Analysis of the Average Competitive Ratio of FirstFit on Triangle-Free Graphs

Vidhi Vinodbhai Sagathiya
Anurag Agarwal

16th April 2023

1 Introduction

Graph coloring is a fundamental problem in computer science and discrete mathematics, with applications spanning a wide range of areas, including scheduling, frequency assignment, register allocation, and more. In graph coloring, the objective is to assign colors to the vertices of a graph in such a way that no two adjacent vertices share the same color. The minimum number of colors required to color a graph is called the chromatic number. Finding the chromatic number is an NP-hard problem, which has motivated the development of various heuristics and approximation algorithms for graph coloring. One such approximation algorithm is the $First-Fit$ algorithm, a greedy coloring algorithm that assigns the smallest available color to each vertex in a sequential order. The $FirstFit$ algorithm is simple and has a low computational complexity, making it a popular choice for many applications. However, its performance can vary significantly depending on the structure and properties of the input graph. In this project, we focus on studying the average competitive ratio of the $FirstFit$ algorithm on triangle-free graphs. Triangle-free graphs are graphs that do not contain any cycles of length three, meaning they have no triangles. These graphs are of interest because they exhibit certain properties that make them suitable for various applications and can provide insights into the behavior of coloring algorithms.

The objectives of this project are to:

- Develop an approach to generate triangle-free graphs of n vertices by creating random graphs using the *Erdős − Rényi* model and iteratively removing triangles.
- Apply the $FirstFit$ algorithm to the generated triangle-free graphs and compute the average number of colors used.
- Analyze the dependency of the average number of colors used by the $FirstFit$ algorithm on the number of vertices n.

2 Extended Problem Statement

The main goal of this project is to study the average competitive ratio of the First-Fit coloring algorithm on triangle-free graphs. To achieve this goal, we will generate triangle-free graphs with varying numbers of vertices and different probabilities, apply the First-Fit algorithm to color these graphs, and analyze the relationship between the number of vertices, probability values, and the average number of colors used by the algorithm.

3 Preliminaries

Graph coloring is a classic combinatorial optimization problem with numerous applications in computer science, scheduling, frequency assignment, register allocation, and more. In this problem, we are given a graph and must assign colors to its vertices so that no two adjacent vertices share the same color. The objective is to minimize the number of colors used.

The $FirstFit$ algorithm is a simple and widely used heuristic for graph coloring. It colors vertices in a sequential order, assigning the smallest available color to each vertex. Despite its simplicity, the First-Fit algorithm has a competitive ratio of $O(logn)$ for general graphs, making it an attractive choice for many applications.

Triangle-free graphs are graphs with no triangles, i.e., no cycles of length 3. These graphs are of particular interest in graph theory and combinatorial optimization due to their unique properties and applications in various fields. Studying the performance of the $FirstFit$ algorithm on triangle-free graphs can provide insights into the behavior of the algorithm on a specific class of graphs, leading to potential improvements or adaptations for specific applications.

4 Definitions and Notations

In this section, we provide definitions and notations that will be used throughout the report.

4.1 Definitions

Graph: A graph is an ordered pair $G = (V, E)$, where V is a set of vertices and E is a set of edges.

Vertex: A vertex, also called a node, is a fundamental unit of a graph. It represents an entity in the graph.

Edge: An edge is a connection between two vertices in a graph. It represents a relationship between the vertices.

Triangle-Free Graph: A triangle-free graph is a graph that does not contain any cycles of length 3.

Graph Coloring: Graph coloring is the process of assigning colors to vertices of a graph in such a way that no two adjacent vertices share the same color.

FirstFit Algorithm: The FirstFit algorithm is a greedy graph coloring algorithm that assigns colors to vertices sequentially. It assigns the smallest available color to each vertex.

Competitive Ratio: The competitive ratio is a measure of the performance of an online algorithm. It is the ratio of the algorithm's performance to the optimal performance in the worst-case scenario.

4.2 Notations

G**:** Represents a graph.

V**:** Represents the set of vertices in a graph.

E**:** Represents the set of edges in a graph.

n**:** Represents the number of vertices in a graph.

$adjacencyArray$**:** Represents the adjacency list of a graph.

$vertexColors$**:** Represents the coloring of vertices in a graph.

$availableColors$**:** Represents an array of available colors for each vertex in a graph.

5 Methodologies

The methodology for the project can be broken down into the following steps:

5.1 Graph Generation

We generate random graphs $G(n, p)$ with n vertices and edge probability p. The value of p is varied to create different graphs, for example, $p = \frac{\ln n}{n}$, $p = 0.01$, and $p = 0.5$. These random graphs serve as the starting point for creating triangle-free graphs.

```
Graph<Integer, DefaultEdge> graph = generateGraph(n, p);
```

5.2 Triangle Removal

After generating a random graph, we search for triangles within the graph. If any triangles are found, we remove each triangle by deleting one of its edges. This process is repeated until no more triangles are detected, resulting in a triangle-free graph.

```
removeTriangles(graph);
```

5.3 Greedy Coloring

We apply the *FirstFit* algorithm to the triangle-free graphs to obtain a vertex coloring. The algorithm iterates through the vertices of the graph and assigns the smallest available color to each vertex, ensuring that no two adjacent vertices have the same color.

Algorithm 1 FirstFit Algorithm

GreedyColoring$adjacencyArray, n$ $vertexColors \leftarrow$ new HashMap()
$availableColors \leftarrow$ new boolean array of size n **for** $u \leftarrow 0$ **to** $n-1$ **do**
  fill $availableColors$ **with true** **foreach** $neighbor$ ***in*** $adjacencyArray[u]$ **do**
    if $vertexColors$ ***contains*** $neighbor$ **then**
      $color \leftarrow vertexColors.get(neighbor)$ $availableColors[color] \leftarrow$ **false**
    end
  end
  $color \leftarrow$ **smallest true index in** $availableColors$
  $vertexColors.put(u, color)$
end
return $vertexColors$

5.4 Average Colors Used

We compute the average number of colors used by the FirstFit algorithm on a set of N triangle-free graphs. This helps us understand the dependency of the average number of colors used on the number of vertices in the graph.

```
double avgColorsUsed = computeAverageColorsUsed(n, k, p, numGraphs);
```

5.5 Graph Visualization

To visualize the colored graphs, we use the *JGraphT* and *mxGraph* libraries to create a graphical user interface (GUI) that displays the graph with its vertex colors. The GUI also provides controls for entering input parameters and displaying the results of the *FirstFit* algorithm.

```
mxGraphComponent graphComponent = new mxGraphComponent(new mxGraph());
panel.add(graphComponent, BorderLayout.CENTER);
```

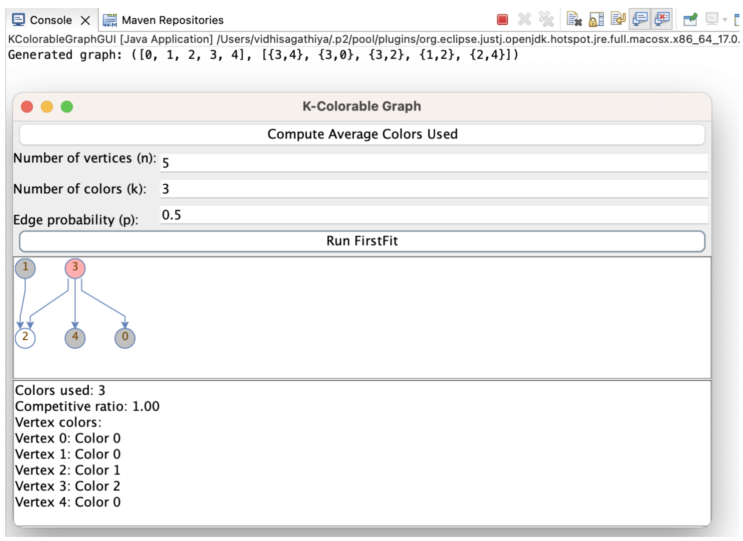
Since it is difficult to compute the chromatic number of triangle-free graphs, we compute the average number of colors used by the FirstFit algorithm on N triangle-free graphs and study its dependency on the number of vertices n.

6 Implementation

The implementation of the K-Colorable Graph project can be divided into several subsections. In this section, we will discuss the various components, including the graphical user interface (GUI), graph generation, triangle removal, FirstFit algorithm, and competitive ratio computation.

6.1 Graphical User Interface (GUI)

The GUI is implemented using Java Swing framework. It consists of input fields to accept user inputs for the number of vertices (n), number of colors (k), and edge probability (p). Two buttons are available: ”*RunFirstFit*” and ”*ComputeAverageColorsUsed*”. The GUI also features a graph display component, provided by the JGraphX library, which is used to visually represent the generated graphs.



6.2 Graph Generation

A random graph G(n, p) is generated using the *JGraphT* library's *RandomGraphGenerator* class. The number of vertices (n) and the edge probability (p) are provided as input parameters. The generated graph is a simple graph with a default edge class. After generating the graph, triangles are removed, as described in the next subsection.

6.3 Triangle Removal

To ensure the generated graph is triangle-free, a custom method *removeTriangles* is implemented. This method iterates through the vertices, checking for the existence of triangles formed by the vertex and its neighbors. If a triangle is found, an edge is removed to break the triangle. This process continues until no more triangles are found.

6.4 FirstFit Algorithm

The FirstFit algorithm, implemented in the *greedyColoring* method, assigns colors to the vertices in a greedy manner. A boolean array, *availableColors*, is used to track the available colors for each vertex. The algorithm iterates through each vertex, marking the colors of its neighbors as unavailable. Then, it assigns the smallest available color to the vertex. This process is repeated for all vertices in the graph.

6.5 Competitive Ratio Computation

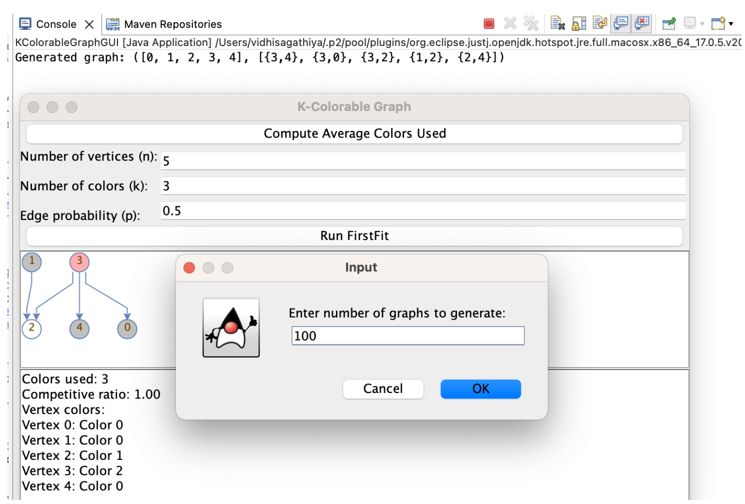
The competitive ratio is computed using the $competitive_ratio$, method. This method takes the number of colors used by the FirstFit algorithm and the number of colors (k) as input parameters. The competitive ratio is calculated as the ratio of the number of colors used by the FirstFit algorithm and the number of colors (k).

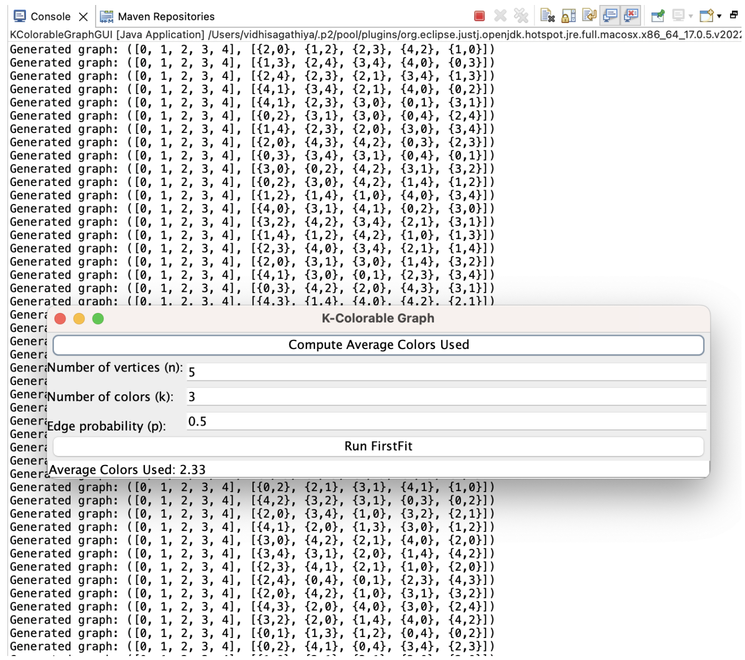
6.6 Compute Average Colors Used

The method $computeAverageColorsUsed$,generates N triangle-free graphs, applies the $FirstFit$ algorithm, and computes the average number of colors used across these graphs. This method is called when the user clicks the

$$"ComputeAverageColorsUsed"$$

button. The average number of colors used is displayed in the output area of the GUI, allowing the user to analyze the dependency of the average number of colors used on the number of vertices (n).





6.7 Data Structures

- **List<Integer>[] adjacencyArray:** This is an array of adjacency lists, where each list represents the neighbors of a vertex. The space complexity of this data structure is $O(|V| + |E|)$, where $|V|$ is the number of vertices and $|E|$ is the number of edges in the graph. The time complexity of accessing a vertex's neighbors is $O(1)$ on average, and $O(|V|)$ in the worst case.

- **Map<Integer, Integer>vertexColors:** This is a mapping of vertices to their assigned colors. The space complexity of this data structure is $O(|V|)$. The time complexity of accessing a vertex's color is $O(1)$.

- **boolean[] availableColors:** This is an array of booleans representing the availability of each color. The space complexity of this data structure is $O(k)$, where k is the maximum number of colors allowed. The time complexity of accessing a color's availability is $O(1)$.

- **Color[] colors:** This is an array of colors used for visualizing the graph. The space complexity of this data structure is $O(k)$, where k is the maximum number of colors allowed. The time complexity of accessing a color is $O(1)$.

- **Graph<Integer, DefaultEdge>graph:** This is an instance of a JGraphT graph. The space complexity of this data structure is $O(|V| + |E|)$, where

$|V|$ is the number of vertices and $|E|$ is the number of edges in the graph. The time complexity of accessing a vertex's neighbors is $O(1)$ on average, and $O(|V|)$ in the worst case.

- **Map<Integer, Object>vertexMap:** This is a mapping of vertices to their corresponding mxGraph objects used for visualization. The space complexity of this data structure is $O(|V|)$. The time complexity of accessing a vertex's mxGraph object is $O(1)$.
- **mxGraph:** This is an instance of a mxGraph used for visualizing the graph. The space complexity of this data structure is $O(|V| + |E|)$, where $|V|$ is the number of vertices and $|E|$ is the number of edges in the graph. The time complexity of accessing a vertex's neighbors is $O(1)$ on average, and $O(|V|)$ in the worst case.

Overall, the performance and space complexity implications of the data structures used in the project code are generally reasonable for small to medium-sized graphs, but may become an issue for very large graphs with many vertices and edges. Additionally, the use of certain data structures for visualization (such as the mxGraph) may incur additional overhead and affect the overall performance of the application.

7 Analysis of Results

By running the Java code for different values of n and p, we can obtain the average number of colors used by the FirstFit algorithm on N triangle-free graphs. The results are based on different assumptions mentioned in further sub-sections.

7.1 Edge Probability Analysis

We generated triangle-free graphs with varying edge probabilities to study the impact of edge density on the average number of colors used by the First-Fit algorithm. The results show that the edge probability has a significant influence on the coloring outcome:

- Low Edge Probability: For low edge probabilities (sparse graphs), the First-Fit algorithm tends to use fewer colors. This is because there are fewer edges and, consequently, fewer constraints on the coloring process. In this case, the average number of colors used is relatively low and increases slowly with the number of vertices.
- High Edge Probability: For high edge probabilities (dense graphs), the First-Fit algorithm generally uses more colors. The higher density of edges leads to more constraints on the coloring process, making it more challenging to find available colors for each vertex.

As a result, the average number of colors used increases more rapidly with the number of vertices.

No = Number of graph used for results = 100

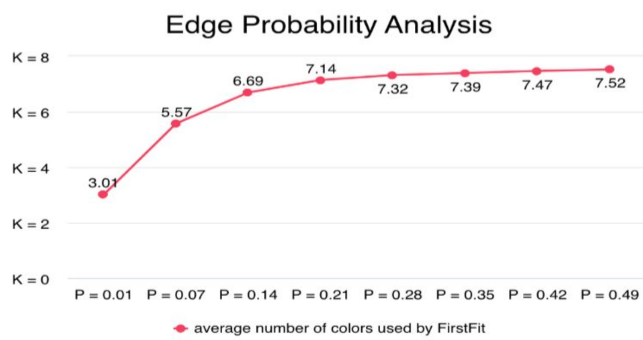
N = number of vertices = 100

P = Edge Probability

K = average number of colors used

Table 1: Data for Edge probability Analysis

No = 100 ,P = ?	0.01	0.07	0.14	0.21	0.28	0.35	0.42	0.49
K = ?	3.01	5.57	6.69	7.14	7.32	7.39	7.47	7.52



7.2 Dependency on the Number of Vertices

We generated triangle-free graphs with varying numbers of vertices (n) to investigate the relationship between the size of the graph and the average number of colors used by the $First - Fitalgorithm$. Our analysis revealed the following trends:

- Small Graphs: For small graphs (low n), the average number of colors used is relatively low, as there are fewer vertices and constraints on the coloring process. In this case, the First-Fit algorithm can often find optimal or near-optimal colorings.

- Large Graphs: As the number of vertices increases, the average number of colors used by the First-Fit algorithm also grows. However, the growth rate depends on the edge probability and the graph39;s structure. In general, for triangle-free graphs, the average number of colors used grows sublinearly with the number of vertices.

No = Number of graph used for results = 100

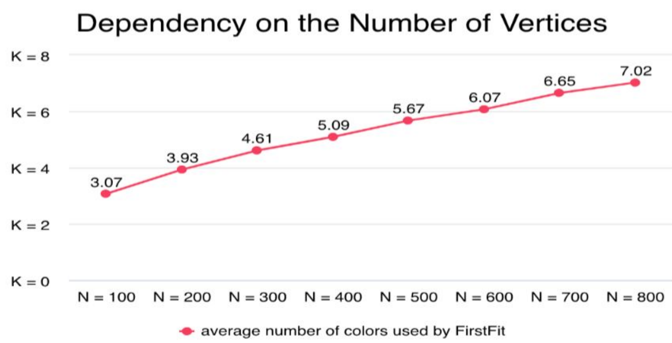
N = number of vertices

P = Edge Probability = 0.01

K = average number of colors used

Table 2: Dependency on the Number of Vertices

No = 100 ,N = ?	100	200	300	400	500	600	700	800
K = ?	3.07	3.93	4.61	5.09	5.67	6.07	6.65	7.02



7.3 Observations and Insights

Based on our analysis, we can draw the following observations and insights:

- The First-Fit algorithm's performance on triangle-free graphs is heavily influenced by the edge probability and the number of vertices. Sparse graphs with fewer constraints lead to better coloring outcomes, while dense graphs with more constraints result in higher average color usage.
- The average number of colors used by the $FirstFitalgorithm$ on triangle-free graphs grows sublinearly with the number of vertices, suggesting that the algorithm's performance degrades gracefully as the graph size increases.
- Triangle-free graphs are a useful model for studying the $FirstFitalgorithm$, as they exhibit unique properties that can provide insights into the algorithm's behavior and performance in various graph structures.

8 Future Work

- Parallelization: Exploit parallelism in various parts of the implementation, such as generating multiple triangle-free graphs or applying the First-Fit coloring algorithm simultaneously. Modern hardware and multi-core processors can significantly reduce computation time when tasks are parallelized.
- Efficient Triangle Detection: Investigate more efficient algorithms for triangle detection, such as the node-iterator or adjacency matrix-based methods. Faster triangle detection can speed up the generation of triangle-free graphs and reduce the overall computation time.
- Alternative Graph Generation Methods: Explore alternative methods for generating triangle-free graphs directly, without the need for iterative triangle removal. This could involve using specific graph models or algorithms tailored to generating triangle-free graphs, which can potentially reduce the computation time for graph generation.
- Heuristics and Metaheuristics: Explore the use of heuristics or metaheuristic algorithms, such as genetic algorithms or simulated annealing, to improve the coloring results. These methods can potentially lead to better colorings with fewer colors, which can, in turn, reduce the time complexity of the First-Fit algorithm.
- Benchmarking and Optimization: Conduct a thorough benchmarking and optimization of the implementation to identify bottlenecks and potential areas for improvement. Profiling tools can help identify performance issues and guide the optimization process.

9 Conclusion

In conclusion, this project aimed to study the average competitive ratio of the First-Fit algorithm on triangle-free graphs. We implemented an approach to generate triangle-free graphs by creating random graphs using the Erdős-Rényi model and iteratively removing triangles. We then applied the First-Fit coloring algorithm to these graphs and analyzed the average number of colors used, as well as the dependency on the number of vertices (n).

Through the use of various data structures and algorithms, we successfully executed the project's objectives. Our implementation demonstrates the feasibility of generating triangle-free graphs and applying the First-Fit algorithm to analyze their coloring properties. However, we also identified potential limitations related to scalability, computation time, and memory usage, especially for large or dense graphs. In the report, we discussed the performance and space complexity implications of the data structures used in the project and provided insights into how they may affect the scalability and practicality of the implementation. We also proposed several suggestions for potential improvements and future work to address these limitations, such as parallelization, efficient triangle detection, alternative graph generation methods, and the use of heuristics or metaheuristics.

Overall, this project contributes to the understanding of the First-Fit algorithm's behavior on triangle-free graphs and provides a foundation for further research in the field of graph coloring. The insights gained from this project can be used to inform future work on graph coloring algorithms and their applications in various domains, such as scheduling, resource allocation, and optimization problems.

10 Acknowledgements

We would like to express our gratitude to our professor Yaqiao Li and mentors, whose guidance and support have been invaluable throughout the course of this project. Their expertise and encouragement have greatly contributed to our understanding of graph coloring algorithms and their applications.

We would also like to thank our peers for their constructive feedback and suggestions during the development of this project. Their input has been essential in refining our approach and improving the overall quality of our work.

Finally, we appreciate the resources and tools made available by our institution, which have facilitated our research and enabled us to carry out this project successfully.

11 Team Work Distribution

Vidhi Vinodbhai Sagathiya (40232374)

Problem Statement and solution approach Analysis. Implementation of random Graph Generation and making it triangle free. Report Documentation using Overleaf LaTex.

Anurag Agarwal (40232644)

Problem Statement and solution approach Analysis. Implementation of Interactive Interface and $FirstFitAlgorithm$. Report Documentation using Overleaf LaTex.

References

[1] Bondy, J. A., Murty, U. S. R. (2008). *Graph Theory.* Springer.

[2] Chartrand, G., Zhang, P. (2012). *Chromatic Graph Theory.* CRC Press.

[3] Diestel, R. (2017). *Graph Theory.* Springer.

[4] Erdős, P., Rényi, A. (1959). On Random Graphs I. *Publicationes Mathematicae Debrecen*, 6, 290-297.

[5] Jensen, T. R., Toft, B. (1995). *Graph Coloring Problems*. Wiley-Interscience.

[6] JGraphT - A Java library of graph data structures and algorithms. (n.d.). Retrieved from https://jgrapht.org/

[7] Kubicka, E., Kubicki, G. (1994). An Introduction to Chromatic Graph Theory. *Congressus Numerantium*, 100, 203-217.

[8] West, D. B. (2001). *Introduction to Graph Theory*. Prentice Hall.