Algorithm Design Techniques (COMP-6651)
Project Report

Analysis of First Fit and CBIP Algorithm on Online Graph Coloring

Team

Akshit Rameshkumar Desai (Student ID: 40233041)
Deepkumar Pareshkumar Raval (Student ID: 40231723)
Dharmil Mukeshbhai Vaghasiya (Student ID: 40230633)
Jaymin Shantilal Suhagiya (Student ID: 40232368)
Ruchit Dobariya (Student ID: 40232238)

Submitted to: Dr. Yaqiao Li

Project Github Link: https://github.com/akshitdesai/gc
Project GUI Link : https://akshitdesai.github.io/gc/

1 Abstract

Graph coloring is one of the very well-know problems in computer science. Its decision variant is known to be $\mathcal{NP}$-Complete. The online graph coloring is a similar problem where vertices come one at a time. In this study, we have implemented FirstFit and CBIP algorithm for online graph coloring. Generation of k-colorable graph is also discussed. Results discusses the change in competitive ratio due to the number of vertices, probability-factor and chromatic number. GUI for simulating the implemented algorithms is also discussed. Our study reveals that as p increases, the competitive ratio decreases and stabilizes for large values of p and n. In bipartite graphs, CBIP slightly outperforms FirstFit.

2 Problem Statement

2.1 Graph coloring

Graph coloring is an important concept in graph theory that involves assigning colors to the vertices (or vertices) of a graph such that adjacent vertices have different colors [1]. The primary goal of graph coloring is to use as few colors as possible, which is known as the chromatic number of the graph. There are several different types of graph coloring, of which vertex coloring is a specific type of graph coloring which is used here. The chromatic number of a graph is the minimum number of colors required to properly color the graph.

2.2 Online Graph coloring

Additionally, an online graph is a graph that is constructed incrementally over time as new vertices and edges are added to it [2]. Online graphs are dynamic in nature and can be updated continuously as new data becomes available. Online graphs are often used in situations where data is streaming in continuously, such as in social networks, traffic networks, or sensor networks. By updating the graph in real-time, it allows for more accurate analysis and predictions based on the most up-to-date information. Finding an optimal coloring for large or complex graphs can be challenging, and it may not always be possible to find a coloring that uses the minimum number of colors.

2.3 First Fit Algorithm

The First Fit graph coloring algorithm is a simple greedy approach for solving the graph coloring problem. In this algorithm, vertices are colored one by one, in the order they appear in the graph. For each vertex, the algorithm tries to find the lowest numbered color that is not already used by any of its neighboring vertices. If all the colors adjacent to the vertex are already used, then the algorithm assigns a new color to the vertex [3].

2.4 CBIP Algorithm

The CBIP algorithm is an online graph coloring algorithm that assumes the offline chromatic number of the graph is known. It operates by maintaining a k-partition (k-coloring) of the vertices of the current partial graph into independent sets, where k is the offline chromatic number of the graph. When a new vertex arrives, the algorithm colors it with the smallest natural number that has not been used by the vertices in the other independent set, apart from the independent set to which the arrived vertex belongs [4].

3 Implementation

See this section which contains instruction on how to run the implementation.

3.1 Graph generation

This algorithm generates a random graph with n vertices and chromatic number k. The first part of the algorithm initializes k groups and assigns the first k vertices to each group. The remaining $n - k$

vertices are randomly assigned to one of the k groups using a uniform distribution. This ensures an even distribution of vertices among groups. After assigning the vertices to groups, the algorithm creates an edge between each vertex of all groups to any vertex of remaining each groups. In other words, each edge connects the vertex to a vertex in a different group than itself, selected randomly. Once these edges are created, the algorithm adds more edges to each vertex to some other vertex in different group based on a probability p. This probability is given as input to the algorithm. To represent the same graph as an online graph, the algorithm prepare an array which specifies the incoming vertices in online graph. This array maintains the pair of a vertex and it's neighbours (which were arrived in graph previously) in a chronological order. Algorithm returns this array structure which is nothing but an online graph.

Algorithm 1 Generate Graph

```
function GenerateGraph($num\_vertex$, $k$, $prob$)
    $groups \leftarrow \phi$
    $edges \leftarrow \phi$
    for $i = 1$ to $k$ do                                   ▷ Generate k empty groups.
        $groups[i] = \{\}$
    end for
    for $vertex = 1$ to $k$ do
        $groups[vertex].add(vertex)$
    end for
    for $vertex = k + 1$ to $num\_vertex$ do
        $groups[random(1, k)].add(vertex)$
    end for
    for $i = 1$ to $groups.size()$ do
        for $vertex$ in $groups[i]$ do
            for $k = 1$ to $groups.size()$ do
                if $groups[k].has(vertex)$ then
                    continue
                end if
                $int\ v = groups[k][random(0, groups.size())]$
                if $vertex < v$ then
                    $edges.add(vertex, v)$
                else
                    $edges.add(v, vertex)$
                end if
            end for
            for $k = 1$ to $groups.size()$ do
                if $groups[k].has(vertex)$ then
                    continue
                end if
                for $v$ in $groups[k]$ do
                    if $random(0, 1) <= p$ then
                        if $vertex < v$ then
                            $edges.add(vertex, v)$
                        else
                            $edges.add(v, vertex)$
                        end if
                    end if
                end for
            end for
        end for
    end for
end function
$alreadyAdded \leftarrow new\ Set()$
$onlineGraph \leftarrow new\ Pair()$
for $u = 1$ to $n$ do
    $neighbours \leftarrow \phi$
    for $edge$ in $edges$ do
        if $edge$ has $u$ then
            $v \leftarrow edge[1]$
            if $alreadyAdded$ has $v$ then
                $neighbours.add(v)$
            end if
        end if
    end for
    $alreadyAdded.add(u)$
    $onlineGraph.add(pair(u, neighbours))$
end for
$return\ onlineGraph$
```

3.2 First Fit Algorithm

First Fit Algorithm is trivial to implement. When a new vertex v arrives, we can first create a set containing colors of all its neighbours. Next we can start incrementing color (starting from 1) until we reach to a color which is not in the set we generated earlier. This color is then can be used to color the vertex v.

Algorithm 2 The First Fit algorithm for online coloring.

```
function FirstFit($vertex$)
    $neighbours\_colors \leftarrow new\ Set()$
    for $neighbour$ in $vertex.neighbours$ do
        $neighbours\_colors.add(neighbour.color)$
    end for
    $min\_color \leftarrow 1$
    while $neighbours\_colors.has(min\_color)$ do
        $min\_color = min\_color + 1$
    end while
    $vertex.color = min\_color$
end function
```

3.3 CBIP Algorithm

The CBIP algorithm finds the k-partition of the current partial graph into independent sets. Here current partial graph is connected component of a vertex upon arrival.

For the the CBIP algorithm we have fixed $k = 2$. Because for k larger than 2, it is computationally heavy to generate a k-partition. For $k = 2$ graph is bipartite so it is easier to generate such partition (this is evident from the below pseudocode) but it is not true for $k \geq 3$.

Upon arrival of vertex v, the $GET\_PARTITION$ distributes all the vertices of current partial graph into two sets.(These will be independent sets as graph is bipartite). It uses BFS to distributes the vertices into two sets. Then from returned partitions we find the partition $v \in H_v$. Then from all the rest of the partitions(only 1 partition because we have fixed $k = 2$) we find the minimum color which is not used by remaining partitions. And using which we can color the vertex v.

Algorithm 3 The CBIP algorithm for online coloring.

```
function CBIP($vertex$)
    $partitions \leftarrow GET\_PARTITIONS(vertex)$
    $neighbours\_colors \leftarrow new\ Set()$
    for $partition$ in $partitions$ do
        if $partition.has(vertex)$ then
            continue
        end if
        for $neighbour$ in $vertex.neighbours$ do
            $neighbours\_colors.add(neighbour.color)$
        end for
    end for
    $min\_color \leftarrow 1$
    while $neighbours\_colors.has(min\_color)$ do
        $min\_color = min\_color + 1$
    end while
    $vertex.color = min\_color$
end function

function GET_PARTITIONS($vertex$)
    $partitions \leftarrow \phi$
    $visited \leftarrow \phi$
    $queue \leftarrow new\ Queue()$
    $queue.push(vertex)$
    $visited[vertex] = true$
    $level \leftarrow 0$
    while $!q.empty()$ do
        $size \leftarrow q.size()$
        for $i = 1$ to $size$ do
            $u \leftarrow q.pop()$
            $partitions[level].push(u)$
            for $neighbour$ in $u.neighbours$ do
                if $not\ visited[neighbour]$ then
                    $visited[neighbour] = true$
                    $q.push(neighbour)$
                end if
            end for
        end for
        $level = level \oplus 1$
    end while
    $return\ partitions$
end function
```

3.4 GUI

The GUI is hosted on github pages and can be accessed via this link.

3.4.1 Technology used

We implemented the GUI in React.js[5] to visualize both graph coloring algorithms. To diplay the graph for any given state we used "react-graph-vis" package which provides nice interactive graph component.

3.4.2 Implementation of the algorithms in Javascript

We implemented previously discussed pseducodes for the graph generation, first fit and CBIP in the javascript. These and C++ codes are equivalent.

3.4.3 Handling the graph states throughout coloring

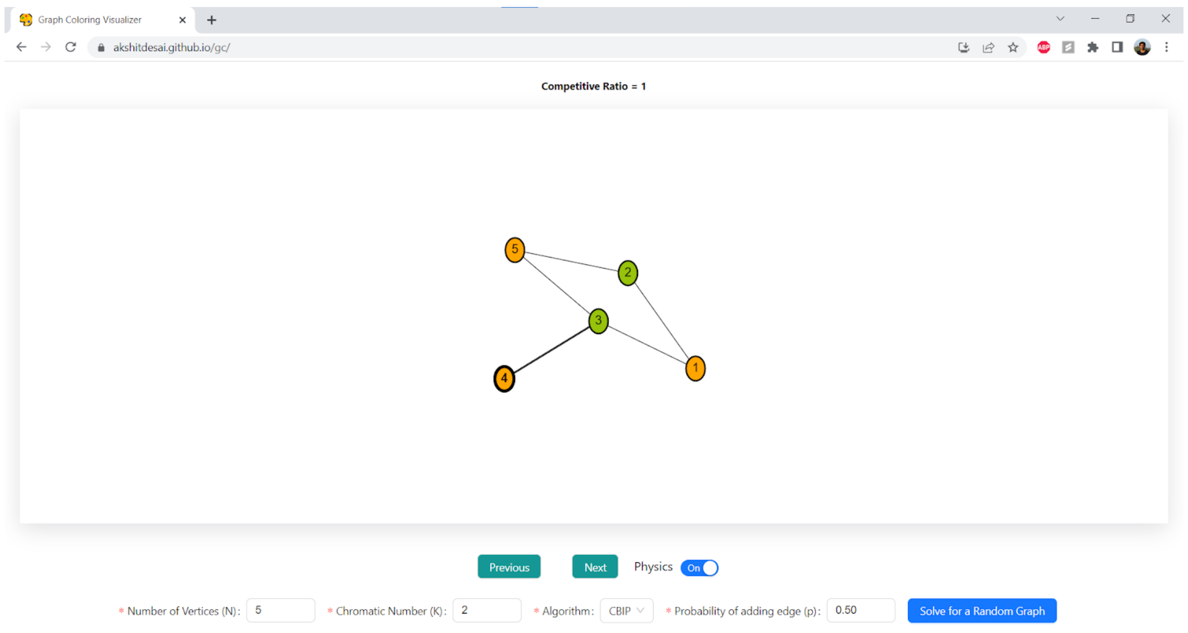
The GUI also supports traversing through the states during coloring. Clicking on "previous" displays the previous state of the graph and clicking "next" displays the next state of the graph. To implement this, we used two separate lists one tracks the current state of the graph and one tracks the history of the graph. When "previous" is clicked, a state is popped from the current graph and it is added to the history. Similarly, when "next" is clicked, a state is popped from the history and added to the current graph.

3.4.4 GUI Walk through

GUI takes the following inputs from the user:

- Number of vertices (N)
- Chromatic Number of the Graph (K)
- Algorithm: "CBIP" or "First Fit"
- Probability of adding edge (p) - This is used internally for genereating a random graph. While generating a graph, p will be used to decide whether to add edge between two vertices or not. For example, $p = 1$ and $k = 2$ will generate a complete bipartite graph.

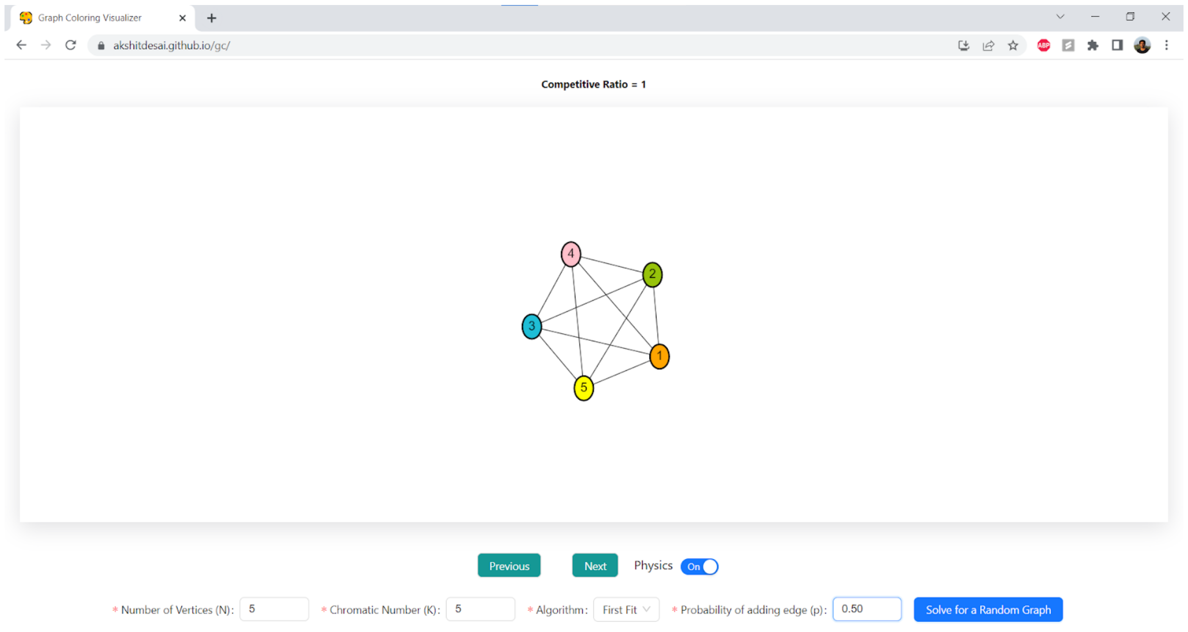
Initially, GUI displays a solution using CBIP on a random graph generated with $N = 5$, $K = 2$, and $p = 0.5$. Note that once the graph coloring is done, algorithm's competitive ratio will be shown at the top of the screen. Below figures shows the demo of the GUI.



Competitive Ratio = 1

* Number of Vertices (N): 5
* Chromatic Number (K): 2
* Algorithm: CBIP
* Probability of adding edge (p): 0.50

Figure 1: N = 5, k = 2, Algorithm: CBIP, Probability of adding edge (p)= 0.50.



Competitive Ratio = 1

* Number of Vertices (N): 5 * Chromatic Number (K): 5 * Algorithm: First Fit * Probability of adding edge (p): 0.50

Figure 2: N = 5, k = 5, Algorithm = FirstFit, Probability of adding edge (p) = 0.50.

4 Results

To generate results we created a script in C++ (i.e. **analysis.cpp**). This script runs the First Fit and CBIP algorithm over 100 random graphs for the given chromatic number k and the number of vertices n. Then the average is taken of the competitive ration is taken and stored inside a CSV file. To make this script faster we also implemented multithreading. Each atomic thread runs the algorithm on 10 random graphs. Main thread calls this thread 10 times to get the average value of the competitive ration for the 100 runs. We used these CSV files to generate the below graphs and tables.

Number of vertices (n)	Average Competitive Ratio over 100 runs		
	p=0.25	p=0.5	p=0.75
50	1.155	1.07	1.015
100	1.17	1.085	1.015
200	1.17	1.115	1.005
400	1.18	1.075	1.01
800	1.15	1.06	1
1600	1.175	1.04	1.01

Table 1: CBIP with k = 2

Number of vertices (n)	Average Competitive Ratio over 100 runs		
	p=0.25	p=0.5	p=0.75
50	1.220	1.040	1.015
100	1.240	1.090	1.015
200	1.215	1.060	1.020
400	1.165	1.080	1.035
800	1.370	1.065	1.010
1600	1.205	1.030	1.010

Table 2: FirstFit with k = 2

Number of vertices (n)	Average Competitive Ratio over 100 runs		
	p=0.25	p=0.5	p=0.75
50	2.375	1.930	1.570
100	2.490	1.840	1.610
200	2.650	1.830	1.600
400	2.710	1.845	1.595
800	2.745	1.895	1.600
1600	2.775	1.965	1.590

Table 3: FirstFit with k = 3

Number of vertices (n)	Average Competitive Ratio over 100 runs		
	p=0.25	p=0.5	p=0.75
50	3.720	2.915	2.245
100	4.730	2.800	2.170
200	5.060	2.785	2.185
400	4.995	2.740	2.230
800	5.380	2.740	2.215
1600	6.425	2.735	2.170

Table 4: FirstFit with k = 4

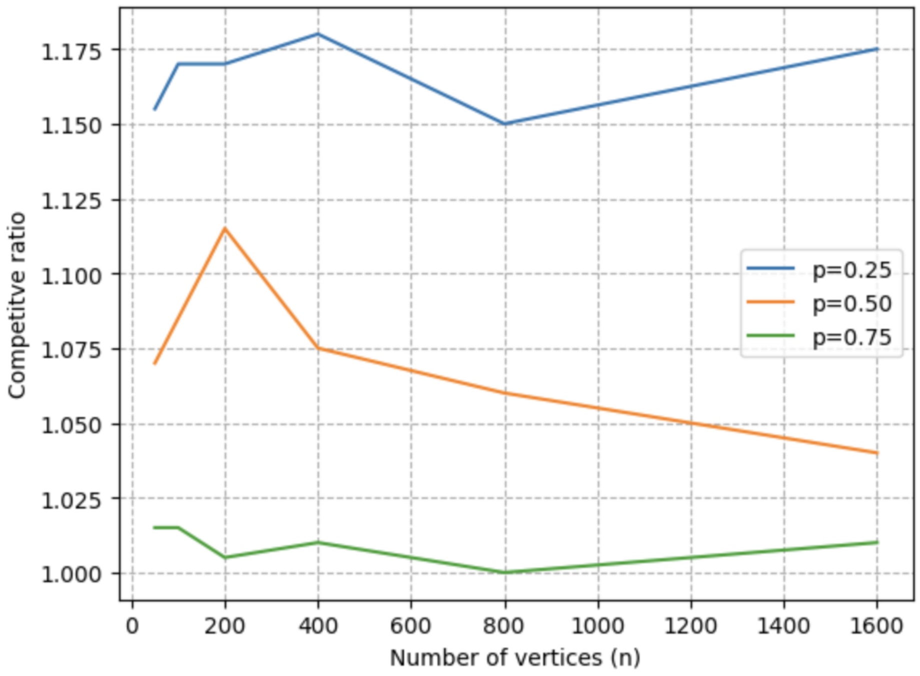


Figure 3: Competitive ratio of CBIP algorithm for $k = 2$ vs Number of vertices (n) for different values of p

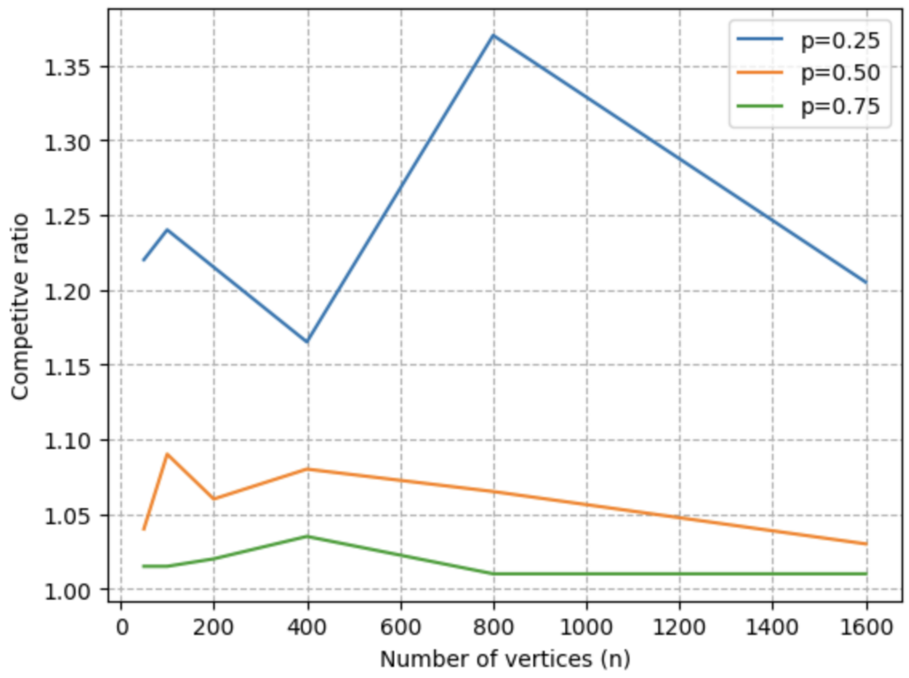


Figure 4: Competitive ratio of FirstFit algorithm for $k = 2$ vs Number of vertices (n) for different values of p

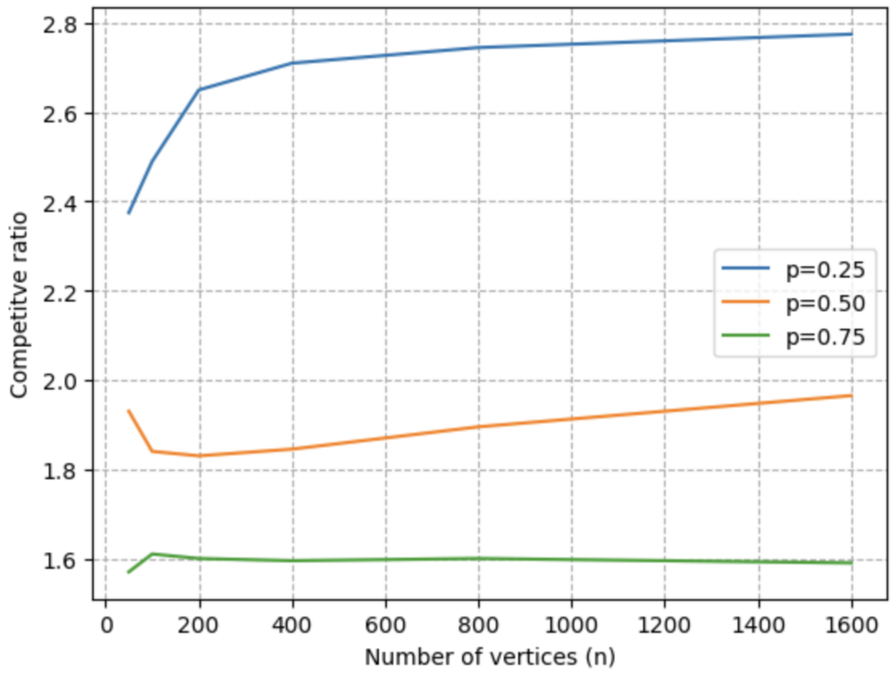


Figure 5: Competitive ratio of FirstFit algorithm for $k = 3$ vs Number of vertices (n) for different values of p

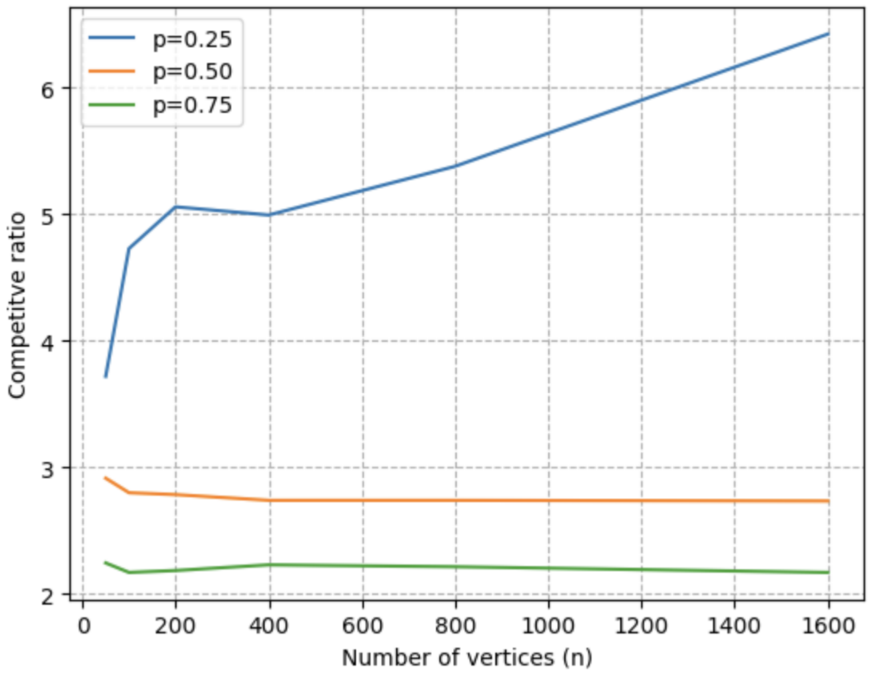


Figure 6: Competitive ratio of FirstFit algorithm for $k = 4$ vs Number of vertices (n) for different values of p

5 Conclusions

Irrespective of the algorithm employed, it has been noted that as the value of p increases, the competitive ratio decreases. Moreover, for large values of p, the competitive ratio stabilizes over varying values of n. As p increases, the graph exhibits a higher degree of connectivity, which could explain this trend. In the case of bipartite graphs, CBIP outperforms FirstFit slightly. Additionally, we have observed a trend for FirstFit where the competitive ratio increases as n increases. In future, we would like to run the algorithms on more finer range of ns, this could help us determine the growth function (if any) of the algorithms.

6 Work Distribution

- All the work was distributed evenly among all the group members. For most of the implementation part we did pair programming in group of two and later explained all the work done to each other.

Graph Generation & Algorithms:

- Graph generation was implemented in both C++ and javascript by Akshit(40233041) & Deepkumar(40231723).
- First Fit Algorithm was implemented in both C++ and javascript by Dharmil(40230633).
- CBIP Algorithm was implemented in both C++ and javascript by Jaymin(40232368) & Ruchit(40232238).

GUI:

- Akshit(40233041) & Deepkumar(40231723) designed web page and connected all the components created in javascript into React App.
- Component to handle previous and next state of graph is done by Jaymin(40232368).

Analysis of Algorithms:

- Final script(C++) to bench mark algorithm and prepare csv files are completed by Deepkumar(40231723).
- Generation of line-graphs from csv file is done in python(matplotlib) by Akshit(40233041).

Report:

- Report and README files are prepared by Dharmil(40230633) & Ruchit(40232238).

References

[1] Philippe Galinier et al. "Recent Advances in Graph Vertex Coloring". In: vol. 38. Jan. 2013, pp. 505–528. ISBN: 978-3-642-30504-7. DOI: 10.1007/978-3-642-30504-7_20.

[2] Sundar Vishwanathan. "Randomized online graph coloring". In: *Journal of Algorithms* 13.4 (1992), pp. 657–669. ISSN: 0196-6774. DOI: https://doi.org/10.1016/0196-6774(92)90061-G. URL: https://www.sciencedirect.com/science/article/pii/019667749290061G.

[3] A. Gyárfás and Jeno Lehel. "On-line and first-fit coloring of graphs". In: *Journal of Graph Theory* 12 (Oct. 2006), pp. 217–227. DOI: 10.1002/jgt.3190120212.

[4] Yaqiao Li, Vishnu V. Narayan, and Denis Pankratov. *Online Coloring and a New Type of Adversary for Online Graph Problems.* 2020. arXiv: 2005.10852 [cs.DS].

[5] URL: https://react.dev/.