Comparative Analysis of Online Graph Coloring Problem

RAJAT SHARMA, 40196467, Concordia University
AMANPREET SINGH, 40221947, Concordia University
TRIPT DHUDI, 40248554, Concordia University
ASHISH UPADHYAY, 40225754, Concordia University
VIKRAM SINGH BRAHM, 40241024, Concordia University

1 PROBLEM DESCRIPTION

Our project is on the Online graph coloring problem, which is a well-known problem in graph theory, where vertices and their corresponding edges are revealed in an online fashion. The objective of an online graph coloring algorithm is to assign a proper color to each vertex upon its arrival, using as few colors as possible, with no two adjacent vertices sharing the same color.

In this paper, we focus on the FirstFit and CBIP algorithms for coloring online graphs and study their average competitive ratios on k-colorable online graphs for k = 2, 3, 4.

Competetive ratio is the ratio of output of our greedy algorithms of First-Fit algorithm and the CBIP algorithm and **k** which is the chromatic number of the graph.

This study performs an empirical study on the performance of the already existing algorithms of First-Fit algorithm and CBIP algorithms by calculating the competetive ratios for 2, 3, and 4 colorable graphs. The k-colorable graph problem has a great number of applications in the fields of resource allocation, network routing, job scheduling etc.

This study will be computationally expensive owing to the fact that although the study is performed on greedy algorithms we are conducting this study on 1000s of graphs

2 IMPLEMENTATION

In this section, we will describe the implementation details of our project, which consists of a backend developed in Python using the Flask framework and a frontend developed in HTML, CSS, and JavaScript. We also utilized visualization libraries like D3.js to display the results of the online graph coloring algorithms.

2.1 Backend (Python Flask)

2.1.1 Graph Generation [2]: We implemented a function to generate k-colorable online graphs based on the user's input for the number of vertices (n), the chromatic

Authors' addresses: Rajat Sharma, 40196467, Concordia University; Amanpreet Singh, 40221947, Concordia University; Tript Dhudi, 40248554, Concordia University; Ashish Upadhyay, 40225754, Concordia University; Vikram Singh Brahm, 40241024, Concordia University.

number (k), and the number of instances (N). For this, we used the Graph Generator Programs mentioned in the references.

2.1.2 Coloring Algorithm [1] : We implemented the FirstFit and CBIP algorithms as separate functions. These functions receive an online graph as an adjacency list as input and return the dictionary of colors used.

2.1.3 Routing: We set up Flask routes to handle API requests for generating graphs and running the coloring algorithms. These routes receive the user's input from the frontend and return the results in a JSON format.

2.2 Frontend (HTML, CSS, JavaScript)

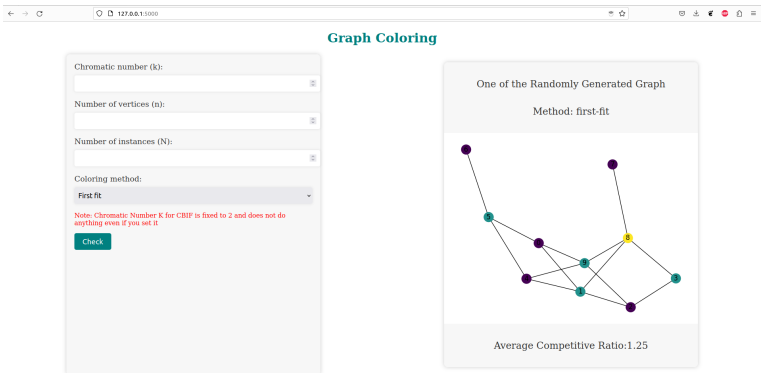

Fig. 1. The Complete Project Implementation

2.2.1 User Interface: We designed an interactive user interface with input boxes for the chromatic number (k), the number of vertices (n), and the number of graph instances (N). We also provided buttons to choose between the FirstFit and CBIP algorithms.

2.2.2 Visualization: We used the D3.js library to visualize the online graph generation process and display the results, including the colors used by the algorithms, the competitive ratio, and other relevant statistics.

2.2.3 API Calls: We implemented JavaScript functions to make asynchronous calls to the backend Flask API routes, passing the user's input and receiving the results in JSON format.

- The user sets the parameters (k, n, and N) and selects the desired coloring algorithm (FirstFit or CBIP).
- The frontend sends a request to the backend with the selected parameters.

- The backend generates the k-colorable online graphs and runs the selected algorithm on each graph.
- The backend calculates the average competitive ratio and returns the results to the frontend.
- The frontend visualizes the results, including the colored graphs and their statistics, allowing the user to explore the performance of the algorithms and understand the dependency of the average competitive ratio on the number of vertices.

This implementation allowed us to efficiently study the average competitive ratios of the FirstFit and CBIP algorithms on k-colorable online graphs, providing valuable insights into their performance and enabling us to draw conclusions about their behavior.

3 ALGORITHMS

3.1 Greedy Algorithms

In the greedy algorithm approach we make a locally optimal choice at each and every stage of the algorithm in hope that our greedy algorithm approach will lead to us finding the global optimum solution to the problem. Here, no backtracking is possible and each decision taken at any point of the algorithm is final and irrevocable.

Salient Features

- Greedy algorithms are easy to implement.
- They might not be able to always produce optimal solutions which can be seen from our implementations of the First-Fit and CBIP algorithm.
- They mostly have lower time complexities as compared to other approaches to solve the same problems.

For our empirical study we deal with two graph coloring algorithms

3.1.1 First-Fit Graph Coloring Algorithm [3]. The First-Fit Graph coloring algorithm is greedy algorithm that tries to m minimize the number of used colors by assigning the first available color that has not been already assigned to any of the neighbors of the vertices. If there exists no available color that can be assigned to a vertex the algorithm assigns a new color. And this is how this algorithm minimizes the number of colors assigns to the graph

Working of the algorithm

- Initialize an empty dictionary of colors.
- Repeat the next three steps for all vertices in the graph
- Get the smallest number that has not been used to color the neighboring vertex of v.
- Now color the vertex with the smallest number found in the initial step.

- If there is no color available to color the vertex as all the colors have been used by it's neighbors then add a new color to the list and color the vertex with that color.

Algorithm 1 First-Fit Greedy Algorithm for Graph Coloring

Input: A graph in the form of Adjacency List called G

Output: A dictionary $Colors$ containing the vertex-color assignments

▷ Initialize an empty dictionary to store the color assignments for each vertex

$colors \leftarrow \{\}$

foreach $vertex$ *in* G **do**

▷ Create a set of colors already been assigned to the vertex's neighbors

$used\_colors \leftarrow \{colors[neighbor] : neighbor \in G[vertex] \text{ and } neighbor \in colors\}$

▷ Initialize a set of available colors

$available\_colors \leftarrow \{1, 2, \ldots, |G|\} - used\_colors$

▷ If there are available colors, assign the first one in the set

if $available\_colors \neq \{\}$ **then**

$colors[vertex] \leftarrow \min(available\_colors)$

end

▷ If there are no available colors, assign a new color to the vertex

else

$colors[vertex] \leftarrow |G| + 1$

end

end

return $colors$

3.1.2 CBIP Greedy Graph Coloring Algorithm. The CBIP or Coloring by Bipartite Partitioning algorithm is an online greedy algorithm in which the vertices are covered in the way they arrive in an online fashion. In this algorithm the offline chromatic number is assumed **k** to known.

Working of the algorithm

- Initialize an empty list called color_list to store the colors of vertices.
- Choose any vertex from G and assign it the color 1.
- Perform a breadth-first search (BFS) starting from the chosen vertex.
- As we traverse the graph using BFS, assign the opposite color (2 if the current vertex is colored 1 and 1 if it is colored 2) to all the neighbors of the current vertex.

- If at any point we encounter a neighbor that is already assigned the same color as the current vertex, then the graph is not 2-colorable and we can stop.
- Repeat steps 3 to 5 until all vertices are colored.
- Return the list of colors assigned to each vertex.

Algorithm 2 CBIP Greedy Graph Coloring Algorithm Part 1

Input: Graph G with n vertices and k colors

Output: List of n integers representing the color of each vertex

```
Function bipartite_is(graph)
    i_set ← {} , i_set_2 ← {} , visited ← {} , st_v ← n − 1
    i_set.add(st_v), visited.add(st_v)
    qu ← [st_v]
    while qu is not empty do
        ver ← qu.pop(0)
        for neighbor in graph[ver] do
            if neighbor not in visited then
                if ver in i_set then
                    i_set2.add(neighbor);
                else
                    i_set.add(neighbor)
                visited.add(neighbor)
                qu.append(neighbor)
    return list(i_set), list(i_set2);
```

Algorithm 3 CBIP Greedy Graph Coloring Algorithm Part 2

```
Function CBIP_is(graph)
    find_minimum_integer ← lambda function that returns the smallest integer
not in a given list
    G ← nx.Graph(G)
    n ← length of G nodes
    if n < 1 then
        return empty list;
    color_list ← list containing 1 as the first element
    for i ← 1 to n do
        other_part_colors ← empty list
        other_part ← the second independent set returned by
        bipartite_is(G.subgraph(range(i + 1)))
        for j ∈ other_part do
            other_part_colors.append(color_list[j])
        color ← find_minimum_integer(other_part_colors)
        color_list.append(color)
    return color_list;
```

4 GRAPH GENERATION

The procedure we use for generating graphs will generate a k-colorable graph making it possible for us color the graph with k or lesser number of colors. So, it is entirely possible that the chromatic number we get for the generated graph from the proposed feature to be less than k depending on the probability that we will generate randomly.

We carry out the process of partitioning of the vertices such that each subset has the number of vertices n/k. Doing this will make it possible for the number of vertces to be **k** in number and in turn the graph would become a **k-colorable** graph.

Working of the algorithm

- Divide the set of vertices into **k** disjoint sets.
- For each vertex v_i and v_j we check if they belong to the same subset. If no, we create an edge between v_i and v_j with a probability p.
- By varying p we are able to generate multiple k-colorable graphs.

Algorithm 4 Algorithm to Generate a Random k-Colorable Graph

Input: Number of vertices n, number of colors k

Output: Adjacency list G representing a random k-colorable graph

▷ Initialize an empty adjacency list of size n

$G \leftarrow$ empty list of size n

▷ p is generated randomly between 0 and 1

$p \in [0, 1]$

▷ Partition the vertices into k disjoint sets

Divide the set of vertices $1, 2, \ldots, n$ into k disjoint subsets $S_1, S_2, \ldots, S_k$

▷ Connect vertices in different sets with probability p

for $i \leftarrow 1$ **to** n **do**
    $G[i] \leftarrow$ empty list **for** $j \leftarrow i + 1$ **to** n **do**
        if *i, j not in the same set S_t for any t* **then**
            if *random number between 0 and 1 $< p$* **then**
                add edge (i, j) to $G[i]$
                add edge (j, i) to $G[j]$

return G;

5 RESULT

5.1 First-Fit Greedy Graph Coloring Algorithm

For the first part we have compared the competitive ratios produced by the $First - Fit - Greedy - Graph - Coloring - Algorithm$. Our algorithm gave us the results as shown in the graph

5.1.1 N=100. Here we compare the output for algorithm for 100 instances of randomly generated graphs.

From Table 1, Table 2, Table 3 we can conclude that the average competitive ratio increases with number of vertices n. Moreover from Fig. 1. we can conclude that the average competitive ratio increase with increase in chromatic number k. So, we can say that the output we are receiving for our algorithm is expected as a larger graph(a graph with more number of vertices) and a larger chromatic number are bound to make the problem of k-coloring tougher. Furthermore, from the results we can conclude that our algorithm many a times fails to produce optimal solutions.

Table 1. Average Comparison Ratios for k=2 and N=100

n	Avg. Comp. Ratio
50	1.48
100	1.915
200	2.05
400	2.18
800	2.515
1600	2.815

Table 2. Average Comparison Ratios for k=3 and N=100

n	Avg. Comp. Ratio
50	1.7333
100	2.0767
200	2.1900
400	3.0567
800	3.3500
1600	4.6500

Table 3. Average Comparison Ratios for k=4 and N=100

n	Avg. Comp. Ratio
50	1.585
100	2.0225
200	2.5675
400	2.9975
800	3.955
1600	4.775

This observation is in line with the behavior of greedy algorithms as they tend to reach a local optima rather than the global optima.

5.1.2 N=1000. In this section we compare the outputs of the First-Fit Algorithm for N=1000, for the chromatic number values of k = 2, 3, 4.

Just like for N=100, in N=1000 we can observe from Table 4., Table 5., Table 6. and Fig2. that the average competitive ratios increase with the graph size and the chromatic number and works as expected. From Fig. 3 we can observe that the First-Fit was able to give correct coloring for the random graph generate with chromatic number equal to 2.

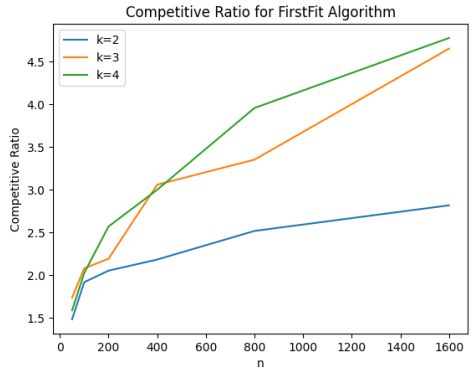

Fig. 2. Comparison of Competitive Ratios generated by the First-Fit Algorithm for N=100

Table 4. Average Comparison Ratios for k=2 and N=1000

n	Avg. Comp. Ratio
50	1.5495
100	1.7495
200	1.955
400	2.042
600	2.206
800	2.2345
1000	2.3875
1400	2.346
1600	2.4265

Table 5. Average Comparison Ratios for k=3 and N=1000

n	Avg. Comp. Ratio
50	1.6799
100	1.9823
200	2.4249
400	2.7463
600	3.0239
800	3.1980
1000	3.4060
1400	3.7743
1600	3.7910

Table 6. Average Comparison Ratios for k=4 and N=1000

n	Avg. Comp. Ratio
50	1.6505
100	2.08075
200	2.56375
400	3.12175
600	3.612
800	3.85875
1000	4.48625
1400	4.55025
1600	4.72475

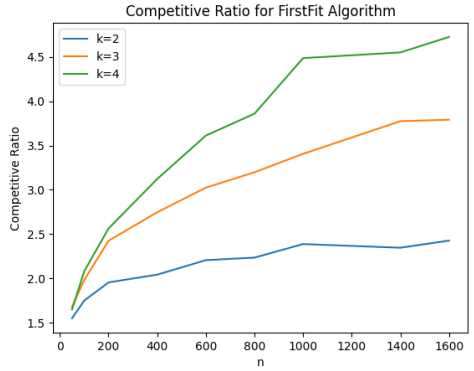

Fig. 3. Comparison of Competitive Ratios generated by the First-Fit Algorithm for N=1000

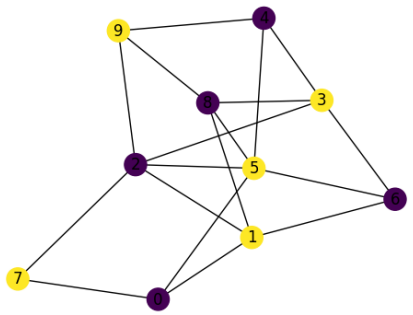

Fig. 4. Coloring of Random graph with First-Fit algorithm with k=2, n=10

5.2 CBIP Greedy Graph Coloring algorithm

In this section we compare the outputs of CBIP algorithm for k=2. Unlike in the previous section we do not compare the outputs for all the 3 chromatic numbers

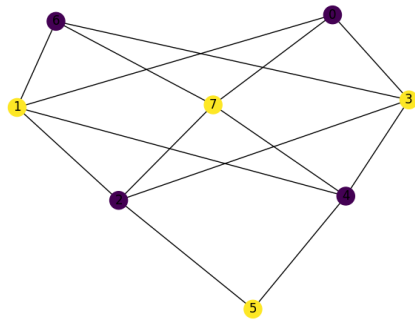

Fig. 5. Coloring of Random graph with CBIP algorithm with k=2, n=8

k=2,3,4. This is due to the case that computing k=3 and 4 is computationally impractical.

5.2.1 . N=100 Here we compare the output for algorithm for 100 instances of randomly generated graphs. The output of the algorithm for n=10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105, 110, 120, 150, 200 is shown in Table 7.

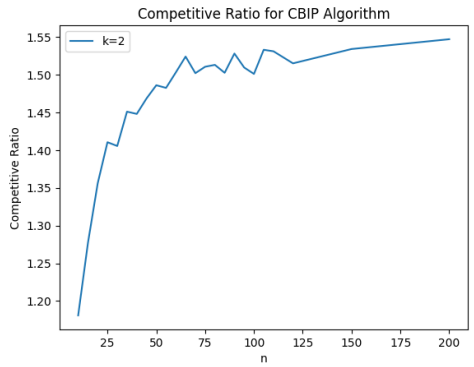

Fig. 6. Visualization of Competitive ratio for CBIP algorithm when k=2

5.2.2 ***Time Complexity and computational impracticality for k>=3.*** The worst-case time complexity of CBIP on a graph with k colors is $O(2^k * n^2)$, where n is the number of nodes in the graph. This is because the algorithm involves recursively exploring all possible ways of partitioning the nodes of the graph into two sets, which gives rise to an exponential running time in the worst case.

In the worst case, the algorithm will need to recursively explore all possible colorings of the graph, which gives rise to a factor of 2^k in the running time.

Table 7. Average Comparison Ratios for k=2 and N=100

n	Avg. Comp. Ratio
10	1.181
15	1.277
20	1.356
25	1.4105
30	1.4055
35	1.451
40	1.448
45	1.4685
50	1.486
55	1.4825
60	1.503
65	1.524
70	1.502
75	1.5105
80	1.513
85	1.5025
90	1.528
95	1.5095
100	1.501
105	1.533
110	1.531
120	1.515
150	1.534
200	1.547

Additionally, for each possible coloring, the algorithm may need to check each node against each node in the other partition, which gives rise to a factor of n^2 in the running time. Therefore, the worst-case time complexity of CBIP on a graph with k colors is $O(2^k * n^2)$.

This means that as the number of colors increases, the time required to perform CBIP on a given graph grows very quickly. For example, when $k = 3$, the worst-case time complexity is $O(8n^2)$, which is already very high. When $k = 4$, the worst-case time complexity is $O(16n^2)$, which is significantly higher. Ask continues to increase, the computational resources required to run CBIP become increasingly impractical.

6 CONCLUSION

In this paper, we presented an empirical study on the average competitive ratio of the FirstFit and CBIP algorithms for coloring online graphs. Our findings suggest that the FirstFit algorithm's performance may degrade with an increase in the number of vertices for 2-colorable online graphs, while the CBIP algorithm maintains a constant average competitive ratio. The computational impracticality of the CBIP algorithm for 3-colorable and 4-colorable graphs arises from the fact that the worst-case time complexity of CBIP on a graph with k colors is $O(2^k * n^2)$, where n is the number of nodes in the graph. Furthermore, even for graphs with a small number of colors, the worst-case running time of CBIP can be quite high, making it impractical for many real-world use cases. As a result, alternative algorithms for graph coloring, such as backtracking or greedy heuristics, are often used in practice.

7 TEAM CONTRIBUTION

Table 8. Contributions to the project

Name	Contribution to the project
Rajat Sharma	Generation of random graphs algorithm and report making
Amanpreet Singh	First-Fit Graph coloring algorithm and report making
Tript Dhudi	CBIP Graph coloring algorithm and report making
Ashish Upadhyay	Analysis of findings and report making
Vikram Singh Brahm	Report making and implementations of algorithms

8 REFERENCES

[1] Li, Y., Narayan, V.V. Pankratov, D. Online Coloring and a New Type of Adversary for Online Graph Problems. *Algorithmica* 84, 1232–1251 (2022). https://doi.org/10.1007/s00453-021-00920-w

[2] Erdős, P., R'enyi, A. (1959). On random graphs I. *Publicationes Mathematicae*, 6, 290-297.

[3] Kierstead, H. Smith, David Trotter, William. (2015). First-fit coloring on interval graphs has performance ratio at least 5. *European Journal of Combinatorics*. 51. https://doi.org/10.1016/j.ejc.2015.05.015.