International Journal on Emerging Technologies 11(2): 854-862(2020)

ISSN No. (Print): 0975-8364
ISSN No. (Online): 2249-3255

Greedy Graph Coloring Algorithm Based on Depth First Search

Sumit Gupta[1] and Dhirendra Pratap Singh[2]
[1]Research Scholar, Department of Computer Science and Engineering, MANIT Bhopal (Madhya Pradesh), India.
[2]Assistant Professor, Department of Computer Science and Engineering, MANIT Bhopal (Madhya Pradesh), India.

(Corresponding author: Sumit Gupta)
(Received 04 January 2020, Revised 03 March 2020, Accepted 05 March 2020)
(Published by Research Trend, Website: www.researchtrend.net)

ABSTRACT: A greedy Graph Coloring Algorithm allocates non-identical colors to the adjacent vertices of a graph such that the number of assigned colors is minimized. In Greedy Coloring of the graph, the ordering of vertices is an essential parameter for allocating the colors to vertices of a graph. Assigning the color to the graph must be time efficient. In this paper, we have proposed a new algorithm in which the Depth First Search algorithm is used to give orders to the vertices of the graph. The objective of this work is to develop a fast algorithm of graph coloring to overcome the problems in existing methods, and it should be efficient for all kinds of graph instances. The proposed algorithm is computed on large and small benchmark graphs and compared with four well known coloring algorithms BC-COL Algorithm, DSATUR Algorithm, A cutting plane Algorithm and new DSATUR Algorithm. The computation result shows that the proposed algorithm has successfully evaluated the known chromatic number for 54 different graphs and perform best with all other compared algorithm. It also tells the minimum number of colors required to color 19 different graphs whose chromatic numbers are not known.

Keywords: Greedy Graph Coloring, Chromatic Number, Depth First Search, Graph Algorithm.

I. INTRODUCTION

Let G (V, E) be an undirected graph having V as a set of vertices and E as a set of edges. A coloring of the graph G (V, E) allocates the colors to the vertices of the graph, in such a way that if an edge (u, v) Є E, then C(u) ≠ C(v). Where C(u) and C(v) are the colors of vertex u and vertex v, respectively. A graph is k-colorable if it has proper k-coloring. In the proper coloring of the graph, the number of colors used is minimized so that every vertex gets a color that is non-identical to its adjacent vertices colors. If a graph can be colored using k colors, then the graph is called k-colorable.

The minimum number of colors needed for coloring a graph is known as the chromatic number of the graph and denoted by ϰ(G). It is the lower bound to colors needed to color the graph. The chromatic number ϰ(G) is least k such that Graph is k-colorable and ϰ(G) exists as allocating non-identical colors to vertices yields a proper k-coloring, and proper coloring of Graph is ϰ(G) coloring. A graph G (V, E) is k-chromatic if ϰ(G) = k. There is no general rule defining a chromatic number, and we instead place an upper bound on the chromatic number of a graph based on the maximum vertex degree of the graph. For a graph G (V, E) with a maximum vertex degree Δ, ϰ(G) ≤ f(Δ) where f(Δ) is some function of the maximum vertex degree. Determining the chromatic number of a graph is the NP-Hard problem [10].

Graph coloring has a various practical application related to scientific and real-life problems such as Register allocation [6], time tabling [5], frequency assignment [28], printed circuit testing [9], bag rationalization [12], Crew Scheduling [8, 14], Various puzzles games like Solving the Latin square completion problem [15] and satellite range scheduling [31].

In most cases, current existing algorithms of greedy and exact approaches solve the graph instances with a very limited number of vertices. For the larger graphs, the existing algorithm of greedy and exact approaches is optimal in terms of computation time but not optimal in terms of the least colors. In a survey by Malaguti and Toth (2010) [18] on the vertex coloring problem on greedy, exact, heuristics and meta-heuristics approaches are reported, and the results on computation time and the number of colors used dictates that specific type of existing algorithm is suitable for the specific type of graph instances. The concern of coloring is all admitted as complex in terms of finding the least colors. So, this paper focuses on finding a polynomial-time greedy algorithm for providing solutions to graph coloring.

The various greedy algorithms proposed on different strategies work by selecting the vertices in predefined order or using some rule. The number of colors used in the graph depends on the order of vertex in which it is processed.

Greedy coloring is coloring the vertex of a graph sequentially, and the order of sequence is decided by some rule and chooses the smallest possible color from the color set and assign it to vertices such that the color of two adjacent vertices is different. If an arbitrary sequence is given, (V_1, V_2, , V_n), of all the vertices of G (V, E), a Greedy coloring algorithm assigns a color to each vertex from V_1 to V_n in turn, using the smallest possible color value that is not already assigned to one of its adjacent vertexes. The assignment of color is sequential from V_1 to V_n. The algorithm is an algorithm used to properly color and ordered graph using k colors considering the order vertex and chromatic number is possible.

The basic greedy algorithm follows the step:

Algorithm 1: Basic Greedy Coloring Algorithm
Input: DIMACS graphs G (V, E).
Output: Colored Graph with the number of colors.

Step 1: Assign an order to the set of colors.

Step 2: Considering the first vertex in the vertex order, assign to it the first color.

Step 3: Considering the next vertex, assign to it the lowest-ordered color that has not already been assigned to a vertex adjacent to it.

Step 4: Repeat step 3 until the graph is colored.
Some Proposition for Greedy Algorithm.

Proposition 1 For any graph G, there is an order that can be assigned to the vertices of G for which the greedy coloring algorithm will use the graph's chromatic number of colors to properly color G.
Proposition 2 For any connected graph, G, there is an order in which one can place the vertices of G such that every vertex has a higher-ordered neighbor, except for the last vertex in the order.

The rest of the paper is organized in five sections. Section II summarizes some previous works done related to graph coloring algorithms. Section III explains the proposed algorithm. Section IV discusses about the test machine setup, test dataset used, and the final results of the algorithms. Section V concludes the proposed work.

II. RELATED WORK

In literature, various exact and greedy approaches are available to solve the graph coloring problem. The first approach is given by Brown [4], it is based on sequential coloring and allocates colors to the graph one by one to each vertex by using the colors that are pre-allocated to the vertex and if the conflict persists then add the new color to allocate. The algorithm is further improved by Brélaz [3] in the DSATUR algorithm. This algorithm works by dividing the problem into sub-problems, and the sub-problem is generated when it tries to optimize the main problem. DSATUR generates partial coloring, which is referred to as a clique, and its size is used as the lower bound. DSATUR determines the future color availability based on degree and runs in $O(n^2)$. Brélaz (1979) asks for coloring the next vertex of the highest chromatic degree, and if it ties with vertex, then select a degree with a maximum uncolored vertex in graph else solves it lexicographically [3].
Sewell (1996) gave the countable improvement in DSATUR algorithm and defined a new rule SEWELL that if vertices are tied at maximum degree, then select one of the vertices maximal number of common available colors in the neighborhood of uncolored vertices. The reported results of the SEWELL are better than DSATUR for benchmark random graph and a small set of graphs describing real-world problems; instead, it runs $O(n^3)$ with more overhead than DSATUR [27].
Segundo (2012) [26] describes a new rule PASS, computed faster than Sewell (1996) [27] as it has reduced the overhead. Due to the restriction to a subset of vertices, i.e., destined to reducing color domains of vertices, which are already known to have the least number of colors that are available.
To optimize the more, Segundo [26] uses PAAS rule selectively to a specific set of vertices, mostly in the later phase of the algorithm, as the number of vertices is less compared to the initial phase. The incorporation of this rule brings to many instances gets optimized in terms of the chromatic number, but using any optimization rule to the existing algorithm, it is always a possibility of overhead to time complexity that leads to an increase in the total computation of the algorithm. So this rule has brought up the new upper bounds to the chromatic number.
Méndez-Díaz and Zabala (2006) propose an improvement based on giving solutions to symmetry inequality constraints towards selecting a vertex for coloring. It includes preprocessing procedures for removing vertices; remove the vertices that, if color allocation to the current graph, would not lead to add a new color in the resulting graph [21]. Méndez-Díaz and Zabala (2008) make further improvements to his own algorithm by removing the symmetry that would come from color in distinguish ability and defines two additional rules [22].
Xu and Jeavons (2015) used a greedy approach and proposed a randomized algorithm for distributed coloring that uses local processing at vertices and messages along the edges explained in two versions of the algorithm. The approach is regarding the processors exchanges message along the edge, the message is in the form of potential color values, and each processor has minimal graph knowledge [29].
Two versions of algorithm compute Greedy coloring, after different expecting steps of which takes $O(\Delta^2 \log^2 n)$, $O(\Delta^2 \log n)$, respectively. Where n is the number of vertices, and is the maximum degree of the graph [11, 13]. Manne and Boman (2005) described the greedy algorithm for sparse random graph towards the balance coloring. The number of available colors is predicted and given by a prediction formula. The predicted color is γ, and the prediction interval is decided as [1, γ]. Three greedy strategies First Fit algorithm, Least used algorithm, and Random algorithm selects the smallest color in the predicted interval based on the rules [19].
Lucet *et al.*, (2006) proposed a technique of the exact method of vertex coloring that is based on the linear decomposition of the graph. The graph is successively decomposed into sub-graphs at each stage of decomposition, and boundary set vertices give the solution of resolved graph. The linear decomposition principle is stated [17]. The benefit gets from this method is its complexity depends on linear width, not on the size of the graph. The coloring algorithm works in the current step, examining only those sub-graphs that have no edge between two vertices of a graph generated in the precedent step, and coloring rules are applied. In the last step, the configuration set gives the chromatic number.
Ouerfelli and Bouziri (2001) [25] propose the greedy algorithm for the dynamic graph coloring exploiting the same approach used by DSATUR [3, 26, 27]. It describes the First Fit algorithm and the three selection rules for ordering the sequence the vertex that is different from the DSATUR. Zaker (2008) defines a set in a graph for greedy algorithms. In most cases, the greedy algorithm uses more colors.

To eliminate the extra color used by greedy algorithms, some of the vertices are pre-colored before the algorithm starts. The pre-colored set of vertices leads to optimal coloring [30].

Lu *et al.*, [16] proposed a parallel balanced coloring algorithm, which is a practical implementation of the equitable coloring [23]. Lu *et al.*, try to distribute an equal number of vertices in every color classes. It can be obtained into two steps: First, by using greedy coloring, they get almost balanced coloring of the graph. In the next step, It uses a vertex-centric parallelization scheme or a color-centric parallelization scheme for getting balanced coloring. In the vertex-centric scheme, the vertex of different colors is parallelly moved from an over-full bin to the under-full bin while in color centric vertices of the same color move parallelly.

Laurent et al. proposed two versions of the new self-replicating algorithm called HEAD' and HEAD [24]. It is based on the Hybrid Evolutionary algorithm (HEA) [7], which is proposed by Galiner and Hao. HEAD' uses TabuCOl and GPX algorithm for escaping from local minima. It is a simple algorithm which contains two TabuCol algorithm which interacts each other. The major drawback of HEAD' algorithm is premature convergence means without getting the legal coloring, it stops. For dealing with this premature convergence, Laurent et al. give HEAD in which they add two other candidate solution for maintaining the population diversity. The key idea is to replace one of the solutions with the solution previously obtain.

Artacho *et al.*, (2018) gives an efficient graph coloring algorithm which obtained the graph coloring with the help of the Douglas–Rachford algorithm. Here graph coloring is done by considering a different formulation, based on semi-definite programming. This new approach is demonstrated with the help of numerous numerical experiments [1].

III. PROPOSED WORK

In the proposed approach, the greedy coloring of vertices that colors the vertices of a graph sequentially and using the color that is pre-allocated to vertices or adds up a new color to allocate is described. The sequencing of vertices is an important parameter in terms of minimizing the number of colors. The results in greedy approaches are obtained in polynomial time if compared with heuristics and Meta-heuristics approaches. The proposed method uses the Depth-first Search for sequencing the vertices of a graph that takes the input sequence of the graph and returns the vertices of a graph in the sequence of its traversal to the graph by assigning a non-zero increasing positive integer called as an index to every vertex. The coloring starts according to the sequence given by the Depth-first traversal and continues until all the vertices get processed and return the number of colors.

The proposed Graph coloring algorithm is based on the Depth First Search. DFS is used to provide the order to the vertices of the graph in which they will be color. It will maintain the connectivity between the vertices which get colored one after another. The algorithm is exploiting the properties of a greedy approach is proposed. The primary objective of using the greedy approach is that the number of colors used in the coloring is to be minimized and secondarily the speed of the algorithm. Greedy algorithms for coloring compute the result in polynomial time. All steps of the algorithm are explained in the next section.

A. Proposed Algorithm

In the proposed algorithm coloring of vertices of the graph is done using a greedy algorithm, and for ordering the vertices, we employed DFS traversal. During the coloring, the connectivity of the graph is preserved. The algorithm works in the following steps.

Algorithm 2: Proposed Coloring Algorithm
Input: DIMACS graphs G(V, E).
Output: Colored Graph with number of color χ.

Step 1: Repeat the step 2-5 for all vertexes of graph one by one.

Step 2: The DFS is applied to the graph and assigned a unique number to each vertex in increasing order starting from 1.

Step 3: The DFS start from vertex selected in step 1 and continue until all the vertices in the graph are visited and returns a DFS sequence or index. The color set is initialized to zero. Two lists are maintained for each vertex.
(i) Adjacent vertexes having an index less than the current vertex index.
(ii) Adjacent vertexes having an index greater than the current vertex index.

Step 4: This phase starts coloring of the graph, it starts with vertex having index 1, following the DFS sequence and continues to color all the vertexes until the graph gets colored. During any iteration of the algorithm, the following rules are followed
(i) Vertex, which is having a greater index than its adjacent vertex, can only get modified its color.
(ii) Always assign a minimum possible color from the current color set. If color conflicts to all adjacent vertexes, then add a new color to the color set.
(iii) The color assignment is done in increasing order of their vertex index.

Step 5: If at any iteration while color allocation, the algorithm is at the current vertex, then the color assignment is done in the following way.
(i) Color Assignment to Current Vertex:.
a. First checks the color with all the vertexes having vertex number less than the current vertex.
b. If, when any vertexes color is matched with the color of vertex having less vertex number than current vertex, then assign a minimum color from the current color set. A color that differs from its all adjacent vertexes color.
c. If all the color of the color set are already assigned to adjacent vertex, then assign a new color to current vertex and add this to color set.
(ii) Color Assignment to Adjacent Vertexes:
a. First, it checks, the color of all vertexes one by one having vertex number greater than the current vertex number.
b. If a vertex is not assigned any color till now, then assign a minimum color from the color set, which is different from the current vertex color.
c. When the current vertex color is matched with a vertexes color having a larger vertex number than the current vertex, then change this adjacent vertex having a larger vertex number.
d. Assign a minimum color from the current color set, which differs from the current vertex color

Step 6: The number of colors in the color set gives the chromatic number of the graph.

B. Complexity of Algorithm

The time complexity of an algorithm exhibits its performance. In our proposed algorithm, the time

complexity is the sum of the Depth First Search sequencing and the Coloring algorithm. Let n be the number vertexes in the Graph, d be the maximum degree of the graph k be the chromatic number of Graph.
Ordering or labeling the vertices of the graph = O(n).
Coloring the vertices of graph = O(kdn).
So, the total Complexity = O(n) + O(kdn) ⊃ O(kdn).
The time required for the algorithm to color vertices of graph = O(kdn).

These operations are repeated n times, so the final complexity of the algorithm is $O(kdn^2)$.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

The proposed algorithm is implemented on a machine having Intel (R) i7-8700 3.20 GHz processor, 8 GB RAM, and windows10 operating system. The C programming language is used to code the proposed algorithm. The key data structure used for storing and manipulation is 'Structure'. The results are computed on benchmark instances of the DIMACS graph dataset [2].

Table 1: Graph instances with matched LB to chromatic number.

Instances	n	m	ℵ	BC-COL Algorithm [23]			DSATUR Algorithm [14]			A cutting plane Algorithm [15]		new DSATUR Algorithm [13]		Proposed Algorithm	
				LB	UB	Time	LB	UB	Time	LB	Time	LB	Time	LB	Time
Myceil3	11	20	4											4	0.47
Myceil4	20	71	5											5	0.48
Myceil5	47	236	6											6	0.88
Myceil6	95	755	7	5	7	7200	2	7	7200			7	tout	7	0.86
Myceil7	191	2360	8	5	8	7200	2	8	7200			8	tout	7	1.45
Fpsol2_i_1	496	11654	65	65	65	0.6	65	65	0.1	65	8	68		65	12.04
Fpsol2_i_2	451	8691	30	30	30	1.2	30	30	0.1	30	1	30		30	10.86
Fpsol2_i_3	425	8688	30	30	30	1.2	30	30	0.1	30	1	30		30	11.89
inithx.i.1	864	18707	54	54	54	Init	54	54	Init			54	0.0	54	39.63
inithx.i.2	645	13979	31	31	31	Init	31	31	Init			31	0.0	31	16.61
inithx.i.3	621	13979	31	31	31	Init	31	31	Init			31	0.0	31	18.53
mulsol.i.1	197	3925	49	49	49	Init	49	49	Init			49	0.0	49	2.69
zeroin.i.1	211	4100	49	49	49	Init	49	49	Init			49	0.0	49	3.2
zeroin.i.2	211	3541	30	30	30	Init	30	30	Init			30	0.0	30	1.4
zeroin.i.3	206	3540	30	30	30	Init	30	30	Init			30	0.0	30	1.3
Anna	138	493	11	11	11	Init	11	11	Init			11	0.0	11	0.34
David	87	406	11	11	11	Init	11	11	Init			11	0.0	11	0.26
Homer	561	1629	13	13	13	Init	13	13	Init					13	1.97
Huck	74	301	11	11	11	Init	11	11	Init			11	0.0	11	1.03
Jean	80	254	10	10	10	Init	10	10	Init			10	0.0	10	1.02
1-insertions_4	67	232	5	5	5	2								5	0.22
3-insertions_3	56	110	4	4	4	1	4	4	5	3	0	4	0.3	4	0.21
4-insertions_3	79	156	4	3	4	7200	2	4	7200	3	0	4	96.9	4	0.49
1-FullIns_4	93	593	5	5	5	0.1				4	0	5	0.0	5	0.41
1-FullIns_5	282	3247	6	4	6	7200	3	6	7200	4	0	6	tout	6	1.31
2-FullIns_3	52	201	5	5	5	0.1	5	5	1014	5	0	5	tout	5	0.53
2-FullIns_4	212	1621	6	5	6	7200	4	6	7200	6	4	6	tout	6	0.77
2-FullIns_5	852	12201	7	5	7	7200	4	7	7200			7	tout	7	9.38

Continued …….

3-FullIns_3	80	346	6	6	6	0.1				6	0	6	tout	6	0.68
3-FullIns_4	405	3524	7	6	7	7200	5	7	7200	6	4	7	tout	7	2.20
3-FullIns_5	2030	33751	8	6	8	7200	5	8	7200	6	292	8	tout	8	86.02
4-FullIns_3	114	541	7	7	7	3				7	0	7	tout	7	0.45
4-FullIns_4	690	6650	8	7	8	7200	6	8	7200	7	16	8	tout	8	5.56
5-FullIns_3	154	792	8	8	8	20						8	tout	8	0.51
5-FullIns_4	1085	11395	?	8	9	7200	7	9	7200	8	55	9	tout	9	10.73
mulsol.i.2	188	3885	31	31	31	Init	31	31	Init			31	0.0	31	1.83
mulsol.i.3	184	3916	31	31	31		31	31				31	0.0	31	1.79
mulsol.i.4	185	3946	31	31	31		31	31				31	0.0	31	1.90
mulsol.i.5	185	3973	31	31	31		31	31				31	0.0	31	1.79
School1	385	19095	14	14	14		14	14				14	0.1	14	20.2
School1_nsh	352	14612	14	14	14		14	14				14	0.4	14	12
games120	120	638	9	9	9		9	9						9	1.13
miles250	128	387	8	8	8	Init	8	8	Init			8	0.0	8	1.20
miles500	128	1170	20	20	20	Init	20	20	Init			20	0.0	20	1.83
Miles750	128	2113	31	31	31	Init	31	31	Init			31	0.0	31	2.76
Miles1000	128	3216	42	42	42	0.02	42	42	0.1	42	0	42	0.0	42	2.89
Miles1500	128	5198	73	73	73	0.1	73	73	0.1	73	0	79	0.0	73	3.42
Fpsol2_i_1	496	11654	65	65	65	0.6	65	65	0.1	65	8	65		65	12.04
Fpsol2_i_2	451	8691	30	30	30	1.2	30	30	0.1	30	1	30		30	10.86
Fpsol2_i_3	425	8688	30	30	30	1.1	30	30	0.1	30	1	30		30	11.89
Mug88_1	88	146	4	4	4	11						4	324	4	0.21
Mug88_25	88	146	4	4	4	184	4	4	4756			4	191	4	0.37
Mug100_1	100	166	4	4	4	60						4	tout	4	0.28
Mug100_25	100	166	4	4	4	60						4	tout	4	0.27

The implementation of our algorithm, above mentioned, as specification keeping in mind and the standard specification that was required for successful execution is taken and tried for the avoidance of all possible hindrance and errors. The ideal platform and requisite language are taken into consideration. In this, we are experimentally demonstrating the result of our algorithm, and results are summarized in the table. The results shown here are executed 25 times on a machine mentioned above. The time shown in the table for execution is average computations. All the computations performed on the standard benchmark instances of the DIMACS graph [2]. We have assumed the computation time of other algorithms on standard parameters as all the algorithms are executed on different hardware.

In our experiment, we test the proposed algorithm and compared benchmark algorithm on the widely used graph coloring library from DIMACS benchmark instances. Various classes of the graphs are present in this library like as random or quasi-random graphs, problems based on register allocation for variables in real codes, or class scheduling graphs, among others. Most of the graph coloring problem uses this DIMACS graph instances. In the table the first column represents the instances, the second column represented as 'n' shows the number of vertices in the graph instances, the third column represented by 'm' shows the number of edges in the graph, the fourth column represented by 'א' shows the known chromatic number of the graph. The upper bound and lower bound of chromatic number is represented by UB, LB, respectively. The last column gives the result of the proposed algorithm. The time in the table is considered in seconds. The 'init' represents the initial time of the respective algorithm. The remaining columns are the result of an existing algorithm with their referenced mark. The results summarized in Table 1 show the graph instance for which the LB result of the proposed algorithm in matched with a chromatic number of the respective graph and results of the rest of instances are shown in Table 2.

Table 1 is a summarization of the best results that we have found in our proposed algorithm. The results are also compared to existing algorithm [20, 21, 22, 26]. For 54 graphs proposed algorithm has found the chromatic numbers similar to current known or proved lower bounds. The graph from Mycielski Transformation family myceil3, myceil4, myceil5, and myceil6, myceil7 found their chromatic number in time less than 1 second.

Chromatic number of graphs 4-insertions_3, 1-FullIns_5, 2-FullIns_3, 2-FullIns_4, 2-FullIns_5, 3-FullIns_4, 3-FullIns_5 and 4-FullIns_4 are evaluated very fast in comparison of other algorithms and for graph 5-FullIns_4 chromatic number similar to given by other algorithms is evaluated in 10 seconds.

Table 2: Experimental Results on Graph Instances.

Instances	n	m	ℵ	BC-COL Algorithm [23]			DSATUR Algorithm [14]			A cutting plane Algorithm [15]		new DSATUR Algorithm [13]		Proposed Algorithm	
				LB	UB	Time	LB	UB	Time	LB	Time	LB	Time	LB	Time
DSJC125.1	125	736	5	5	5	0.9	5	5	0.1	5	1	5	0	7	0.80
DSJC125.5	125	3891	?	13	20	7200	9	19	7200	12	7	19	tout	23	1.46
DSJC125.9	125	6961	?	42	47	7200	29	45	7200	42	354	46	tout	54	1.89
DSJC250.1	250	3218	?	5	9	7200	4	9	7200	5	11	9	tout	12	1.84
DSJC250.5	250	15668	?	13	36	7200	9	35	7200	14	3339	34	tout	40	6.80
DSJC250.9	250	27897	?	48	88	7200	34	87	7200	48	3605	82	tout	95	22.01
DSJC500.1	500	12458	?	5	15	7200	5	15	7200			14	tout	18	6.70
DSJC500.5	500	62624	?	13	63	7200	9	63	7200	13	538	62	tout	68	99
DSJC500.9	500	224874	?	59	161	7200	43	160	7200	59	5870			169	366.3
DSJC1000.1	1000	49629	?	6	26	7200	5	25	7200			25	tout	30	164
DSJC1000.5	1000	249826	?	15	116	7200	10	114	7200			110	tout		
DSJC1000.9	1000	449449	?	65	301	7200	53	300	7200	66	4546	300	tout		
DSJR500.1	500	3555	12	12	12	Init	12	12	Init			12	0	13	2.82
DSJR500.1c	500	121275	?	78	88	7200	70	88	7200	80	4470	85	tout	107	367.6
DSJR500.5	500	58862	122	119	130	Init	103	130	Init	119	1211	130	tout	140	94.58
Latin_sq_10	9000	307350	?	90	129	7200	90	129	7200						
le_450_5a	450	5714	5	5	9	7200	5	9	7200					12	3.84
le_450_5b	450	5734	5	5	9	7200	5	9	7200					12	5.21
le_450_5c	450	9803	5	5	5	7200	5	5	Init			5	0	14	5.06
le_450_5d	450	9757	5	55	10	Init	5	8	7200			5	98.1	14	5.41
le_450_15a	450	8168	15	15	17	7200	15	17	7200			16	tout	20	6.13
le_450_15b	450	8169	15	15	17	7200	15	16	7200			16	tout	20	8.72
le_450_15c	450	16680	15	15	24	7200	13	23	7200			22	tout	29	10.86
le_450_15d	450	16750	15	15	23	7200	13	23	7200			23	tout	29	15.17
le_450_25a	450	8260	25	25	25	Init	25	25	Init			25	0	27	6.81
le_450_25b	450	8263	25	25	25	Init	25	25	Init			25	0	26	7.83
le_450_25c	450	17343	25	25	28	7200	20	28	7200					34	12.69
le_450_25d	450	17425	25	25	28	7200	21	27	7200					33	10.98
queen5_5	25	160	5											5	1.62
queen6_6	36	290	7											8	1.72
queen7_7	39	476	7											10	0.98
queen8_8	64	728	9	9	9	3	9	9	18			9	3.0	11	1.14
queen8_12	96	1368	12	12	12	Init	12	12	Init			12	0.0	14	0.88
queen9_9	81	1056	10	9	11	7200	9	10	7200			10	466	12	1.40
queen10_10	100	2940	11	10	12	7200	10	12	7200			12	tout	14	1.50
queen11_11	121	3960	11	11	14	7200	11	13	7200			13	tout	15	1.40

Continued.....

queen12_12	144	5192	12	12	15	7200	12	14	7200			14	tout	17	2.00
queen13_13	169	6656	13	13	16	7200	13	15	7200			15	tout	18	2.34
Queen14_14	196	8372	14	14	17	7200	14	17	7200			16	tout	20	1.89
Queen15_15	225	10360	15	15	18	7200	15	18	7200			18	tout	21	2.12
Queen16_16	256	12640	16	16	20	7200	13	19	7200			19	tout	22	2.14
1-insertions_5	202	1227	?	4	6	7200	2	6	7200	3	0	6	tout	6	0.48
1-insertions_6	607	6337	?	4	7	7200	2	7	7200	3	3	7	tout	7	2.56
2-insertions_4	149	541	?	4	5	7200	2	5	7200	3	0	5	tout	5	0.55
2-insertions_5	597	3936	?	3	6	7200	2	6	7200	3	3	6	tout	6	3.07
3-insertions_4	281	1046	?	3	5	7200	2	5	7200	3	0	5	tout	5	0.80
3-insertions_5	1406	9695	?	3	6	7200	2	6	7200	3	61			6	9.80
4-insertions_4	475	1795	?	3	5	7200	2	5	7200	3	2	5	tout	5	1.4
wap01	2368	110871	?	41	46	7200	39	48	7200			47	tout		
wap02	2464	111742	?	40	45	7200	39	46	7200			46	tout		
wap03	4730	286722	?	40	56	7200	40	55	7200						
wap04	5231	294902	?	40	50	7200	20	48	7200						
wap05	905	43081	?	50	51	7200	27	51	7200			50	50	50	297.1
wap06	947	43571	?	40	44	7200	33	45	7200			48	48	48	129.87
wap07	1809	103368	?	40	46	7200	23	46	7200			45	45		
wap08	1870	104176	?	40	47	7200	23	45	7200			45	45		
qg_order30	900	26100	30	30	30	Init	30	30	Init			30	0.0		
qg_order40	1600	62400	40	40	42	7200	40	42	720			40	0.2		
qg_order60	3600	212400	60	60	63	7200	60	63	7200			61	tout		
ash331GPIA	662	4185	4	4	4	51	4	4	0.7	4	46	4	0.0	6	3.64
ash608GPIA	1216	7844	4	4	4	692	4	4	3	4	692	4	0.1	6	8.73
Ash958GPIA	1916	12506	?	4	5	7200	3	5	7200	4	4236	4	0.4	7	34
abb313GPIA	1557	46546	?	8	10	7200	6	10	7200			10	tout		
will199GPIA	701	6772	7				7	7	1.2			7	tout	8	8.62

All fpsol graphs have found the known chromatic number and the chromatic number of fpsol_i_1, fpsol_i_2, fpsol_i_3 are 65, 30, 30 respectively. All fpsol instances are taking a time approximately 12 seconds. Mugg graphs are solved with equal to the chromatic number in the minimum of time compared to other algorithms. It is solved within 1 second for all executions. The ash graph, abb graph, and will graphs are solved with equal to chromatic number with a maximum of computation time 32 seconds. School graphs have found the known chromatic number.

Results of the register graphs show that chromatic numbers with equal to their known chromatic number. Results for the family of mulsol graph also equal to a chromatic number.Every graph is solved within 3 seconds. All zeroin graphs are solved within 4 seconds. Results of Anna, david homer, huck, and jean are equal to a chromatic number. Games graph is equal to the chromatic number, and all miles graph are evaluated to their known chromatic number in less than 4 seconds.

Table 2 shows the computation results on remaining benchmark instances of large and small graphs and it contains the lower bound to the chromatic number given by our algorithm, the time taken to compute the results and the comparison with the existing algorithm [17-19, 23]. In this, we have compared our resulted lower bounds to the chromatic number and the computation time overall the family of graphs.

So starting with the random graph we have tested 15 random graph with most of them are unknown chromatic numbers and the result is compared to other existing algorithms [20-22, 26] and it shows that for all the random graph the result is very bad in terms of chromatic number and it clear that except for DSJC125.1 in which result is very close to lower bound of chromatic number and it is equal to 5. Out of 15 random graphs 13 are solved in efficient time with the highest time of 366.3. The Latin graph is unsolved in our case.

When we are comparing the results of a family of le_450_x graph with existing algorithms, the results are more than the upper bound that is given in the Table 2. The upper bound for le450_5a counted is 12 in 3.84 seconds. The last graph of this type of le450_25d is gained a chromatic number of 33. Every 15 instances are solved with a minimum time of 3.84 seconds and a maximum time of 15.17.

For the queen graph, we have obtained the chromatic number more than the upper bound and with every instance is solved within 3 seconds. Only Queen5_5 is

equal to its chromatic number; remaining instances are higher than the upper bound. Every Queen graph is solved.

For a family of fullIns graph, results are equal to the chromatic number. Every FullIns graph is solved; the computation time is efficient than other algorithms. 4-fullIns-5 has remained unsolved. The time taken to compute results for these graphs is less than 10 seconds except for 3-fullIns-5, which takes 86 seconds. So it has an advantage that with the equal chromatic numbers, these instances have given the best results in terms of time.

For the results on the family of the wap graph, only two instances are solved others remained unsolved, and it is exited from the execution giving a large integer value. In this type of graph, the density ratio of edges with respect to the vertex is very high. All the qg_ order graph instances are unsolved to our algorithm. The wap05 graph is solved with equal to chromatic number with very efficient time compared to other algorithms and is equal to 297 seconds.

IV. CONCLUSION

This paper has presented an effective method of graph coloring, that is based on the greedy approach, and the Depth First Search decides the sequencing. In order to decrease the computation time, a greedy algorithm is employed. Coloring of the vertices in their Depth-First search ordering keeps the connectivity between vertices, which are colored in sequence. The proposed algorithm computes nearly all instances in very less time. Challenging graph instances are tested on the proposed algorithm, and the result shows that for most of the graph known chromatic number are matched. The comparison with the existing algorithm is conducted, and it shows that the objective is justified. The proposed algorithm has matched with the currently known chromatic number for 54 graph instances and minimized the time of other graph instances as compared to other algorithms.

V. FUTURE SCOPE

Greedy graph coloring can be applied to various real-life applications where the sequential algorithm is converted into the parallel. By the use of coloring nodes of the same color can directly apply parallelly. There are some graphs where our algorithm cannot give the results like Wap and q_g_order or on some graph; it did not provide the optimal results also algorithm can be applied to some vast dataset. In the future, we will try to apply this algorithm to find the community of the densely connected nodes to retrieve the useful information.

ACKNOWLEDGMENTS

We are very thankful to all the faculty members, research scholars, and supporting staff of Maulana Azad National Institute of Technology, Bhopal, India, for the continued encourages and support to complete this work.

Conflict of Interest. The authors declare that there is no conflict of interest of any sort on this research.

REFERENCES

[1]. Artacho, F. J. A., Campoy, R., & and Elser, V. (2018). An enhanced formulation for solving graph coloring problems with the Douglas–Rachford algorithm. *Journal of Global Optimization*, 1-21.

[2]. Bader, D. A., Henning, M., & Sanders P. (2004). DIMACS graph dataset. https://mat.tepper.cmu.edu/COLOR/instances.html.

[3]. Brélaz, D. (1979). New methods to color the vertices of a graph. *Communications of the ACM, 22*(4), 251–256.

[4]. Brown, J. R. (1972). Chromatic scheduling and the chromatic number problem. *Management Science, 19*(4-part-1), 456–463.

[5]. Burke, E. K., Meisels, A.,Petrovic, S., & Qu, R. (2004). A graph-based hyper heuristic for timetabling problems. Computer Science Technical Report No. NOTTCS-TR-2004-9, University of Nottingham.

[6]. de Werra, D., Eisenbeis, C.,Lelait, S., & Marmol, B. (1999). On a graph-theoretical model for cyclic register allocation. Discrete Applied Mathematics, *93*(2-3), 191–203.

[7]. Galinier, P. & Hao, J. K. (1999). Hybrid evolutionary algorithms for graph coloring. *Journal of combinatorial optimization, 3*(4), 379–397.

[8]. Gamache, M., Hertz, A., & Ouellet J. O. (2007). A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding. *Computers & operations research, 34*(8), 2384–2395.

[9]. Garey, M., Johnson, D., & So, H. (1976). An application of graph coloring to printed circuit testing. IEEE Transactions on circuits and systems, *23*(10), 591–599.

[10]. Garey, M. R. & Johnson, D. S. (1979). Computers and intractability. Freeman San Francisco, Vol. 174.

[11]. Gavoille, C., Klasing, R., Kosowski, A., Kuszner, L., & Navarra, A. (2009). On the complexity of distributed graph coloring with local minimality constraints. Networks. *An International Journal, 54*(1), 12–19.

[12]. Glass, C. (2002). Bag rationalization for a food manufacturer. *Journal of the Operational Research Society, 53*(5), 544–551.

[13]. Grundy, P. M. (1939). Mathematics and games. *Eureka, 2*, 6–9.

[14]. Izadkhah, H. (2019). Learning based genetic algorithm for task graph scheduling. *Applied Computational Intelligence and Soft Computing*, 1-16.

[15]. Jin, Y., & Hao, J. K. (2019). Solving the Latin square completion problem by memetic graph coloring. *IEEE Transactions on Evolutionary Computation, 23*(6), 1015-1028.

[16]. Lu, H., Halappanavar, M., Chavarria-Miranda, D., Gebremedhin, A. H., Panyala, A., & Kalyanaraman, A. (2016). Algorithms for balanced graph colorings with applications in parallel computing. *IEEE Transactions on Parallel and Distributed Systems, 28*(5), 1240–1256.

[17]. Lucet, C., Mendes, F., & Moukrim, A. (2006). An exact method for graph coloring. *Computers & operations research, 33*(8), 2189–2207.

[18]. Malaguti, E., & Toth, P. (2010). A survey on vertex coloring problems. *International transactions in operational research, 17*(1), 1–34.

[19]. Manne, F., & Boman, E. (2005). Balanced greedy colorings of sparse random graphs. The *Norwegian Informatics Conference, NIK,* 113–124.
[20]. Mehrotra, A., & Trick, M. A. (1996). A column generation approach for graph coloring. Informs *Journal on Computing, 8*(4), 344–354.
[21]. Méndez-Díaz, I., & Zabala, P. (2006). A branch-and-cut algorithm for graph coloring. Discrete Applied Mathematics, 826–847.
[22]. Méndez-Díaz, I., & Zabala, P. (2008). A cutting plane algorithm for graph coloring. *Discrete Applied Mathematics*, 159–179.
[23]. Meyer, W. (1973). Equitable coloring. The American Mathematical Monthly, *80*(8), 920–922.
[24]. Moalic, L., & Gondran, A. (2018). Variations on memetic algorithms for graph coloring problems. *Journal of Heuristics, 24*(1), 1–24.
[25]. Ouerfelli, L., & Bouziri, H. (2011). Greedy algorithms for dynamic graph coloring. *2011 International Conference on Communications, Computing and Control Applications (CCCA), IEEE,* 1–5.
[26]. San Segundo, P. (2012). A new Dastur-based algorithm for exact vertex coloring. *Computers & Operations Research, 39*(7), 1724–1733.
[27]. Sewell, E. (1996). An improved algorithm for exact graph coloring. DIMACS series in discrete mathematics and theoretical computer science, *26*, 359–373.
[28]. Smith, D. H., Hurley, S., & Thiel, S. (1998). Improving heuristics for the frequency assignment problem. *European Journal of Operational Research, 107*(1), 76–86.
[29]. Xu, L., & Jeavons, P. (2015). Patterns from nature: Distributed greedy coloring with simple messages and minimal graph knowledge. *Information Sciences,* 550–566.
[30]. Zaker, M. (2008). Greedy defining sets in latin squares. *Ars Combin, 89*, 205–222.
[31]. Zufferey, N., Amstutz, P., & Giaccari, P. (2008). Graph coloring approaches for a satellite range scheduling problem. *Journal of Scheduling, 11*(4), 263–277.

How to cite this article: Gupta, S. and Singh, D. P. (2020). Greedy Graph Coloring Algorithm based on Depth First Search. *International Journal on Emerging Technologies*, *11*(2): 854–862.