Visualization of graph algorithms

LE Van Linh NGO Tuan Hau

April 23, 2015

Abstract

Graph coloring is an interesting topic in graph theory. It can be vertex coloring, edge coloring or face coloring. The number of colors that we use to color the graph is important. Our program visualizes the Vizing's theorem indicating the number of colors for edge coloring of a valid graph. Here, a valid graph does not have to eject the same color of incident edges of a vertex.

1 Introduction

Graph theory is a combination of mathematics and computer science. It has been developed for a long time ago and many applications in this field are well developed. One of the topics is the graph coloring which is applied to many fields such as education, science and technology, etc.

Depending on the applications, it is flexible to choose the appropriate graph coloring algorithms to apply. However, the number of colors which applies to the projects needs to be considered. Vizing's theorem indicated how to use the number of colors to color the edges of a valid graph.

Our project is mainly based on the framework "Graph" from the supervisor. We applied the core of framework and the improvement of it to display on the graphical user interface (GUI). Furthermore, our program is a visualization for a coloring graph theorem, which is based on the paper of J. Misra and David Gries: "A constructive proof of Vizing's Theorem". It to simplify the algorithm by Béla Bollobás[2]. It displays the result on GUI. The functionalities are built as following:

- Draw the graph: we display it into the GUI.
- Implement the algorithm Breadth-first search (BFS): From the previous step, we implement the Breadth-first search algorithm[3]. On the other hand, we also construct the BFS tree.
- Implement the Misra and Gries edge coloring algorithm [4]

2 Definitions

Edge coloring: *"An edge coloring of a graph G is a coloring of the edges of G in such a way that two edges which share a vertex have different colors."*

Chromatic index: *"The chromatic index of a graph G, denoted by $\chi'(G)$ is the minimum number of colors necessary for the edge-coloring of G."*

The maximum degree of a graph G is denoted by $\Delta(G)$. For any graph G, $\chi'(G) \geq \Delta(G)$. It means that all the edges around a vertex with degree $\Delta(G)$ must have different colors.

Vizing's theorem[1] *"For any graph G, $\chi'(G) \leq \Delta(G) + 1$".*

Thus the only possible values for $\chi'(G)$ are $\Delta(G)$ or $\Delta(G)+1$.
To prove the Vizing's theorem, we use "A constructive proof of the theorem of Vizing"[4] to obtain a $\Delta(G)+1$ edge-coloring for graphs.

The fan[4]
They defined a data structure called "fan". A fan $< f..l >$ of X (X is a vertex), is a sequence of vertices that satisfies the following properties:

- F0: $< f..l >$ is a nonempty sequence of distinct neighbors of X
- F1: edge Xf is uncolored (the other edges of the fan are colored)
- F2: ($\forall$ u:: color of edge Xu^+ is free at u (u^+ is the successor of u in the fan))

For example, on the graph below (see figure 1). Edge AB is not colored (dashed line) and another edges are coloring. A fan $F1 = \{B, D, C\}$ of A is maximal fan. And $F2 = \{B, C\}$ is another fan of A but it is not maximal.

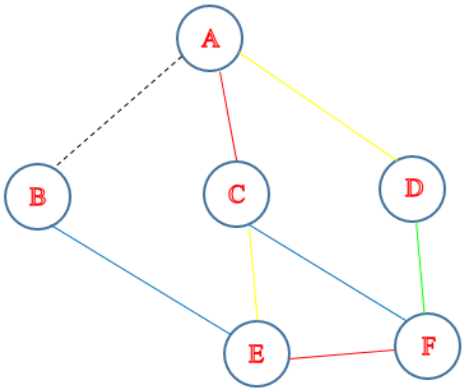


Figure 1: Graph with a edge uncolored

Inverting a path[4]
A cd-path of X is a path that includes vertex X, has edges colored only c or d, and is to be maximal. The operation "invert the cd-path" switches every edge on the path colored c to d and vice versa.
As above example (figure 1), $\{AD, DF\}$ is a yellow-green path of A. And after invert, the its color change from yellow to green and vice versa. As the followed result (see figure 2).

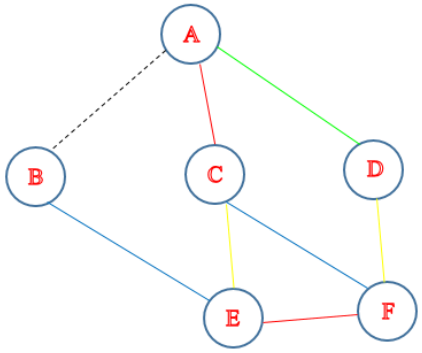


Figure 2: Graph after inversion the yellow-green path of A

Rotating a fan[4]
Given a fan $F[1:k]$ of a vertex X, the "rotate fan" operation does the following in parallel:

- $c(F[i], X) = c(F[i+1], X)$
- Uncolor $F[k]$

This operation leaves the coloring valid, as for each i, $c(F[i+1], X)$ was free on $(F[i], X)$.

With the graph in figure 2, we can easy find a fan of A: $F = \{B, C, D\}$. We apply the rotation on fan F. The result change the color of fan-edge as bellow image (see figure 3).

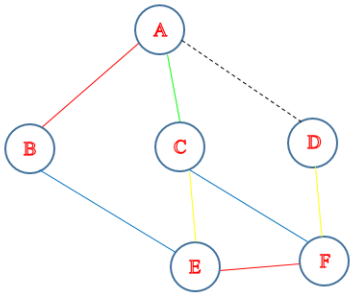


Figure 3: Graph G after rotation a fan $F = \{B, C, D\}$

The algorithm[4]

Data: A graph G
Result: A proper coloring c of the edges of G
Let $U := E(G)$
while *U is not empty* **do**

1. Let (u, v) be any edge in U
2. Let $F[1:k]$ be a maximal fan of u starting at $F[1] = v$
3. Let c be a color that is free on u and d be a color that is free on $F[k]$
4. Invert the cd_u path
5. Let $w \in V(G)$ be such that $w \in F, F' = [F[1]..w]$ is a fan and d is free on w
6. Rotate F' and set $c(v, w) = d$
7. $U := U - \{(u, v)\}$

end

Proof: Refer to [1].

The following example illustrate the algorithm. A graph with four vertices (A, B, C, D) and five edges including two edges are not coloring (AC, BD) and another edges are coloring (figure 4).
To reduce the time execution algorithm, we only consider the un-colored edges. So, the set of edges is $U = \{AC, BD\}$.
The round 1: consider AC edge.

- Maximal fan of A: $F = \{C\}$ (because blue not free on C)
- Color c is free on A, and color d is free on C. We choose c is red and d is yellow (because both red and blue are not free on C). The red-yellow path of A is empty.

- We choose $w = C$ because the fan F has only one member.
- Set the color of AC is yellow. The set $U = \{BD\}$

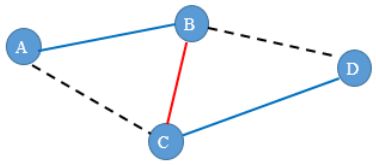


Figure 4: Graph G

After round 1, AC colored and the graph is following:

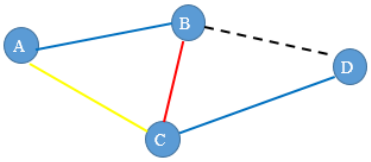
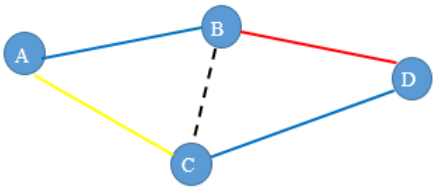


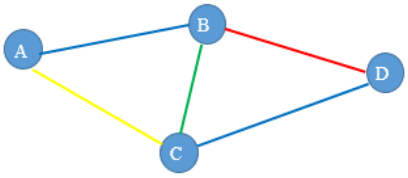
Figure 5: Graph G after AC colored by yellow

The round 2, consider BD edge:

- Maximal fan of B: $F = \{D, C\}$ (because blue not free on A and C)
- Color c is free on B, and color d is free on C. We choose c is yellow and d is green (because both red, blue and yellow are not free on C). The yellow-green path of B is empty.
- We choose $w = C$ because no fan edge has color green. The new fan is also F: $F_1 = F = \{D, C\}$
- Rotate fan F_1 (figure 6a) and color BC by green (figure 6b). The set $U = empty$



(a) Graph G after rotation the fan F_1



(b) Graph G after BD colored by green

Figure 6: The graph was colored after execute the edge coloring algorithm

The algorithm stopped by the set U empty.

3 Specifications

The crucial requirement of our program is to visualize the graph's algorithms. The input does not contain any information for the graph in the GUI. The program allows the user to display the graph on the GUI

which follows the style coding conventions defined by the programmer. The vertices are represented by the labeled circles. The transitions are represented by the straight lines, it also displays the labels for transitions.

This application represents the graph on the circle model. Depend on the number of vertices, program automatically defines the position of each vertex.

Program allows the users to run the breadth-first search on the GUI. The vertices automatically change its colors when program executes on it (push, pop a vertex). Besides, it also changes the color of edge to display the BFS tree.

At last, the application implements the Misra and Gries edge coloring algorithm. It calculates the number of colors to color the edges of graph and coloring it.

4 Implementation

4.1 Class diagram

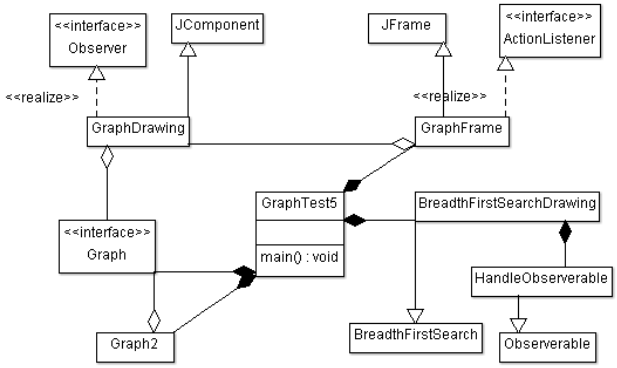


Figure 7: Class diagram of application

The class diagram (figure 7) show mainly classes of the functions in program. The *main* method located in the *GraphTest5* class . To present the graph on the GUI, we use *GraphDrawing* and *GraphFrame* class. For the breadth-first search, we have *BreadthFirstSearch* and *BreadthFirstSearchDrawing* class. Final, the edge coloring located in the *Graph2* class. Besides, we also have another classes to support the main classes in the diagram. The detail in the sections below.

4.2 Represent the graph on the GUI

Based on the project provided by the supervisor. Program defines the classes to implement the functions of program. To display the graph on the GUI, vertices and the edges must be have coordinate. By default, vertices represented by the background is yellow, the border is red and the color for label is blue.

Classes to represent the graph on GUI are:

- **VertexCoordinates**: represent the states with the position.
- **DirectedEdgeWithColor** and **DirectedEdgeWithLabel**: represent the transitions with the label and color.
- **GraphDrawing**: represent the graph

- **Drawing**: support the drawing methods (draw state, transition,...)

Our proposal for display the graph on GUI:

1. Get the number of vertices
2. Calculate the position of each vertex on GUI (based on the layout)
3. Re-create the vertices with the coordinate.
4. Drawing the states
5. Drawing the transition and label

4.3 The Breadth-first search algorithm

To implement the breadth-first search algorithm, we have added the classes **VertexWithAction**, **BreadthFirstSearch**, **BreadthFirstSearchDrawing** and added the methods into **GraphDrawing** class to represent the change when run algorithm.

Classes defined BFS algorithm are:

- **VertexWithAction**: creates a vertex on the action when algorithm run.
- **BreadthFirstSearch**: implements the BFS algorithm with two methods that are enqueue and dequeue a vertex.
- **BreadthFirstSearchDrawing**: extends from BreadthFirstSearch. It represents the algorithm step by step.
- **GraphDrawing**(the **update** method): updated the state of vertex when algorithm is running on it.

4.4 The Misra and Gries edge coloring algorithm

The algorithm indicates the number of color to coloring all the edges of the graph with at most $\Delta(G) + 1$ colours. Based on the set colors of edges, program can add a new color or not. After that, program applied the algorithm to coloring all the edges of the graph.

Classes implemented the edge coloring algorithm are:

- The previous classes: represent the graph, edge with coloring.
- **Graph2**: implements the algorithm

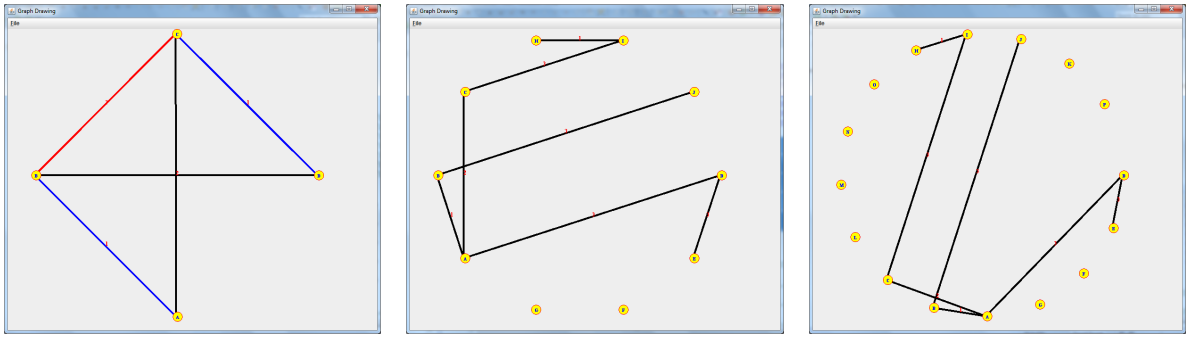
Our proposal for edge coloring:

1. Get the set color of edges
2. Get the maximal "fan" $<f..l>$ of vertex X
3. Get two colors are free on X (color c) and the vertex latest (color d) in fan of X
4. Get the cd-path of X
5. Invert the cd-path
6. Find the vertex v in fan such that it remains is fan and color d is free on v
7. Rotate the new fan and coloring the Xv with d

5 Results

In this section, we consider the function represent the graph on GUI. As you known in specifications, based on the number of states, program automatically pinpoint the vertex's exactly locality.

Sure, distance between states depend on the radius of circle. As example, with radius is 10 and the graphs are displayed following (see Figure 8).



(a) Graph with 4 vertices (b) Graph with 10 vertices (c) Graph with 16 vertices

Figure 8: The graph represented on GUI

For the graph with 4 states, the angular distance between states are 90 degree. Similar, the graph with 10 states, the distance between it is 360 degrees divide by 10, and so on.
The BFS algorithm executes on the graph, when it run on vertex, it make corresponding changes to a action. Besides, the colors of edge also change to form a BFS tree (see Figure 9).

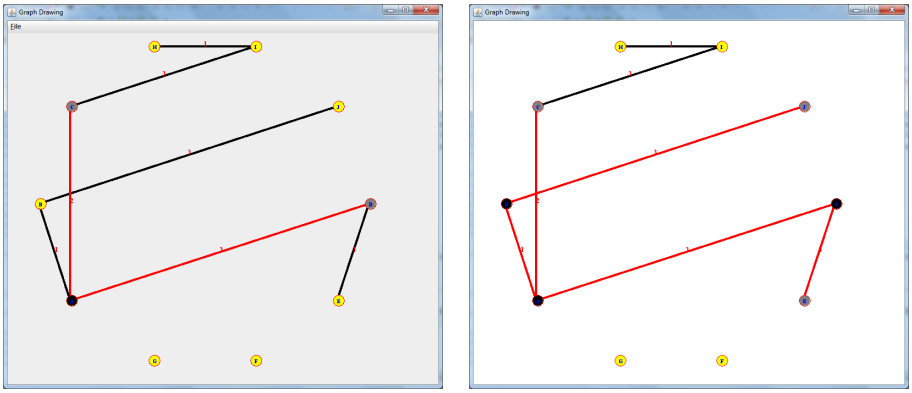


Figure 9: The BFS algorithm run on graph

When the BFS algorithm enqueue a vertex, its color change from yellow to gray. When dequeue it, its color change from gray to black. And, the edge connect by dequeue and enqueue vertex is change from black to red.
The edge coloring algorithm indicate the number of colors to color all edge of graph. After that, it colored the edge such that the graph is valid (Figure 10).

The edge have black color when it does not colored. We use the colors different black to colored the edge of graph. For every run, the algorithm colored a edge what does not colored. The end, we will have all edge colored. And the number of colors satisfy the Vizing's theorem.

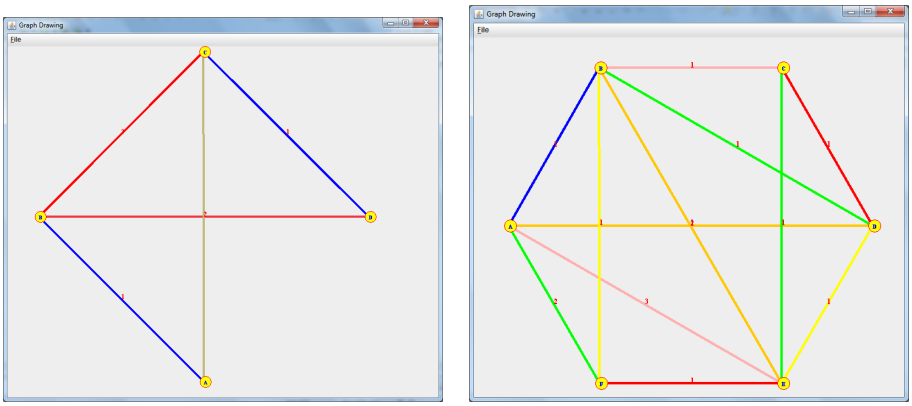


Figure 10: The graph was colored after execute the edge coloring algorithm

6 Conclusion

The application fully the work requirements. Some features are tested in console not in the GUI. The future work would mainly focus on the following functions:

- Change the way to put the parameters (on GUI).
- Optimize the code.
- Integrate more graph algorithm in project.

References

[1] Claude Berge and Jean Claude Fournier. A short proof for a generalization of vizing's theorem. *Journal of graph theory*, 15(3):333–336, 1991.

[2] Béla Bollobás. *Graph theory*, page 94. Elsevier, 1982.

[3] Shimon Even. *Graph algorithms*, pages 11–14. Cambridge University Press, 2011.

[4] Jayadev Misra and David Gries. A constructive proof of vizing's theorem. *Information Processing Letters*, 41(3):131–133, 1992.