Mathematics Interdisciplinary Research **9** (2) (2024) 215 – 236

Original Scientific Paper

Solving Graph Coloring Problem Using Graph Adjacency Matrix Algorithm

Hanifa Mosawi, Mostafa Tavakoli and Khatere Ghorbani-Moghadam*

Abstract

Graph coloring is the assignment of one color to each vertex of a graph so that two adjacent vertices are not of the same color. The graph coloring problem (GCP) is a matter of combinatorial optimization, and the goal of GCP is determining the chromatic number $\chi(G)$. Since GCP is an NP-hard problem, then in this paper, we propose a new approximated algorithm for finding the coloring number (it is an approximation of chromatic number) by using a graph adjacency matrix to colorize or separate a graph. To prove the correctness of the proposed algorithm, we implement it in MATLAB software, and for analysis in terms of solution and execution time, we compare our algorithm with some of the best existing algorithms that are already implemented in MATLAB software, and we present the results in tables of various graphs. Several available algorithms used the largest degree selection strategy, while our proposed algorithm uses the graph adjacency matrix to select the vertex that has the smallest degree for coloring. We provide some examples to compare the performance of our algorithm to other available methods. We make use of the Dolan-Moré performance profiles to assess the performance of the numerical algorithms, and demonstrate the efficiency of our proposed approach in comparison with some existing methods.

Keywords: Chromatic number, Coloring number, Graph coloring algorithms, Graph adjacency matrix, Degree of vertex.

2020 Mathematics Subject Classification: 05C15, 05C85, 05C90.

How to cite this article
H. Mosawi, M. Tavakoli and Kh. Ghorbani-Moghadam, Solving graph coloring problem using graph adjacency matrix algorithm, *Math. Interdisc. Res.* **9** (2) (2024) 215-236.

*Corresponding author (E-mail: m_tavakoli@um.ac.ir)
Academic Editor: Gholam Hossein Fath-Tabar
Received 25 July 2023, Accepted 13 December 2023
DOI: 10.22052/MIR.2023.253223.1428

1. Introduction

The graph coloring problem is a combinatorial optimization problem that has most studied in computer science and mathematics. Two types of vertex and edge coloring are defined for graph coloring. Both of them aim to color the entire graph without contradiction [1]. Therefore, adjacent vertices and edges should be colored with different colors. For a graph G and a set $C = \{1, 2, \ldots, k\}$, $f : V(G) \to C$ is a coloring for G with k colors if $f(u) \neq f(v)$ for each pair u and v of $V(G)$. The k-coloring problem is the problem of finding the minimum number of colors, k, needed for coloring of a graph, see [2]. The chromatic number $\chi(G)$ is equal to $\min\{k \mid f : V(G) \to \{1, \ldots, k\}$ is a coloring of $G\}$, see [3, 4] for more details.
Garey and Johnson [5] proved that the k-coloring problem belongs to NP-complete class, and determining the chromatic number $\chi(G)$ is an NP-hard problem; for this reason many approximation algorithms have been proposed. The graph coloring problem has many applications in various fields such as solving biological problems, communication and the internet and also, graph coloring algorithms are used for plenty of real-world problems [1] including map coloring [6], timetable and schedule issues [7, 8], registration allocation issues [9, 10], sudoku issue [5], and frequency allocation issues [11]. For solving GCP, the number of innovative and meta-heuristic algorithms were expanded to obtain better answers. Innovative algorithms were usually used for less vertex problems but for complete graphs meta-heuristic algorithms can find better answers [12]. Tabu search algorithm [13], Refrigeration simulation algorithm [14], Genetic algorithm [8], Ant colony algorithm [15], Cuckoo algorithm [12] are some of the meta-heuristic algorithms for graph coloring. When the vertices of a graph were colored with greedy algorithms, it makes the best choice at each execution step, and for this reason these algorithms were called greedy algorithms. Greedy algorithms usually provided sufficient and effective results [12]. As examples of these algorithms we have first fit algorithm (FF) [16], welsh and powell (WP) [2], incidence degree ordering (IDO) [16], The largest degree ordering algorithm (LDO) [16], recursive largest first algorithm (RLF) [17] and degree of saturation algorithm (DSATUR) [18]. All of these algorithms are greedy for vertex coloring and have been tested on standard graphs provided by DIMACS [19]. Lima et al. [20] gave polynomial-time algorithms for rainbow vertex coloring on permutation graphs, powers of trees and split strongly chordal graphs.

Here, we propose a new algorithm that uses graph adjacency matrix to select the vertex that has the smallest degree, for finding the chromatic number or approximation of chromatic number. The rest of the article is arranged as follows: In Section 2, we provide some necessary definitions related to graph coloring. Our proposed algorithm for finding the graph coloring number is described in Section 3. In Section 4, the numerical results are given and it is compared with some existing methods. In Section 5, we apply algorithm (GCA) in coloring the faces of $C80$ to $C240$ as an application of our algorithm. We provide the conclusion in Section 6.

2. Preliminaries

A pair of vertices in a graph are called adjacent if they are linked by an edge. A vertex u is a neighbor of a vertex v if there exists an edge $\{u, v\}$. The neighborhood of a vertex v, $N(v)$, is the set of all vertices that are adjacent to it. The problem of *graph coloring* is to assign a color to each vertex of the graph so that two adjacent vertices are not of the same color. A suitable k-coloration of vertices in a graph $G = (V, E)$ is a function of $C : V(G) \to \{1, 2, \ldots, k\}$ such that $C(x) \neq C(y)$ holds for each $(x, y) \in E$. The number that corresponds to the vertex x is called the color x and the vertices with the same color indicate a color class that each color class is an independent set. The *chromatic number* $\chi(G)$ is a suitable coloring with the smallest number of colors used for all the vertices of the graph G. Since finding the chromatic number is an NP-hard problem, so an approximation of it has been computed and is called *coloring number*. As the number of vertices increases, the complexity of the problem also increases because it becomes difficult to color the graph with the fewest possible colors, so we need special methods to color the graphs with the fewest different colors. The graph *adjacency matrix* is given in the following relation based on the conditions that there is an edge between the two vertices.

$$A(i, j) = \begin{cases} 1, & \text{if there is an edge between the two vertices } v_i \text{ and } v_j, \\ 0, & \text{otherwise.} \end{cases}$$

A is adjacency matrix to a graph G. The vertices of the graph G are displayed in a set of $V = \{1, 2, \ldots, n\}$.

3. Proposed method for finding the coloring number

Suppose $G = (V, E)$ is a simple and undirected graph with $|V| = n$. This proposed algorithm performs operations on the adjacency matrix of the graph G which we call A with the dimension $n \times n$ and the set of vertices $V = \{1, 2, \ldots, n\}$ in order to achieve a proper separation of the graph vertices. Unlike the usual algorithms compared to it, this algorithm uses the vertex with the smallest degree for coloring every time, and when the coloring is finished with one color, at the end of each iteration of the algorithm, if the matrix A is not zero, the set V is updated to start recoloring the graph with the next color for the uncolored vertices, and until the vertices of the graph are finished, this operation is repeated. To obtain x_k sets, we use the ordered set $x = \{1, \cdots, n\}$. The value of k is zero at the beginning, and it increases by one at each stage when the coloring of the graph is finished with one color. The set x_k is colored with the k-th color, and the largest k in x_k is the number of different colors, which is used to color the graph. First, with an example, we describe how to use the graph adjacency matrix to color the graph, and then we describe the algorithm (GCA).
Consider the graph $G = (5, 7)$ depicted in Figure 1 with $V = \{1, \cdots, 5\}$ and

$x = \{1, \cdots, 5\}$. The matrix A is the adjacency matrix of the graph $G = (5, 7)$.

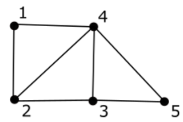


Figure 1: Graph $G = (5, 7)$.

Then it has 5 rows and 5 columns. The first row of A corresponds to the vertex (1) and the second row corresponds to the vertex (2) and so on. In the first row of A, if vertex (1) is adjacent to another vertex, $A(1, j) = 1$, otherwise $A(1, j) = 0$ and also $A(1, 1) = 0$ and the next rows are constructed in the same way and the symmetric matrix A is obtained.

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}, V = \{1, 2, 3, 4, 5\}.$$

Now, using the matrix A, we color the graph $G = (5, 7)$. We obtain the total rows of the matrix A: $sum(A^{'}) = \{2, 3, 3, 4, 2\}$ (the matrix $A^{'}$ is the transproduct of the matrix A and $sum(A^{'})$ is the column sum of the matrix $A^{'}$). $min(sum(A^{'})) = 2$ and the sum of the **first** row is 2, so, $i = 1$. $j = \{2, 4\}$ is the neighbors of i. We obtain t and $V(i)$. $t = i \cup j = \{1, 2, 4\}$ and $V(i) = 1$. Now, in the order of the numbers that are in t, from the largest to the smallest, we remove the row and column of the matrix A and from the ordered set V, and the new matrix A and new V are obtained.

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, V = \{3, 5\}.$$

Since $sum(sum(A^{'}))^{'} > 0$, we repeat the above operation again and $sum(A^{'}) = \{1, 1\}$ is obtained and $min(sum(A^{'})) = 1$ is in the **first** row, so $i = 1$. The neighbor of i is $j = \{2\}$. We obtain $t = i \cup j = \{1, 2\}$ and $V(i) = 3$. Now, we remove the rows and columns of the matrix A and from the ordered set V, in the order of the numbers in t, from the largest to the smallest. The new A and V will be empty.

$$A = [\,], V = \{\}.$$

Now, since $sum(sum(A^{'}))^{'} = 0$, coloring with one color is finished and $V = \emptyset$. But if $V \neq \emptyset$, it would be placed in the set of $r = V(i) \cup V = \{1, 3\}$. At first, $k = 0$ and we put $k = k + 1$. $x_k = x_1 = x(r) = \{x(1), x(3)\} = \{1, 3\}$ is obtained. We remove $x(r)$ from x and $x = \{2, 4, 5\}$ remains. We construct the new matrix

A with neighbors of (i)s that are not colored, by the method of constructing the induced subgraph matrix $G[V \setminus r]$. In this process, according to the numbers in the set r are removed from the largest to the smallest, the rows and columns of the matrix A and the matrix A is updated and V has the same number of members as the rows of the matrix A and is updated.

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, V = \{1, 2, 3\}.$$

Figure 2 shows the induction subgraph of $G[V \setminus r]$.

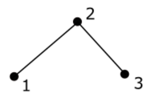


Figure 2: Graph $G = (3, 2)$.

Since $V \neq \emptyset$, then we repeat all the above operations. While $sum(sum(A^{'}))^{'} > 0$, we find the vertices of i and color them with the color $k = k + 1$. The ordered set $sum(A^{'}) = \{1, 2, 1\}$ and $min(sum(A^{'})) = 1$. It corresponds to the **first** row. So, $i = 1$ and adjacent to it, $j = 2$. Therefore, $t = \{1, 2\}$ and $V(i) = 1$. Now, we remove the rows and columns of the matrix A and from the ordered set V in the order of the numbers in t, from the largest to the smallest. A new matrix of A and V is obtained.

$$A = \begin{bmatrix} 0 \end{bmatrix}, V = \{3\}.$$

The matrix A has one member and so $sum(sum(A^{'}))^{'} = 0$. Therefore the operation is not performed, the remaining members of V are single vertices and are placed in the set $r = V(i) \cup V = \{1, 3\}$. Now, we put $k = k + 1$, $x_k = x_2 = x(r) = \{x(1), x(3)\} = \{2, 5\}$ and remove $x(r)$ from x and $x = \{4\}$ is obtained. We construct the new matrix A with the neighbors of (i)s that are not colored, by the method that the induction subgraph matrix $G[V \setminus r]$. In this way, according to the numbers in the set r, from the largest to the smallest, the rows and columns of the matrix A are removed and the matrix A is updated and V has the same number of members as the rows of the matrix A and is updated.

$$A = \begin{bmatrix} 0 \end{bmatrix}, V = \{1\}.$$

Figure 3 shows the induced subgraph $G[V \setminus r]$ with only one vertex.
Since $sum(sum(A^{'}))^{'} = 0$, the operation is not performed and we have only one vertex, and it is colored with the next color. In this case, we put $k = k + 1$ and put the remaining members of x in x_k, $x_k = x_3 = \{x\} = \{4\}$ and the algorithm ends and x_k sets are obtained as below.

$$x_1 = \{1, 3\}, x_2 = \{2, 5\}, x_3 = \{4\}.$$

Figure 3: Graph $G = (1, 0)$.

Now, we summarize our proposed method as Algorithm Algorithm 1.

Algorithm 1 Graph Coloring Algorithm (GCA)

Input A is the adjacency matrix of the graph G.
Step-1 Let $k = 0, n = A.rows, V = \{1, \cdots, n\}, x = \{1, \cdots, n\}$.
Step-2 **while** $V \neq \phi$ **do**:

- 2-1 **while** $sum(sum(A'))' > 0$ **do**:
  - 2-1-1 Find i which is the smallest row and is chosen randomly and the sum of it, is not zero.
  - 2-1-2 Let $w =$ the neighbors of (i).
  - 2-1-3 Let $t = \cup(i, w)$ and order t.
  - 2-1-4 Remove rows and columns containing t from the largest to the smallest from A and V.
- 2-2 Let $k = k + 1$, $r = (i)s \cup V$ and sort r. So, take the corresponding numbers of r from the largest to the smallest index of x and put them in x_k. ($x_k = x(r)$ and remove $x(r)$ from x.)
- 2-3 Let $c =$ the neighbors of (i)s.
- 2-4 **if** $|c| == 1$, **then** $k = k + 1$ and $x_k = x$ and the algorithm terminates.
- 2-5 **if** $|c| > 1$, **then** $B =$ adjacency matrix of induced subgraph $G[c]$. The matrix B is constructed in such a way that the algorithm removes the rows and columns of the matrix A according to the ordered set r, from the largest to the smallest, and the matrix B is created.
- 2-6 **if** $sum(sum(B'))' = 0$, **then** $k = k + 1$ and $x_k = x$ and the algorithm terminates, **otherwise** $A = B$ and $V = 1, \cdots, length(c)$, **then** go to **Step 2**.

Output: Return x_k(it is sets of separation and k is number of sets.)

The details of the steps associated with Algorithm 1 are described below.

(1) *Initialization*: First, the value of k is equal to zero, n is the number of rows of matrix A, the set of $V = \{1, \cdots, n\}$, which is the vertices of the graph and also the ordered set $x = \{1, \cdots, n\}$ are determined

(2) In this step, the following commands are performed until the set V is not empty:
Until $A > 0$ and to detect it, the algorithm obtains the transpose of the matrix A, denoted here by A' and $sum(A')$ is the vector obtained from the sum of each row of A and also $sum(sum(A'))'$ is a number that shows the sum of a row and if $sum(sum(A'))' > 0$ was, the algorithm executes the following commands.
First it finds the smallest non-zero row i, and then the neighbors of i are put in the set w and places the sets i and w in t and sorts t. Then it deletes rows and columns of A and V from large to small as shown in the set t. If the matrix A is not zero, the loop commands are executed. When the matrix A becomes zero, it means that there are not vertices or the graph is empty and some vertices may remain from V, which are the only vertices. To obtain the set colored by k, use the ordered set $x = \{1, \cdots, n\}$. It can be done in such a way that the set of is and only vertices is placed in r and sorted, and the members of this set are the index of vertices with color k are colored and the corresponding numbers of each index is removed from the x set from large to small as shown in the set r and placed in x_k. Then, it places the neighbors of is in c and checks the following conditions:

If the set c has only one member, the algorithm puts $k = k + 1$ and puts the set x in x_k, that is, the set x has one member. It is colored with the color k and the algorithm terminates. If the size of the set c is greater than one, it obtains the adjacency matrix of the induced subgraph $G[c]$. In this way, it removes the set r sorted from large to small from the matrix A and the matrix B is obtained.

Next, the algorithm checks the matrix B. To determine if the matrix B is zero or not, the algorithm obtains the transpose of the matrix B, denoted here by B' and $sum(B')$ is the vector obtained from the sum of each row of B and also $sum(sum(B'))'$ is a number that shows the sum of a row and if $sum(sum(B'))' = 0$ was in this case, the matrix B is zero, then it puts $k = k + 1$ and $x_k = x$, then the algorithm terminates because the matrix B becomes zero in the last iteration of the algorithm. Otherwise, $A = B$ and V has the same number of members as c, that is, $V = \{1, \cdots, length(c)\}$ and the algorithm goes to the beginning of step 2. The algorithm is repeated until the set V is not empty, and the set of x_k is produced as the output of the algorithm in each iteration.

Our algorithm is repeated as many times as the obtained coloring number minus one, because at the end of the last iteration, a empty graph is created and the color k is assigned to its vertices.

2.1 Illustrative example

In this section, by using an example, we want to explain how we can find the coloring number by using the algorithm GCA.

Let $G = (9, 13)$ be a simple graph with nine vertices and thirteen edges (see Figure 4). At the beginning, $k = 0, 9 = A.rows, V = 1, \cdots, 9, x = 1, \cdots, 9$. A is the adjacency matrix of the graph $G = (9, 13)$ and the number of rows of A is equal to 9. According to Algorithm 1, since $V \neq \emptyset$, the algorithm goes to 2-1.

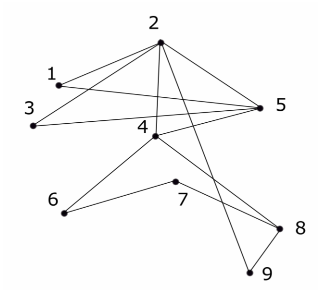


Figure 4: G=(9, 13).

While $|E(G)| = 0$, the loop commands are executed. The vertex $i = 1$ is specified as the smallest vertex and its neighbors. Then, i and its neighbors $w = \{5, 2\}$ are removed from the graph or adjacency matrix (see Figure 5). Since the graph is not

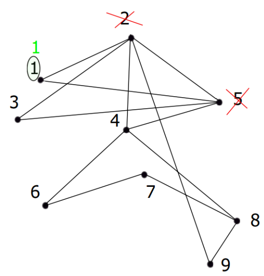


Figure 5: The first iteration of the algorithm, the first iteration of the loop and step 2-1-2.

empty, the loop commands are repeated again. Then the vertex $i = 9$ is selected as the smallest non-zero vertex (see Figure 6). Thus, i and its neighbor $w = \{8\}$ are removed from the graph or adjacency matrix, and the graph of Figure 7 is obtained. Since $A > 0$, it means that the graph is not empty or only vertices, the loop commands are executed again, and $i = 4$ is chosen as the smallest vertex,

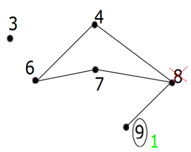


Figure 6: The first iteration of the algorithm, the first iteration of the loop and step 2-1-4, then the second iteration of the loop and step 2-1-2.

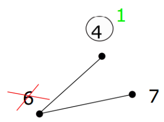


Figure 7: The first iteration of the algorithm, the second iteration of the loop and step 2-1-4, and then the third iteration of the loop and step 2-1-2.

and then i and $w = \{6\}$ are removed from the graph or adjacency matrix. The graph in Figure 8 is obtained, which is an empty graph. Therefore, in this case, $A = 0$, so the loop ends, and the algorithm goes to step 2-2. According to step

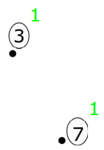


Figure 8: The first iteration of the algorithm, because $A = 0$, then step 2-2 is executed.

2-2: $k = k + 1$ and because $k = 0$ is assumed, then $x_k = x1$ and union (i) and the only vertices are put in r and this set is sorted and removed from the set x from large to small, as shown in r and puts it in x_1 and $x_1 = \{1, 3, 4, 7, 9\}$ is obtained and the set $x = \{2, 5, 6, 8\}$ remains.

According to step 2-3, put the set of neighbors of i in c so $c = \{2, 5, 6, 8\}$. By using step 2-4, since $|c| > 1$, the step 2-5 is implemented and the induced subgraph G by c, the graph of the form Figure 9, or the adjacency matrix B is created. To construct the matrix B of the ordered set r from large to small, we delete the rows and columns of the matrix B as shown in r and the matrix B is updated.

According to step 2-6: since $B > 0$, $A = B$ and $V = 1 : length(c)$ or $V = \{1, 2, 3, 4\}$, and again the algorithm is repeated from step-2. Since $V \neq \emptyset$, the algorithm goes to 2-1. As one can see in Figure 9, $i = 1$ is selected as the smallest nonzero vertex, and it is removed from the graph or adjacency matrix with its neighbor $w = \{2\}$. The graph in Figure 10 is obtained, which is an empty graph

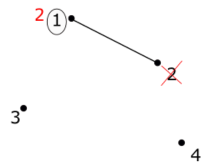


Figure 9: The first iteration of the algorithm and step 2-5 and the creation of the induced subgraph $G[c]$ and going to the second iteration of the algorithm and performing steps 2-1-1 and 2-1-2.

and $A = 0$, so the loop ends again, then step 2-2 is implemented. Therefore,

Figure 10: The second iteration of the algorithm and step 2-1-4.

$k = k+1$ and the union (i) and the only vertices are put in r and this set is sorted and removed from the set x from large to small, as shown in r and puts it in x_2 and $x_2 = \{2, 6, 8\}$ is obtained and the set $x = \{5\}$ remains.

The algorithm goes to 2-3, and $c = \{2\}$ is obtained, and then the algorithm goes to 2-4. Since $|c| = 1$, then $k = k + 1$ and the set x is put in x_3. Then, $x_3 = \{5\}$, see Figure 11 and the implementation of algorithm is terminated. This

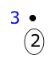


Figure 11: The second iteration of the algorithm and step 2-4.

algorithm divides the graph into x_1, x_2, x_3, or it is colored with $k = \{1, 2, 3\}$ colors.

Every vertex that is selected for a color is removed with its neighbors, so other vertices that get the same color cannot be adjacent to each other, so the algorithm works correctly.

According to these observations, we believe that the algorithm can obtain a good approximation of the optimal solution. We define f, which actually shows the number of neighbors of vertices i in each row of matrix A [21]. Their color must be different from the vertex i, which is defined as follows:

$$f = \sum_{(i,j)\in E} A(i,j).$$

Suppose that $C(i)$ is the vertex color of i. The following function shows the f probability complement to all colors:

$$1 - \frac{f(x_i)}{\displaystyle\sum_{j=1}^{N} f(x_j)},$$

where, $f(x_i) = \sum_{(i,j)} A(i,j)$ means the vertices adjacent to x_i that are not of the same color as x_i, and $\sum_j f(x_j)$ means the sum of all rows. The reason we take $f(x_i)$ the smallest is to increase the probability of the remaining colors for vertices other than x_i.

Suppose that, we have a sufficient number of colors. If the vertex with the smallest non-zero degree is selected, the probability of remaining colors increases for vertices other than x_i.

4. Numerical results

Here, we present some numerical results obtained by applying MATLAB 9.3, and all experiments were run on a PC with CPU Intel Core (TM) i7-7700K CPU at 4.20GHz, 32G bytes of SDRAM memory, and Windows 10 operating system. In [22], the algorithms FF, LDO, WP, IDO, DSATUR, and RLF were tested on benchmark graphs provided by DIMACS [12]. Algorithm 1, has been tested on the same benchmark graphs and is attached to the last two columns of tables [22]. In Tables 1 to 6, the first column, entitled Graph, shows the name of the benchmark graphs, the column entitled V, shows the number of vertices, the column entitled E, shows the number of edges, the column entitled Den, displays the density of edges obtained from the relation $Den = 2E/V(V-1)$, and $\chi(G)$ or best is a chromatic number or the best number ever known, the column entitled RLF, shows the name of algorithm and in this column the column R shows the number of different colors obtained by this algorithm, and T displays the implementation time (seconds). And the rest of the columns show the name of the algorithm and its results in the same way. Our proposed method is like this, that first we run the same algorithm in the mode of selecting the smallest vertex in terms of degree in a non-random mode, and in this mode, an approximation of the chromatic number or k is obtained, which we call the initial solution. Then, we test the GCA algorithm 15 times for each example. If the smallest solution obtained in these tests was smaller than the initial solution, this solution is selected; otherwise, the same initial solution as the example solution is selected. We display the best solution for each example in the R column and also the average execution times of the best solution in every 15 test times in the T column. The "GCA" algorithm can be repeated automatically until the desired value is generated. Coloring algorithms are presented and the performance of these algorithms on standard graphs (presented in [19]) is compared. In the last row of each table, the sum of the data of the R columns is given, by comparing the total data of each $Best/\chi(G)$ column, we can know the algorithm that got a better answer.

In Table 1, if we compare our results with other methods on Table 1, we observe that our proposed algorithm improved the implementation time in comparison with DSATUR and IDO on twelve instances (*mycies15, mycies16, mycies17, miles1000, miles1500, miles500, miles750, anna, david, huck, jean, games120*). In Table 1, concerning the coloring numbers obtained by FF, on the five instances (*miles1500, miles500, miles750, anna, david*), our proposed algorithm has improved the result, but in comparison with other algorithms the coloring numbers of our proposed method are remained identical.

Table 1: The results and computation times for Mycielski and SGB graphs.

Graph	V	E	Den.	Best/$\chi(G)$	RLF		DSATUR		WP		LDO		IDO		FF		GCA	
					R	***T***	***R***	***T***	***R***	***T***	***R***	***T***	***R***	***T***	***R***	***T***	***R***	***T***
mycie13	11	20	0.33	4	4	0.0004	4	0.0023	4	0.0001	4	0.0002	4	0.0009	4	0.0001	4	0.0022
mycie14	23	71	0.27	5	5	0.0009	5	0.0077	5	0.0002	5	0.0005	5	0.0028	5	0.0003	5	0.0039
mycie15	47	236	0.21	6	6	0.0024	6	0.0255	6	0.0003	6	0.0010	7	0.0085	6	0.0006	6	0.0069
mycie16	95	755	0.17	7	7	0.0075	7	0.0876	7	0.0004	7	0.0024	7	0.0309	7	0.0016	7	0.0134
mycie17	191	2360	0.13	8	8	0.0293	8	0.3254	8	0.0006	8	0.0068	8	0.1366	8	0.0054	8	0.0301
miles1000	128	3216	0.39	42	42	0.1065	42	1.2942	43	0.0016	43	0.0123	43	0.7013	44	0.0121	44	0.1136
miles1500	128	5198	0.63	73	73	0.3158	73	2.6877	73	0.0024	73	0.0219	73	1.6033	76	0.0220	73	0.1968
miles500	128	1170	0.14	20	20	0.0250	20	0.3249	20	0.0010	20	0.0055	20	0.1360	22	0.2249	21	0.0530
miles750	128	2113	0.26	31	31	0.0522	31	0.7112	32	0.0013	32	0.0083	31	0.3443	34	0.0081	32	0.0852
anna	138	493	0.05	11	11	0.0135	11	0.1231	11	0.0006	11	0.0037	11	0.0457	12	0.0026	11	0.0232
david	87	406	0.11	11	11	0.0076	11	0.1026	11	0.0005	11	0.0025	11	0.0346	12	0.0017	11	0.0203
huck	74	301	0.11	11	11	0.0062	11	0.0745	11	0.0005	11	0.0021	11	0.0244	11	0.0014	11	0.0179
jean	80	254	0.08	10	10	0.0060	10	0.0616	10	0.0004	10	0.0020	10	0.0198	10	0.0013	10	0.0162
games120	20	638	0.09	9	9	0.0180	9	0.1537	9	0.0006	9	0.0039	9	0.0577	9	0.0032	9	0.0325
Sum of R	-	-	-	248	248	-	248	-	250	-	250	-	249	-	260	-	252	-

In Table 2, when comparing our computation times on Queen graphs with other existing method, we observe that the implementation time of our proposed algorithm is better than DSATUR and IDO algorithms. Also, our obtained coloring numbers are best in comparision with WP, LDO, IDO and FF, and in comparision with DSATUR our results are better on eight graphs (*queen 5-5, queen 6-6, queen 8-8, queen 9-9, queen 10-10, queen 11-11, queen 12-12 and queen 14-14*), we obtain the same values for the rest of instances. Our obtained coloring number is better than RLF on one graph (*queen 7-7*), we obtain the same values for the rest of instances.

In Table 3, versus to the results presented in DSATUR, on nineteen tested instances (*1-Fullins-4, 1-Fullins-5, 1-Insertios-4, 1-Insertions-5, 1-Insertions-6, 2-Fullins-3, 2-Fullins-4, 2-Fullins-5, 2-Insertion-4, 2-Insertion-5, 3-Fullins-3, 3-*

Table 2: The results and computation times for Queen graphs.

Graph	V	E	Den.	Best/$\chi(G)$	RLF		DSATUR		WP		LDO		IDO		FF		GCA	
					R	*T*	*R*	*T*	*R*	*T*	*R*	*T*	*R*	*T*	*R*	*T*	*R*	*T*
queen 5-5	25	160	0.51	5	5	0.0012	5	0.0332	7	0.0003	7	0.0006	7	0.0109	8	0.0005	5	0.0052
queen 6-6	36	290	0.45	7	8	0.0028	9	0.0634	9	0.0003	9	0.0010	10	0.0218	11	0.0008	8	0.0095
queen 7-7	49	476	0.40	7	9	0.0045	11	0.1090	12	0.0005	12	0.0016	12	0.0461	10	0.0013	7	0.0153
queen 8-12	96	1368	0.30	12	13	0.0202	14	0.3851	15	0.0007	15	0.0045	15	0.1779	15	0.0041	14	0.0410
queen 8-8	64	728	0.36	9	11	0.0081	12	0.1783	13	0.0005	13	0.0024	15	0.0923	13	0.0020	11	0.0222
queen 9-9	81	1056	0.32	10	12	0.0154	13	0.2853	15	0.0007	15	0.0036	15	0.1312	16	0.0030	12	0.0291
queen 10-10	100	2940	0.59	11	13	0.0215	14	0.4351	17	0.0009	17	0.0052	17	0.1876	16	0.0044	13	0.0388
queen 11-11	121	3960	0.54	11	14	0.0345	15	0.6300	17	0.0009	17	0.0072	18	0.2964	17	0.0063	14	0.0521
queen 12-12	144	5192	0.50	13	15	0.0550	16	0.9163	19	0.0010	19	0.0100	20	0.4604	20	0.0092	15	0.0681
queen 13-13	169	6656	0.47	13	16	0.0800	17	1.3226	23	0.0013	23	0.0134	22	0.6869	21	0.0125	17	0.0863
queen 14-14	196	8372	0.44	16	17	0.1227	19	1.8404	25	0.0015	25	0.0170	24	1.0488	23	0.0169	18	0.1136
Sum of R	-	-	-	114	133	-	145	-	172	-	172	-	169	-	170	-	134	-

Fullins-4, 3-Insertion-3, 3-Insertion-4, 4-Fullins-3, 4-Fullins-4, 4-Insertions-3, 4-Insertions-4, 5-Fullins-3), and IDO algorithm on sixteen tested instances (*1-Fullins-4, 1-Fullins-5, 1-Insertios-4, 1-Insertions-5, 1-Insertions-6, 2-Fullins-3, 2-Fullins-4, 2-Fullins-5, 2-Insertion-4, 2-Insertion-5, 3-Fullins-3, 3-Fullins-4, 3-Insertion-3, 3-Insertion-4, 4-Fullins-3, 5-Fullins-3*), we improved the computation times.

Table 3: The results and computation times for CAR graphs.

Graph	V	E	Den.	Best/$\chi(G)$	RLF		DSATUR		WP		LDO		IDO		FF		GCA	
					R	*T*	*R*	*T*	*R*	*T*	*R*	*T*	*R*	*T*	*R*	*T*	*R*	*T*
1-Fullins-4	93	593	0.14	5	5	0.0069	5	0.0869	5	0.0003	5	0.0021	6	0.0298	11	0.0016	5	0.0113
1-Fullins-5	282	3247	0.08	6	6	0.0612	6	0.5026	6	0.0007	6	0.0104	7	0.2167	14	0.0098	6	0.0507
1-Insertios-4	67	232	0.10	5	5	0.0058	5	0,0258	5	0.0002	5	0.0013	5	0.0090	5	0.0008	5	0.0086
1-Insertions-5	202	1227	0.06	6	6	0.0314	6	0.1527	6	0.0005	6	0.0055	6	0.0569	6	0.0038	6	0.0307
1-Insertions-6	607	6337	0.03	7	7	0.3575	7	1.2288	7	0.0023	7	0.0344	7	0.6011	7	0.0308	7	0.5414
2-Fullins-3	52	201	0.15	5	5	0.0027	5	0.0237	5	0.0002	5	0.0010	5	0.0076	10	0.0008	5	0.0072
2-Fullins-4	212	1621	0.07	6	6	0.0329	6	0.2116	6	0.0006	6	0.0060	6	0.0796	14	0.0052	6	0.0295
2-Fullins-5	852	12201	0.03	7	7	0.8307	7	3.2494	7	0.0044	7	0.0703	7	1.8256	18	0.0763	7	1.4973
2-Insertion-4	149	541	0.05	5	5	0.0188	5	0.0621	5	0.0004	5	0.0036	5	0.0230	5	0.0021	5	0.0191
2-Insertion-5	597	3936	0.02	6	6	0.3721	6	0.6322	6	0.0023	6	0.0276	6	0.2985	6	0.0211	6	0.5105
3-Fullins-3	80	346	0.11	6	6	0.0060	6	0.380	6	0.0003	6	0.0017	6	0.0134	12	0.0012	6	0.0106
3-Fullins-4	405	3524	0.04	7	7	0.1476	7	0.5354	7	0.0012	7	0.0157	8	0.2370	17	0.0150	7	0.1365
3-Fullins-5	2030	33751	0.02	8	8	10.3646	8	18.3988	8	0.0254	8	0.3786	9	11.5821	22	0.4600	8	22.2276
3-Insertion-3	56	110	0.07	4	4	0.0033	4	0.0125	4	0.0002	4	0.0010	4	0.0047	4	0.0006	4	0.0063
3Insertion-4	281	1046	0.03	5	5	0.0731	5	0.1288	5	0.0007	5	0.0073	5	0.0498	5	0.0048	5	0.0497
3-Insertion-5	1406	9695	0.01	6	6	3.6966	6	2.3609	6	0.0128	6	0.1101	7	1.2766	6	0.0996	6	7.1841
4-Fullins-3	114	541	0.08	7	7	0.0107	7	0.0610	7	0.0004	7	0.0026	7	0.0216	14	0.0020	7	0.0147
4-Fullins-4	690	6650	0.03	8	8	0.5299	8	1.2976	8	0.0031	8	0.0386	8	0.6677	20	0.0387	8	0.7614
4-Fullins-5	4146	77305	0,01	9	9	89.5661	9	85.9533	9	0.1131	9	1.6550	9	55.6602	26	2.0289	9	198.1971
4-Insertions-3	79	156	0.05	4	4	0.0065	4	0.0180	4	0.0002	4	0.0015	4	0.0068	4	0.0009	4	0.0086
4-Insertions-4	475	1795	0.02	5	5	0.2527	5	0.2467	5	0.0015	5	0.0188	5	0.1009	5	0.0103	5	0.2407
5-Fullins-3	154	792	0.07	8	8	0.0220	8	0.0922	8	0.0005	8	0.0036	8	0.0332	16	0.0029	8	0.0201
5-Fullins-4	1085	11395	0.02	9	9	1.8848	9	2.9627	9	0.0080	9	0.0874	9	1.6530	23	0.0923	9	3.0829
Sum of R	-	-	-	144	144	-	144	-	144	-	144	-	149	-	270	-	144	-

In Table 4, compared to the DSTUR and IDO methods tested on six graphs (*DSJC125-1, DSJC125-5, DSJC125-9, DSJC250-1, DSJC250-5, DSJR500-1*) we provide best computation time. Also, we observe that the coloring number of our proposed algorithm on the six tested instances (*DSJC125-1, DSJC125-5, DSJC125-9, DSJC250-1, DSJC250-5, DSJR500-1*) are better than FF, and on the five tested instances are better than IDO algorithm and on one tested graph

(*DSJC125-5, DSJC125-9, DSJC250-1, DSJC250-5*) remained identical. Also, our results is better than LDO and WP on the three tested graphs (*DSJC125-5, DSJC125-9, DSJC250-5*), and our result in comparison with DSATUR is better on two graphs (*DSJC125-1, DSJC125-5*).

Table 4: The results and computation times for Random and Flat graphs.

Graph	V	E	Den.	Best/$\chi(G)$	RLF		DSATUR		WP		LDO		IDO		FF		GCA	
					R	***T***	***R***	***T***	***R***	***T***	***R***	***T***	***R***	***T***	***R***	***T***	***R***	***T***
DSJC125-1	125	736	0.09	5	6	0.0135	6	0.0846	7	0.0005	7	0.0032	7	0.0320	8	0.0024	7	0.0263
DSJC125-5	125	3891	0.50	17	21	0.0468	22	0.6111	23	0.0011	23	0.0079	25	0.2966	26	0.0074	21	0.0783
DSJC125-9	125	6961	0.89	44	49	0.1811	51	1.4215	53	0.0019	53	0.0162	54	0.7575	56	0.0154	51	0.1701
DSJC250-1	250	3218	0.10	8	10	0.0665	10	0.4791	11	0.0011	11	0.0142	12	0.2086	13	0.0097	11	0.0852
DSJC250-5	250	15668	0.50	28	35	0.4661	37	4.9399	41	0.0025	41	0.0371	40	3.0145	43	0.0394	35	0.2451
DSJR500-1	500	3555	0.03	12	12	0.2863	13	0.5829	13	0.0023	13	0.0237	13	0.2609	15	0.0199	14	0.4393
Sum of R	-	-	-	114	133	-	139	-	148	-	148	-	151	-	161	-	139	-

In Table 5, if we compare our computation times of DSATUR, IDO and RLF on fourteen tested instances (*fpso12-i1, fpso12-i2, fpso12-i3, mulsol-i1, mulsol-i2, mulsol-i3, mulsol-i4, mulsol-i5, inithx-i1, inithx-i2, inithx-i3, zeroin-i1, zeroin-i2, zeroin-i3*), we have improved fourteen results for graphs and we obtain the same values of coloring numbers for fourteen instances.

Table 5: The results and computation times for Register Allocation graphs.

Graph	V	E	Den.	Best/$\chi(G)$	RLF		DSATUR		WP		LDO		IDO		FF		GCA	
					R	***T***	***R***	***T***	***R***	***T***	***R***	***T***	***R***	***T***	***R***	***T***	***R***	***T***
fpso12-i1	496	11654	0.09	65	65	0.9869	65	3.1791	65	0.0044	65	0.0646	65	1.8096	65	0.0552	65	0.3890
fpso12-i2	451	8691	0.09	30	30	0.5217	30	1.9960	30	0.0024	30	0.0442	30	1.1139	30	0.0409	30	0.2479
fpso12-i3	425	8688	0.10	30	30	0.5184	30	1.9752	30	0.0022	30	0.0427	30	1.0739	30	0.0407	30	0.2096
mulsol-i1	197	3925	0.20	49	49	0.1299	49	0.6347	49	0.0021	49	0.0153	49	0.2924	49	0.0137	49	0.1182
mulsol-i2	188	3885	0,22	31	31	0.1171	31	0.6423	31	0.0015	31	0.0145	31	0.2899	31	0.0133	31	0.0773
mulsol-i3	184	3916	0.23	31	31	0.1164	31	0.6189	31	0.0015	31	0.0143	31	0.2805	31	0.0134	31	0.0772
mulsol-i4	185	3946	0.23	31	31	0.1243	31	0.6328	31	0.0015	31	0.0145	31	0.2994	31	0.0130	31	0.0777
mulsol-i5	186	3973	0.23	31	31	0.1253	31	0.6286	31	0.0015	31	0.0145	31	0.2900	31	0.0128	31	0.0761
inithx-i1	864	18707	0.05	54	54	2.7427	54	6.7614	54	0.0066	54	0.1337	54	4.2802	54	0.1266	54	1.5789
inithx-i2	645	13979	0.07	31	31	1.4014	31	4.2319	31	0.0037	31	0.0839	31	2.5214	31	0.0800	31	0.6985
inithx-i3	621	13969	0.07	31	31	1.3034	31	4.1724	31	0.0035	31	0.0819	31	2.5577	31	0.0780	31	0.6160
zeroin-i1	211	4100	0.18	49	49	0.1427	49	0.6636	49	0.0020	49	0.0157	49	0.3188	49	0.0139	49	0.1197
zeroin-i2	211	3541	0.16	30	30	0.1062	30	0.5390	30	0.0015	30	0.0136	30	0.2504	30	0.0124	30	0.0753
zeroin-i3	206	3540	0.17	30	30	0.1150	30	0.5439	30	0.0014	30	0.0134	30	0.2530	30	0.0123	30	0.0681
Sum of R	-	-	-	523	523	-	523	-	523	-	523	-	523	-	523	-	523	-

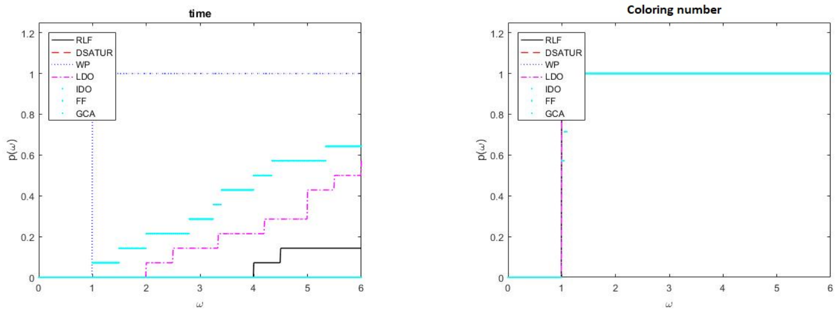
In Table 6, our algorithm provides better computation time than DSATUR and IDO on six graphs (namely *le450-15b, le450-25a, le450-25b, le450-25c, le450-5c, le450-5d*). If we compare the obtained coloring number of our proposed method with FF, we observe that on four tested instances (*le450-15b, le450-5c, le450-25c, le450-5d*) and our obtained coloring number on two instances (*le450-5c, le450-5d*) is better than coloring number of IDO, LDO, WP and DSATUR.

We used the performance profiles given by the Dolan-Moré diagrams (see details in [23]). The performance profile provides, for each value of w, the proportion $\rho(w)$ of

Table 6: The results and computation times for Leighton graphs.

Graph	V	E	Den.	Best/$\chi(G)$	RLF		DSATUR		WP		LDO		IDO		FF		GCA	
					R	***T***	***R***	***T***	***R***	***T***	***R***	***T***	***R***	***T***	***R***	***T***	***R***	***T***
le450-15b	450	8169	0.08	15	17	0.3071	16	1.7589	18	0.0025	18	0.0348	18	0.9585	22	0.0337	21	0.3964
le450-25a	450	8260	0.08	25	25	0.3502	25	1.7952	26	0.0029	26	0.0367	25	1.0172	28	0.0355	28	0.4299
le450-25b	450	8263	0.08	25	25	0.3583	25	1.9924	25	0.0028	25	0.0371	25	1.0341	27	0.0355	28	0.3998
le450-25c	450	17343	0.17	25	28	0.7839	29	5.9978	29	0.0034	29	0.0626	31	3.6658	37	0.0674	35	0.5392
le450-5c	450	9803	0.10	5	5	0.2226	10	2.4336	12	0.0020	12	0.0352	12	1.3233	17	0.0375	6	0.2260
le450-5d	450	9757	0.10	5	6	0.2315	12	2.4037	14	0.0025	14	0.0362	13	1.2504	18	0.0382	5	0.2180
Sum of R	-	-	-	100	106	-	117	-	124	-	124	-	124	-	149	-	123	-

test problems where each considered algorithmic variant has a performance within a factor of w of the best. Thus, based on the Dolan-Moré performance profile as shown in Figures 12 to 17, we conclude that the our proposed method (GCA) performs are better than others.



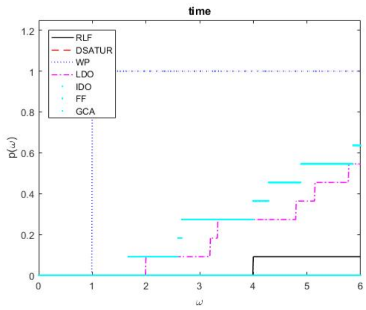
(a) The Dolan-Moré performance profile for comparison of computation times for Mycielski and SGB graphs.

(b) The Dolan-Moré performance profile for comparison of coloring number for Mycielski and SGB graphs.

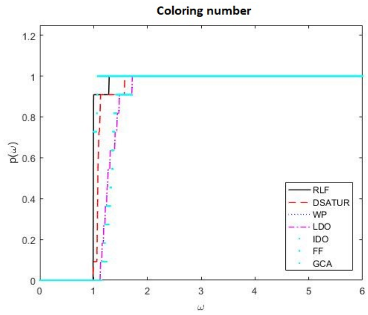
Figure 12: The Dolan-Moré performance profile for Mycielski and SGB graphs.

5. Using (GCA) in coloring the faces of $C80$ to $C240$

Using Algorithm (GCA) in coloring the faces of $C80$ to $C240$ Graph coloring algorithm can be used to color the face $C80$ to $C240$ in the article [24]. First, we make the graph of each of the shapes $C80$ to $C240$ and then we color each of these graphs with the algorithm (GCA). The method of making the graph is as follows:
1- We name pentagonal and hexagonal regions with numbers, and each number represents a vertex of the graph, thus the vertices of the graph are obtained.
2- We add an edge from each vertex to all the vertices, which have different colors.
3- We color the resulting graph with the algorithm (GCA).
Because each of the obtained graphs from $C80$ to $C240$ is large, we construct

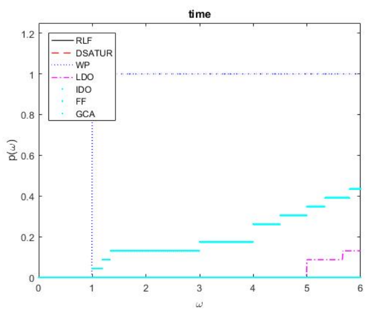


(a) The Dolan-Moré performance profile for comparison of computation times for Queen graphs.

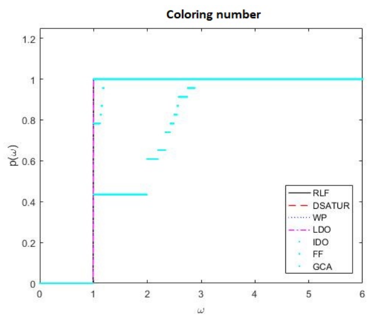


(b) The Dolan-Moré performance profile for comparison of coloring number for Queen graphs.

Figure 13: The Dolan-Moré performance profile for Queen graphs.



(a) The Dolan-Moré performance profile for comparison of computation times for CAR graphs.



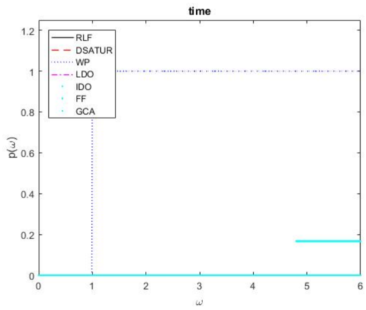
(b) The Dolan-Moré performance profile for comparison of coloring number for CAR graphs.

Figure 14: The Dolan-Moré performance profile for CAR graphs.

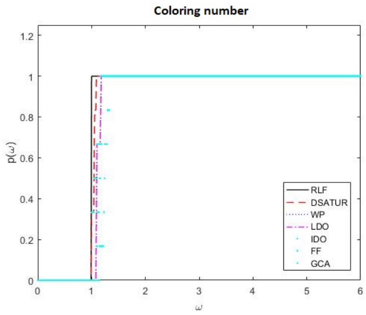
the adjacency matrix of the graph obtained from $C80$ and $C240$ with MATLAB software. The (GCA) algorithm receives each of them as input and produces sets that have the same color. The adjacency matrix of the graph $G = (42, 360)$ resulting from $C80$ is constructed as follows. First, we name pentagonal and hexagonal regions with numbers, each of these numbers represents a vertex. So, $V = \{1, 2, \ldots, 42\}$.

$$A(i,j) = \begin{cases} 1, & \text{If there is an edge between the two vertices } v_i \text{ and } v_j, \\ 0, & \text{otherwise.} \end{cases}$$

Figure 18 shows $C80$, which is numbered.

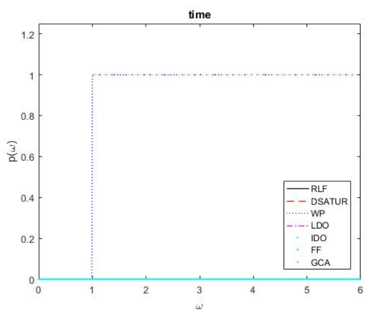


(a) The Dolan-Moré performance profile for comparison of computation times for Random and Flat graphs.

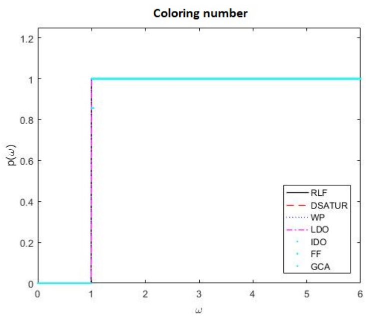


(b) The Dolan-Moré performance profile for comparison of coloring number for Random and Flat graphs.

Figure 15: The Dolan-Moré performance profile for Random and Flat graphs.



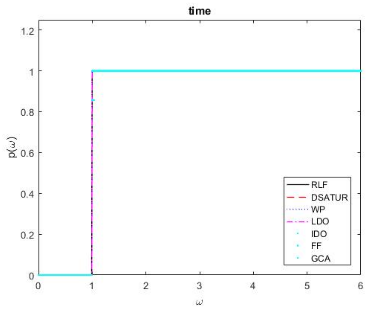
(a) The Dolan-Moré performance profile for comparison of computation times for Register Allocation graphs.



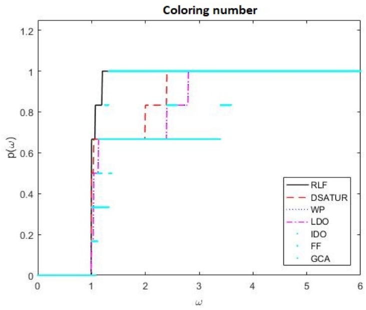
(b) The Dolan-Moré performance profile for comparison of coloring number for Register Allocation graphs.

Figure 16: The Dolan-Moré performance profile for Register Allocation graphs.

After running the algorithm on the adjacency matrix $G = (42, 360)$, the following sets are obtained and show that X_1 is colored with the color $K = 1$ and X_2 is colored with the color $K = 2$.
$x_1 = [2, 3, 4, 5, 6, 8, 10, 12, 14, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 28, 30, 32, 34, 36, 37, 38, 39, 40, 41]$,
$x_2 = [1, 7, 9, 11, 13, 15, 27, 29, 31, 33, 35, 42]$.
Figure 19 shows $C240$, which is numbered.
Also, after running the algorithm on the adjacency matrix $G = (122, 4795)$, the following sets are obtained and it shows that X_1 with color $K = 1$, X_2 with color $K = 2$, X_3 is colored with the color $K = 3$ and X_4 is colored with the color $K = 4$.
$x_1 = [2, 3, 4, 5, 6, 17, 18, 20, 21, 23, 24, 26, 27, 29, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50,$

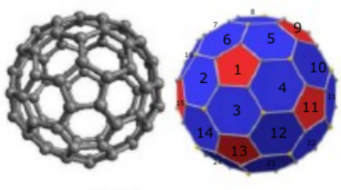


(a) The Dolan-Moré performance profile for comparison of computation times for Leighton graphs.



(b) The Dolan-Moré performance profile for comparison of coloring number for Leighton graphs.

Figure 17: The Dolan-Moré performance profile for Leighton graphs.



.

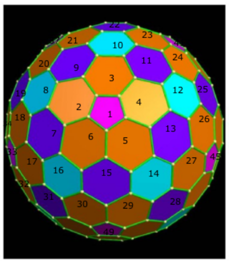
Figure 18: $C80$.

$53, 57, 61, 65, 69, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 93, 95, 96, 98, 99, 101, 102, 104, 105, 117, 118, 119, 120, 121]$,
$x_2 = [7, 9, 11, 13, 15, 19, 22, 25, 28, 31, 35, 39, 43, 47, 51, 52, 54, 55, 56, 58, 59, 60, 62, 63, 64, 66, 67, 68, 70, 71, 75, 79, 83, 87, 91, 94, 97, 100, 103]$,
$x_3 = [1, 33, 37, 41, 45, 49, 73, 77, 81, 85, 89, 122,]$,
$x_4 = [8, 10, 12, 14, 16, 108, 110, 112, 114, 116]$.

6. Conclusion

We presented a new algorithm for finding the chromatic number or approximation of chromatic number by using the adjacency matrix of a graph to colorize or separate a graph and specify separation sets. We provided some challenging benchmark graphs to compare the performance of our proposed algorithm to other available methods from the DIMACS library.

The best algorithm in terms of efficiency is the RLF algorithm, and the DSATUR algorithm ranks second in terms of the solution and it is slower than RLF in terms of execution time. Our proposed algorithm has worked very well for the graphs



.

Figure 19: $C240$.

of the queen category, and for the queen7-7 graph, the best answer or $\chi(G)$ is obtained while the RLF algorithm could not obtain it and also in the case of the graph le450-5d as well. In general, our proposed method is better than DSATUR algorithm in terms of execution time, which is after RLF in terms of solution, except for a few, it has performed better. Also, for Register Allocation graphs, the Algorithm 1 is similar to RLF in terms of its answer, but it is better than RLF in terms of implementation time has done. In the last row of Tables 1 to 6, we have given the total results for each algorithm. Based on these results, we can conclude that our proposed algorithm has worked very well and is similar to RLF. We made use of the Dolan–Moré performance profiles to assess the performance of the numerical algorithms and demonstrated the efficiency of our proposed approach in comparison with some existing methods.

Acknowledgement

Research of the second author, Mostafa Tavakoli, supported in part by the Ferdowsi University of Mashhad. Research of the third author, Khatere Ghorbani-Moghadam, supported in part by the Mosaheb Institute of Mathematics, Kharazmi University.

Conflicts of Interest. The authors declare that they have no conflicts of interest regarding the publication of this article.

References

[1] J. L. Gross, J. Yellen and M. Anderson, *Graph Theory and its Applications*, Chapman and Hall/CRC, (2018).

[2] F. Ge, Z. Wei, Y. Tian and Z. Huang, Chaotic ant swarm for graph coloring, *IEEE international conference on intelligent comput-*

ing and intelligent systems, Xiamen, China, **2010** (2010) 512 – 516, https://doi.org/10.1109/ICICISYS.2010.5658530.

[3] D. J. A. Welsh and M. B. Powell, An upper bound for the chromatic number of a graph and its application to timetabling problems, *Comput J.* **10** (1967) 85 – 86, https://doi.org/10.1093/comjnl/10.1.85.

[4] I. M. Díaz and P. Zabala, A generalization of the graph coloring problem, *Investig. Oper.* **8** (1999) 167 – 184.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman (1979).

[6] B. H. Gwee, M. H. Limand and J. S. Ho, Solving four colouring map problem using genetic algorithm, *Proceedings of First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems, Dunedin, New Zealand*, (1993) 332 – 333.

[7] K. A. Dowsland and J. M. Thompson, Ant colony optimization for the examination scheduling problem, *J. Oper. Res. Soc.* **56** (2005) 426 – 438, https://doi.org/10.1057/palgrave.jors.2601830.

[8] N. Chmait and K. Challita, Using simulated annealing and ant-colony optimization algorithms to solve the scheduling problem, *Comput. Sci. Inf. Tech.* **1** (2013) 208 – 224, https://doi.org/10.13189/csit.2013.010307.

[9] G. J. Chaitin, M. A. Auslander, A. K. Chandra, J. Cocke, M. E. Hopkins and P. W. Markstein, Register allocation via coloring, *Comput. Lang.* **6** (1981) 47 – 57, https://doi.org/10.1016/0096-0551(81)90048-5.

[10] F. C. Chow and J. L. Hennessy, The priority based coloring approach to register allocation, *ACM Trans. Program. Lang. Syst.* **12** (1990) 501 – 536, https://doi.org/10.1145/88616.88621.

[11] W. K. Hale, Frequency assignment: theory and applications, *in Proceedings of the IEEE* **68** (1980) 1497 – 1514. https://doi.org/10.1109/PROC.1980.11899.

[12] S. Mahmoudi and S. Lotfi, Modified cuckoo optimization algorithm (MCOA) to solve graph coloring problem, *Appl. Soft Comput.* **33** (2015) 48 – 64, https://doi.org/10.1016/j.asoc.2015.04.020.

[13] A. Hertz and D. de Werra, Using tabu search techniques for graph coloring, *Computing* **39** (1987) 345 – 351, https://doi.org/10.1007/BF02239976.

[14] M. Chams, A. Hertz and D. de Werra, Some experiments with simulated annealing for coloring graphs, *European J. Oper. Res.* **32** (1987) 260 – 266, https://doi.org/10.1016/S0377-2217(87)80148-0.

[15] S. Ahn, S. Lee and T. Chung, Modified ant colony system for coloring graphs, information, communications and signal processing, 2003 and fourth pacific rim conference on multimedia, Proceedings of the joint conference of the fourth international conference on, 1849 – 1853, (2003).

[16] H. Al-Omari and K. E. Sabri, New graph coloring algorithms, *American journal of mathematics and statistics* **2** (2006) 439 – 441.

[17] F. T. Leighton, A graph coloring algorithm for large scheduling problems, *J. Res. Nat. Bur. Standards*, **84** (1979) 489 – 506.

[18] D. Brélaz, New methods to color the vertices of a graph, *Comm. ACM* **22** (1979) 251 – 256, https://doi.org/10.1145/359094.359101.

[19] C. Dimacs, graph coloring instances, *(2016) instances homepage on CMU. [online]. Available: http:mat.gsia.cmu.edu/COLOR/instances.html.*

[20] P. T. Lima, E. J. van Leeuwen and M. van der Wegen, Algorithms for the rainbow vertex coloring problem on graph classes, *Theoret. Comput. Sci.* **887** (2021) 122 – 142, https://doi.org/10.1016/j.tcs.2021.07.009.

[21] S. M. Douiri and S. Elbernoussi, Solving the graph coloring problem via hybrid genetic algorithms, *J. King Saud Univ. Eng. Sci.* **27** (2015) 114 – 118, https://doi.org/10.1016/j.jksues.2013.04.001.

[22] M. Aslan and N. A. Baykan, A performance comparison of graph coloring algorithms, *Int. J. Intell. Syst. Appl. Eng.* **4** (2016) 1 – 7.

[23] E. D. Dolan and J. J. Moré, Benchmarking optimization software with performance profiles, *Math. Program.* **91** (2002) 201 – 213, https://doi.org/10.1007/s101070100263.

[24] K. Balasubramanian, O. Ori, F. Cataldo, A. R. Ashrafi and M. V. Putz, Face colorings and chiral face colorings of icosahedral giant fullerenes: C80 to C240, *Fuller. Nanotub. Carbon Nanostructures* **29** (2021) 1 – 12, https://doi.org/10.1080/1536383X.2020.1794853.

Hanifa Mosawi
Department of Applied Mathematics,
Faculty of Mathematical Sciences,
Ferdowsi University of Mashhad,
P.O. Box 1159, Mashhad 91775, I. R. Iran
e-mail: hmousv@gmail.com

Mostafa Tavakoli
Department of Applied Mathematics,
Faculty of Mathematical Sciences,
Ferdowsi University of Mashhad,
P.O. Box 1159, Mashhad 91775, I. R. Iran
e-mail: m_tavakoli@um.ac.ir

Khatere Ghorbani-Moghadam
Mosaheb Institute of Mathematics,
Kharazmi University, Tehran, I. R. Iran
k.ghorbani@khu.ac.ir