Vertex Sequence Independent Approach for Graph Coloring

Aayush Chhabra[1], Gouravjeet Singh[2], Srishti Mathur[3]
[1]Information Technology and Web Science, Rensselaer Polytechnic Institute, Troy, NY, USA
[2]Information Technology and Web Science, Rensselaer Polytechnic Institute, Troy, NY, USA
[3]Information Technology, IET DAVV, Indore, MP, India

Abstract-

Most of the algorithms used for Graph Coloring may produce optimal coloring for particular ordering of the vertices, but a less fortuitous ordering of vertices may lead to an extremely poor coloring. In this paper, we put forth a technique, which is independent of the initial ordering of the vertices, for coloring a graph with minimum number of colors and in significantly lesser time than many other techniques.

Keywords: Graph coloring, chromatic number, edge table, minimum coloring.

I. INTRODUCTION

A graph can be defined as an abstract representation of a set of objects where some pairs of the objects are connected by links. The interconnected objects are represented by mathematical abstractions called *vertices*, and the links that connect some pairs of vertices are called *edges*.
Graph Coloring Problem can be formally defined as follows: The coloring/labeling of the vertices of a graph G= (V, E), with V as set of vertices, and E as set of edges, is a map F: V->N, where adjacent vertices receive distinct colors in N; i.e., if uv ε E, then F (u) ≠F (v).
A graph can be properly colored with k colors if and only if its set of nodes can be partitioned into k pairwise-disjoint independent sets. In general, Graph Coloring Problem is a way of coloring the vertices of a graph such that no two adjacent vertices share the same color.
The problem of Graph coloring can be broken into two sub problems:
Firstly, the graph needs to be colored correctly. This means that no adjacent nodes must be assigned the same color. And secondly, the total number of colors used should be minimized. The first condition ensures correctness of the algorithm and the second ensures optimality.Coloring an undirected graph with a small number of colors as possible, has direct influence on how efficiently a certain target problem can be solved. A coloring using at most *k* colors is called a *k*-coloring. The smallest number of colors needed to color a graph is called its chromatic number [6]. Thus we can say that a graph that can be assigned a *k*-coloring is *k*-colorable, and it is *k*-chromatic if its chromatic number is exactly the same meaning.
This paper focuses on providing an optimized solution to crown graphs, which are a specialized case of graphs.
A crown graph is an undirected graph with two sets of vertices u_i and v_i and with an edge from u_i to v_j whenever $i \neq j$. One can view the crown graph as a complete bipartite graph from which the edges of a perfect matching have been removed. It is a particularly bad case for greedy coloring and other algorithms, as if the vertex ordering places two vertices consecutively whenever they belong to one of the pairs of the removed matching, then a greedy coloring will use n colors, i.e., half of the total number of vertices, while the optimal number of colors for this graph is two.

II. PREVIOUS WORK

Various approaches have been proposed for finding a solution to the graph coloring problem [2]. One such approach is the greedy algorithm, which at each stage makes a locally optimal choice in order to find the global optimum. In case of crown graphs Greedy Algorithm does not provide an optimal solution in terms of chromatic number.
DSATUR[3] by Brelaz is a sequential coloring[5] algorithm .It starts by assigning color 1 to the vertex maximal degree. Next a vertex with maximum saturation degree is chosen and assigned the lowest available color. This process is repeated until all vertices are colored. Though, this algorithm colors the crown graph with minimum number of colors but its time complexity is very high.
RLF[4] is a contraction algorithm[5]. It initially assigns color 1 to the node with maximal degree in the given graph. Once *i* nodes have been assigned color 1, it finds a set of uncolored nodes not adjacent to any colored node. Next, it finds the vertex from the above set, which has maximum number of common neighbors with the given vertex and colors it with color 1. Ties are broken by choosing the node that has minimal degree in the given set. If no such selection is possible, then the entire

process is repeated recursively on the sub graph induced by the uncolored nodes using next available color. This recursion is then repeated until all the nodes in the graph are colored. RLF Algorithm colors the crown graphs using two colors but its time complexity is very high as compared to the proposed algorithm.

In Edge Table Scanning algorithm [1] the graph is represented using an edge table. Each row of the edge table represents the two connected vertices of the graph. Initially all the vertices are assigned color 1. The entire edge table is scanned starting from the first row of the table. If the vertex in the first column has the same color as the vertex in the second column then the color of the vertex in the second column is incremented by one. The time complexity of this approach is less but it does not color crown graphs with minimum number of colors for particular ordering of vertices.

III. PROPOSED ALGORITHM

The proposed approach colors the graph with minimum number of colors, irrespective of the relative ordering of vertices. The algorithm can color all the graphs but it may be tailored for particularly coloring crown graphs. The approach is tested for complex graph as well as simple graphs. The Algorithm is described with the help of two lists L1 and L2, where L1 stores the nodes that are yet to be explored and L2 stores the already explored nodes. Initially all the vertices are assigned color 1. Then L1 is initialized with node. The algorithm scans the entire list L1 starting with the first entry. All the nodes connected to the vertex being scanned are found. Check for all the connected nodes: If it is present in L2 then ignore it. Otherwise compare its color with that of the node being scanned in L1. If both the nodes have the same color then increment the color of the connected node by 1. If the connected node is not present in L1, then append it to L1. After all the connected nodes are examined, remove the vertex currently being scanned from L1 and add it to L2. This procedure is repeated until L1 becomes empty.

ALGORITHM

```
    1. Assign color 1 to all nodes.
    2. Create a list L1 and initialize it with node1.
    3. Create an empty list L2.
    4. Loop until list L1 is empty
A. Remove the first node from list L1, and call it E.
B. Find all the nodes connected to E. C. For each connected node, c, do:
      a) For crown graph: If c is present in list L2, replace c with next connected node. If no more connected nodes,
      then go to 4.A. Otherwise, for normal graphs, continue.
      b) If color(E) = color(c) , then, color node c with next available color.
i. For crown graph: If c is not present in L1, append c to list L1. Otherwise, for normal graph check if c is absent in both L1
and L2, is so, append c to list L1.
D. Add E to list L2.
```

IV. RESULT

The proposed algorithm is an optimized algorithm which colors the crown graph with minimum number of colors, irrespective of the sequencing of vertices. The execution time and the number of colors used by the five algorithms namely greedy, RLF, Dsatur, ETS and the proposed algorithm are compared for different number of vertices. The results are stated in Table 1.

V. APPLICATIONS

Many complex problems can be solved by representing them as graph coloring problem. The various applications are listed below.

A. Scheduling

Many scheduling problems can be solved by representing them as vertex coloring problems. In certain problems where a set of jobs are given, each job needs to be allotted a time slot. The jobs that use the same resources cannot be allotted the same time slot. Hence, to perform this scheduling the jobs are represented by the vertices of a graph and the jobs which cannot be assigned the same time slot are connected with edges. The graph is then colored to obtain time slots for the given set of jobs.

B. Register Allocation

Register allocation is one of the techniques of compiler optimization in order to improve the execution time of the code. Fast processor registers are used to store the most frequently used values of the compiled program. An interference graph is constructed by the compiler. The vertices represent the registers. The edges connect two nodes if they are needed at the same time. The chromatic number of the graph so formed represents the number of registers required to store the variables.

C. Other Applications

A number of applications, including map-coloring, pattern matching, Sudoku puzzle solving etc. can be solved by representing them as graph coloring problems.

Table 1: Result

ALGORITHM	NO. OF VERTICES	EXECUTION TIME	CHROMATIC NUMBER
GREEDY	100	00:00:00:0000645	50
RLF	100	Table 1: Result	2
DSATUR	100	00:00:00:0607955	2
ETS	100	00:00:00:0004156	50
PROPOSED	100	00:00:00:0101834	2
GREEDY	200	00:00:00:0000806	100
RLF	200	00:00:00:0660421	2
DSTAUR	200	00:00:00:5016083	2
ETS	200	00:00:00:0017873	100
PROPOSED	200	00:00:00:0513453	2
GREEDY	300	00:00:00:0000816	150
RLF	300	00:00:00:3402479	2
DSTAUR	300	00:00:01:6595305	2
ETS	300	00:00:00:0035517	150
PROPOSED	300	00:00:00:2664480	2
GREEDY	400	00:00:00:0000986	200
RLF	400	00:00:01:3318033	2
DSTAUR	400	00:00:03:9196352	2
ETS	400	00:00:00:0100552	200
PROPOSED	400	00:00:00:6423458	2
GREEDY	500	00:00:0:0010006	250
RLF	500	00:00:02:5822682	2
DSTAUR	500	00:00:07:1147923	2
ETS	500	00:00:00:0125953	250
PROPOSED	500	00:00:01:4038308	2

VI. CONCLUSION

According to the results in the table, Dsatur algorithm takes the maximum time and the time and the time and the time complexity of RLF is also considerably high. Though greedy and ETS algorithms take less time as compared to the proposed algorithm, but they don’t give an optimized solution in terms of chromatic number. The proposed algorithm not only colors the graph with minimum number of colors but also takes significantly less time, as its complexity is O(n^2). Thus, the proposed algorithm can be used to produce optimized solutions for graph coloring problems, both in terms of execution time and chromatic number.

REFERENCES

[1] Priyadarshini. J, Anandhakumar. P. A Heuristic Approach for Graph Coloring using Combinatronics.
[2] Brelaz, D. (1979). New methods to color vertices of a graph. Communications of the ACM 22, 251-256.
[3] Leighton, F.T., A Graph Coloring Algorithm for Large Scheduling Problems, Journal of Research of the National Bureau of Standards 84 (1979), 489-503.
[4] Walter Klotz, Graph Coloring Algorithms.
[5] Brown, J. R., Chromatic scheduling and the chromatic number problem, Management Science 19 (1972), 456-463.