An analysis between different algorithms for the graph vertex coloring problem

Velin Kralev, Radoslava Kraleva
Department of Informatics, South-West University "Neofit Rilski", Blagoevgrad, Bulgaria

Article Info

Article history:

Received May 14, 2022
Revised Sep 5, 2022
Accepted Oct 1, 2022

Keywords:

Chromatic number
Computational complexity
Graph coloring
Graph theory
Vertex coloring

ABSTRACT

This research focuses on an analysis of different algorithms for the graph vertex coloring problem. Some approaches to solving the problem are discussed. Moreover, some studies for the problem and several methods for its solution are analyzed as well. An exact algorithm (using the backtracking method) is presented. The complexity analysis of the algorithm is discussed. Determining the average execution time of the exact algorithm is consistent with the multitasking mode of the operating system. This algorithm generates optimal solutions for all studied graphs. In addition, two heuristic algorithms for solving the graph vertex coloring problem are used as well. The results show that the exact algorithm can be used to solve the graph vertex coloring problem for small graphs with 30-35 vertices. For half of the graphs, all three algorithms have found the optimal solutions. The suboptimal solutions generated by the approximate algorithms are identical in terms of the number of colors needed to color the corresponding graphs. The results show that the linear increase in the number of vertices and edges of the analyzed graphs causes a linear increase in the number of colors needed to color these graphs.

Corresponding Author:

Velin Kralev
Department of Informatics, Faculty of Mathematics and Natural Science, South-West University
66 Ivan Michailov str., 2700 Blagoevgrad, Bulgaria
Email: velin_kralev@swu.bg

1. INTRODUCTION

Graph theory has been studied extensively in recent decades [1]. Graph structures are used to represent, study and analyze processes in many different real objects and therefore they are very useful [2]–[4]. Many complex, significant and important problems can be presented and studied with graphs. Most often, these types of problems are analyzed and solved by software that executes specific algorithms [5]. This is one of the reasons why many researchers are researching and improving different algorithms for solving certain classes of problems, related and presented directly and indirectly through graph structures [6]–[8]. This also includes the development of various software products (applications). This process is usually done through integrated development environments and event-oriented programming. These environments allow the use of different programming languages and different compilers for different target platforms [9].

Structurally, each graph is represented by two sets-one for vertices V and one for edges E. The set of vertices V cannot be empty and must have at least one element (vertex). In contrast, the set of edges E may be empty and not contain even a single element. This is usually not the case, because the edges actually represent connections between pairs of vertices, thus realizing the basic idea of the graph structure. In a given graph the edges can be unoriented. In this case, it does not matter which of the two incident vertices is the starting one and which is the final one. When the edge is oriented, one vertex is called the initial vertex and the other vertex is called the final vertex. In this case, the edge is called an arc [10]. If a numerical value is set for each edge, then the graph is called a weighted [11], [12]. Once these definitions have been presented, the graph vertex coloring problem can also be presented as well.

The graph vertex coloring problem is an NP-complete problem [13]. This problem is still being actively studied [14], [15]. Scientific publications have described many variants of this problem. For instance, the reconfiguration graph for vertex colorings of weakly chordal graphs [16], the facial unique-maximum colorings of plane graphs with restriction on big vertices [17], the vertex coloring with communication constraints in synchronous broadcast networks [18], and other. Different variants use different approaches [19], [20], techniques [21], [22] and algorithms [23], [24]. Similar approaches have been used to solve other problems in graph theory [25]. Detailed reviews of the specifics of the graph vertex coloring problem are discussed in [26], [27].

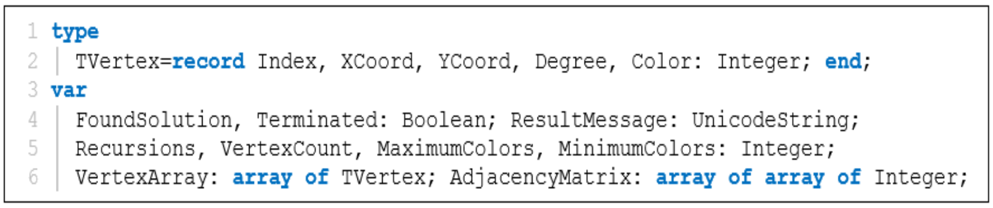
The most important feature of a graph vertex coloring algorithm is its computational complexity. In fact, it has to do with determining the chromatic number of a graph. A graph can be colored with only one color when it is composed of only vertices, and the set of edges is an empty set, i.e. the graph is empty. The graphs that can be colored exactly with two colors are the so-called bipartite graphs. The characteristic of these graphs is that the algorithms that can "recognize" these graphs as 2-colorable (and respectively the algorithms with which these graphs can be colored) are executed for polynomial time. In all other cases, when $k \geq 3$ the graph coloring problem is NP-complete [13]. Moreover, even determining to approximate the chromatic number is an NP-hard problem [1], [28].

All graphs that are not complete and do not have an odd-length cycle have a chromatic number that is less than or at most equal to the greatest degree of a vertex in that graph, i.e. $\chi(G) \leq \Delta(G)$, which is proved in [29]. In addition, if all the vertex degrees in a graph are greater than 2, then the chromatic number of the graph will be equal to the largest degree of the vertex plus one only when there is a full clique in that graph of exactly $\Delta(G) + 1$ [29]. Other results related to the graph vertex coloring and the determination of the bounds for the chromatic number are published in [30], [31].

In this study, three different algorithms for the graph vertex coloring problem will be studied-one exact and two approximate [32]. The exact algorithm is based on the backtracking method and always finds the optimal solutions for the analyzed graphs. In contrast, the other two algorithms-greedy coloring (GCA) and Welsh-Powell (WPA) are approximate and it is not always guaranteed that the solutions they find will be optimal. There are other algorithms that are discussed in the scientific literature [33], [34].

2. RESEARCH METHOD

This section introduces an implementation of the exact algorithm that can be used to solve optimally the graph vertex coloring problem. This algorithm is based on the backtracking method, and it always finds the exact solution. The other two used algorithms (GCA and WPA) are greedy and can be used to solve the graph vertex coloring problem approximately. For the implementation of the exact algorithm, it is necessary to declare (and initialize) some variables and dynamic structures. These declarations are presented in Figure 1 (in Delphi language).



```
1 type
2   TVertex=record Index, XCoord, YCoord, Degree, Color: Integer; end;
3 var
4   FoundSolution, Terminated: Boolean; ResultMessage: UnicodeString;
5   Recursions, VertexCount, MaximumColors, MinimumColors: Integer;
6   VertexArray: array of TVertex; AdjacencyMatrix: array of array of Integer;
```

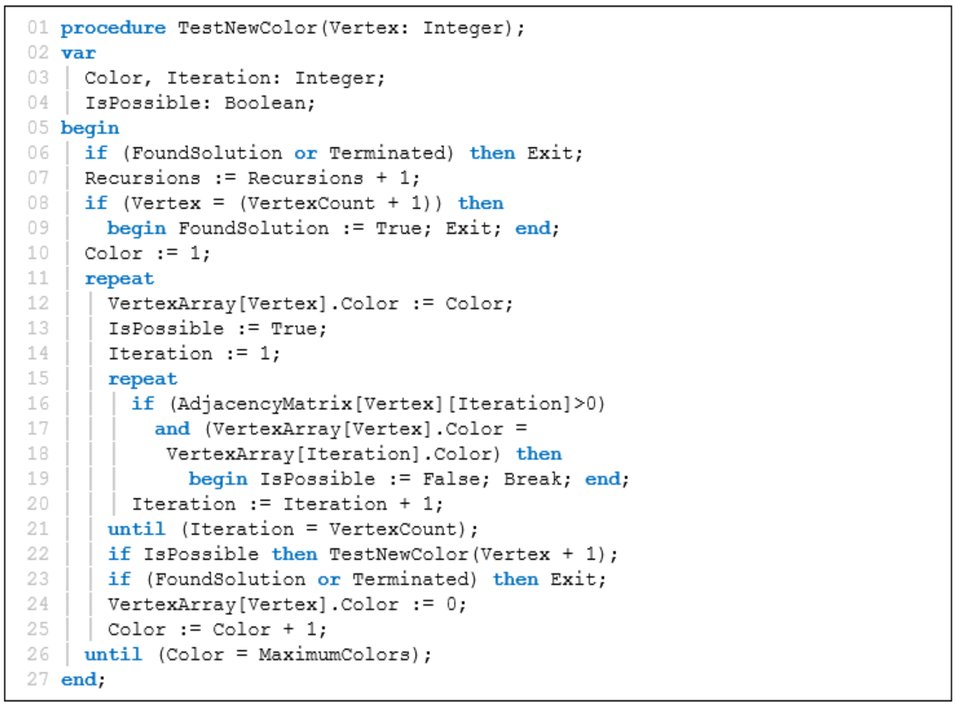
Figure 1. Source code of the global declarations

The *TVertex* structure (line 2) is a record through which a vertex of a certain graph can be represented. The Index field contains the index of the corresponding vertex. The *XCoord* and *YCoord* fields contain the screen coordinates at the vertex. The Degree field stores the degree of the corresponding vertex. The value of this field indicates the number of vertices with which the vertex is adjacent. The Color field is used by the coloring algorithm, and it contains information about the color with which the corresponding vertex is colored. The value in this field changes dynamically in the process of finding a solution.

The global variable *VertexCount*, which is declared on line 5, stores the number of vertices in the graph. The graph is represented by a list of vertices (the dynamic array *VertexArray* declared on line 6) and the two-dimensional dynamic array (matrix) *AdjacencyMatrix*, which is also declared on line 6. The *AdjacencyMatrix* structure is actually a dynamic array that stores other one-dimensional dynamic arrays (of the same size). Each one-dimensional array is associated with a specific vertex of the graph and has as many elements as the vertices in the graph. The variables maximum colors and minimum colors (declared on line 5) are used in the solution search process. The *ResultMessage* variable (declared on line 4) is used to display the values of the different variables after finding a solution.

The global variable *FoundSolution* (declared on line 4) has a true value when a solution is found, i.e., when all vertices in the graph are colored and there is a false value otherwise. The global variable terminated (also declared on line 4) is used when it is necessary to interrupt the process of finding a solution (usually by the user). The variable recursions (declared on line 5) stores the number of recursive calls that the exact algorithm has made in the solution search process.

The source code of the recursive *TestNewColor* procedure is shown in Figure 2. This procedure checks whether the vertex with the Vertex index (submitted as a procedure parameter) can be colored with some of the colors already used. At the beginning of the procedure, a check is made whether one of the two conditions for the end of the recursive process is fulfilled. The first condition is whether a solution has been found (i.e., whether the *FoundSolution* variable has a true value). The second condition is whether the process of finding a solution is interrupted by the user (i.e., whether the variable terminated has a value of true-line 6). If neither condition is met, the value of the recursions variable increases (line 7). In this way, the next recursive call of the *TestNewColor* procedure is registered.



```
01 procedure TestNewColor(Vertex: Integer);
02 var
03   Color, Iteration: Integer;
04   IsPossible: Boolean;
05 begin
06   if (FoundSolution or Terminated) then Exit;
07   Recursions := Recursions + 1;
08   if (Vertex = (VertexCount + 1)) then
09     begin FoundSolution := True; Exit; end;
10   Color := 1;
11   repeat
12     VertexArray[Vertex].Color := Color;
13     IsPossible := True;
14     Iteration := 1;
15     repeat
16       if (AdjacencyMatrix[Vertex][Iteration]>0)
17         and (VertexArray[Vertex].Color =
18          VertexArray[Iteration].Color) then
19            begin IsPossible := False; Break; end;
20       Iteration := Iteration + 1;
21     until (Iteration = VertexCount);
22     if IsPossible then TestNewColor(Vertex + 1);
23     if (FoundSolution or Terminated) then Exit;
24     VertexArray[Vertex].Color := 0;
25     Color := Color + 1;
26   until (Color = MaximumColors);
27 end;
```

Figure 2. Source code of the *TestNewColor* procedure

The verification of whether a solution has been found is performed on line 8. If the index of the current vertex has become one greater than the number of vertices in the graph, it means that all vertices in the graph are colored and a solution is found. Therefore, the *FoundSolution* variable is set to true, and then the recursive procedure is terminated by calling the exit method (line 9). If no solution is found, the algorithm checks whether it can color the current vertex with some of the other available colors (lines 10-26). This color must be chosen so that none of the adjacent vertices of the current one is colored with it (lines 15-25). If the algorithm colors the current vertex with one of the available colors, the *TestNewColor* procedure is called recursively, and the index of the next vertex after the current one is passed as a parameter (line 22). The process of searching for possible coloring continues until all available colors (stored in the *MaximumColors* variable) are tested. The algorithm optimizes the process of finding a solution, interrupting the construction of any partial but unacceptable solution.

The source code of the *ColorGraphExact* procedure is shown in Figure 3. This procedure executes the exact algorithm. At the beginning of this method, the variables *FoundSolution*, terminated, recursions, and the local variable Iteration are initialized. The local variable Iteration controls the construction repeat-until (lines 9-19). Through this construction, the idea of the algorithm to check whether the given graph can be colored with 1, 2, 3, ..., and *VertexCount* colors is realized. Initially (on line 10) the *MaximumColor* variable is initialized with the next value of the Iteration variable. In the color field of the structure *TVertex*, the value 0 is set. (line 13). The recursive *TestNewColor* procedure is then called. The vertex with index 1 is passed to it as a parameter, i.e., the coloring of the vertices starts from the first vertex. If the recursive procedure finds the color of the graph with the possible number of colors (stored in the variable *MaximumColors*), then the global variable *FoundSolution* will be set to True and the execution of the algorithm will end. If the recursive procedure fails to find a solution, i.e., fails to color the vertices of the graph with the available colors, then the global variable *FoundSolution* will be set to false. This means that coloring with only the available colors is not possible and a new color will have to be added to the existing ones (line 18). The value of the variable *MaximumColors* is set at the beginning of the construction repeat-until (line 10). The execution of the algorithm can also be interrupted by the user (during the solution search process). If this occurs, the Terminated variable will be set to true. This will immediately interrupt the execution of the repeat-until loop (line 17). Therefore, the repeat-until loop can only be completed in two ways. First, a solution is found (the variable *FoundSolution* is true). Second, the solution search process is interrupted by the user (the variable Terminated is True). Depending on these values the *ResultMessage* variable will be set (lines 20-24).

```
01 procedure ColorGraphExact;
02 var
03   Iteration, Vertex: Integer;
04 begin
05   FoundSolution := False;
06   Terminated := False;
07   Recursions := 0;
08   Iteration := 1;
09   repeat
10     MaximumColors := Iteration;
11     Vertex := 1;
12     repeat
13       VertexArray[Vertex].Color := 0;
14     until (Vertex = VertexCount);
15     TestNewColor(1);
16     if FoundSolution then Break;
17     if Terminated then Break;
18     Iteration := Iteration + 1;
19   until (Iteration = VertexCount);
20   if FoundSolution then
21     ResultMessage :=
22       IntToStr(MaximumColors) + ',' + IntToStr(Recursions);
23   if Terminated then
24     ResultMessage := 'Terminated!';
25 end;
```

Figure 3. Source code of the *ColorGraphExact* procedure

The two approximate algorithms are known and often discussed and analyzed in the scientific literature [32]. The GCA algorithm iterates sequentially through the vertices of the graph, passing through each vertex once. For each vertex, a certain color is chosen from the available ones, but it must have the lowest index. For each vertex, this color does not change once it has been set. The other heuristic algorithm-WPA colors non-adjacent vertices with a pre-selected color. Only when the current color can no longer be used does the algorithm select the next color. With this new color, the algorithm colors only those of the other uncolored vertices that are not adjacent. These vertices form a new chromatic class. This process is performed until the number of colored vertices is equal to the number of vertices in the graph. The computational complexity of the two approximate algorithms is quadratic and depends on the number of vertices in the graph. The exact algorithm generates all possible colorings of the graph and therefore its computational complexity is exponential.

3. RESULTS AND DISCUSSION

The results from two experiments will be presented. First, for graphs with how many vertices and edges, the exact algorithm can be used so that the time to find the optimal solution is acceptable (for example, on the order of minutes). Second, a comparative analysis between the algorithms, comparing the solutions found and the execution time of the algorithms (for the same input data).

3.1. Methodology of the experiments

In this study, 27 graphs, respectively with 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 100, 200, 300, 400, 500, 600, 700, 800, and 900 vertices were used. They were divided into two different groups, the first containing 18 graphs and the second the remaining 9 graphs. In this distribution, the first group included the graphs with 9÷43 vertices, and the second group, the graphs with 100÷900 vertices. These graphs are shown in Tables 1 and 2. The first group of graphs is used to perform experiments with all three algorithms, and the second group of graphs is used to perform experiments only with approximate algorithms. All graphs are generated randomly, and for each of them, the minimum, maximum, and average degrees of the vertices are presented.

Table 1. Graphs from the first group

Graph	Vertex	Edge	Vertices degree			Graph	Vertex	Edge	Vertices degree		
abbreviation	count	count	Min	Max	Avg	abbreviation	count	count	Min	Max	Avg
G_9_14	9	14	1	6	3	G_27_140	27	140	6	15	10
G_11_22	11	22	2	6	4	G_29_162	29	162	8	17	11
G_13_31	13	31	2	8	5	G_31_186	31	186	7	17	12
G_15_42	15	42	4	8	6	G_33_211	33	211	7	21	13
G_17_54	17	54	3	9	6	G_35_238	35	238	9	20	14
G_19_68	19	68	3	11	7	G_37_266	37	266	9	20	14
G_21_84	21	84	4	12	8	G_39_296	39	296	9	22	15
G_23_101	23	101	5	15	9	G_41_328	41	328	11	23	16
G_25_120	25	120	5	13	10	G_43_361	43	361	10	25	17

Table 2. Graphs from the second group

Graph	Vertex	Edge	Vertices degree			Graph	Vertex	Edge	Vertices degree		
abbreviation	count	count	Min	Max	Avg	abbreviation	count	count	Min	Max	Avg
G_100_990	100	990	9	28	20	G_600_35940	600	35 940	93	162	120
G_200_3980	200	3 980	28	54	40	G_700_48930	700	48 930	111	169	140
G_300_8970	300	8 970	41	79	60	G_800_63920	800	63 920	117	194	160
G_400_15960	400	15 960	56	107	80	G_900_80910	900	80 910	139	218	180
G_500_24950	500	24 950	76	127	100						

3.2. Experimental results

The experimental conditions are 64-bit OS Windows 11 and hardware configuration: Processor: Intel (R) Core (TM) i7-7700HQ at 2.80-3.80 GHz; RAM: 8GB DDR4. In Table 3, the “Recursions” column shows the number of recursive calls that the exact algorithm has made to find the optimal solutions for all graphs. These solutions show the minimum number of colors needed to distribute the vertices of the analyzed graphs into chromatic classes. This number is displayed in the “Colors” column under the “Exact algorithm” column. To execute these recursive calls, the exact algorithm has been running for some time. This time is displayed in the “Time (ms)” column in milliseconds and in the “Time (h, min, s)” column, but formatted in the format: “hours, minutes, seconds”.

Table 3 and the chart of Figure 4 show that with a linear increase of the vertices and the edges in a graph, the execution time for the exact algorithm increases exponentially. In contrast to this trend, with a linear increase in the number of vertices and edges in a graph, the number of colors required to distribute the vertices of the graph in different chromatic classes also increases linearly. The chart in Figure 5 shows the distribution of the solutions generated by the three algorithms in terms of the quality of the solutions. In half of the analyzed graphs, the approximate algorithms have found the optimal solutions. In addition, the same optimal solutions have been found by both approximate algorithms. Accordingly, the suboptimal solutions found by the two approximate algorithms also coincide.

The results obtained from the two approximate algorithms are shown in the “Colors” and “Solution” columns under the “GC algorithm” and “WP algorithm” columns. The values in the “Colors” columns have the same meaning as the values in the “Colors” column associated with the exact algorithm, i.e., the number of the chromatic classes but indicated by colors. The “Solution” columns (for both approximate algorithms) indicate whether the solutions found by these algorithms match the optimal solutions or not.

Table 3. Results of the three algorithms for the first group of graphs

Graph abbreviation	Exact algorithm				GC algorithm		WP algorithm	
	Recursions	Time (ms)	Time (h, min, s)	Colors	Colors	Solution	Colors	Solution
G_9_14	111	0	< 0.1 s	4	4	Optimal	4	Optimal
G_11_22	377	0	< 0.1 s	4	4	Optimal	4	Optimal
G_13_31	1 283	0	< 0.1 s	4	4	Optimal	4	Optimal
G_15_42	6 881	1	< 0.1 s	4	4	Optimal	4	Optimal
G_17_54	13 720	2	< 0.1 s	4	4	Optimal	4	Optimal
G_19_68	54 636	8	< 0.1 s	5	5	Optimal	5	Optimal
G_21_84	197 409	29	< 0.1 s	5	6	Suboptimal	6	Suboptimal
G_23_101	671 191	102	0.1 s	5	6	Suboptimal	6	Suboptimal
G_25_120	2 550 525	416	0.42 s	6	6	Optimal	6	Optimal
G_27_140	9 691 994	1 583	1.58 s	6	6	Optimal	6	Optimal
G_29_162	35 860 377	5 931	5.93 s	6	7	Suboptimal	7	Suboptimal
G_31_186	129 097 359	22 271	22.27 s	6	7	Suboptimal	7	Suboptimal
G_33_211	503 479 699	89 142	1 min, 29 s	6	7	Suboptimal	7	Suboptimal
G_35_238	1 812 526 917	337 465	5 min, 37 s	7	7	Optimal	7	Optimal
G_37_266	6 706 349 594	1 306 008	21 min, 46 s	7	8	Suboptimal	8	Suboptimal
G_39_296	22 801 588 618	4 577 713	1 h, 16 min	7	8	Suboptimal	8	Suboptimal
G_41_328	79 805 560 163	16 794 065	4 h, 39 min	7	8	Suboptimal	8	Suboptimal
G_43_361	265 752 515 342	58 113 386	16 h, 8 min	8	9	Suboptimal	9	Suboptimal

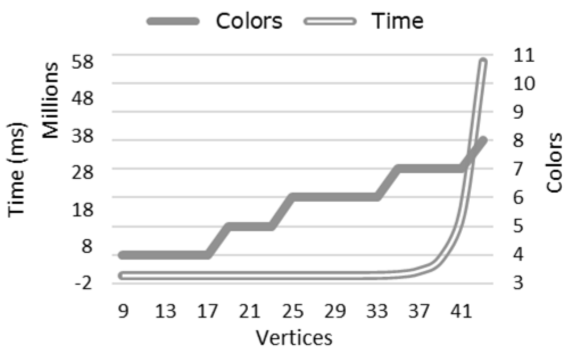


Figure 4. Influence of increasing the number of vertices and edges (x-axis) in each graph on the execution time of the exact algorithm (left y-axis, in milliseconds) and the number of colors (right y-axis) for optimal solutions

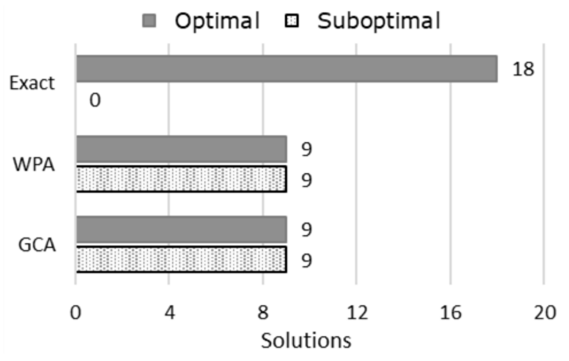


Figure 5. Comparison of the results of the algorithms according to the quality of the solutions

Since the execution time of the exact algorithm is unacceptably long for graphs with more than 35 vertices, only the approximate algorithms will be used to conduct the second experiment. The results of the approximate algorithms with the second group of graphs are shown in Table 4. In Table 4, the “Colors” columns (under the “GC algorithm” and “WP algorithm” columns) show the number of colors needed to distribute the vertices of the analyzed graphs (from the second group) into chromatic classes. The execution times of both algorithms are shown in the “Time (ms)” columns. The results of the approximate algorithms are identical and are summarized in Figure 6 and Figure 7.

Table 4. Results of the two approximate algorithms for the second group of graphs

Graph abbreviation	Vertex count	Edgecount	GC algorithm		WP algorithm	
			Colors	Time (ms)	Colors	Time (ms)
G_100_990	100	990	10	15	10	19
G_200_3 980	200	3 980	16	34	16	42
G_300_8 970	300	8 970	21	54	21	66
G_400_15 960	400	15 960	25	64	25	72
G_500_24 950	500	24 950	29	70	29	115
G_600_35 940	600	35 940	35	90	35	132
G_700_48 930	700	48 930	39	112	39	147
G_800_63 920	800	63 920	43	136	43	192
G_900_80 910	900	80 910	47	171	47	207

Table 4 and the chart of Figure 6 show that with a linear increase of the vertices and the edges in a graph, the number of colors required to distribute the vertices of the graph in different chromatic classes increases in a polynomial way. In contrast to the exact algorithm, with a linear increase of the vertices and the edges in a graph, the execution time for the approximate algorithms increases linearly Figure 7. The execution times of both approximate algorithms (for all analyzed graphs) are commensurate and very short. This is due to the fact that the computational complexity of both algorithms is quadratic [32].

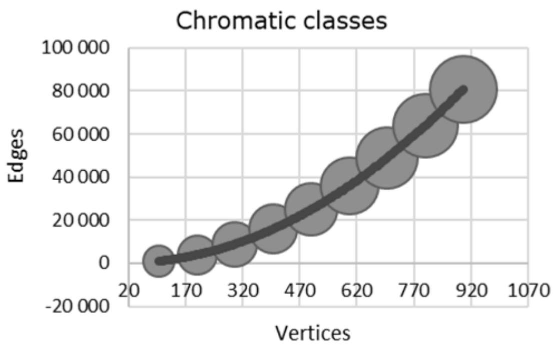


Figure 6. Influence of increasing the number of vertices (x-axis) and the number of edges (y-axis) on the number of required colors (chromatic classes) generated by the approximate algorithms

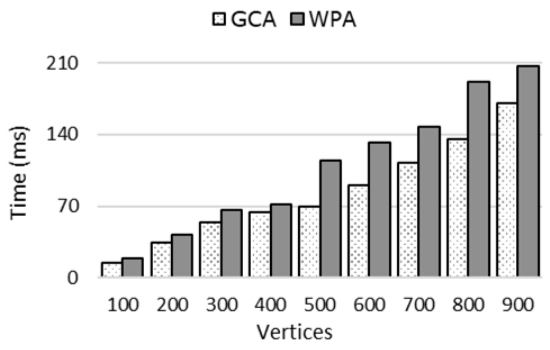


Figure 7. Comparison between the number of vertices and the execution times of both algorithms

4. CONCLUSION

In this paper, a study of the graph vertex coloring problem has been presented. Different approaches and algorithms to its solution have been discussed. The implementation of an exact algorithm has been presented as well. The declarations of different data structures-arrays and matrices have been shown. The source code of the methods for the exact algorithm has been discussed in detail. When analyzing the time for execution of the exact algorithm, the multitasking mode of the operating system has been taken into account. The methodology and conditions for the experiments have been presented. For conducting the experiments, twenty-seven graphs were generated randomly.

The results show that with a linear increase in the size of the graph, the execution time for the exact algorithm increases exponentially. The execution times of both heuristic algorithms are very short. In addition, with a linear increase in the size of the graph, the number of colors required to color the vertices of the graph increases linearly. For half of the tested graphs, the heuristic algorithms found the optimal solutions. Furthermore, both algorithms found the same optimal solutions. Also, both algorithms found the same suboptimal solutions. Both approximate algorithms generated identical results. The question remains whether this is true for every single graph or not. Additional (extended) experiments need to be conducted to establish this fact or to find an example that contradicts this statement. This study is beyond the scope of the present one and will therefore be presented in another scientific paper.

REFERENCES

[1] V. E. Alekseev, R. Boliac, D. V. Korobitsyn, and V. V. Lozin, "NP-hard graph problems and boundary classes of graphs," *Theoretical Computer Science*, vol. 389, no. 1–2, pp. 219–236, Dec. 2007, doi: 10.1016/j.tcs.2007.09.013.

[2] S. V. Kurapov, M. V. Davidovsky, and A. V. Tolok, "A modified algorithm for planarity testing and constructing the topological drawing of a graph. The thread method," *Scientific Visualization*, vol. 10, no. 4, pp. 53–74, Oct. 2018, doi: 10.26583/sv.10.4.05.

[3] B. Natarajan, "Computation of chromatic numbers for new class of graphs and its applications," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 8, pp. 396–400, 2019.

[4] V. Kralev and R. Kraleva, "Methods for software visualization of large graph data structures," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 10, no. 1, Feb. 2020, doi: 10.18517/ijaseit.10.1.10739.

[5] V. S. Kralev and R. S. Kraleva, "Visual analysis of actions performed with big graphs," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 1, pp. 2740–2744, Nov. 2019, doi: 10.35940/ijitee.A4978.119119.

[6] S. Slamin, N. O. Adiwijaya, M. A. Hasan, D. Dafik, and K. Wijaya, "Local super antimagic total labeling for vertex coloring of graphs," *Symmetry*, vol. 12, no. 11, Nov. 2020, doi: 10.3390/sym12111843.

[7] J. Xu, X. Qiang, K. Zhang, C. Zhang, and J. Yang, "A DNA computing model for the graph vertex coloring problem based on a probe graph," *Engineering*, vol. 4, no. 1, pp. 61–77, Feb. 2018, doi: 10.1016/j.eng.2018.02.011.

[8] A. Parihar, N. Shukla, M. Jerry, S. Datta, and A. Raychowdhury, "Vertex coloring of graphs via phase dynamics of coupled oscillatory networks," *Scientific Reports*, vol. 7, no. 1, Dec. 2017, doi: 10.1038/s41598-017-00825-1.

[9] T. Kosar, S. Gaberc, J. C. Carver, and M. Mernik, "Program comprehension of domain-specific and general-purpose languages: replication of a family of experiments using integrated development environments," *Empirical Software Engineering*, vol. 23, no. 5, pp. 2734–2763, Oct. 2018, doi: 10.1007/s10664-017-9593-2.

[10] D. S. S, D. R. Arunadevi, D. B. Kanisha, and D. R. Kesavan, "Mining of sequential patterns using directed graphs," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 11, pp. 4002–4007, Sep. 2019, doi: 10.35940/ijitee.K2242.0981119.

[11] K. D. Rangaswamy and M. Gurusamy, "Application of graph theory concepts in computer networks and its suitability for the resource provisioning issues in cloud computing-a review," *Journal of Computer Science*, vol. 14, no. 2, pp. 163–172, Feb. 2018, doi: 10.3844/jcssp.2018.163.172.

[12] V. Kralev, "An analysis of a recursive and an iterative algorithm for generating permutations modified for travelling salesman problem," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 7, no. 5, Oct. 2017, doi: 10.18517/ijaseit.7.5.3173.

[13] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified NP-complete graph problems," *Theoretical Computer Science*, vol. 1, no. 3, pp. 237–267, Feb. 1976, doi: 10.1016/0304-3975(76)90059-1.

[14] S. Nicoloso and U. Pietropaoli, "Vertex-colouring of 3-chromatic circulant graphs," *Discrete Applied Mathematics*, vol. 229, pp. 121–138, Oct. 2017, doi: 10.1016/j.dam.2017.05.013.

[15] R. Chitnis, L. Egri, and D. Marx, "List H-coloring a graph by removing few vertices," *Algorithmica*, vol. 78, no. 1, pp. 110–146, May 2017, doi: 10.1007/s00453-016-0139-6.

[16] C. Feghali and J. Fiala, "Reconfiguration graph for vertex colourings of weakly chordal graphs," *Discrete Mathematics*, vol. 343, no. 3, Mar. 2020, doi: 10.1016/j.disc.2019.111733.

[17] B. Lidický, K. Messerschmidt, and R. Škrekovski, "Facial unique-maximum colorings of plane graphs with restriction on big vertices," *Discrete Mathematics*, vol. 342, no. 9, pp. 2612–2617, Sep. 2019, doi: 10.1016/j.disc.2019.05.029.

[18] H. Lakhlef, M. Raynal, and F. Taiani, "Vertex coloring with communication constraints in synchronous broadcast networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 7, pp. 1672–1686, Jul. 2019, doi: 10.1109/TPDS.2018.2889688.

[19] M. Zaker, "A new vertex coloring heuristic and corresponding chromatic number," *Algorithmica*, vol. 82, no. 9, pp. 2395–2414, Sep. 2020, doi: 10.1007/s00453-020-00689-4.

[20] A. Dey, L. Son, P. Kumar, G. Selvachandran, and S. Quek, "New concepts on vertex and edge coloring of simple vague graphs," *Symmetry*, vol. 10, no. 9, Sep. 2018, doi: 10.3390/sym10090373.

[21] T. Karthick, F. Maffray, and L. Pastor, "Polynomial cases for the vertex coloring problem," *Algorithmica*, vol. 81, no. 3, pp. 1053–1074, Mar. 2019, doi: 10.1007/s00453-018-0457-y.

[22] M. Cavers and K. Seyffarth, "Reconfiguring vertex colourings of 2-trees," *Ars Mathematica Contemporanea*, vol. 17, no. 2, pp. 653–698, Dec. 2019, doi: 10.26493/1855-3974.1813.7ae.

[23] C. C. Bolton, G. Gatica, and V. Parada, "Automatically generated algorithms for the vertex coloring problem," *Plos One*, vol. 8, no. 3, Mar. 2013, doi: 10.1371/journal.pone.0058551.

[24] P. San Segundo, "A new DSATUR-based algorithm for exact vertex coloring," *Computers and Operations Research*, vol. 39, no. 7, pp. 1724–1733, Jul. 2012, doi: 10.1016/j.cor.2011.10.008.

[25] V. Kralev, R. Kraleva, V. Ankov, and D. Chakalov, "An analysis between exact and approximate algorithms for the k-center problem in graphs," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 12, no. 2, pp. 2058–2065, Apr. 2022, doi: 10.11591/ijece.v12i2.pp2058-2065.

[26] C. Yang, B. Yao, and Z. Yin, "A new vertex distinguishing total coloring of trees," *AIMS Mathematics*, vol. 6, no. 9, pp. 9468–9475, 2021, doi: 10.3934/math.2021550.

[27] K. S. Lyngsie and L. Zhong, "Vertex colouring edge weightings: a logarithmic upper bound on weight-choosability," *The Electronic Journal of Combinatorics*, vol. 28, no. 2, Apr. 2021, doi: 10.37236/6878.

[28] C. Lund and M. Yannakakis, "On the hardness of approximating minimization problems," in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing-STOC '93*, 1993, pp. 286–293, doi: 10.1145/167088.167172.
[29] R. L. Brooks, "On colouring the nodes of a network," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 37, no. 2, pp. 194–197, Apr. 1941, doi: 10.1017/S030500410002168X.
[30] B. Reed, "A strengthening of brooks' theorem," *Journal of Combinatorial Theory, Series B*, vol. 76, no. 2, pp. 136–149, Jul. 1999, doi: 10.1006/jctb.1998.1891.
[31] B. Reed, "ω, Δ, and χ," *Journal of Graph Theory*, vol. 27, no. 4, pp. 177–212, Apr. 1998, doi: 10.1002/(SICI)1097-0118(199804)27:4<177::AID-JGT1>3.0.CO;2-K.
[32] A. Frieze and C. McDiarmid, "Algorithmic theory of random graphs," *Random Structures and Algorithms*, vol. 10, no. 1–2, pp. 5–42, Jan. 1997, doi: 10.1002/(SICI)1098-2418(199701/03)10:1/2<5::AID-RSA2>3.0.CO;2-Z.
[33] Z. Huanping, X. Dangqin, and S. Huojie, "Strong vertex-distinguishing total coloring algorithm for complete graphs based on equitable coloring," *Journal of Engineering Science and Technology Review*, vol. 13, no. 1, pp. 126–132, Feb. 2020, doi: 10.25103/jestr.131.17.
[34] M. Miri, K. Mohamedpour, Y. Darmani, and M. Sarkar, "DIAMOND: a distributed algorithm for vertex coloring problems and resource allocation," *IET Networks*, vol. 8, no. 6, pp. 381–389, Nov. 2019, doi: 10.1049/iet-net.2018.5204.

BIOGRAPHIES OF AUTHORS

Velin Kralev is an associate professor of Computer Science at the Faculty of Mathematics and Natural Sciences, South-West University, Blagoevgrad, Bulgaria. He defended his Ph.D. Thesis in 2010. His research interests include database systems development, optimization problems of the scheduling theory, graph theory, and component-oriented software engineering. He can be contacted at email: velin_kralev@swu.bg.

Radoslava Kraleva is an associate professor of Computer Science at the Faculty of Mathematics and Natural Sciences, South-West University "Neofit Rilski", Blagoevgrad, Bulgaria. She defended her Ph.D. Thesis "Acoustic-Phonetic Modeling for Children's Speech Recognition in Bulgarian" in 2014. Her research interests include child-computer interaction, speech recognition, mobile app development and computer graphic. She is an editorial board member and reviewer of many journals. She can be contacted at email: rady_kraleva@swu.bg.