Advanced Algorithm Project Work: Report

Prepared By:
21CSB0B58 : Tarachan Rana

21CSB0B56: Sunny Kumar

21CSB0B61: Ujjwal Kumar Singh

PROBLEM STATEMENT:

Advanced Approach to Graph Coloring Using Divide and Conquer and Greedy Heuristics:

- GRAPH COLORING PROBLEM:

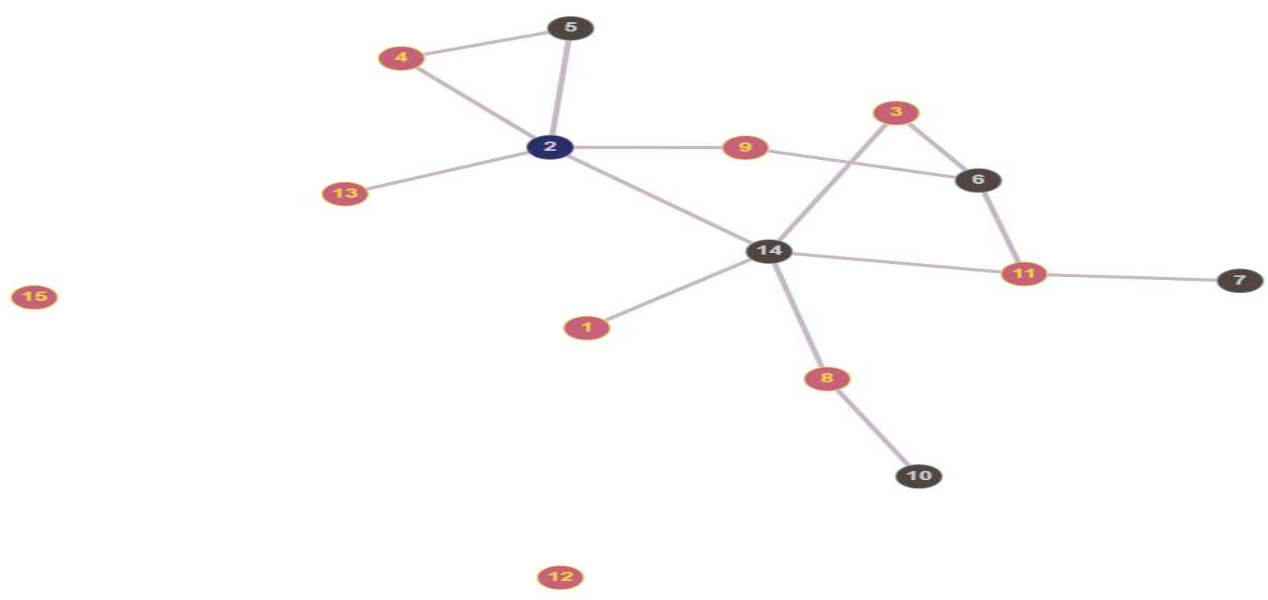
Vertex Set and Edge Set:

- Vertex set V(G) of a simple graph G is {v1, v2, …, vn}
- Edge set E(G) is {e1, e2, …, em}, with each edge ek mapped to a pair of disordered end vertices (vj, vi).

Graph Coloring Problem (GCP):

- Minimum number of colors required to color V(G) such that adjacent vertices are assigned different colors is denoted as $\chi(G)$ and referred to as the graph coloring problem (GCP).

Example:



Colored Graph : (3Colors)

PROPOSED SOLUTIONS:

GCP is a NP hard problem . Through this project work, we are trying to implement an efficient algorithm which uses **greedy Heuristics** which also accompanied by the **Divide and Conquer** Strategy for large inputs.

- # Greedy Heuristics Method

Case 1: SetGreedyColor(i)

A heuristic procedure is executed on all vertices whose degree ≥µ½ (G), employing a greedy approach as follows:

```
Choose a vertex i such that color[i] = 0 and degree (i) ≥ µ½ (G).
For j = 1 to mincolor:
      Boolean t = false;
            For k = 1 to n:
                  If (Aik = 1), then:
                        If (color[k] = = j), then t = false; break;
                        Else t = true;
            If (t = true), then color[i] = j; break;
Repeat Steps i and ii for other vertices whose degree ≥µ½ (G).
```

Case 2:

For all the vertices whose degree <µ½ (G), SetGreedyColor (i) is invoked.

Compute f(G). If f(G) ≠ 0, set mincolor = mincolor + 1;

repeat case 1 and case 2 until f(G) = 0.

This greedy procedure assigns color[i] in [1,mincolor] for all 1 ≤ i ≤ n.

- # Divide and Conquer Strategy:

This iterative method is evolved by applying the divide & conquer strategy in method 1; by initializing mincolor as 3 and color[i] = 0 for 1 ≤ i ≤ n.

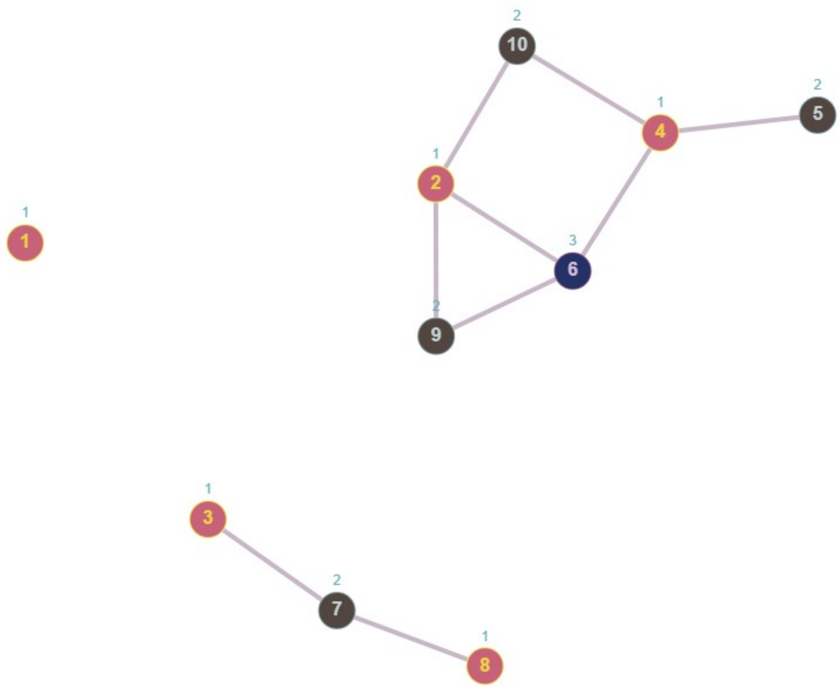
•Call Split(V(G)) to partition V(G) into Vl(G) and Vr(G).
•If n > 100:
  - Call Split(Vl(G)) to partition Vl(G) into Vll(G) and Vlr(G).
  - Call Split(Vr(G)) to partition Vr(G) into Vrl(G) and Vrr(G).
•Calculate the number of conflicting edges from Vl(G) to Vr(G) and from Vr(G) to Vl(G).
•Resolve (Vl(G), Vr(G)) and (Vlr(G), Vrr(G)) in concurrence using method 1, addressing the subset with more conflicting edges first.
•Compute f(G).
If f(G) ≠ 0 or color[i] > mincolor for all i in [1, n],
set mincolor = mincolor + 1
and repeat steps 3 and 4 until f(G) = 0.

IMPLEMENTED CODE IN C++

The code for the algorithm is implemented using C++ language
.
It is available along with the project work and can also be accesses from the following link:
https://drive.google.com/drive/u/0/folders/1F7tLFiTUXy3h-rTQnlgZiubXCLnJTuAV

OUTPUTS:

1.GRAPH-1(n=10,m=9):(O1: Simulator, Output2: New GCP)

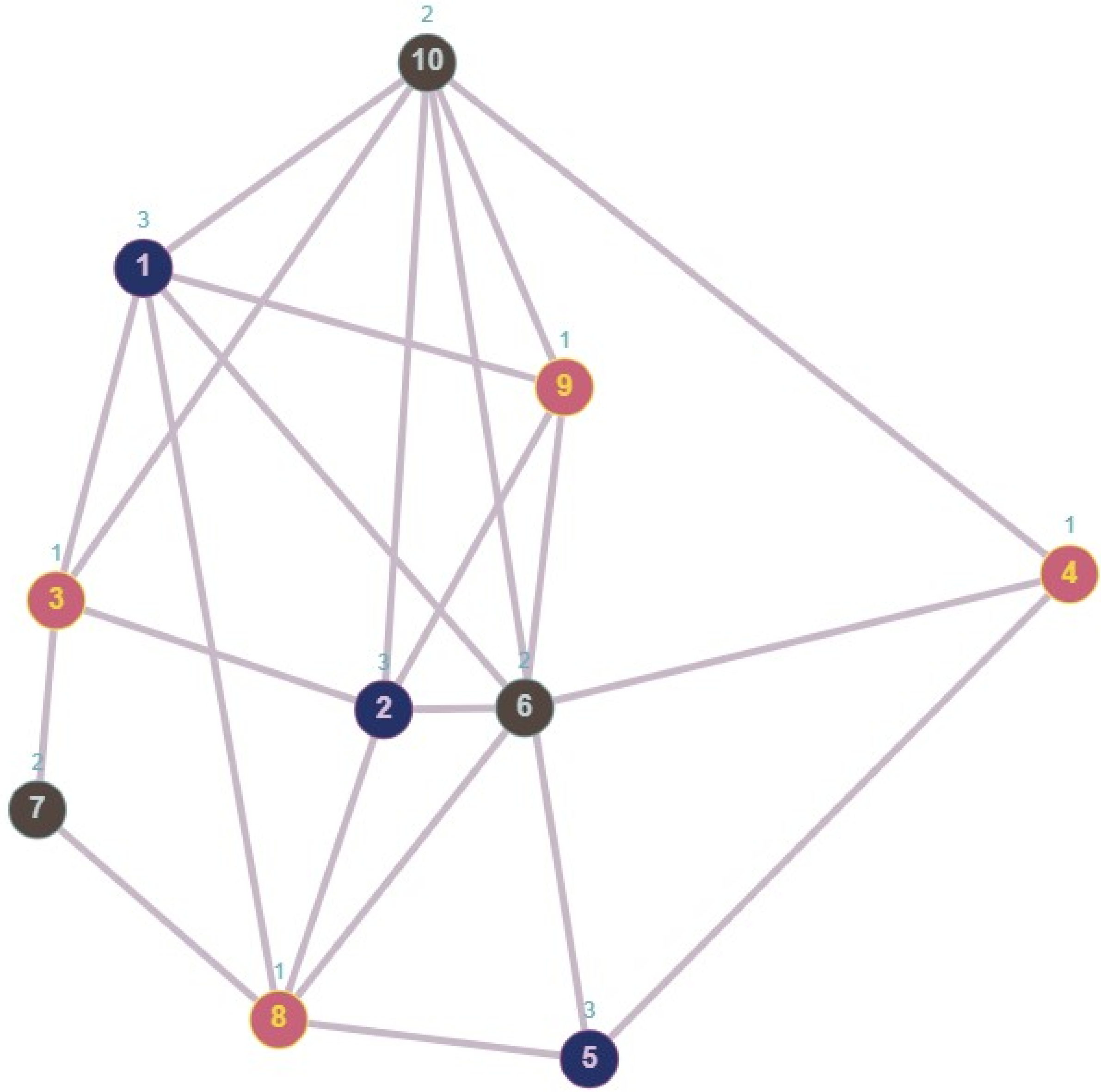



```
No of Vertices for Graph:
10
Constructing grapgh by making an adjacency matrix:
Adjacency Matrix:
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 1 1
0 0 0 0 0 0 1 0 0 0
0 0 0 0 1 1 0 0 0 1
0 0 0 1 0 0 0 0 0 0
0 1 0 1 0 0 0 0 1 0
0 0 1 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0 0
0 1 0 0 0 1 0 0 0 0
0 1 0 1 0 0 0 0 0 0

No of edges is :9

Colors of Vertices:
Vertex 0: Color 1
Vertex 1: Color 1
Vertex 2: Color 2
Vertex 3: Color 1
Vertex 4: Color 2
Vertex 5: Color 2
Vertex 6: Color 1
Vertex 7: Color 2
Vertex 8: Color 3
Vertex 9: Color 2
```

2.GRAPH-2(n=10,m=21):(Simulator Output & New GCP Output)

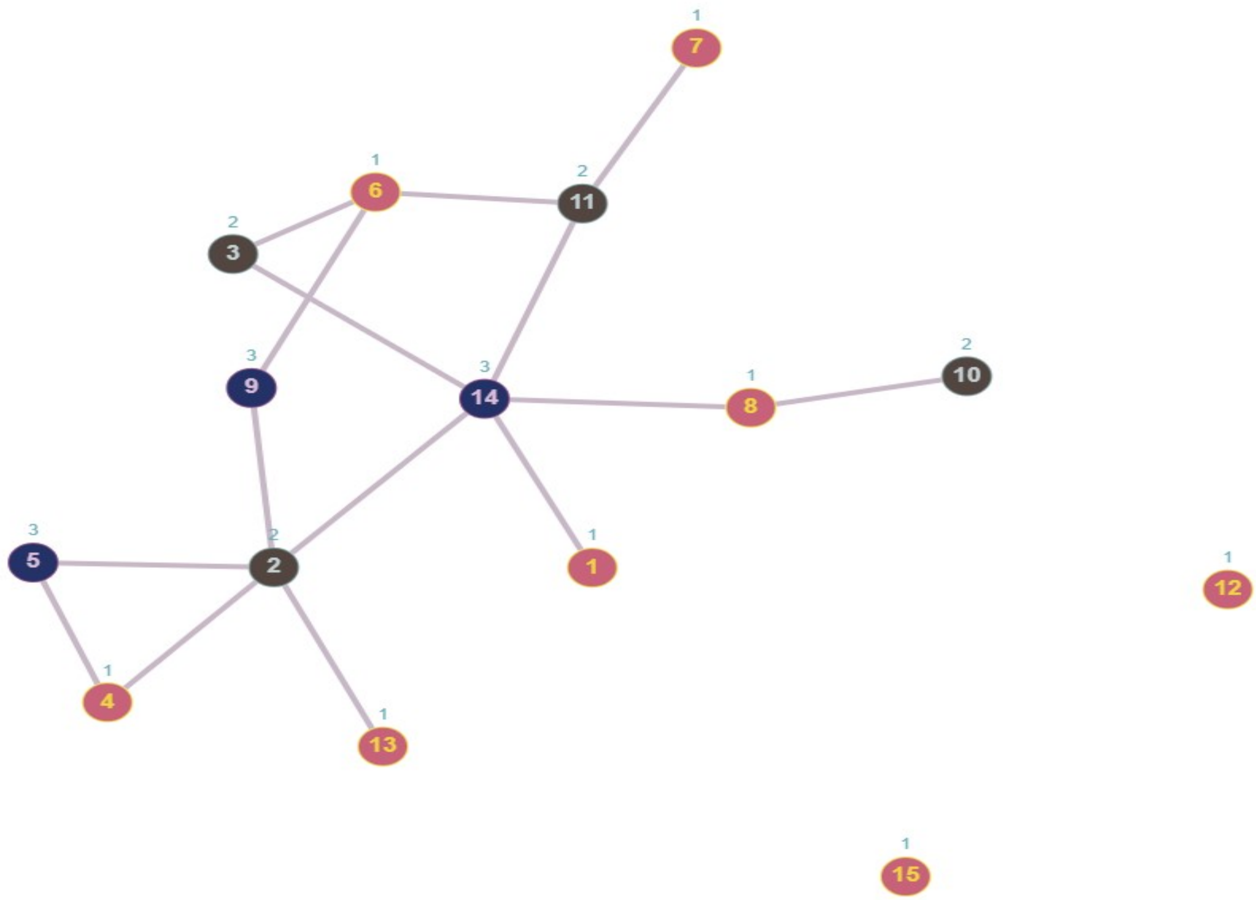


```
No of Vertices for Graph:
10
Constructing grapgh by making an adjacency matrix:
Adjacency Matrix:
0 0 1 0 0 1 0 1 1 1
0 0 1 0 0 1 0 1 1 1
1 1 0 0 0 0 1 0 0 1
0 0 0 0 1 1 0 0 0 1
0 0 0 1 0 0 0 1 0 1
1 1 0 1 0 0 0 1 1 0
0 0 1 0 0 0 0 1 0 0
1 1 0 0 1 1 1 0 0 0
1 1 0 0 0 1 0 0 0 1
1 1 1 1 1 0 0 0 1 0

No of edges is :21

Colors of Vertices:
Vertex 0: Color 1
Vertex 1: Color 1
Vertex 2: Color 3
Vertex 3: Color 1
Vertex 4: Color 4
Vertex 5: Color 2
Vertex 6: Color 1
Vertex 7: Color 3
Vertex 8: Color 3
Vertex 9: Color 2
```

3.GRAPH-3(n=15,m=15):(Simulator Output & New GCP Output):



```
No of Vertices for Graph:
15
Constructing grapgh by making an adjacency matrix:
Adjacency Matrix:
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 1 1 0 0 0 1 0 0 0 1 1 0
0 0 0 0 0 1 0 0 0 0 0 0 0 1 0
0 1 0 0 1 0 0 0 0 0 0 0 0 0 0
0 1 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 1 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 1 0
0 1 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 1 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

No of edges is :15

Colors of Vertices:
Vertex 0: Color 1
Vertex 1: Color 1
Vertex 2: Color 1
Vertex 3: Color 2
Vertex 4: Color 3
Vertex 5: Color 2
Vertex 6: Color 2
Vertex 7: Color 1
Vertex 8: Color 3
Vertex 9: Color 2
Vertex 10: Color 1
Vertex 11: Color 1
Vertex 12: Color 2
Vertex 13: Color 2
Vertex 14: Color 1
```

4.Graph-4(n=101,m=1048)

```
No of Vertices for Graph:
101
Constructing grapgh by making an adjacency matrix:

No of edges is :1048

Colors of Vertices:
Total no of colors used is :6
PS C:\Users\Adminstrator\Desktop\Projects\advalgo>
```

CONCLUSION

Best Case Time Complexity:

- The best-case scenario occurs when the graph can be evenly divided into subsets with balanced colors and minimal conflicts, resulting in a time complexity closer to O(V log V) for the color_graph_divide_and_conquer function.

Worst Case Time Complexity:

- The worst-case scenario occurs when the graph divisions lead to unbalanced subsets with high conflict resolution needs, resulting in a time complexity approaching O(V^2) for the color_graph_divide_and_conquer function.

Why This Algorithm and Any Improvement?

New approximation methods to find $\chi(G)$ using greedy and divide & conquer strategies are presented in this paper. The methods are based on the median degree $\mu_{½}(G)$. The devised methods are evaluated using benchmark graphs and near optimal solution are obtained; and optimal solution is obtained for most of the graphs. The divide & conquer strategy is performed well for most graphs compared to method 1.

ACKNOWLEDGEMENT

The authors would like to acknowledge the support rendered by the management of SASTRA University by the way of providing necessary infrastructure and financial support for this research. The authors would like to thank Gary Lewandowski, and Michael Trick for their support

rendered in http://mat.gsia.cmu.edu/COLOR/instances.html to accessing the graph coloring instances available in DIMACS which are the main breeding source of this research.

References:

1.Research Paper:

https://www.sciencedirect.com/science/article/pii/S2213020916301057#sec0030

2.Online Graph Algo Simulation Website:

https://graphonline.ru/en/