Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Technology 4 (2012) 261 – 266

Procedia Technology

C3IT 2012

A novel scheme for graph coloring

Vishal Donderia[a], Prasanta K. Jana[a]

[a]*Department of Computer Science and Engineering*
Indian School of Mines, Dhanbad-826 004, India

Abstract

Graph coloring is an important problem with its wide variety of applications. The problem is *NP*-hard in nature and no polynomial time algorithm is known for it. In this paper, we propose a new method for graph coloring. The proposed scheme is efficient with respect to simplicity, robustness and computation time.

Keywords:Graph coloring; *NP*-hard problem; chromatic number; time complexity.

1. Introduction

Let $G(V, E)$ be an undirected graph with the set of vertices (nodes) V and the set of edges E. A coloring of the graph $G(V, E)$ is a mapping $C: V \rightarrow S$, such that if an edge $(u, v) \in E$, then $C(u) \neq C(v)$. In other words, it is defined as coloring the vertices of a graph such that no two adjacent vertices have the same color. The minimum number of colors needed for coloring a graph is known as chromatic number. It is known that the graph coloring problem is *NP*-hard [1]. Despite its *NP*-hardness, the graph coloring problem has been studied extensively due to its wide variety of applications such as in scheduling [2], [3], register allocation [4], printed circuit testing [5], manufacturing [6], frequency assignment [7], timetabling [8] and satellite range scheduling [9]. Many heuristics have been proposed to handle large graph coloring problem. Greedy based heuristics were proposed in which the vertices of the graph are colored one by one guided by a predefined greedy function. The largest saturation degree (DSATUR) [10] and the recursive largest first (RLF) [11] are the two best known heuristics in this class. Various local search algorithms based on Tabu search [12], [13], [15] and simulated annealing [16], [17]. Other LSA can be found in [18], [19], [20], [21]. In the recent year, many other algorithms have been proposed which can be found in [22], [23]. However, many of these algorithms, particularly genetic algorithms and simulated annealing based, suffer from high running time, when dealing with large graph problems. Some of the algorithms are not robust enough and produce unsatisfactory results for many applications. In this paper, we propose a method which is simple and robust in dealing large graph problems. Our method is capable of producing comparable results in lesser time than the best known algorithm DSATUR [10].

2. Proposed algorithm

The basic idea of the algorithm is as follows. We maintain a table having the four attributes namely, vertex, color, flag and color-assigned. The initial configuration of this table is shown in Fig. 1 for the augmented cube shown in Fig. 2. Here, we assume that the number of colors available is equal to the number of the nodes of the graph. At any iteration, we choose one vertex, say '*x*' of the graph to color it. At the beginning of the iteration, we reset all the flag bits to 0. We now test every vertex adjacent to '*x*' to examine whether it is colored. If it is so, we check its color and put the flag bit 1 corresponding to its color. We now scan the flag bit vector from the beginning to search for the first flag bit as 0. This indicates that the corresponding color has not yet been used. So we assign this color for the vertex '*x*'. We consider the next vertex and follow the same procedure to color it. The method is repeated until all the vertices of the graph are colored.

vertex	1	2	3	4	5	6	7	8	9
color	1	2	3	4	5	6	7	8	9
flag	0	0	0	0	0	0	0	0	0
color-assigned									

Fig. 1. Initial table configuration

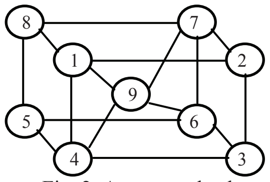

Fig. 2. Augmented cube

Let us first illustrate the method stepwise for coloring the augmented cube shown in Fig. 2.

Step 1: We randomly choose any one vertex from the graph. Without any loss of generality, we start coloring with the vertex 1. Initially all the flag bits are zero. This indicates that no color has been used so far. Therefore, we assign color 1 to the vertex 1 and set the corresponding flag bit 1. The situation is shown in Fig. 3.

node	1	2	3	4	5	6	7	8	9
color	1	2	3	4	5	6	7	8	9
flag	1	0	0	0	0	0	0	0	0
color-assigned	1								

Fig. 3. Table configuration after 1st iteration

Step 2: Now we move to the next vertex, i.e., the vertex 2. We first reset all the flag bits to 0 as shown in Fig. 4.

node	1	2	3	4	5	6	7	8	9
color	1	2	3	4	5	6	7	8	9
flag	0	0	0	0	0	0	0	0	0
color-assigned	1								

Fig. 4. Table configuration after resetting flag bits to 0

The adjacent vertices of the vertex 2 are 1, 3 and 7 (see Fig. 1).We now consult the color-assigned vector in Fig. 4 and observe that out of these adjacent vertices the vertex 1 is colored with color 1. Therefore we put 1 in the flag bit corresponding to color 1 (see Fig. 5). As the vertices 3 and 7 are not colored so far, their flag bits remain same. Now to color vertex 2, we traverse flag vector from left to right to encounter the 1st zero flag bit, (which means that the corresponding color is not used). It is seen from Fig. 5 that flag bit corresponding to color 2 is 0. Therefore, we can assign color 2 to the vertex 2 and we update color-assigned vector as shown in Fig. 5.

node	1	2	3	4	5	6	7	8	9
color	1	2	3	4	5	6	7	8	9
flag	1	0	0	0	0	0	0	0	0
color-assigned	1	2							

Fig. 5. Table configuration after 2nd iteration

Step 3: Move to next vertex 3 and reset all the flag bits to 0 as shown in Fig. 6.

node	1	2	3	4	5	6	7	8	9
colors	1	2	3	4	5	6	7	8	9
flag	0	0	0	0	0	0	0	0	0
color-assigned	1	2							

Fig. 6. Table configuration after resetting flag bits to 0

Vertex 2, 6, 4 are adjacent to the vertex 3. Out of these, the vertex 2 is colored with color 2, therefore we put 1 in the corresponding flag bit (see Fig. 7) and search for the 1st zero flag bit from the beginning in left to right fashion and find that the 1st flag bit is zero. Therefore, we can assign the color 1 to the vertex 3 as shown in Fig. 7.

node	1	2	3	4	5	6	7	8	9
color	1	2	3	4	5	6	7	8	9
flag	0	1	0	0	0	0	0	0	0
color-assigned	1	2	1						

Fig. 7. Table configuration after 3rd iteration

The above method is continued and at the end of the final iteration we obtain the final solution in Fig. 8.

node	1	2	3	4	5	6	7	8	9
color	1	2	3	4	5	6	7	8	9
flag	1	0	0	0	0	0	0	0	0
color-assigned	1	2	1	2	1	2	1	2	3

Fig. 8. Table configuration after final iteration

The algorithm is formally presented as follows.

Algorithm NGC(*G*)

```
1.  //n is the number of vertices.
2.  // ar[1: n,1: n] is an adjacency matrix represent the G.
3.  // a[1: n] is solution vector, initially a is assigned zero.
4.  //c[1:2, 1: n] is the color vector.
5.  // cn counts the number of colors used.
6.   {
7.     cn = 1; p = 0; a[1] =1;
8.     for i:= 2 to n   do
9.       {
10.        for  j:= 1 to n do
11.          {
12.           if ((ar[i][j] = 1) && (a[j] > 0) && (i ≠ j))
13.             {
14.               for k:= 1 to n do
15.                 {
16.                   if(c [1] [k] = a[j]) then
17.                         { c[2][k] = 1; break;}
18.                     }
19.                 }
20.             }
21.         for k = 1 to n
22.             {
23.               if((c[2][k] ≠ 1) && (p = 0)) then  //p is a temporary variable
24.                 {
25.                   a[i] = c[1][k];
26.                   p =1;
27.                   if(cn < c[1][k])
28.                    cn = c[1][k];
29.                 }
30.             else
31.             c[1][k] = 0;
32.             }
33.         p = 0;
34.       }
35.    Write (a[1:n]); Write ("Number of colors used:",  cn);
36. }
```

Explanation: Line 12 checks whether v_j is adjacent to v_i and also examines whether v_j is colored, $\forall$ i and j, $1 < i$, $j <= n$ and $i \neq j$. If it is so line 16 checks the color of v_j and set the flag bit of the corresponding color to 1 in the color vector. Line 23 examines the 1st zero bit flag bit and if it is found, line 25 assigns the color to v_i. This ensures that no two adjacent vertices will have the same color. Line 28 counts the total number of colors used and Line 31 resets the flag bit to zero.

Time Complexity: For each iteration of the 'for' loop from line 10 to 22 takes $O(n^2)$ time, 'for' loop comprising lines 14 to 17 is executed a maximum of n iterations. Lines 21 to 31 is implemented in $O(n)$, therefore lines 10 to 31 requires $O(n^2)$ time. As the 'for' loop in line 8 is executed in $n - 1$ iterations, therefore the overall complexity of the algorithm NGC is $O(n^3)$.

3. Experimental Results

We have tested our algorithm on several randomly generated graphs by varying the number of nodes as well as edges and calculated the run time and the minimum number of colors required by the proposed algorithm NGC. The results are compared by running the DSATUR algorithm [10] and shown in Fig. 9. It is observed that although minimum number of colors for both the algorithms are same except for the problem instances 3, 7 and 8, the running time of the proposed algorithm is significantly less than the DSATUR. The comparisons of the running times are also shown by the bar charts in Fig 10.

Problem Instance	No. of Nodes	No. of Edges	NGC(Ours)		DSATUR	
			Time(secs.)	No. of colors	Time(secs.)	No.of colors
01	07	12	1.18	4	2.38	4
02	20	190	1.07	20	1.61	20
03	60	1500	1.10	26	370.76	25
04	120	80	1.08	3	3.13	3
05	150	70	1.12	3	4.27	3
06	250	190	1.08	3	8.42	3
07	520	810	2.79	5	39.56	4
08	600	450	3.47	4	41.36	3

Fig. 9. Comparison results of the proposed algorithm and DSATUR [10]

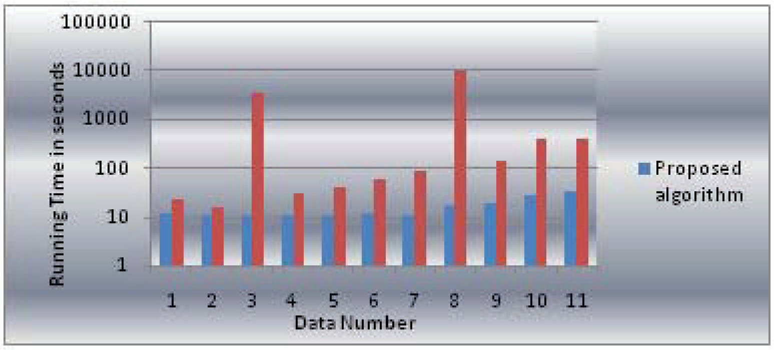

Fig. 10. Bar charts for comparing running time

4. Conclusion

In this paper, we have presented a novel method for graph coloring problem. The algorithm has been shown to run in $O(n^3)$ time. We have tested the proposed algorithm on various kinds of graph. The experimental results have been compared and shown in tabular form as well as bar charts. The proposed algorithm has been shown to produce comparable results in far better running time.

References

1. Garey MR, Johnson DS. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: W.H. Freeman and Company; 1979.
2. Bianco L, Caramia M, Dell'Olmo P. *Solving a preemptive scheduling problem using coloring technique*, in: J. Weglarz (Ed.), Project Scheduling Recent Models, Algorithms and Applications, Kluwer Academic Publishers, Dordrecht;1998.
3. Blazewicz J, et al.. *Scheduling Computer and Manufacturing Processes*. Berlin: Springer; 1996.
4. Werra D de, Eisenbeis C, Lelait S, Marmol. On a graph-theoretical model for cyclic register allocation. *Discrete Applied Mathematics* 1999;**93**:191–203.
5. Garey MR et al. An application of graph coloring to printed circuit testing. *IEEE Trans. on Circuits and Systems* 1976; **23**:591–599.
6. Glass C. Bag rationalisation for a food manufacturer. *Journal of the Operational Research Society 2002;***53:544**–551.
7. Smith DH, Hurley S, Thiel SU. Improving heuristics for the frequency assignment problem. *European Journal of Operational Research* 1998;**107 (1)**:76–86.
8. Burke EK, McCollum B, Meisels A, Petrovic S, Qu R. A graph-based hyper heuristic for timetabling problems. *European Journal of Operational Research* 2007; **176**:177–192.

9. Zufferey N et al. Graph colouring approaches for a satellite range scheduling problem. *J. of Scheduling 2008;***11(4)**:263–277.
10. Brelaz D. New methods to color the vertices of a graph. *Communications of the ACM* 1979;**22(4)**: 251–256.
11. Leighton FT. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards* 1979;**84(6):**489–506.
12. Hertz A, Werra D de. Using tabu search techniques for graph coloring. *Computing* 1987;**39:**345–351.
13. Dorne R, Hao JK. Tabu search for graph coloring, T-colorings and set Tcolorings, Meta-Heuristics: *Advances and Trends in Local Search Paradigms for Optimization.*1998.
14. Fleurent C, Ferland JA. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research* 1996;**63**:437–461.
15. [15] Galinier P, Hao JK. Hybrid evolutionary algorithms for graph coloring. *J. of Combinatorial Opt. 1999;***3(4)**:379–397.
16. Chams M et al. Some experiments with simulated annealing for coloring graphs. *European J. of Opr. Res.* 1987; **32**:260–266.
17. Johnson DS, Aragon CR, McGeoch LA, Schevon C . Optimization by simulated annealing: An experimental evaluation; Part II. Graph coloring and number partitioning, *Operations Research 1991;***39(3)**:378–406.
18. Avanthay C, Hertz A, Zufferey N. A variable neighborhood search for graph coloring. *European Journal of Operational Research* 2003;**151(2)**:379–388.
19. Blochliger I, Zufferey N. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers and Operations Research* 2008;**35(3)**:960–975.
20. Porumbel DC, Hao JK, Kuntz P. A search space ‘‘cartography” for guiding graph coloring heuristics*. Computers and Operations Research, in press*, doi:10.1016/j.cor.2009.06.024.2009.
21. Galinier P et al. A survey of local search methods for graph coloring. *Comp. and Operations Research* 2006;**33 (9)**:2547–2562.
22. Massimiliano Caramia and Paolo Dell'Olmo. Coloring graphs by iterated local search traversing feasible and infeasble solutions. *Discrete Applied Mathematics*, 2008;**156**:201-217.
23. Zhipeng Lü, Jin-Kao Hao. A memetic algorithm for graph coloring. *European J. of Operational Research,* 2010; **203**: 241-250.