On Graph Coloring Analysis Through Visualization

Adam Dudáš
Department of Computer Science
Faculty of Natural Sciences
Matej Bel University
Banská Bystrica, Slovakia
email: adam.dudas@umb.sk

Jarmila Škrinárová
Department of Computer Science
Faculty of Natural Sciences
Matej Bel University
Banská Bystrica, Slovakia
email: jarmila.skrinarova@umb.sk

Adam Kiss
Department of Computer Science
Faculty of Natural Sciences
Matej Bel University
Banská Bystrica, Slovakia
email: adam.kiss@student.umb.sk

***Abstract*—The focus of the presented article is put on the analysis of edge coloring of selected sets of graphs - we are specifically interested in edge 3-coloring of graphs called snarks. Previous research suggests, that while using a single coloring algorithm and using various initial graph coloring edges, coloring of such graph may take anywhere from time lower than one millisecond to the time ranging in hundreds of milliseconds. In our case, we use recursive backtracking coloring algorithm based on breadth-first search and implement the change of initial graph coloring edge via permutation of adjacency matrix of graph. In this article, we present a tool created for the needs of analysis of edge coloring of graphs which is based on visualization of edge coloring and we present several problematic subgraphs and patterns which increase the time of edge coloring of cubic graphs.**

I. Introduction

Graph theory includes number of problems which can be solved by operations on graphs. In our research we deal with edge coloring of graphs, which is relevant in several areas - scheduling, radio frequency allocation, compiler optimization or SAT solvers. [1], [2].

Edge coloring of graphs is an NP-complete problem [3], which can be simply defined as assigning colors to the edges of given graph. In the presented research, we specifically deal with proper edge *k*-coloring of selected sets of graphs - edge *k*-coloring is proper when no adjacent edges are colored with the use of same color of *k* colors used for given graph (see Fig.1).

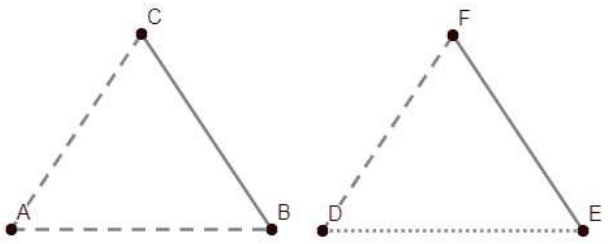

Figure 1. Example of improper (left) and proper coloring of the same graph

Fig. 1 presents graph colored with two different edge colorings - on the left is the graph described by the vertices *(A, B, C)*, which is edge colored improperly. Coloring of edges incident to vertex *A* is problematic since the two edges are colored with the use of the same color (in this case represented by a dashed line). On the same graph, now described by the vertices *(D, E, F)* the proper edge *k*-coloring is presented - where *k* is equal to three - therefore, we use three colors represented by dashed line, dotted line and solid line.

We use a specific type of graphs, which are suitable for our purposes - a group of cubic graphs (graphs in which each vertex is incident with exactly three edges) called snarks. Snarks are cubic graphs, which cannot be colored with the use of three colors [4]. However, there is no algorithm which is able to compute the "non-colorability" of a graph directly - the standard procedure for determining whether a cubic graph is a snark or not is such that the graph is edge-3-colored by possible colorings until we exhaust all possibilities. In the case, that at the end of this process the graph is edge 3-colored properly, then it is not a snark, otherwise the graph in question is a snark. Such coloring is thus an extreme case of edge coloring, in which it is necessary to recolor the graph several times.

In the research presented in this article, we use a simple recursive edge backtracking algorithm. It is known that while using this algorithm, the time of computation of edge *k*-coloring of a graph depends on the initial edge of coloring of the graph. The differences in these computational times of the edge *k*-coloring of the graph range from less than one millisecond to several hundred milliseconds.

This article presents a tool created for the needs of analysis of this edge coloring of graphs based on visualization of edge coloring and searching for subgraphs and patterns, which prolong computation of the problem. Presented paper is structured as follows:

- In the section II of the paper, we present our previous research in the area and research of other authors, which is relevant to problems described in the paper.
- Section III is focused on the principle of proper edge k-coloring of graph, cubic graphs, edge backtracking algorithm used in the proposed solution and change of initial edge of coloring of graph.
- Section IV describes design and implementation of proposed tool for edge coloring visualization purposes.
- In the section V, we compare proposed tool with other, comercialy available tools focused on graph visualization and present several subgraphs, which are problematic in the edge k-coloring of graph.

978-1-6654-3692-2/21/$31.00 ©2021 IEEE

II. Related Works

Graph coloring is NP-compelete problem, which can solved with the use of several algorithms such as:

- Edge-color algorithm presented by author of [5]. This algorithm uses polynomial space which improves over the previous, O($2^{n/2}$) algorithm of authors Beigel and Eppstein [6]. Author of [5] uses natural approach of generating inclusion-maximal matchings of the graph.
- Different approach to the graph coloring was introduced by authors of [7] who present simple but empirically efficient heuristic algorithm for the edge-coloring of graphs. The basic idea of this algorithm is the displacement of so called conflicts (adjacent edges colored with the use of same color, see. Fig 1) along paths of adjacent vertices whose incident edges are recolored by swapping alternating colors - with the use of Kempe interchange.
- Simple backtracking approach to edge coloring of graph, which uses recursive functions was presented in several research articles and publications [8], [4]. We describe this algorithm in detail in the section III of this paper.

Research presented in this article is continuation of research related to use of parallel and distributed computing in coloring of cubic graphs and visualization of graphs presented in:

- In the paper [9], we briefly introduced the proper edge coloring of graphs in the context of parallel and distributed computations while using various initial edges for coloring of the graph implemented via permutation of adjacency matrix of graph.
- The paper [10] presented use of adjacency matrix permutation as a way to minimize time of computation of proper edge coloring of large sets of graphs.
- Research presented in the [11] was focused on the visualization of graphs with specific objectives - clarity of diagram of graph, possibility of various input formats for graph visualization and possibility of simultaneous visualization of sets of graphs.

III. Edge Coloring of Cubic Graphs

Graph *G*, as we are considering it in the scope of the presented research, consists of [12]:

- vertices - elements of set *V(G)*. In the Fig. 1 vertices are labeled with capital letters A, B, C, D, E and F.
- edges – elements of set *E(G)* - edge is connection between two vertices and we can label it with the use of labels of these two vertices.

Therefore, graph *G* is pair of sets *V* and *E*, where elements of the set *E* are double element subsets of the set *V* [13]:

$$G = (V, E), E \subseteq [V]^2 \tag{1}$$

The concept of degree of vertex represents number of edges incident to the given vertex (since we work strictly with undirected graphs, by incident, we mean connected to the vertex in any way). In the case vertex V is of degree equal to three, we use notation *deg(V) = 3*. Highest (maximal) degree of vertex in graph *G* is denoted by $\Delta(G)$. In this paper we consider strictly cubic graphs. Graph is cubic when all of it's vertices are of degree equal to three.

Main objective of presented research is edge coloring of cubic graphs - operation of assignment of colors to individual edges of graph. Coloring is called proper when there is no conflict in the coloring of given graph, this means that no vertex of given graph is incident to two or more edges colored with the same color. Lowest number of colors usable in proper edge coloring of graph *G* is called edge chromatic index of graph G. For this property of graphs, we use notation $\chi'(G)$ [13].

Vizing's theorem [13], which says that minimal number of colors needed for coloring of graph is in the interval $<\Delta(G), \Delta(G)+1>$, holds true. Formal notation of Vizings' theorem focused on minimal number of colors:

$$\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1 \tag{2}$$

where $\Delta(G)$ is maximal degree of vertex in the graph *G* and $\chi'(G)$ is edge chromatic index of the graph *G*. Since every vertex of cubic graph is of degree equal to three, we consider three or four colors for proper coloring of cubic graph.

There is only small group of cubic graphs which need four colors for their proper edge coloring. Graphs from this group of cubic graphs are called edge 3-uncolorable graphs or snarks [4].

Chromatic index of snarks is $\chi'(G) = 4$. In order to find out whether given graph *G* is snark, we need to edge color it with the use of three colors. Therefore coloring algorithm needs to verify every possibility of edge 3-coloring of the graph *G*. Algorithms are able to decide whether the graph is snark or not after checking all possible edge colorings with the use of three colors.

The subsection A servers as an introduction to algorithm used for proper edge 3-coloring of graphs and in the subsection B, we introduce principle and method for the change of initial edge of coloring.

A. Edge Coloring of Cubic Graphs Using Edge Backtracking Algorithm

In the presented paper, we use algorithm on the basis of breadth-first search called edge backtracking algorithm as algorithm for edge coloring of graphs.

Edge backtracking algorithm works on the basis of edge coloring of graph with predetermined succession of three colors. In the case, that algorithm finds conflict in the coloring of graph, it backtracks to the previous edge, recolors the edge and continues in coloring. If there are no possible proper colorings of problematic edge, algorithm backtracks even further back - to the edge which precedes both of the recolored edges.

Algorithm continues in this approach until either whole graph is colored properly, or until algorithm examines all possible ways of edge coloring of given graph.

Time complexity of edge backtracking algorithm is $O(2^{n-1})$, where *n* is number of vertices of given graph.

Algorithm itself is represented in these steps:

978-1-6654-3692-2/21/$31.00 ©2021 IEEE

1) Algorithm takes three colors and colors consecutive edges of graph until:
   - either graph is colored properly,
   - or there is conflict in the graph coloring.
2) In the case of conflict in the coloring of graph, algorithm backtracks to edges which were already colored and recolors them with the use of next color in predetermined succession of colors until:
   - either conflict is solved – in this case, algorithm can continue in further edge coloring of graph as stated in the first step of the algorithm.
   - or all possibilities of edge coloring of the graph are improper. In this case the colored graph is snark (cubic graph which cannot be properly colored with the use of three colors).

B. Change of Initial Edge of Coloring

While using algorithm presented in the subsection A the time of computation of edge k-coloring of a graph depends on the initial edge of coloring of the graph. We showed this in the work [10] and [11]. The differences in these computational times of the edge k-coloring of the graph range from less than one millisecond to several hundred milliseconds. For the change of initial edge of coloring, we use graph isomorphism.

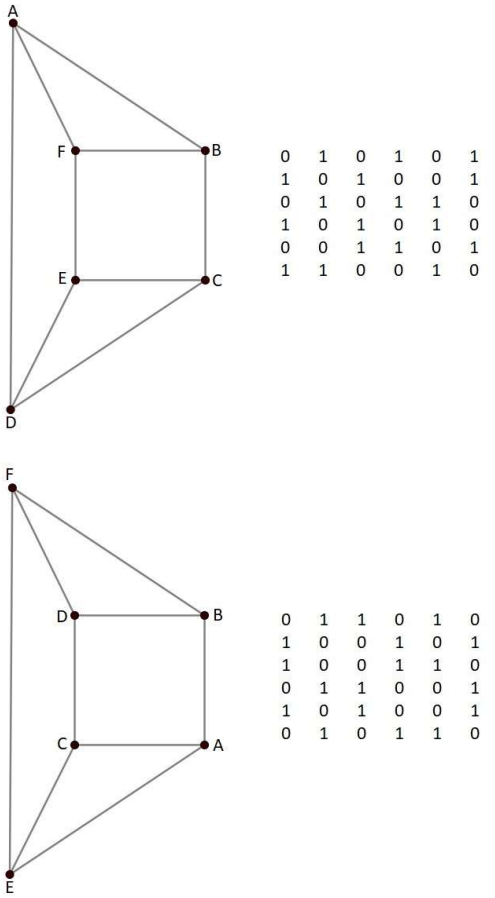

Figure 2. Example of isomorphic cubic graphs and their adjacency matrices

Let the graph G' and the given graph G be isomorphic graphs. An important feature of isomorphic graphs is that any pair of such graphs has different adjacency matrices but identical diagrams (presented in the Fig. 2 - example of isomorphic cubic graphs for simple visualization). In our case, this means that these are identical graphs with different sequences of vertices and edges. If the graph G is represented by the adjacency matrix A, we can create a different sequence of edges of the graph G (an isomorphic graph G') by permuting the adjacency matrix of the graph G.

This permutation of adjacency matrix A of graph G can ve computed with the use of following matrix multiplication formula:

$$A' = P^{-1} * A * P \tag{3}$$

where A' is permuted adjacency matrix of G, A is original adjacecny matrix of graph G, P represents (randomly generated) permutation matrix (this matrix needs to be generated in proper format - containing exactly one value 1 in each row and column of the matrixm otherwise filled with the value 0) and P^{-1} is transposed permutation matrix P.

The permuted adjacency matrix A' obtained in the described way represents the changed order of the edges of the original graph G. Hence, while coloring graph represented by A', algorithm uses different initial edge for coloring of the graph as in the case when the matrix A is used.

IV. Tool for Visualization of Graph Coloring

In this section of the article we describe the design and implementation of the software tool which can be used to analyze edge coloring of graphs. While designing and implementing the tool, we focused on several basic criteria applicable to graph drawing tools:

- Input formats - it is necessary to be able to insert data in various formats of graph notation. The standard input formats in commercially available tools are the adjacency matrix of graph, the incidence matrix of graph, or the adjustment list of graph. However, in addition to these formats, there is number of formats that are practical and space effective for storing graphs in the memory of a computer.
- Drawing algorithm - there is large number of various graph drawing algorithms, which are the basis for graph visualization in the form of a diagram. The main required properties of graph drawing algorithms are the lowest possible computational time needed to draw the graph and clarity of the diagram itself - the clarity of the diagram is implemented as minimization of edge crossing in the diagram and maximization of symmetry of the diagram of graph.
- Specific functions - all available tools which can be used for graph visualization also contain a set of implemented graph functions. Such functions include shortest path search in the graph, shortest cycle search in the graph, Hamiltonian cycle search and so on. Rarely is any form of graph coloring among the implemented functions -

when it is included in the set of functions, only the vertex coloring is implemented.

In the rest of this section, we describe how we proceeded in design and implementation of these three criteria in the case of our proposed software tool for graph coloring.

A. Design of Tool for Visualization of Graph Coloring

Since our tool for graph visualization has a specific objective of edge coloring visualization, we focused on adapting the criteria described above to this goal.

Within the presented tool, we require three input formats for visualization of graph:

- Adjacency matrix and adjacency list - these are conventional methods of representation of graphs. The adjacency matrix is a matrix of size n x n, where n is the number of vertices of given graph. Each row and column of this matrix represents one of the vertices of the given graph - in the case there is an edge connecting two vertices in the graph, the value 1 is recorded in the adjacency matrix, the value 0 is recorded otherwise. The adjacency list is a list of vertices with assigned names of the vertices to which they are connected by an edge.
- Graph6 format - with the use of this format, we are able to store graphs in a compact manner, using vectors of printable ASCII characters. A graph in graph6 format is stored in a single line of a file, which allows us to efficiently store entire sets of graphs in a single file. Although this graph format is unreadable to humans (as opposed to the adjacency matrix and mainly adjacency list), its advantages are versatility and efficiency of use.

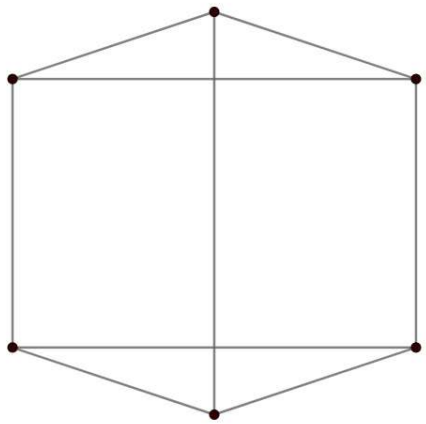

Figure 3. Example of use of circular layout drawing algorithm applied on the graph from Fig.2

After reading input graph in one of the selected formats, the application draws the graph using the selected graph drawing algorithm. For the purposes of this work, we have chosen circular layout - a popular method of graph drawing, which offers high clarity of diagram of graph (Fig. 3). Regarding number of edge crossings in the graph - this algorithm is able to draw a graph with no edge crossings only in the case of outerplanar graphs. In other cases, edge crossing occurs, but it is possible to implement optimization techniques which minimize number of crossings.

In this way we implement clarity of the diagram of graph based on minimization of edge crossing in the diagram and maximization of symmetry of the diagram of graph.

The third - and arguably the most important - criterion of the design of the graph visualization tool is the specification of a set of functions which can be used while working with the tool:

- Translation between formats of graph description - the examined graph is inputted in one of three possible formats (adjacency matrix, adjacency list or graph6 format). Due to the readability of the graph and in order to increase other possibilities of working with the graph, it is practical to translate the examined graph from the input format into the other two defined formats.
- Edge coloring of graph - the core of the presented application is edge coloring of graphs. The Edge Backtracking Algorithm presented in section III, subsection A is used for implementation of this action.
- Visualization of edge coloring of graph - an extension of the standard edge coloring of graph is the element of visualization of the computed coloring. For the edge colored graph, a graph diagram is drawn and the edges of this diagram are colored according to the computed coloring. In such case, that the graph is not proper edge k-colorable, the edges of the graph remain colored with the use of default color. The user is notified that there is no proper edge k-coloring for the given graph.
- Step-by-step edge coloring - from the point of view of analysis of edge coloring, it is necessary for the user to be able to browse through a specific computing coloring in the step-by-step manner. Each step represents coloring or recoloring of one of the edges of the graph visualized in the application. Since edge coloring can take several tens to several thousand of steps, it is advisable to implement the stepping options set to 1 step, 5 steps, 10 steps and 50 steps at a time. It is also necessary to be able to step edge colorgin in both directions (forward and backward) with the possibility of 1, 5, 10 or 50 steps at a time.

B. Implementation of Tool for Visualization of Graph Coloring

In the implementation of the required solution, we followed the design presented in the section IV, subsection A. The application was implemented using C/C++ language as follows:

- The input graph for the program is specified in a text file, which is located in the directory with executable file of the program. This graph can be stored in adjacency matrix, adjacency list or graph6 format.
- After starting the program, the input file is loaded and the format in which the graph is stored is specified. In the case of the adjacency matrix, the program proceeds to the next step of computation. In other cases, the input format is translated to adjacency matrix and the computation continues. In this step, two new files are created - each

978-1-6654-3692-2/21/$31.00 ©2021 IEEE

Table I
COMPARISON OF FUNCTIONALITIES OF OPEN SOURCE, PAID AND PROPOSED TOOLS

	Graph Online	CS Academy	Matlab	Wolfram Mathematica	Proposed Tool
Input: adjacency matrix	YES	YES	YES	YES	YES
Input: graph6 format	NO	NO	NO	YES	YES
Edge coloring of graph	NO	NO	YES	YES	YES
Step analysis of edge coloring	NO	NO	NO	NO	YES

containing the graph stored in one of the two remaining formats.
- Since the graph is represented as an adjacency matrix, it can be edge colored using the $edgeColorise$ function. The function uses Edge Backtracking Algorithm and edge 3-colors the graph. The output of this function is information on whether the graph can be colored with the use of three colors or not and a set of steps for edge coloring of the graph. These steps are later visualized.
- The graph is then visualized using the $drawVertices$ and $drawEdges$ functions in the graphical user interface run by the $drawGUI$ function. This GUI contains space for plotting graphs and buttons for stepping options which consist of eight buttons: +1, +5, +10, +50 and -1, -5, -10, -50 steps.
- Finally, the $drawSteps$ function is called, which uses step data fo edge coloring computed in the $edgeColorise$ function and, after the user interacts with the GUI buttons, draws the individual coloring steps.

The application uses two windows for its operation. The first window serves as a console in which the user is provided with information about the input graph - number of vertices, adjacency matrix, adjacency list and graph6 format of the graph, computed information about edge 3-colorability of the graph, number of steps needed in edge coloring of graph and current step of coloring visualized in the other window of application.

The second window serves as a GUI - the visualization of the graph itself takes place in it and it provides buttons for step-by-step edge coloring.

V. EXPERIMENTS ON GRAPH COLORING VISUALIZATION

We test the proposed application from the two perspectives. The first perspective is to compare the basic functionalities of the presented tool and a group of other - open source or paid - tools that are designed for a similar purpose. The properties, we focuse on include input data formats for tools, functions implemented in individual tools, the possibility of analyzing graphs in the tool or the duration of edge coloring of graphs in the application.

The second point of view of application testing is the use of the presented application in the identification of several patterns and subgraphs that cause an increase in the time of computation of edge coloring of graphs.

A. Comparison of Proposed Graph Visualization Tool with Other Available Tools

There are several tools available that are commonly used to draw graphs - whether open source tools like $GraphOnline$ or $CSacademy$, or professional paid tools like $Matlab$ or $WolframMathematica$. All of these tools share common features with small variations.

In the Tab.I we present the properties available in the individual tools and compare them with each other and at the same time with the application proposed in this paper.

From the Tab.I it is clear, that we can divide the properties essential for our research into following three groups:
- Input data formats for tools - each of the tools is able to work with an adjacency matrix as input, but only a minority of tools can work with input data in the graph6 format. This format is practical and space effective way of storing a graph in the memory of computer - it encodes an adjacency matrix of graph into a one-line character string. This format is supported by $WolframMathematica$ and proposed application.
- Functions implemented in the tool - each of the presented tools contains several implemented functions, that can be applied on graphs (searching for the shortest path in the graph, searching for the shortest cycle of graph, and so on). The function that interests us is the edge coloring of graph. This function is present only in the tools $Matlab$, $WolframMathematica$ and the presented application. Although it is not possible to edge color a graph in the mentioned open source tools, a function for the vertex coloring of the graph is present in the $GraphOnline$ tool. Thus, we would be able to project the input graph into the line graph [13] and thus obtain the edge colorability of the graph. However, such a method of coloring is not visual and the transformation of the graph itself is not available in the given tool.
- Specialized analytical functions - none of the available tools offers the possibility of visualization of the edge coloring of graph step-by-step - therefore it is not possible to analyze the coloring in any way. This feature is unique among graphing tools.

It is important to mention that the possibility of step-by-step visualization of edge coloring of the graphs is advantageous from the point of view of the analysis of the coloring itself, but it brings increased costs in the form of substantial increase of computational time needed for the problem. In the Tab.II we present a comparison of computation times for coloring of the followings: Petersen graph - simplest example of snark (10

vertices), the first Blanuša snark (18 vertices) and 34-vertex snark generated by the $snarkhunter$ tool [15]. All of these graphs are cubic, so the number of edges which are colored, can be computed as $(3/2) * n$, where n is number of vertices of the graph.

It is clear that the presented tool needs much higher time to color the graph itself. This is due to the need to compute and store all the edge coloring steps of the input graph so that the graph can be subsequently analyzed. We do not consider this shortcoming to be important at the moment, as this tool is used to analyze graphs and not as a tool which computes the edge colorability of a graph in the shortest possible time (we presented such an algorithm in [10]).

Table II
COMPARISON OF COMPUTATION TIME OF EDGE COLORING FOR CHOSEN SNARKS

	Wolfram Mathematica	Proposed Tool
Petersen graph	550 ms	670 ms
Blanuša snark	540 ms	840 ms
34-vertex snark	1080 ms	12 440 ms

B. Patterns and Subgraphs of Interest

In this subsection, we present point of view of application testing focused on the use of the presented application in the identification of patterns and subgraphs that cause an increase in the time of computation of edge coloring of graphs.

In the Tab. III we present the number of necessary recolorings of the edges of individual graphs needed for the algorithm to be able to determine whether the graph is edge 3-colorable or not. These values are measured on graphs in standard form - in the form without applying any permutation on the graph - while using Edge Backtracking Algorithm presented in the section III, subsection A.

Table III
COMPARISON OF NUMBER OF RECOLORINGS FOR CHOSEN GRAPHS IN STANDARD FORM

	Number of Recolorings
Petersen graph	91
Blanuša snark	1029
34-vertex snark	18 253

The graph with the lowest number of vertices we worked with was 10 vertex (15 edge) snark - Petersen graph. Edge 3-colorability (or more precisely uncolorability) of this graph was computed in 91 steps. On the Fig. 4 we see that the edge coloring of the graph was started on the edge marked with vertices $(Initial_S, Initial_E)$ and the graph was being properly edge colored until the coloring of edge represented by dashed line. Uncolored edges are marked by a dotted line.

The coloring of this graph was done in 91 steps - out of these 91 step 80 were dedicated to the recoloring of graph related to the problematic edge marked with the use of dashed line. This represents 87.9 percent of computation steps needed for the edge 3-coloring of the graph.

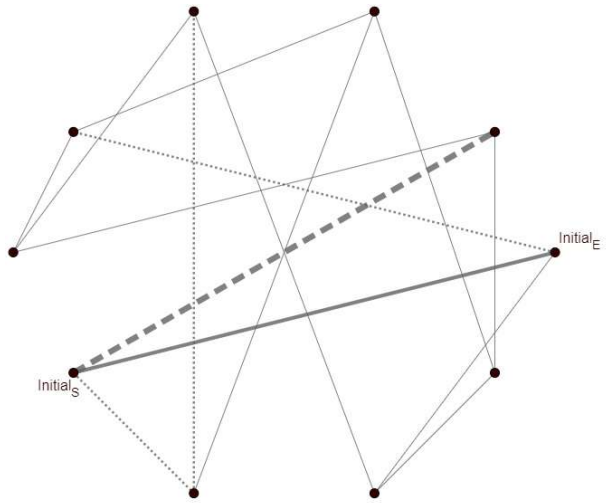

Figure 4. Analysis of edge coloring of Petersen graph

From the Fig.4, we can identify some properties and information about the problematic edge and edge coloring of this graph:

- problematic edge is incident with initial edge of coloring,
- three of four edges which were not colored are incident to the initial edge of coloring,
- fourth edge, which was not colored is incident to other uncolored edge.

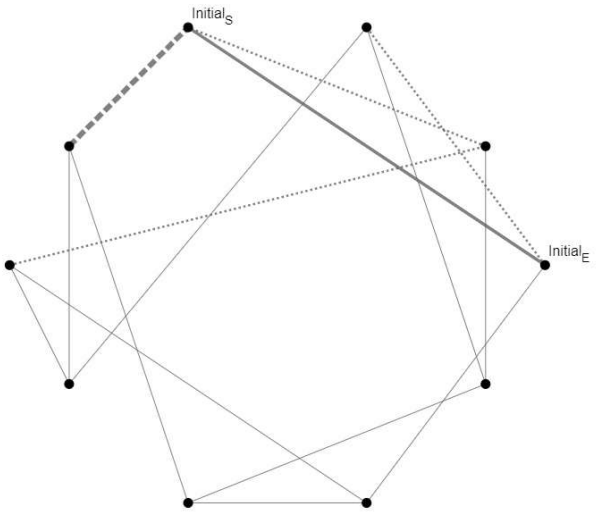

Figure 5. Analysis of edge coloring of permuted Petersen graph

To compare this edge coloring of Petersen graph, we used edge 3-coloring of the permutation of the same graph as presented in the section III, subsection B (see Fig.5). The permutation used for the coloring of the graph was randomly generated, while complying the criteria for permutation matrix presented in the section III, subsection B. The edge coloring of graph presented in the Fig.5 was computed in 73 steps. After analysis of this edge coloring we can identify same properties and information about the problematic edge and edge coloring

978-1-6654-3692-2/21/$31.00 ©2021 IEEE

of this graph as in previous case. The only difference is number of steps needed in order to find the edge coloring of the graph (see Tab.IV).

Table IV
COMPARISON OF NUMBER OF RECOLORINGS AND TIME OF COMPUTATION OF EDGE COLORING FOR PETERSEN GRAPH IN STANDARD FORM AND PERMUTED FORM

	10-vertex, 15-edge snark
Number of Recolorings in SF	91
Number of Recolorings in PF	73
Edge Coloring Time in SF	670 ms
Edge Coloring Time in PF	660 ms

As we can see from the Tab.IV, the number of steps needed for computation was lower while using permuted graph, but the time of computation of the edge coloring of graph was almost the same. The values of computational time presented in the Tab.IV were computed as an average from five measurements.

In order to further test the application we used the first Blanuša snark in the same way as the Petersen graph above. The first Blanuša snark consists of 18 vertices and 27 edges. The edge coloring of the graph was computed in 1029 steps. Out of these 1029 steps, steps 1-64 colored edges properly with minor recolorings needed. When algorithm found the problematic edge, it took remaining 965 recolorings to try and color this edge (which corresponds to 93.8 percent).

Before recoloring, only four edges are not colored properly - in this case, problematic edge is not incident to the initial edge of coloring (as oppose to the first examined case), three of these four edges are incident to the initial edge of coloring and fourth edge, which was not colored (the problematic one) is incident to other uncolored edge. We present this coloring on the Fig.6.

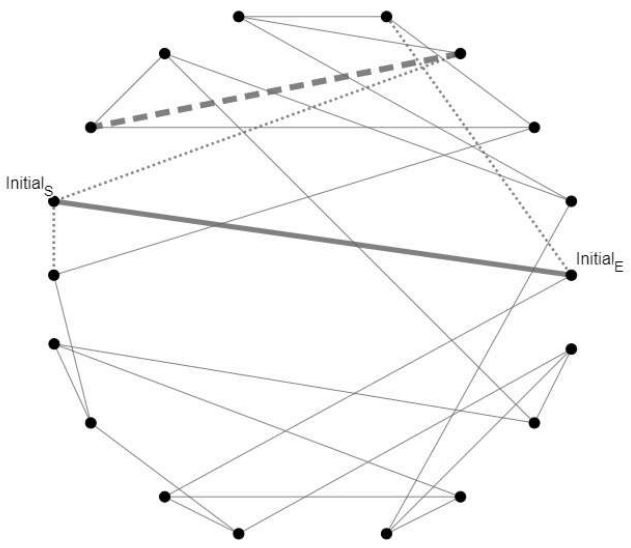

Figure 6. Analysis of edge coloring of first Blanuša snark

We also edge colored permuted first Blanuša snark. The edge coloring of the permuted graph was computed in 401 steps and contained no single problematic edge. While coloring this graph, algorithm encountered number of smaller problems in coloring which were recolored in 14 - 60 steps. Therefore, we need to study permutation used in this case and try to generalize its properties in order to further optimize the edge 3-coloring of graphs.

In the Tab.V we present comparison of computation times for edge coloring of the first Blanuša snark and comparison of number of step needed for edge coloring of given graph.

Table V
COMPARISON OF NUMBER OF RECOLORINGS AND TIME OF COMPUTATION OF EDGE COLORING FOR THE FIRST BLANUŠA SNARK IN STANDARD FORM AND PERMUTED FORM

	18-vertex, 27-edge snark
Number of Recolorings in SF	1029
Number of Recolorings in PF	401
Edge Coloring Time in SF	840 ms
Edge Coloring Time in PF	710 ms

VI. CONCLUSION

Main objective of the article was to present a tool created for the need of analysis of edge 3-coloring of graphs based on visualization of edge coloring and searching for subgraphs and patterns, which prolong computation of the chosen problem.

We designed and implemented software tool, which can be used for step-by-step analysis of edge coloring of graphs. On the basis of this step analysis, we can decide which edge of the graph is most fitting as an initial edge of coloring of given graph.

In the section V, subsection B, we presented use of step-by-step analysis on chosen types of graphs - Petersen graph and the first Blanuša snark. With the use of our tool, we were able to identify some properties and information about the problematic edges and edge coloring of these graphs.

Even though the function of step analysis of edge coloring of graph is unique and practical, it also has some drawbacks. Main shortcoming of proposed tool is time needed for computation of all steps of coloring - in some cases this visualization is computed in the time 10-times higher than standard edge coloring of graph.

Future work in this area contains:

- Use of parallel and distributed computing for optimization of computational time needed for visualization of edge coloring of graph similar to [10], [16]. The edge coloring can be computed in advance and the visualization of individual steps of the coloring can be done in parallel via either simple thread-based model or via use of distributed computing.
- Implementation of methods of artificial intelligence, which would be able to analyze graph coloring and propose fitting permutation of given graph. We call permutation fitting in such case, that after applying it on the

graph (via formula 3), the computational time for edge coloring of the graph is reduced.
- Using proposed model for analysis of large sets of edge colored graphs.
- Implementation of user interface for the general use [17] - mainly formats of data input for application are not user friendly at the moment.

ACKNOWLEDGMENT

The research was partially supported by the grant of The Ministry of Education, Science, Research and Sport of Slovak Republic - Implementation of new trends in computer science to teaching of algorithmic thinking and programming in Informatics for secondary education, project number KEGA 018UMB-4/2020.

Computing was performed in the High Performance Computing Center of the Matej Bel University in Banská Bystrica using the HPC infrastructure acquired in project ITMS 26230120002 and 26210120002 (Slovak infrastructure for high-performance computing) supported by the Research & Development Operational Programme funded by the ERDF.

REFERENCES

[1] Marx D.: Graph colouring problems and their applications in scheduling. In: Periodica Polytechnica, Electrical Engineering, Vol. 48, 2004, No. 1-2, pp. 11-16.

[2] Chaitin G. J.: Register allocation & spilling via graph colouring. Proc.1982 SIGPLAN Symposium on Compiler Construction, pp. 98-105, 1982. ISBN 0-89791074-5

[3] Holyer I.: The NP-Completeness of Edge-Colouring. In: SIAM J.COMPUT, Vol. 10, 1981, No. 4, pp. 718–720. ISSN 0097-5397.

[4] Karabáš J.—Máčajová E.—Nedela R.: 6-decomposition of snarks. In: European Journal of Combinatorics: 20th International workshop on combinatorial algorithms (IWOCA), Elsevier, Vol. 34, 2013, No. 1, pp. 111–122. ISSN 0195-6698.

[5] Kowalik L.: Improved edge-coloring with three colors. In: Theoretical computer science, Vol. 410, 2009, No. 38-40, pp. 3733-3742. ISSN 0304-3975.

[6] Beigel R., Eppstein D.: 3-coloring in time O(1.3289^n). In: J. Algorithms 54(2), 168–204, 2005

[7] Fiol M. A.–Vilaltella J.: A Simple and Fast Heuristic Algorithm for Edge-coloring of Graphs. In: AKCE International Journal of Graphs and Combinatorics, Vol. 10, 2013, No. 3, pp. 263-272

[8] Nedela R., Karabáš J., Škoviera M.: Nullstellensatz and Recognition of Snarks. In: 52th Czech-Slovak Conference Grafy, 2017

[9] Dudáš A., Škrinárová J., Voštinár P., Siláči J.: Improved process of running tasks in the high performance computing. In: ICETA 2018: Proceedings: 16th IEEE International Conference on Emerging eLearning Technologies and Applications. pp 133-140. ISBN 978-1-5386-7912-8.

[10] Dudáš A., Škrinárová J, Vesel E.: Optimization design for parallel coloring of a set of graphs in the High-Performance Computing. In: Proceedings of 2019 IEEE 15th International Scientific Conference on Informatics. pp 93-99. ISBN 978-1-7281-3178-8.

[11] Dudáš A., Janky J., Škrinárová J.: Web applicaiton for graph visualization purposes. In: ICETA 2020: Proceedings: 18th IEEE International Conference on Emerging eLearning Technologies and Applications. pp 90-96. ISBN 978-0-7381-2366-0.

[12] Palúch S.— Peško Š.: Quantitative methods in logistics ($in\ Slovak$). EDIS, Žilina, Slovakia, 2006, ISBN 80-8070-636-0.

[13] Diestel R.: Graph theory. Springer - Verlag, Heidelberg, 2016, ISBN 978-3-662-53621-6.

[14] Häglund J.: On snarks that are far from being 3-edge-colorable. In: The Elcectronic Journal of Combinatorics, Vol. 23, 2016, No. 2, Paper P2.6. ISSN: 1077-8926.

[15] Brinkmann G.—Coolsaet K.— Goedgebeur J.—Mélot H.: House of Graphs: a database of interesting graphs, Discrete Applied Mathematics, 161(1-2):311-314, 2013 (DOI). Available at http://hog.grinvin.org

[16] Melicherčík M., Siladi V., Svítek M., Huraj L.: Spreading High Performance Computing Skills with E-Learning Support, ICETA 2018 - 16th IEEE International Conference on Emerging eLearning Technologies and Applications, 2018, pp. 361–366.

[17] Sedlacek P., Kmec M., Rusnak P.: Software Visualization Application for Threads Synchronization Handling in Operating Systems. ICETA 2020 - 18th IEEE International Conference on Emerging eLearning Technologies and Applications, 2020, pp. 580–585

