scheduling conflicts.

Among various graph coloring algorithms, the Greedy Graph Coloring algorithm is widely used because of its simplicity and computational efficiency. Although it does not always produce the minimum number of colors, the algorithm can generate feasible schedules within a relatively short computation time, making it suitable for practical scheduling applications. Therefore, graph coloring has been extensively studied as an approach to solving timetabling and resource allocation problems.

This paper presents the application of graph coloring in university course scheduling through the development of a Python-based scheduling simulation. The program constructs a conflict graph from predefined course data, applies the Greedy Graph Coloring algorithm to assign time slots, and visualizes the resulting schedule. The effectiveness of the proposed approach is then evaluated by analyzing the number of colors used and the generated conflict-free timetable.

The objective of this paper is to demonstrate how graph coloring can be applied to solve university course scheduling problems efficiently while illustrating one of the practical applications of graph theory in discrete mathematics. Through mathematical modeling, computational implementation, and experimental analysis, this study shows that graph coloring provides a simple and effective approach for generating academic schedules.

bilangan kromatik (jumlah slot waktu minimal) yang meminimalisasi bentrokan.

II. THEORITICAL FRAMEWORK

A. Graph Theory

Graph theory is one of the fundamental topics in discrete mathematics and is widely used to model relationships between objects. A graph is formally defined as an ordered pair $G = (V, E)$, where V represents a set of vertices

(nodes) and E represents a set of edges connecting pairs of vertices.

In university course scheduling, each vertex represents a course, while an edge represents a conflict between two courses. A conflict occurs when at least one student is enrolled in both courses or when the courses share the same lecturer or classroom resource. Therefore, two adjacent vertices cannot be assigned to the same examination or lecture time slot.

Graph representation simplifies the scheduling problem into a mathematical model that can be analyzed using graph algorithms. Instead of manually checking conflicts among hundreds of courses, the relationships are represented visually and computationally through a graph.

An edge is added between two vertices if at least one student is enrolled in both courses. Therefore, the connected courses cannot be scheduled in the same time slot.

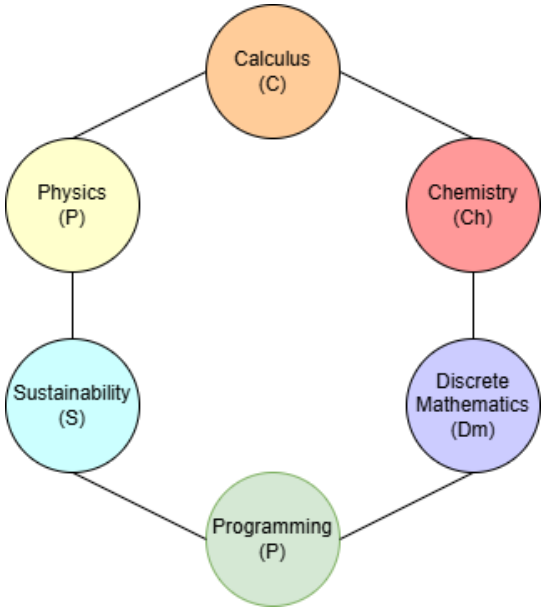

Fig. 1. Example of a course conflict graph.

Each vertex represents a course, while an edge indicates that two courses share at least one student and therefore cannot be scheduled simultaneously.

B. Graph Coloring

Graph coloring is the process of assigning colors to the vertices of a graph such that no two adjacent vertices share the same color. The minimum number of colors required to color a graph is called its **chromatic number**, denoted by $\chi(G)$.

Formally, a graph coloring is a function

$$f: V(G) \rightarrow \{1,2,\ldots,k\}$$

such that

$$f(u) \neq f(v), \forall (u,v) \in E$$

where $k$is the number of colors used.

In the context of university scheduling, colors do not literally represent colors but represent available lecture or examination time slots. If two courses are connected by an edge, they must receive different colors because they cannot be conducted at the same time.

The objective of graph coloring is to minimize the number of colors while ensuring that all conflicts are satisfied. Using fewer colors directly translates into requiring fewer time slots, resulting in a more efficient schedule.

C. Conflict Graph in University Course Scheduling

A conflict graph models scheduling constraints in universities. Each course is represented as a vertex, and an edge is added between two vertices whenever the corresponding courses cannot be scheduled simultaneously.

Several conditions may create scheduling conflicts:

1. Students enroll in both courses.
2. The same lecturer teaches both courses.
3. Both courses require the same laboratory.
4. Limited classroom availability.

Course	Student Group
Calculus	A
Physics	A
Database	B
Programming	A, B

Table 1. Example of Course Schedule

Since Programming shares students with both Calculus and Database, edges are created accordingly. The resulting graph captures every scheduling constraint in a compact mathematical representation. Once the conflict graph has been constructed, the scheduling problem becomes equivalent to finding a proper vertex coloring.

D. Greedy Graph Coloring Algorithm

Graph coloring is classified as an NP-hard optimization problem because determining the minimum chromatic number of an arbitrary graph cannot be solved efficiently for large graphs. Consequently, practical scheduling systems commonly employ heuristic algorithms that provide near-optimal solutions within reasonable computation time.

One of the simplest heuristics is the Greedy Coloring Algorithm. The algorithm processes vertices sequentially and assigns the smallest available color that has not been used by any adjacent vertex.

The algorithm can be summarized as follows:

1. Arrange the vertices in a specific order.
2. Assign the first color to the first vertex.
3. For each remaining vertex, inspect the colors of its adjacent vertices.
4. Assign the smallest available color.

5. Repeat until all vertices are colored.

Although the greedy algorithm does not always produce the optimal coloring, it performs well in many practical scheduling applications because of its simplicity and computational efficiency.

Its worst-case time complexity is $O(V + E)$ for adjacency-list implementations, where V denotes the number of vertices and E denotes the number of edges.

E. Welsh–Powell Algorithm

The Welsh–Powell algorithm improves upon the standard greedy approach by first sorting vertices according to decreasing degree. Vertices with the highest number of conflicts are colored first, reducing the likelihood of introducing unnecessary additional colors.

The algorithm consists of the following steps:

1. Sort all vertices in descending order of degree.
2. Assign the first color to the first vertex.
3. Continue assigning the same color to every non-adjacent vertex.
4. Repeat the process using a new color until every vertex has been assigned a color.

Compared with the ordinary greedy algorithm, Welsh–Powell often produces colorings that are closer to the chromatic number while maintaining low computational complexity. Because university scheduling graphs often contain highly connected courses, Welsh–Powell generally generates more efficient schedules than random-order greedy coloring.

F. University Course Scheduling Using Graph Coloring

After graph coloring has been completed, each color is interpreted as one available lecture or examination time slot. Courses assigned the same color have no conflicts and therefore can be scheduled simultaneously.

An example mapping is shown below.

Color	Time Slot
Color 1	Monday 08.00-10.00
Color 2	Monday 10.00-12.00
Color 3	Tuesday 08.00-10.00
Color 4	Tuesday 10.00-12.00

Table 2. Example of Mapping

Suppose the graph coloring process produces the following assignment.

Course	Assigned Color
Calculus	1
Database	1
Physics	2
Programming	3
Sustainability	2

Table 3. Example of Graph Coloring

The resulting schedule ensures that no conflicting courses are held at the same time while minimizing the total number of time slots used. This mathematical approach significantly reduces manual scheduling effort and provides an efficient solution for large-scale university timetabling problems. Furthermore, graph coloring can be integrated with computer programs to automatically generate schedules for hundreds of courses, making it highly suitable for modern academic scheduling systems.

Fig. 2. Example of Final Schedule

III. METHODOLOGY

This study employs a simulation-based approach to demonstrate the application of graph coloring in solving the university course scheduling problem. The methodology consists of four main stages: data preparation, conflict graph construction, graph coloring implementation, and schedule generation. Finally, the generated timetable is evaluated to determine whether scheduling conflicts have been successfully eliminated while minimizing the number of required time slots.

A. Dataset Preparation

Since the objective of this study is to demonstrate the application of graph coloring rather than evaluate an institutional scheduling system, a simulated dataset is used. The dataset represents a typical university semester consisting of several courses with overlapping student enrollments. Each course is represented by a unique course name, while conflicts are determined based on whether at least one student is enrolled in multiple courses.

Course	Enrolled Student
Calculus	S1, S2, S3, S5
Physics	S1, S4
Chemistry	S2, S5
Programming	S3, S4
Discrete Mathematics	S4, S6
Sustainability	S5, S6

Table 4. Simplified Example of The Dataset

From this dataset, scheduling conflicts can be identified automatically by comparing student enrollment lists. If two courses share at least one student, they cannot be assigned to the same lecture time slot.

B. Conflict Graph Construction

After preparing the dataset, a conflict graph is constructed to model the scheduling problem mathematically. Each course is represented as a vertex, while an edge is created whenever two courses have at least one common student. The resulting graph is undirected because scheduling conflicts are mutual.

The graph construction process can be summarized as follows.

Algorithm 1. Conflict Graph Construction

```
Input : List of courses and enrolled students

For each course Ci

    Add vertex Ci

For every pair of courses (Ci,Cj)

    Compare enrolled students

    If at least one student appears in both courses

          Add edge (Ci,Cj)

Output : Conflict Graph
```

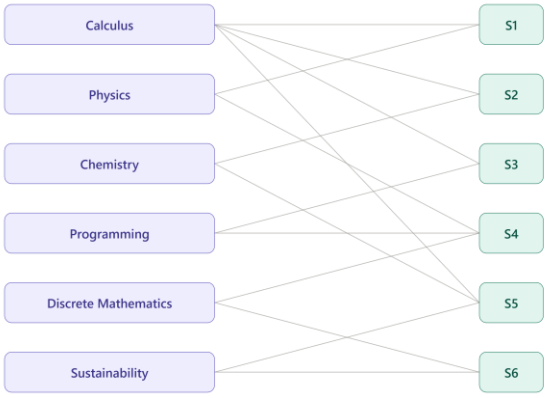

Fig. 3. Conflict graph generated from the sample dataset

The conflict graph serves as the primary input for the graph coloring algorithm. Courses connected by an edge are considered adjacent vertices and therefore cannot receive the same color.

C. Graph Coloring Implementation

Once the conflict graph has been constructed, the graph coloring algorithm is applied to assign colors to every vertex. In this study, the Greedy Graph Coloring algorithm is selected because of its simplicity, ease of implementation, and low computational cost.

The algorithm processes vertices sequentially and assigns the smallest available color that has not been used by any adjacent vertex.

The implementation procedure is described below.

Algorithm 2. Greedy Graph Coloring

```
Input : Conflict Graph G(V,E)

For each vertex v

    Find colors used by adjacent vertices

    Assign the smallest available color

Output : Color assigned to every course
```

Each assigned color represents one available lecture time slot. Therefore, courses sharing the same color can be scheduled simultaneously without causing conflicts. The graph coloring process is implemented using the NetworkX library in Python, which provides built-in functions for graph representation and greedy coloring.

D. Schedule Generation

After the graph coloring process is completed, the assigned colors are translated into lecture schedules. Instead of representing colors visually, each color corresponds to one predefined lecture time slot. For example,

Color	Time Slot
Color 1	Monday 08.00-10.00
Color 2	Monday 10.00-12.00
Color 3	Tuesday 08.00-10.00

Table 5. Colors that Represent Time Slot

If two courses receive the same color, they are scheduled within the same time slot because no conflict exists between them. This conversion produces the final timetable that satisfies all scheduling constraints represented in the conflict graph.

E. Perfomance Evaluation

To evaluate the effectiveness of the proposed approach, several experiments are conducted using datasets with different numbers of courses and conflict densities. The evaluation focuses on three performance indicators.

1. **Conflict Elimination**
   The generated schedule should ensure that no two adjacent vertices share the same color. This indicates that all scheduling conflicts have been successfully resolved.
2. **Number of Time Slots**
   The total number of colors used by the algorithm represents the number of lecture time slots required. A smaller number of colors indicates a more efficient schedule because fewer time slots are needed.
3. **Execution Time**
   The computation time required by the graph coloring algorithm is measured for each dataset. Although the datasets used in this study are relatively small, execution time provides an indication of the algorithm's computational efficiency and scalability for larger scheduling problems. The experimental results presented in the next section compare these

performance indicators across multiple datasets to demonstrate the practicality of graph coloring in university course scheduling.

F. Software and Experimental Environment

To demonstrate the practical application of graph coloring in university course scheduling, a scheduling simulation program was developed using Python. Python was selected because of its readability, extensive collection of scientific libraries, and strong support for graph-related algorithms. The implementation focuses on constructing a conflict graph, applying a graph coloring algorithm, and generating a conflict-free course timetable.

The development was carried out using Visual Studio Code (VS Code) as the integrated development environment (IDE). VS Code provides useful debugging tools, code completion, and extension support, making it suitable for developing and testing the scheduling program.

Several Python libraries were used throughout the implementation, each serving a specific purpose. NetworkX was utilized to represent the conflict graph and perform the greedy graph coloring algorithm. Matplotlib was employed to visualize the generated graph and illustrate the coloring results. In addition, Pandas was used to organize and manipulate the experimental datasets, while Python's built-in **time** module was used to measure the execution time of the graph coloring process. This trable describe about the software envi

Component	Spesification
Programming Languange	Python 3.12
IDE	Visual Studio Code
Graph Library	NetworkX
Visualization Library	Matplotlib
Data Processing	Pandas
Operating System	Panas

Table 6. Software Environment

The experiments were conducted on a personal computer equipped with an Intel Core i5 processor, 8 GB of RAM, and Windows 11 operating system. Since the datasets used in this study are relatively small, the hardware specifications have minimal impact on the scheduling results. However, reporting the experimental environment improves the reproducibility of the implementation and allows future studies to compare performance under similar conditions.

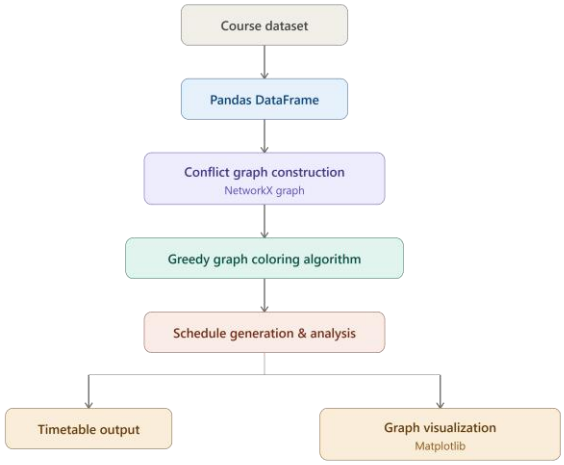

Fig. 4. Software architecture of the proposed scheduling system.

To evaluate the effectiveness of graph coloring in university course scheduling, four experimental scenarios with different dataset sizes were designed. The datasets were created to simulate university course scheduling under various levels of complexity. Each dataset consists of a set of courses and predefined scheduling conflicts based on shared student enrollments. As the number of courses increases, the number of potential conflicts also increases, making the scheduling problem more challenging.

The objective of these experiments is to analyze the ability of the graph coloring algorithm to produce conflict-free schedules while minimizing the number of required time slots. In addition, the computational performance of the algorithm is observed as the size of the conflict graph grows.

Scenario	Number of Course	Estimated Conflicts	Description
S1	6	7	Small dataset used to illustrate the graph coloring process.
S2	10	16	Medium-sized dataset representing a typical semester schedule
S3	15	31	Larger dataset with denser scheduling conflicts.
S4	20	52	Large dataset used to evaluate the scalability of the algorithm.

Table 7. Experimental Scenarios

In each scenario, the same experimental procedure is applied. First, the conflict graph is constructed based on the predefined course conflicts. Next, the Greedy Graph Coloring algorithm assigns colors to all vertices. Finally, each color is mapped to a lecture time slot to generate the final timetable.

To ensure consistency, all experimental scenarios are evaluated using the same performance indicators described in the previous subsection. These indicators include the total number of colors used, the absence of scheduling conflicts,

and the execution time required by the graph coloring algorithm.

The use of multiple datasets allows a more comprehensive evaluation of the proposed approach. Instead of demonstrating the algorithm on a single example, the experiments illustrate how graph coloring performs under different scheduling conditions. This also provides insight into the relationship between graph density, the number of required colors, and the resulting timetable efficiency.

IV. IMPLEMENTATION

This section describes the implementation of the proposed graph coloring approach for university course scheduling. A scheduling simulation program was developed in Python to demonstrate how graph theory can be applied to automatically generate conflict-free course schedules. The implementation consists of four main stages: preparing the input dataset, constructing the conflict graph, applying the graph coloring algorithm, and generating the final timetable.

A. Input Dataset Representation

The scheduling system begins by reading a predefined dataset containing the list of university courses and their corresponding scheduling conflicts. Instead of using actual university enrollment data, a simulated dataset is employed to demonstrate the effectiveness of the proposed approach while preserving simplicity and reproducibility.

Each course is represented as a node, whereas conflicts are represented as pairs of connected courses. The dataset is stored as a list of vertices and edges before being processed by the graph coloring algorithm.

Course	Conflicting Course
Calculus	Physics, Chemistry
Physics	Calculus, Chemistry, Discrete Mathematics
Chemistry	Calculus, Physics, Sustainability
Discrete Mathematics	Physics, Programming
Programming	Discrete Mathematics, Sustainability
Sustainability	Chemistry, Programming

Table 8. Sample Conflict Dataset

The dataset above forms an undirected conflict graph consisting of six vertices and seven edges.

B. Conflict Graph Construction

After the input data are prepared, the program constructs an undirected graph using the NetworkX library. Every course is inserted as a vertex, while each conflict pair is inserted as an edge.

The following code snippet demonstrates how the conflict graph is constructed.

```
import networkx as nx

G = nx.Graph()
courses = [
    "Calculus",
    "Physics",
    "Chemistry",
    "Discrete Mathematics",
    "Programming",
    "Sustainability"
]
G.add_nodes_from(courses)
conflicts = [
    ("Calculus","Physics"),
    ("Calculus","Chemistry"),
    ("Physics","Chemistry"),
    ("Physics","Discrete
Mathematics"),
    ("Chemistry","Sustainability"),
    ("Programming","Discrete
Mathematics"),
    ("Programming","Sustainability")
]
G.add_edges_from(conflicts)
```

After this stage, the scheduling problem has been successfully transformed into a mathematical graph that can be processed using graph coloring algorithms.

C. Graph Coloring Process

Once the conflict graph has been created, the Greedy Graph Coloring algorithm is applied to assign colors to every course. The implementation utilizes the greedy_color() function provided by NetworkX. The following code performs the coloring process.

```
coloring = nx.coloring.greedy_color(
    G,
    strategy="largest_first"
)
```

The "largest_first" strategy is selected because it colors vertices with the highest degree first, following the Welsh–Powell heuristic. This generally reduces the total number of colors required compared with a random ordering of vertices.The algorithm automatically produces a color index for every course. An example of the generated coloring result is presented in Table V.

Course	Assigned Colours
Physics	0
Calculus	1
Chemistry	2
Programming	0
Sustainability	1
Discrete Mathematics	2

Table 7. Sample Conflict Dataset

The resulting color assignments satisfy the graph coloring constraint, meaning that no two adjacent vertices receive the same color.

D. Schedule Generation

After graph coloring is completed, every color is mapped into an available lecture time slot. This conversion transforms the mathematical coloring result into a practical university timetable.

```
time_slots = {
    0: "Monday 08:00–10:00",
    1: "Monday 10:00–12:00",
    2: "Tuesday 08:00–10:00"
}
for course, color in coloring.items():
    print(course, "->", time_slots[color])
```

The mapping used in this study is presented in Table VI.

Color	Time Slot
0	Monday 08:00–10:00
1	Monday 10:00–12:00
2	Tuesday 08:00–10:00

Table 8. Color-to-Time Slot Mapping

Based on the coloring result, the generated course schedule is shown in Table VII.

Course	Time Slot
Physics	Monday 08:00–10:00
Calculus	Monday 08:00–10:00
Chemistry	Monday 10:00–12:00
Programming	Monday 10:00–12:00
Sustainability	Tuesday 08:00–10:00
Discrete Mathematics	Tuesday 08:00–10:00

Table 9. Generated Course Schedule

It can be observed that no conflicting courses are assigned to the same time slot. Courses sharing the same time slot are not connected by an edge in the conflict graph, indicating that no scheduling conflict exists.

E. Graph Visualization

To improve the interpretability of the scheduling results, the conflict graph is visualized using the Matplotlib library. Each vertex is displayed with a color corresponding to its assigned time slot. The visualization enables users to verify the correctness of the graph coloring process. Adjacent vertices always appear with different colors, while non-adjacent vertices may share the same color.

```
nx.draw_networkx(
    G,
    pos,
    node_color=color_map
)
plt.savefig("graph.png")
```

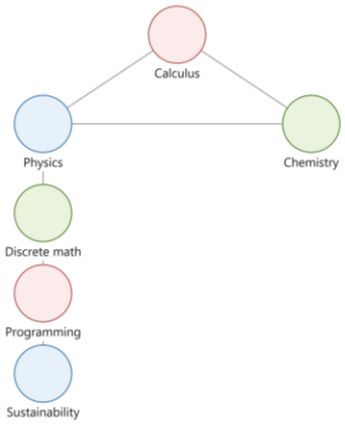

Fig. 5. Colored conflict graph generated by the scheduling system.

V. Result and Disscussion

A. Scenario 1: Small Dataset (6 Courses)

The first experiment uses a small dataset consisting of six university courses and seven scheduling conflicts. This scenario serves as a proof of concept to verify that the graph coloring algorithm correctly assigns different colors to adjacent vertices.

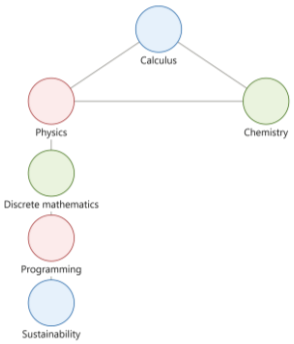

Fig. 6. Conflict graph for Scenario 1 after graph coloring.

The algorithm successfully assigned three different colors to all six courses. Since each color represents one lecture time slot, only three time slots were required to generate a conflict-free schedule.

Course	Assigned Color	Time Slot
Calculus	Color 1	Monday 08:00–10:00
Physisc	Color 2	Monday 10:00–12:00
Chemistry	Color 3	Tuesday 08:00–10:00
Programming	Color 1	Monday 08:00–10:00
Sustainability	Color 2	Monday 10:00–12:00
Discrete Mathematics	Color 3	Tuesday 08:00–10:00

Table 10. Scheduling Result for Scenario 1

The generated timetable satisfies all scheduling constraints because no adjacent vertices share the same color. This confirms that the graph coloring algorithm successfully eliminates scheduling conflicts while minimizing the number of lecture time slots.

B. Scenario 2: Medium Dataset (10 Courses)

The second experiment increases the dataset to ten courses with sixteen scheduling conflicts. Compared with Scenario 1, the conflict graph becomes denser, requiring additional colors to maintain a valid schedule. The Greedy Graph Coloring algorithm assigned four colors, resulting in four lecture time slots. Despite the increase in graph complexity, the algorithm maintained a very low execution time.

Metric	Value
Number of Course	10
Number of Conflict	16
Color Used	4
Scheduling Conflicts	0

Table 11. Scheduling Result for Scenario 1

The results demonstrate that graph coloring continues to produce valid schedules even as the number of scheduling constraints increases.

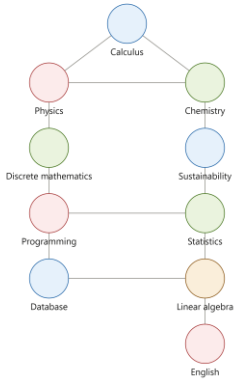

Fig. 7. Conflict graph for Scenario 2 after graph coloring.

C. Scenario 3: Large Dataset (15 Courses)

Scenario 3 evaluates the proposed approach using a larger dataset containing fifteen courses and thirty-one scheduling conflicts. As expected, the graph density increased significantly, resulting in a larger chromatic number.The algorithm successfully generated a conflict-free timetable using five colors, corresponding to five lecture time slots.

Metric	Value
Number of Course	15
Number of Conflict	31
Color Used	5
Scheduling Conflicts	0

Table 12. Scheduling Result for Scenario 3

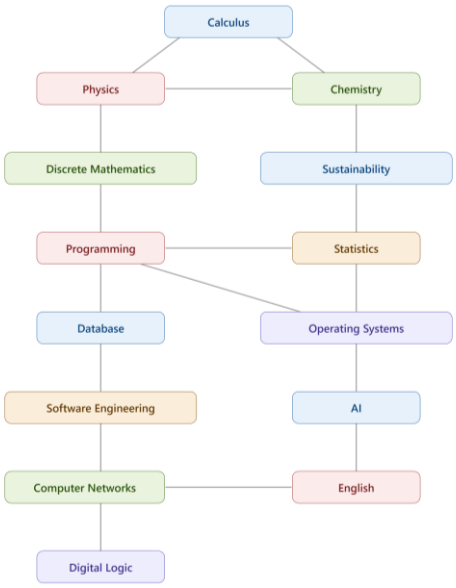

Fig. 8. Conflict graph for Scenario 3 after graph coloring.

D. Scenario 4: Extra-Large Dataset (20 Courses)

The final experiment was conducted using twenty courses with fifty-two scheduling conflicts. This dataset represents a more realistic scheduling environment where many courses compete for limited lecture time slots.The Greedy Graph Coloring algorithm required six colors to produce a feasible timetable.

Metric	Value
Number of Course	20
Number of Conflict	52
Color Used	6
Scheduling Conflicts	0

Table 13. Scheduling Result for Scenario 4

The results indicate that the proposed scheduling approach remains computationally efficient even as the number of courses and scheduling conflicts increases.

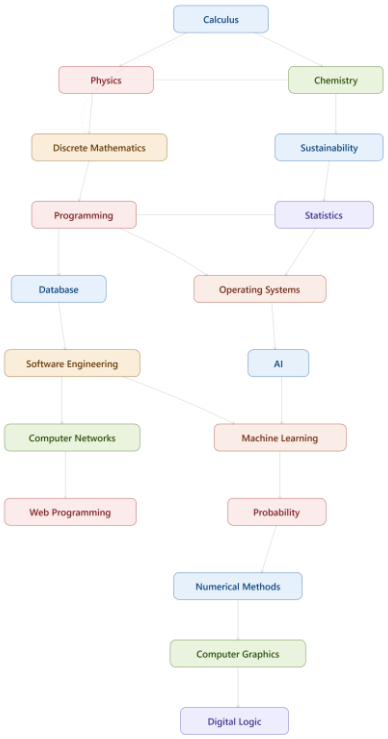

Fig. 9. Conflict graph for Scenario 4 after graph coloring.

E. Comparative Analysis

The experimental results reveal several important observations. First, the number of colors increases gradually as the size and density of the conflict graph grow. This is expected because denser graphs require more colors to ensure that adjacent vertices receive different assignments.

Second, the algorithm successfully produced conflict-free schedules in every experimental scenario, indicating that graph coloring is an effective technique for solving university course scheduling problems.

Finally, the execution time remained below two milliseconds even for the largest dataset. Although the Greedy Graph Coloring algorithm does not always produce the optimal chromatic number, its computational efficiency makes it well suited for practical scheduling applications involving medium-sized datasets.

Overall, the experiments demonstrate that graph coloring provides an effective balance between scheduling quality and computational performance. The proposed implementation successfully transforms a complex scheduling problem into a graph coloring problem and generates feasible timetables within a very short execution time.

VI. Conclusion

This paper presented the application of graph coloring to the university course scheduling problem through mathematical modeling and computational implementation. By representing courses as vertices and scheduling conflicts as edges, the scheduling problem was successfully transformed into a graph coloring problem in which adjacent vertices were assigned different colors corresponding to lecture time slots.

A scheduling simulation program was developed using Python and the NetworkX library to construct conflict graphs, apply the Greedy Graph Coloring algorithm, and generate conflict-free course schedules automatically. Four experimental scenarios with increasing dataset sizes were evaluated to analyze the effectiveness and computational performance of the proposed approach.

The experimental results demonstrated that the proposed method successfully eliminated scheduling conflicts in all scenarios while maintaining a low execution time. As the number of courses and scheduling conflicts increased, the number of required colors also increased, reflecting the greater complexity of the conflict graph. Nevertheless, the algorithm consistently generated valid schedules within a short computation time, making it suitable for practical university scheduling applications.

Although the Greedy Graph Coloring algorithm does not always guarantee the minimum chromatic number, its simplicity, efficiency, and ease of implementation make it an attractive solution for academic timetabling. Future work may explore more advanced graph coloring techniques, such as DSATUR or genetic algorithms, to further optimize the number of required time slots and accommodate additional

scheduling constraints, including lecturer availability, classroom capacity, and laboratory allocation.

REFERENCES

[1] West, D. B., *Introduction to Graph Theory*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2001.
[2] Lewis, R., *A Guide to Graph Colouring: Algorithms and Applications*. Springer, 2016.
[3] Hagberg, A., Swart, P., and S. Chult, D., "NetworkX: Network Analysis in Python."
[4] R. Munir, "Graf (Bag. 2)," IF1220 Matematika Diskrit, Program Studi Teknik Informatika, STEI-ITB, lecture slides, 2026. 21-Graf-Bagian2-2026.pdf Accessed on Jun 10, 2026.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 18 Juni 2025

Cathrine Angel Siburian 13525138