Minimizing Channel Interference in Wireless Networks Using Graph Coloring Techniques

Stefani Angeline Oroh - 13524064
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: angelineoroh15@gmail.com , 13524064@std.stei.itb.ac.id

***Abstract*—This paper discusses a practical strategy for reducing wireless network interference by leveraging graph coloring techniques. In densely populated wireless environments, overlapping signals from multiple access points (APs) can result in signal disruption. In our model, each AP will be represented as a vertex in a graph, and an edge between any two vertices indicates possible interference. To better reflect real world signal behavior, we construct a weighted graph based on the Euclidean distance between APs and apply a distance threshold to determine which APs are likely to interfere. This filtered interference graph is then colored using the Welsh-Powell algorithm, assigning wireless channels (as colors) to APs such that adjacent vertices receive different assignments. A simulation involving six APs demonstrates the effectiveness of applying graph theory concepts to address real world challenges in wireless communications.**

Keywords—graph coloring; interference reduction; wireless network; weighted graph; Welsh-Powell algorithm.

I. INTRODUCTION (*HEADING 1*)

Wireless networks play a significant role in modern communication systems, enabling the transmission of data over radio frequencies without using physical cables. These networks typically consist of multiple access points (APs) that broadcast signals within a limited radius. In environments, where several APs are installed in proximity, such as apartment buildings, educational institutions, or offices, signal overlap is inevitable. It causes interference, which can significantly degrade the quality of services, including dropped connections, reduced data throughput, and higher latency.

One key method of interference mitigation is channel assignment—allocating different frequency channels to APs to prevent overlapping usage. However, the limited number of available channels and varying distances between APs make this problem non-trivial. This is when graph theory from math discrete becomes a valuable tool.

The channel allocation issue can be abstracted as a graph coloring problem, where APs are represented as vertices, and edges are drawn between APs that may interfere. To make this model more realistic, we use a weighted graph based on the Euclidean distance between APs and only connect APs that have distance less than the threshold. Each vertex will be colored with no two same vertices share the same color, thereby reducing the channel interference.

In this work, we apply the Welsh Powell graph coloring algorithm to this problem. This greedy algorithm is well-suited for such applications as it prioritizes highly connected nodes and assign colors in a structured way.

II. BASIC THEORY

A. *Graph Theory*

In general, graph is defined as G = (V,E) with V as vertices and E as edges. V is a set of non-empty vertices, while E is a set of edges which connect a pair of vertices. In short, a graph must consist at least one vertex, but it is allowed to not have any edges at all. Additionally, the number of edges connected to that vertex is defined as degree.There are two types of graphs, namely simple graph and unsimple-graph. As it is sound, simple graph means that there are not any loops or multiple edges, whereas unsimple-graph has the opposite definition of simple graph.

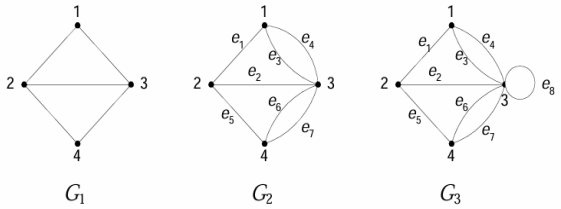


Fig 1. Simple and Unsimple graphs examples (source : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf)

Graph G_1 shows the instance of simple graph, whilst G_2 and G_3 represent the usimple graph. From G_2, the edges e_3 = (1,3) and e_4 = (1,3) are known as multiple or parallel edges because these two edges are connecting the same two vertices (1 and 3). Furthermore, in G_3, e_8 = (3,3) is called as a loop because it starts and ends at the same vertex.

Unsimple graph is divided into two categories, multi-graph which has multiple edges, but loops are not permitted and pseudo-graph which is allowed to contain both parallel edges and loops.

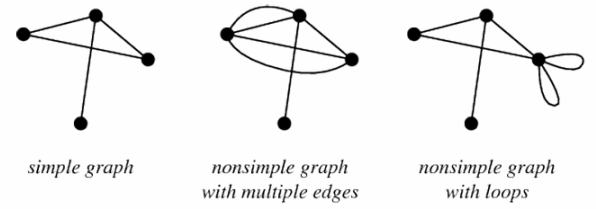


Fig 2. Types of unsimple graph
(source : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf)

Lastly, based on orientation of direction, there are undirected graphs and directed graphs. The difference between them is that in every edge of directed graph equipped with direction or an arrow.

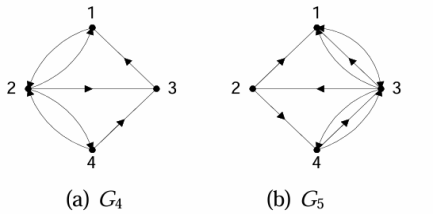
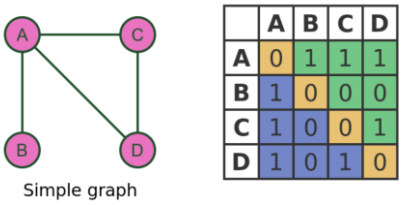


Fig 3. Directed graph examples
(source : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf)

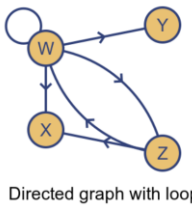
B. Adjacency Matrix

There are three ways to represent the graphs, namely adjacency matrix, incidency matrix, and adjacency list. However, in this paper, we simply apply adjacency matrix and list. A graph is called adjacency when two vertices are directly connected with an edge. Each element a_{ij} of the matrix is defined as 1 if vertex i is adjacent (connected) to vertex j, and 0 if they are not connected. This matrix is useful for understanding the structure of both undirected and directed graphs.



	A	B	C	D
A	0	1	1	1
B	1	0	0	0
C	1	0	0	1
D	1	0	1	0

Fig 4. Simple graph in adjacent matrix
(source: https://graphicmaths.com/computer-science/graph-theory/adjacency-matrices/)



	W	X	Y	Z
W	1	1	1	1
X	0	0	0	0
Y	0	0	0	0
Z	1	1	0	0

Fig 5. Directed graph in adjacent matrix
(source: https://graphicmaths.com/computer-science/graph-theory/adjacency-matrices/)

In the case of a simple (undirected) graph, the adjacency matrix is symmetric. For example, in the graph with vertices A, B, C, and D, the matrix shows that A is connected to B, C, and D; C is also connected to D; and so on. Since the connections go both ways in an undirected graph, if A is connected to B, then B is also connected to A. Hence, the matrix is mirrored along its diagonal.

On the other hand, for a directed graph, the adjacency matrix does not need to be symmetric. Here, each entry a_{ij} indicates a directed edge from vertex i to vertex j. For instance, vertex W has directed edges to W itself (a loop), as well as to X, Y, and Z. Meanwhile, vertex Z has outgoing edges to W and X. This directed relationship is clearly reflected in the matrix, where rows represent the source vertices and columns represent the destination vertices.

C. Adjacency List

Another way to represent the graph is using an adjacency list. Unlike the adjacency matrix, which uses 2D array, an adjacency list represents a graph as a collection of lists which contain the vertices that are directly connected. For instance, in an undirected graph, if vertex A is connected to vertices B and C, then the adjacency list will have an entry for A listing B and C, and entries for B and C listing A.

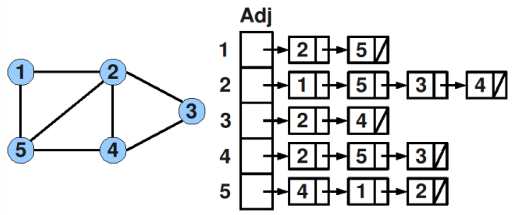


Fig 6. Simple graph in adjacent list
(source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/21-Graf-Bagian2-2024.pdf)

D. Isomorphic Graph

Two graphs that are structurally the same but look different geometrically are called isomorphic graphs.

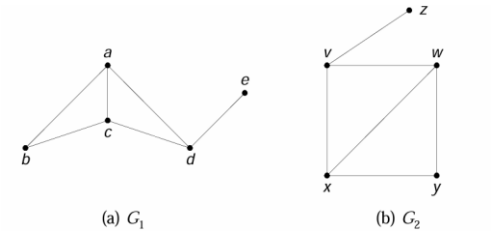


Fig 7. Isomorphic graph examples (source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/21-Graf-Bagian2-2024.pdf)

Graph G_1 and G_2 are said to be isomorphic because there is a one-to-one correspondence between their vertices and between their edges, such that the adjacent relationships are preserved. In other words, if an edge in graph G_1 connects vertices *u* and *v,* then the corresponding edge *e'* in graph G_2 must connect the corresponding vertices *u'* and *v'*.

E. Graph Coloring

In general, there are two ways to color a graph: edge coloring and vertex coloring. However, in this paper, we will focus solely on vertex coloring, as it directly relates to the assignment of wireless channels in a network. In vertex coloring, each vertex is assigned to a color, and two adjacent vertices are not allowed to share the same color

The primary objective of graph coloring is to minimize the number of colors used while satisfying the adjacent constraint. The minimum total of colors needed to color the graph is called chromatic numbers, denoted $\chi(G)$.

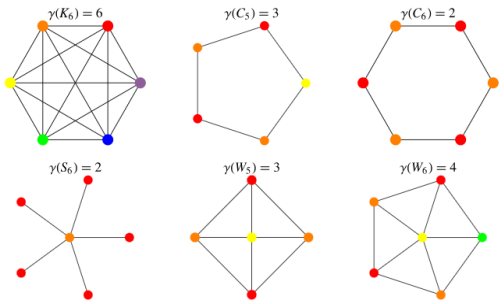


Fig 8. Graph with chromatic numbers (source: https://mathworld.wolfram.com/ChromaticNumber.html)

An empty graph has chromatic number 1 because all vertices are not connected, so we simply need one color to color the vertices. Complete graph—a simple graph in which every vertex connected to other vertices has chromatic number exact same as the total of vertices. For bipartite graph—the vertices can be split into two subsets, V_1 and V_2 have 2 chromatic numbers. Additionally, a circular graph—a simple graph in which each vertex has 2 degrees, for odd vertices $\chi(G) = 3$, whereas even vertices $\chi(G) = 2$. The rest of the graphs cannot be stated in general. Thus, we need Welsh-Powell Algorithm to discover the chromatic numbers of random graphs.

Graph coloring has a wide range of practical applications, for instance scheduling, compiler optimization, and wireless communication. Those fields are applying graph coloring methods to assign resource efficiently while avoiding conflict between connected elements.

F. Weighted Graph

A weighted graph is a type of graph where each edge is assigned a numerical value called a weight. These weights usually represent some form of cost, distance, time, capacity, or strength of connection between two vertices.

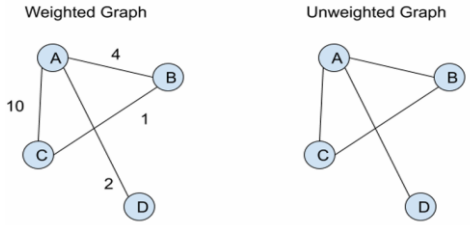


Fig 9. Weighted graph example (source : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf)

G. The Welsh-Powell Algorithm

The Welsh-Powell Algorithm is a popular greedy technique used to approximate a valid vertex coloring of a graph. There are several steps to apply this algorithm, first, we need to calculate the degree of each vertex and sort the vertices in descending order based on the degrees. Second, assign the first color to the highest degree of vertex. Then, set the same color as the previous color to the vertex which is not adjacent to the vertices that have been colored. Finally, once the current color cannot be used further, move on to the next uncolored vertex and assign a new different color. Those steps are repeated until all vertices have been specified with no such same color for vertex which is adjacent with others.

III. Implementation

This section presents the practical implementation of graph coloring techniques aimed at minimizing channel interference in wireless networks. The simulation is built upon the theoretical concepts discussed earlier, including graph modeling, weighted graphs, and the Welsh-Powell coloring algorithm. Specifically, the implementation involves constructing an interference graph based on the physical proximity of access points (APs), applying a graph coloring strategy to assign non-conflicting channels, and visualizing the results for clarity.

To accomplish this, we utilize the Python programming language along with several essential libraries. The NetworkX library is employed to manipulate and analyze the graph structures, while Matplotlib is used for graphical visualization of the network and its colored nodes. The standard math module supports geometric distance calculations (Euclidean distance), which is fundamental for

constructing the weighted interference graph. These tools collectively enable a comprehensive simulation of a wireless network and facilitate the evaluation of coloring strategies in a realistic, distance-aware environment. In the following parts, we provide detailed explanations of the python script which are used to build the graph.

A. *Setting Access Points Position*

The implementation begins with defining the spatial configuration of wireless networks. A dictionary named *ap_positions* is used to store the coordinate of each AP in a two-dimensional plane. These positions are vital because later they will be used to calculate the physical distance between APs using the Euclidean formula. The assumption here is that the spatial proximity of APs directly influences the likelihood of signal interference, which will be modeled through a weighted graph.

B. *Interference Detection Function*

To determine whether two access points are close enough to interfere with one another, a function called *is_interfering* is defined. The function takes two positions tuples as inputs and an optional parameter *threshold* which default to 15 meters. Within the function, the Euclidean distance between the two points is calculated using the formula $\sqrt{((x_1 - x_2)^2 + (y_1 - y_2)^2)}$. The function returns True if the calculated distance is less than or equal to the threshold.

```python
def is_interfering(pos1, pos2, threshold=15):
    x1, y1 = pos1
    x2, y2 = pos2
    return ((x1 - x2)**2 + (y1 - y2)**2)**0.5 <= threshold
```

Fig 10. Function of interference detection (source : Author's code)

C. *Constructing Interference Graph*

Furthermore, a weighted graph is constructed to represent the potential interference relationship between APs. The graph is initialized as an undirected graph using the NetworkX library.

```python
threshold = 15
G = nx.Graph()
for ap in ap_positions:
    G.add_node(ap)

for ap1 in ap_positions:
    for ap2 in ap_positions:
        if ap1 != ap2 and is_interfering(ap_positions[ap1], ap_positions[ap2],
        threshold):
            distance = round(((ap_positions[ap1][0] - ap_positions[ap2][0])**2 +
            (ap_positions[ap1][1] - ap_positions[ap2][1])**2) ** 0.5, 2)
            G.add_edge(ap1, ap2, weight=distance)
```

Fig 11. Code for constructing interference graph (source : Author's code)

Each AP is first added as node in the graph using a simple loop over the *ap_positions* dictionary. Later, a nested loop compares every possible pair of APs. For each pair (excluding self-comparisons), the *is_interfering* function is called to determine if the APs close enough to interfere. If they are, the Euclidean distance between them will be calculated and rounded to two decimal places. After that, an edge is added to the graph and form a weighted interference graph as a result. The presence of an edge signifies that two APs are withing interference range and the edge weight shows the physical distance between them.

D. *Implementing Welsh-Powell Algorithm*

The Welsh-Powell Algorithm is implemented to prevent two interfering APs using the same channel (colors). This greedy coloring algorithm begins with sorting the nodes of the graph in descending order based on their degree. A dictionary called *color_assignment* is used to store the assigned color for each AP. The algorithm iterates through the sorted list of nodes and assigns the first available color to each node, so it does not conflict with any of its neighbors. Once all nodes have been processed, the function will return the final color assignment which maps each AP to its assigned channel.

```python
def welsh_powell(graph):
    sorted_nodes = sorted(graph.nodes(), key=lambda x: len(graph[x]), reverse=True)
    color_assignment = {}
    color = 1
    for node in sorted_nodes:
        if node not in color_assignment:
            color_assignment[node] = color
            for other in sorted_nodes:
                if other not in color_assignment:
                    if all(color_assignment.get(n) != color for n in graph[other]):
                        color_assignment[other] = color
            color += 1
    return color_assignment
```

Fig 12. Function of Welsh-Powell algorithm (source : Author's code)

E. *Coloring and Chromatics Number*

The result of the Welsh-Powell Algorithm will be stored in *color_result* which is a dictionary that maps each AP to channel number (represented as color). The total number of channels used is stored in *chromatic_number* by converting the color values into a set and measuring its length. This value represents the minimum number of channels needed to ensure that no interfering APs share the same channel.

```python
color_result = welsh_powell(G)
chromatic_number = len(set(color_result.values()))

print("Channel Assignments:")
for ap in sorted(color_result):
    print(f"{ap} → Channel {color_result[ap]}")
print(f"\nChromatic Number (Number of Channels Used): {chromatic_number}")
```

Fig 13. Chromatics number output (source : Author's code)

F. *Generating Adjacency Matrix and List*

Moreover, to provide a more formal representation of graph structure, we added adjacency matrix and list for the output. To form an adjacency matrix, first, a sorted list of APs is created to standardize the order of nodes. Then, a dictionary *index_map*, maps each AP name to a numerical index. A square matrix of zeros is initialized using list comprehension. After that, the code loops through each edge and sets the corresponding entries in the matrix to 1, indicating that an edge exists between those two APs. The

matrix is symmetrical because we created an undirected graph.

In addition to the adjacency matrix, the adjacency list is also generated. It is formed by looping through each node in the graph and retrieving its neighbors using the *G.neighbors()* function. Each list of neighbors is sorted for readability and stored in a dictionary.

```python
nodes = sorted(G.nodes())
index_map = {node: i for i, node in enumerate(nodes)}
adj_matrix = [[0]*len(nodes) for _ in range(len(nodes))]

for u, v in G.edges():
    i, j = index_map[u], index_map[v]
    adj_matrix[i][j] = 1
    adj_matrix[j][i] = 1  # simetris

print("\nAdjacency Matrix:")
print("     " + "       ".join(nodes))
for i, row in enumerate(adj_matrix):
    print(f"{nodes[i]}  " + "       ".join(str(val) for val in row))


print("\nAdjacency List:")
adj_list = {node: sorted(list(G.neighbors(node))) for node in nodes}
for node in nodes:
    print(f"{node}: {adj_list[node]}")
```

Fig 14. Code for Adjacency matrix and list (source : Author's code)

IV. TESTING

To evaluate the performance and adaptability of the proposed graph coloring method in different deployment conditions, three test cases have been designed with distinct positions of access points (APs). Each scenario simulates a specific type of network density and signal overlaps.

A. Test Case 1

This test case will lead to the worst scenario, all APs are placed close enough that the distance between any two of them is less than or equal to the interference threshold. This leads to a complete graph, where every APs must use a unique channel

```python
ap_positions = {
    "AP1": (0, 0),
    "AP2": (5, 0),
    "AP3": (0, 5),
    "AP4": (5, 5),
    "AP5": (3, 2),
    "AP6": (2, 4)
}
```

Fig 15. (Test case 1) AP positions

```
Channel Assignments:
AP1 → Channel 1
AP2 → Channel 2
AP3 → Channel 3
AP4 → Channel 4
AP5 → Channel 5
AP6 → Channel 6

Chromatic Number (Number of Channels Used): 6

Adjacency Matrix:
     AP1       AP2       AP3       AP4       AP5       AP6
AP1  0         1         1         1         1         1
AP2  1         0         1         1         1         1
AP3  1         1         0         1         1         1
AP4  1         1         1         0         1         1
AP5  1         1         1         1         0         1
AP6  1         1         1         1         1         0

Adjacency List:
AP1: ['AP2', 'AP3', 'AP4', 'AP5', 'AP6']
AP2: ['AP1', 'AP3', 'AP4', 'AP5', 'AP6']
AP3: ['AP1', 'AP2', 'AP4', 'AP5', 'AP6']
AP4: ['AP1', 'AP2', 'AP3', 'AP5', 'AP6']
AP5: ['AP1', 'AP2', 'AP3', 'AP4', 'AP6']
AP6: ['AP1', 'AP2', 'AP3', 'AP4', 'AP5']
```

Fig 16. (Test case 1) Output

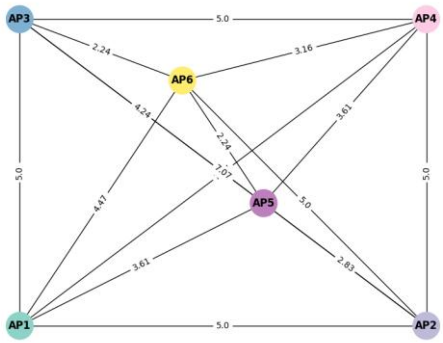


Fig 17. (Test case 1) Graph

The outcome is a fully connected graph, requiring the maximum number of distinct channels, $\chi(G) = 6$.

B. Test Case 2

This case simulates a more realistic layout. Some APs are close enough to interfere, while others are positioned far apart.

```python
ap_positions = {
    "AP1": (0, 0),
    "AP2": (10, 0),
    "AP3": (5, 10),
    "AP4": (25, 0),
    "AP5": (30, 10),
    "AP6": (50, 5)
}
```

Fig 18. (Test case 2) AP positions

```
Channel Assignments:
AP1 → Channel 2
AP2 → Channel 1
AP3 → Channel 3
AP4 → Channel 2
AP5 → Channel 1
AP6 → Channel 1

Chromatic Number (Number of Channels Used): 3

Adjacency Matrix:
     AP1  AP2  AP3  AP4  AP5  AP6
AP1   0    1    1    0    0    0
AP2   1    0    1    1    0    0
AP3   1    1    0    0    0    0
AP4   0    1    0    0    1    0
AP5   0    0    0    1    0    0
AP6   0    0    0    0    0    0

Adjacency List:
AP1: ['AP2', 'AP3']
AP2: ['AP1', 'AP3', 'AP4']
AP3: ['AP1', 'AP2']
AP4: ['AP2', 'AP5']
AP5: ['AP4']
AP6: []
```

Fig 19. (Test case 2) Output

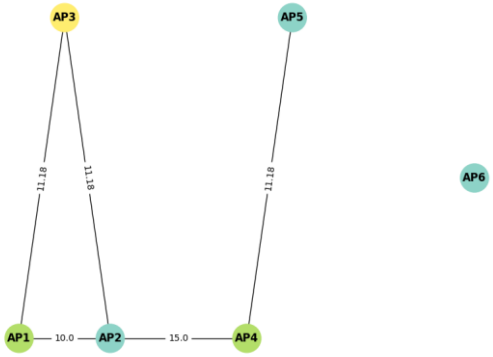


Fig 20. (Test case 2) Graph

The graph shows that several APs are placed far apart from each other, resulting in a sparse interference graph. For instance, there are no edges connected to AP6 which indicates that the distance between AP6 and all other APs exceed the threshold and thus is not considered to interfere with any of them. As a result, AP6 can safely use channel 1, same as AP2 and AP5.

C. Test Case 3

In this scenario, every AP is deliberately placed at distance greater than the interference threshold.

```
ap_positions = {
    "AP1": (0, 0),
    "AP2": (20, 0),
    "AP3": (0, 25),
    "AP4": (25, 25),
    "AP5": (50, 0),
    "AP6": (75, 25)
}
```

Fig 21. (Test case 3) AP positions

```
Channel Assignments:
AP1 → Channel 1
AP2 → Channel 1
AP3 → Channel 1
AP4 → Channel 1
AP5 → Channel 1
AP6 → Channel 1

Chromatic Number (Number of Channels Used): 1

Adjacency Matrix:
     AP1  AP2  AP3  AP4  AP5  AP6
AP1   0    0    0    0    0    0
AP2   0    0    0    0    0    0
AP3   0    0    0    0    0    0
AP4   0    0    0    0    0    0
AP5   0    0    0    0    0    0
AP6   0    0    0    0    0    0

Adjacency List:
AP1: []
AP2: []
AP3: []
AP4: []
AP5: []
AP6: []
```

Fig 22. (Test case 3) Output

Fig 23. (Test case 3) Graph

The result is a graph with no edges (empty graph). Thus, every APs can share the same channel, $\chi(G) = 1$.

V. Conclusion

Graph coloring offers a structured and efficient solution to the challenge of minimizing channel interference in wireless networks. By assuming access points (APs) as vertices and modelling interference through edges based on physical distance, the frequency assignment problem can be transformed into mathematical framework.

Through the implementation of the Welsh-Powell Algorithm, frequency channels are assigned to avoid conflicts between interfering APs, whilst also minimizing the total number of channels used. The results of the three test cases show that physical layout of APs significantly influences the channel efficiency (chromatic number).

The visualizations of the graphs, adjacent matrices and lists provide intuitive and analytical confirmation of the algorithm's behavior. These representations also make the observations clearer of how interference spreads spatially across different AP arrangements.

To conclude, by leveraging additional Python libraries, NetworkX and Matplotlib, it is possible to simulate and visualize graph-based models of wireless networks effectively. The integration of theoretical foundations with practical implementation allows the simulation to minimize the number of channels required and adapt efficiently to various network configurations.

VI. APPENDIX

Github Link:
https://github.com/stefaniangeline/Makalah-Matematika-Diskrit.git

Youtube Link:
https://youtu.be/XjBWSZc8XI0?si=q-uO9IdpKFk_C5mP

VII. ACKNOWLEDGEMENT

First of all, author would like to thank God Almighty for giving the health and strength to complete this paper, titled "Minimizing Channel Interference in Wireless Networks Using Graph Coloring Techniques".

The author also extends appreciation to Mr. Arrival Dwi Sentosa, S.Kom.,M.T., lecturer of Discrete Mathematics (IF1220) for class K02, owing to his guidance and continuous support throughout the learning process.

The deepest thanks are also addressed to the author's parents, whose support and encouragement have been a constant source of motivation to finish this paper.

Lastly, the author would like to thank all friends and classmates who have contributed through discussion, collaboration, and moral support during the writing of this paper.

REFERENCES

[1] Munir, Rinaldi. 2023. "Graf (Bag. 1)". Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf.[Accessed: 16 June 2025].

[2] Munir, Rinaldi. 2023. "Graf (Bag. 2)". Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian2-2024.pdf. [Accessed: 16 June 2025].

[3] Munir, Rinaldi. 2023. "Graf (Bag. 3)". Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/22-Graf-Bagian3-2024.pdf. [Accessed: 16 June 2025].

[4] M. Mujahid, "Graph Coloring with NetworkX," *Medium*, Aug. 22, 2018. Available: https://medium.com/data-science/graph-coloring-with-networkx-88c45f09b8f4. [Accessed: 17 June 2025].

[5] Mayank, Mohit. 2021. "Visualizing Networks in Python". Available: https://towardsdatascience.com/the-new-best-python-package-for-visualising-network-graphs-e220d59e054e/. [Accessed: 17 June 2025].

[6] McBride, Martin. 2023. "Adjacency matrices". Available: https://graphicmaths.com/computer-science/graph-theory/adjacency-matrices/. [Accessed: 16 June 2025].

[7] Weisstein, Eric W. 2025. "Chromatic Number". Available: https://mathworld.wolfram.com/ChromaticNumber.html. [Accessed: 16 June 2025].

STATEMENT

I hereby declare that the paper I wrote is my own writing, not an adaptation or translation of someone else's paper, and it is not plagiarized.

Bandung, 20 June 2025

Stefani Angeline Oroh, 13524064